Dipartimento di Informatica, Bioingegneria,
Robotica ed Ingegneria dei Sistemi

Computational and Theoretical Issues of Multiparameter Persistent Homology for Data Analysis

by

Sara Scaramuccia

Theses Series **DIBRIS-TH-2018-16**

DIBRIS, Università di Genova
Via Opera Pia, 13 16145 Genova, Italy http://www.dibris.unige.it/

Università degli Studi di Genova

Dipartimento di Informatica, Bioingegneria,

Robotica ed Ingegneria dei Sistemi

Ph.D. Thesis in Computer Science and Systems Engineering
Computer Science Curriculum

Computational and Theoretical Issues of Multiparameter Persistent Homology for Data Analysis

by

Sara Scaramuccia

May, 2018

Dottorato di Ricerca in Informatica ed Ingegneria dei Sistemi
Indirizzo Informatica
Dipartimento di Informatica, Bioingegneria, Robotica ed Ingegneria dei Sistemi
Università degli Studi di Genova

DIBRIS, Univ. di Genova
Via Opera Pia, 13
I-16145 Genova, Italy
http://www.dibris.unige.it/

Ph.D. Thesis in Computer Science and Systems Engineering
Computer Science Curriculum
(S.S.D. INF/01)

Submitted by Sara Scaramuccia
DIBRIS, Univ. di Genova
sara.scaramuccia@dibris.unige.it

Date of submission: January 2018

Title: Computational and Theoretical Issues of Multiparameter Persistent Homology for Data Analysis

Advisors:
Leila De Floriani
Dipartimento di Informatica, Bioingegneria, Robotica ed Ingegneria dei Sistemi
Università di Genova, Italy
leila.defloriani@unige.it

Claudia Landi
Dipartimento di Scienze e Metodi dell'Ingegneria
Università di Modena e Reggio Emilia, Italy
claudia.landi@unimore.it

Ext. Reviewers: *Silvia Biasotti, $^{+}$Heike Leitte
*Istituto di Matematica Applicata e Tecnologie Informatiche "Enrico Magenes"
Consiglio Nazionale delle Ricerche: Genova, Italy
silvia.biasotti@ge.imati.cnr.it
$^{+}$ Fachbereich Informatik
Technische Universität Kaiserslautern, Germany
leitte@cs.uni-kl.de

Abstract

The basic goal of topological data analysis is to apply topology-based descriptors to understand and describe the shape of data. In this context, homology is one of the most relevant topological descriptors, well-appreciated for its discrete nature, computability and dimension independence. A further development is provided by persistent homology, which allows to track homological features along a one-parameter increasing sequence of spaces. Multiparameter persistent homology, also called multipersistent homology, is an extension of the theory of persistent homology motivated by the need of analyzing data naturally described by several parameters, such as vector-valued functions. Multipersistent homology presents several issues in terms of feasibility of computations over real-sized data and theoretical challenges in the evaluation of possible descriptors. The focus of this thesis is in the interplay between persistent homology theory and discrete Morse Theory. Discrete Morse theory provides methods for reducing the computational cost of homology and persistent homology by considering the discrete Morse complex generated by the discrete Morse gradient in place of the original complex. The work of this thesis addresses the problem of computing multipersistent homology, to make such tool usable in real application domains. This requires both computational optimizations towards the applications to real-world data, and theoretical insights for finding and interpreting suitable descriptors. Our computational contribution consists in proposing a new Morse-inspired and fully discrete preprocessing algorithm. We show the feasibility of our preprocessing over real datasets, and evaluate the impact of the proposed algorithm as a preprocessing for computing multipersistent homology. A theoretical contribution of this thesis consists in proposing a new notion of optimality for such a preprocessing in the multiparameter context. We show that the proposed notion generalizes an already known optimality notion from the one-parameter case. Under this definition, we show that the algorithm we propose as a preprocessing is optimal in low dimensional domains. In the last part of the thesis, we consider preliminary applications of the proposed algorithm in the context of *topology-based multivariate visualization* by tracking critical features generated by a discrete gradient field compatible with the multiple scalar fields under study. We discuss (dis)similarities of such critical features with the state-of-the-art techniques in topology-based multivariate data visualization.

Contents

List of Figures

List of Tables

Introduction

In the recent years, the increasing amount of data has led to the improvement and the development of information handling techniques. The basic goal of Topological Data Analysis (TDA) is to apply topology, one of the major branches of mathematics, to understand and organize qualitative information about data. Because much of the data arising in scientific applications is in high-dimensional spaces, the focus is on developing suitable tools for studying geometric features in high-dimensional data. Thus, it might be convenient to capture finer features than clusters of data, for instance, loops, holes and their higher-dimensional analogues.

Homology is one of the most relevant descriptor for higher-dimensional loops. That descriptor has gained its popularity in the applications thanks to its discrete nature, computability and dimension independence. *Persistent Homology (1-PH)* allows for comparisons of homologies of different spaces. It consists in a tracking of each homological feature when comparing one space to another along a single parametrized direction. It is used in data analysis to study evolutions of qualitative features of data and it is appreciated for its computability, robustness to noise, and dimension independence. *Multiparameter Persistent Homology (MPH)* is an extension of the PH Theory motivated by the fact that data analysis and comparisons often involve the examination of properties that are naturally described by multiple parameters, for instance in computer vision with respect to photometric properties. Alternatively, for point cloud data, several criteria might be chosen in order to filter the input datum for investigating both the domain itself under several criteria at once, and the explanatory power of the different criteria over the same domain. Another relevant tool for TDA is provided by *discrete Morse Theory*. This theory plays a key role in two main directions. First, it provides compact and robust segmentations of a shape according to its critical points. It also provides an important preprocessing tool for homology computations to reduce the size of the input datum.

Contributions. The focus of this thesis is on evaluating how reduction techniques inspired by Discrete Morse Theory can improve the feasibility of multipersistent homology (MPH) computations. Indeed, computing MPH requires to be able to deal with any reasonably sized data set, and theoretical insights for finding and interpreting suitable descriptors. Richer mathematical constructions are in need when moving from one to multiple parameters. Our contribution in the first direction consists in proposing, in Chapter 4 a discrete Morse-based preprocessing algorithm for computing MPH and more. A theoretical contribution of this thesis consists in proposing, in Chapter 5, a new notion of *optimality* for a Morse-based preprocessing in the multiparameter context. That is to say, we propose a property for determining whether the output of a discrete Morse-based preprocessing algorithm is the best possible one. We show that the algorithm we

propose is optimal in the sense proposed in this thesis. Finally, in Chapter 7, we discuss applications of our approach to the context of topology-based multivariate data visualization.

The remainder of the thesis is organized as described below.

In Chapter 1, we review the necessary background notions for the work of this thesis. In the first section, we introduce cell complexes. This section is crucial to understand the requirements for the algorithm presented in Chapter 4. In the second section, we introduce homology and simplicial homology since they are the main topological invariants treated in this thesis. In the third section, we introduce the basic notions for Lefschetz complexes, useful to capture in a unified way cell complexes and the discrete Morse complex as introduced in the following section. The notions related to relative homology in this section are of great interest for the content of Chapter 5. In the fourth section, we introduce discrete Morse Theory which is the main tool we exploit in this thesis to address computational and theoretical issue in MPH. The sixth and the seventh sections are devoted to the introduction of Persistent Homology (PH), the former in the one-parameter case (*one-persistent homology* (1-PH)), and the latter in the multiparameter case (*multipersistent homology* (MPH)). We presented the two theories in a parallel way in order to highlight (dis)similarities. The last section of the chapter presents the invariants associated with PH, both in the one and in the multiparameter cases, where most of dissimilarities between the two theories come up.

In Chapter 2, we present a review of the literature of persistent homology (PH) along the pipeline common to all different approaches. First, we review how the cell and simplicial complexes is represented, that is to say, the data structures for representing topological structures. Secondly, we review the main constructions for getting a filtration of cell complexes. We distinguish between filtration of a geometrical object like in the context of *Size Theory* and filtrations from point cloud data. The third step is the actual computational step of major interest for this work of thesis. We address the 1-PH and the MPH separately and we provide our comments on the different issued in the two cases. We briefly review methods for the last step of the pipeline which consists in interpreting the PH invariants.

Before introducing our algorithm in Chapter 4, we present some priors works on very closely related state-of-the-art algorithms. We present in a unified way the algorithm in [162] as a suitable 1-PH preprocessing, then generalized to MPH in [4]. Moreover, we explain our classification of the former, as a *local* algorithm, and, of the latter, as a *global* algorithm. This difference is crucial to understand the novelty of the algorithm we propose in Chapter 4. Additionally, the proof of correctness of the algorithm in Chapter 4 makes use of the description of the algorithm from [4]. Finally, the complexity analysis of our algorithm proposed in Chapter 4 has to be compared to the complexity analysis provided in this chapter.

In Chapter 4, we present the algorithm ComputedDiscreteGradient that we introduced in a preliminary version in [115]. ComputeDiscreteGradient retrieves a discrete gradient compatible with a vector-valued filtering function generalizing to multiparameter filtrations the *local* nature of the algorithm treated in the previous chapter, as opposed to its first generalization to MPH which has a global nature. We prove our algorithm to preserve MPH. We provide a detailed complexity analysis of the proposed algorithm.

In Chapter 5, we restate the optimality proved for the 1-PH preprocessing algorithm treated in Section 3.2 over cubical complexes embedded in $\mathbb{R}^3$ into more general terms as a property of a discrete gradient compatible with a one-parameter filtration. This optimality notion relates relative homology to the number of necessary critical cells to capture the MPH properties of the filtration. The proposed optimality notion admits a multiparameter generalization we have introduced in this thesis. Under this definition, we show that the algorithm we propose as a preprocessing in Chapter 4 is optimal for two cell complex subclasses: abstract simplicial complexes up to dimension 2, and cubical complexes embedded in $\mathbb{R}^3$.

Chapter 6 is structured into four sections, one for each experimental evaluation undertaken by our MPH preprocessing algorithm proposed in Chapter 4. Our MPH preprocessing algorithm is evaluated in comparison to state-of-the-art methods, in the impact on MPH computations and in the feasibility of dealing with large sized data which makes it suitable for being adapted to application domains.

real-sized domains.

In Chapter 7, we present our contribution in the context of data visualization of multivariate fields. In order to do this, we present some preliminaries in the research field, we review related methods, and provide our own contribution. In the final part, we discuss comparison to other methods and further developments. Both theoretical and experimental aspects for this part of the thesis have been developed in collaboration with Federico Iuricich (University of Maryland (MD), USA).

We close the work of this thesis with Chapter 8 where we summarize the results obtained in this work, the current developments of the topics, and future perspectives in the field.

1 Background notions

In this chapter, we review the necessary background notions for the work of this thesis. In Section 1.1, we introduce the topological structures of *simplicial complexes* and *cubical complexes* and the more general notion of *regular cell complex with the intersection* property useful to include both simplicial and cubical complexes into a unique formalism. This will be used in Chapter 3 and Chapter 4 to define and compare the input domains of the presented algorithms. In Section 1.2, we introduce the notion of *homology* as topological invariant for a simplicial complex. In Section 1.3, we introduce the *Lefschetz complex* as a combinatorial structure which translates homology in a common language for all kinds of complexes considered in this thesis. In doing this, we introduce *relative homology* for Lefschetz complexes which is necessary for our contributions of Chapter 5. In Section 1.4, we introduce some results from *discrete Morse Theory* which are necessary for stating the properties of the algorithms described in Chapter 3 and Chapter 4, and to describe our results in the context of topology-based visual analytics in Chapter 7. In Section 1.5, we introduce the concept of persistent homology, here called *one-parameter persistent homology*, or simply *one-persistent homology* to highlight its special character with respect to the *multiparameter persistent homology* introduced in Section 1.6 and sometimes called *multipersistent homology*. The latter constitutes the main topological feature we focus on in this thesis. In Section 1.7, we introduce the topological invariants persistent homology for one-parameter and multiparameter settings. This, on the one hand, defines the target of computing techniques related to persistent homology, on the other hand, it is the proper landscape to understand the different character when moving from one-persistent to multipersistent homology.

1.1 Cell complexes

In this section, we introduce the topological structure of a cell complex and we set the terminology adopted for the rest of the thesis. In Section 1.1.1, we introduce the notion of a simplicial complex, that is collection of elementary discrete entities called *simplices* which generalize triangles to dimensions higher than 2. The simplicial complexes here considered are of two main kinds: *embedded in a Euclidean space* or combinatorially described and called *abstract*. In particular, this difference is made explicit to better express our contribution of Section 5.4. In Section 1.1.2, we introduce the notion of a cubical complex, that is a collection of elementary discrete entities called *(hyper)cubes* which generalize cubes to dimensions higher than 3. This time cubical complexes will be only considered as embedded in a Euclidean space. We close the chapter by defining the superclass of regular cell complexes with the intersection property which includes

both simplicial and cubical complexes and it formalizes a requirement for the algorithm we propose in Chapter 4.

1.1.1 Simplicial complexes

In this section, we introduce simplicial complexes. For the main definitions, we follow the formalism of [149]. Differently from the definitions in the given reference, since we are not interested in simplicial complexes with infinitely many simplices, we define all complexes to be finite. We first address the case of a simplicial complex embedded in a Euclidean space and, then, we address the case of abstract simplicial complexes.

A k-dimensional simplex s, or k-simplex for short, is the convex hull of $k+1$ affinely independent points of a p-dimensional Euclidean space with $p \geqslant k+1$. Often, we will simply write s^k to mean a k-simplex and $\dim s$ to indicate its dimension. A *face* t of s is the convex hull of any subset of points generating s. The partial order relation "t is face of s" is denoted $t \ll s$. If the dimensions of s and t differ by one we call t a *facet* of s and write $t < s$. Reciprocally, s is called a *coface* of t or a *cofacet* when the two dimensions differ by one (respectively denoted $s \gg t$ and $s > t$).

1.1.1 Definition (Simplicial complex)**.** A *simplicial complex* S is a finite collection of simplices such that:

(i) every face of a simplex in S is also in S *(closure property)*;

(ii) the intersection of any two simplices in S is either empty or a unique simplex in S *(intersection property)*.

The *embedding space* of a simplicial complex is the Euclidean space where all of its vertexes lie. The *embedding dimension* of a simplicial complex S is the dimension p of the embedding space.

We will denote by S_k the set of k-simplices in S:

- an element in S_0 is also called a *vertex*;
- an element in S_1 is also called an *edge*;
- an element in S_2 is also called a *triangle*;
- an element in S_3 is also called a *tetrahedron*.

A simplicial complex S has *dimension* d (and S is called a d-complex for short) if the maximum of the dimensions of its simplices is d. Given a subset T in S, the *closure of* T in S, denote by $\overline{T}$,

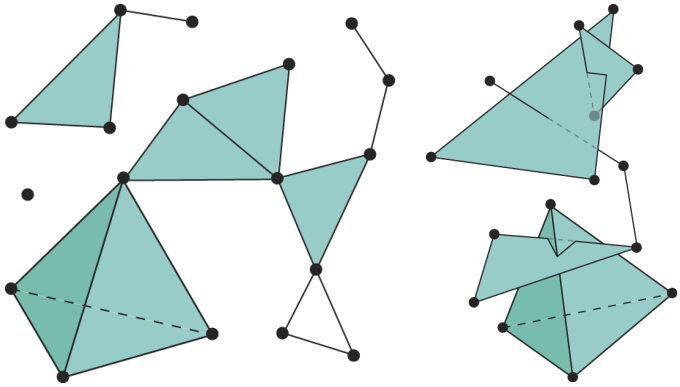


Figure 1.1: (a) a simplicial complex; (b) a se of simplices which is not a simplicial a complex.

Source: image taken from Wikipedia.

is the subset of S containing all faces of s for $s \in T$. The *boundary of* s in S is the set $\mathrm{bd}_S(s)$ equal to $\overline{s}\backslash\{s\}$. The *star of* s *in* S is the set, denoted $\mathrm{Star}_S(s)$, of cofaces of s in S. The *coboundary of* s is the set $\mathrm{cb}_S(s)$ given by $\mathrm{Star}_S(s)\backslash\{s\}$. The subset T is *closed in* S if $\overline{T} = T$. The subset T is *open in* S if $S\backslash T$ is closed. The topology τ_S generated by all open subsets in S is the *simplicial topology* on S. Indeed, arbitrary intersections of open subsets are still open and finite intersections of closed subsets are still closed. Later, in Definition 1.1.10i), we will see how simplicial topology is a particular case of the *weak topology* for CW-complexes.

In our finite setting, the simplicial topology corresponds to the topology induced by the embedding space. Indeed, denoting by $\mathrm{Real}(S)$ the union of all simplices, we get that, since in Definition 1.1.1 we only admit simplicial complexes with finitely many simplices, the simplicial topology corresponds to the subspace topology induced on $\mathrm{Real}(S)$ by the embedding space.

In the following, we report the main definitions and properties that are useful in this thesis.

1.1.2 Definition (Simplicial subcomplex)**.** A simplicial complex T is a *subcomplex of* S if $T \subseteq S$.

It is easy to see that a subcomplex is always closed and, conversely, a closed subset in a simplicial complex is always a subcomplex.

1.1.3 Example. *The boundary of any simplex in a simplicial complex is a subcomplex.*

1.1.4 Definition (k-path)**.** In a simplicial complex S, a *k-path* of simplices *from* s^k *to* t^k is a finite sequence of simplices $(s_0^k, t_0^{k+1}, s_1^k, t_1^{k+1}, \ldots, t_{l-1}^{k+1}, s_r^k)$ such that:

- $l \geqslant 0$;

- $s_0^k = s^k$ and $s_l^k = t^k$;
- $s_0^k < t_0^{k+1} > s_1^k < t_1^{k+1} > \cdots < t_l^{k+1} > s_r^k,$.

If s^k is equal to t^k, we say that the k-path is *closed*. If all s_i^k's are different, we say that the k-path is *simple*. If $l = 0$, we say that the k-path is *trivial*.

Here, we introduce an inductive definition of connectedness for a collection of simplices T in S.

1.1.5 Definition (Connectedness)**.** A subset T in S is:

- 0-*connected* if T contains at least a 0-simplex and for any pair of 0-simplices there exists a 0-path;
- k-*connected* if T is $(k-1)$-connected and, for any pair of simplices s^k, t^k in T, there is a k-path from s^k to t^k of simplices in T.

A maximal k-connected subset with respect to the inclusion is called a k-*connected component*.

A simplicial complex can be defined in a combinatorial way without referring to its embedding space.

1.1.6 Definition (Abstract simplicial complex)**.** An *abstract simplicial complex* S on a set V is a collection of subsets of V such that if $t \in S$ and $s \subseteq t$, then $s \in S$. Elements in S are called *(abstract) simplices* and their dimension $\dim s$ is k if s contains $k+1$ elements in V.

It is easy to see that simplicial complexes can be seen as abstract simplicial complexes on the set $V = S_0$. Indeed, the closure property is implicit in the abstract definition and the intersection property follows by noticing that the intersection of two subsets in V is again a subset in V. The face and coface relations are given respectively by the set inclusions $\subseteq$ and $\supseteq$ between abstract simplices.

1.1.2 Cubical and cell complexes

In this section, we introduce some necessary definitions for the discrete spaces we treat all along the thesis. First, we introduce cubical complexes [119]. Then, we introduce cell complexes as a class including both simplicial and cubical complexes. Finally, we give the definition of regular cell complex with the intersection property of special interest for the algorithms presented in Chapter 3 and Chapter 4.

We begin by defining the elementary pieces in a cubical complex.

1.1.7 Definition ((Hyper)cube)**.** An *elementary interval I* is a closed interval either of the form $[l, l+1] \subseteq \mathbb{R}$ or $[l, l] \subseteq \mathbb{R}$, with $l \in \mathbb{R}$. In the latter case, we call it *degenerate interval* or a 0-cube, or a vertex. A 0-*cube*, also called a vertex is a point in $\mathbb{R}$. A *(hyper)cube q* of dimension $k > 0$, also called a *k-cube*, is a finite product of non-degenerate closed intervals in $\mathbb{R}$ of equal length

$$I_1 \times \cdots I_p \subseteq \mathbb{R}^k.$$

Often, we will simply write q^k to mean a k-cube and $\dim q$ to indicate its dimension. A *face t* of q is a cube defined either by a proper subset of the elementary intervals defining q or by the 0-cube corresponding to some ending point of the elementary intervals defining q. The definitions of *facets, cofacets* are defined and denoted in the same way as the analogous notions for simplices introduced at the beginning of Section 1.1.1.

1.1.8 Definition (Cubical complex)**.** A *cubical complex Q* embedded in $\mathbb{R}^p$ is a finite collection of cubes in $\mathbb{R}^p$ such that:

(i) every face of a cube in Q is also in Q *(closure property)*;

(ii) the intersection of any two cubes in Q is either empty or a unique cube in Q *(intersection property)*.

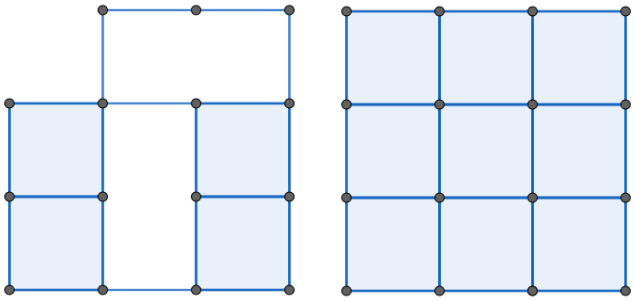


Figure 1.2: (a) cubical complex in $\mathbb{R}^2$ which is not a regular grid and (b) a regular grid in $\mathbb{R}^2$.

The *embedding space* of a cubical complex is the Euclidean space $\mathbb{R}^p$ where all of its vertexes lie. The *embedding dimension* is p. Cubical complexes allow to formalize *regular grids* which constitutes an important subclass of cubical complexes particularly used in computational context, for instance in our tests of Section 6.4 and Section 7.3.2. A regular grid is cubical complex in $\mathbb{R}^p$ obtained by axis-wise regular subdivisions of a limited range per axis of $\mathbb{R}^p$. In Figure 1.2, we show how general cubical complexes differ from regular grids.

Now, we introduce the structure of a cell complex having as special cases simplicial and cubical complexes. A k-dimensional cell (or a k-cell) is a topological space a homeomorphic to the interior of the k-dimensional ball $\mathbb{B}^k$, sometimes shortly denoted by a^k to underline that $\dim a = k$. A set of k-cells is denoted by X_k.

1.1.9 Definition (Cell complex)**.** A *cell complex* X is a Hausdorff space whose elements are a finite number of cells graded by dimensions $X = \bigsqcup_{k\geqslant 0} X_k$ along with, for each cell $a \in X_k$, a continuous *attaching map* $\phi_a : \mathbb{B}^k \longrightarrow a \in X$ that satisfies:

- the interior of $\mathbb{B}^k$ is homeomorphically mapped to a;
- the $(k-1)$-sphere $\mathbb{S}^{k-1} := \partial\mathbb{B}^k$ is mapped into X_{k-1}.

The *dimension* of a cell complex is the smallest integer k such that $\bigcup_{i=0,\dots,k} X_i = X$. The dimension is well-defined by finiteness. A cell b is a *face* of a in X if and only if b belongs to the image of ϕ_a. Analogously to the simplicial case, the partial order relation generated by "b is face of a" is denoted $b \ll a$. If the dimensions of b and a differ by one we call b a *facet* of a and denote it by $b < a$. Reciprocally, a is a *coface* of b, and a *cofacet* when the two dimensions differ by one (respectively indicated $a \gg b$ and $a > b$).

As a topological space, a cell complex is usually asked to have the finest topology being coherent with the topology τ_a over each cell a. This topology is usually referred to as the *weak topology* with respect to all subspaces given by the cells in X.

1.1.10 Definition (CW complex)**.** A *CW-complex* X is a cell complex endowed with the topology τ_{CW} such that:

i) a set $Y \subseteq X$ is closed in X if and only if, for all $a \in X$, $a\backslash(Y \cap a) \in \tau_a$ *(weak topology)*

We denote by $\overline{Y}$ the closure of a subset Y of X with respect to the topology τ_{CW}. The *boundary of* a in X is the set $\mathrm{bd}_X(a)$ equal to $\overline{a}\backslash\{a\}$. The *star of* a *in* X is the set, denoted $\mathrm{Star}_X(a)$, of cofaces of s in X. The *coboundary of* s is the set $\mathrm{cb}_X(a)$ given by $\mathrm{Star}_X(a)\backslash\{a\}$. All the given definitions of (co)boundary, star and closure of a cell generalize the analogous definitions of Section 1.1.1 holding for simplicial complexes.

1.1.11 Definition. A *regular cell complex* X is a CW complex such that all attaching maps are homeomorphisms.

In other words, no identifications occur under the restriction of the attaching maps to the $(k-1)$-spheres. The topological structures of CW complexes and regular cell complexes admit in general multiple cells in the intersection of two different cells.

1.1.12 Definition. A *cell complex X with intersection property* is a CW complex such that, for every two cells in X, the corresponding closures have either empty intersection or their intersection is a unique cell in X

1.1.13 Remark. *All simplicial complexes and cubical complexes are regular cell complexes with the intersection property. The intersection property comes by definition. The regularity follows since, for each k-simplex or k-cube, we have only non-degenerate faces. Moreover the j-faces are in a fixed number, respectively $\binom{k+1}{j+1}$ and $2^{k-j}\binom{k}{j}$.*

1.2 Homology

In this section, we introduce the main topological invariant we deal with in the work of this thesis. The mathematical formalism of this section lay the basis for the language used in Section 1.5 and Section 1.6 to define persistent homology. First, we see how homology is associated with the algebraic notion of a *chain complex*. Then, in Section 1.2.1, we specialize to the case of simplicial homology as an example of how to build a chain complex out of a topological structure like the ones presented in the previous section.

Intuitively, the homology of a complex structure (simplicial, cubical or cellular) is used to detect independent k-dimensional holes in a topological shape. A hole can be given a dimension by following this intuitive idea: different connected components are separated by 0-dimensional holes, tunnels are 1-dimensional holes (we have two of them in Figure 1.3 bounded by red edges), voids are 2-dimensional holes (we have one of them in Figure 1.3 bounded by the whole torus), and so on. The idea can be generalized by looking at *k-cycles*, i.e., formal combinations of k-dimensional cells to determine whether or not they bound a k-dimensional hole. More precisely, the incidence relations among cells in the complex are combinatorially translated into algebraic terms. This allows to arrange k-chains into a *chain complex*. Homology is used to detect holes since it manages in discriminating among k-chains bounding a k-dimensional hole and those bounding a $(k+1)$-chain.

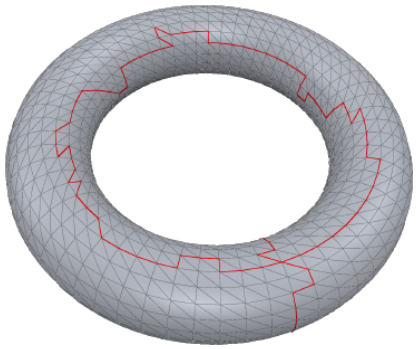


Figure 1.3: a triangularization of a hollow torus with (in red) two 1-chains bounding two 1-dimensional holes.

1.2.1 Definition (Chain complex)**.** A *chain complex* $C = (C_*, \partial_*)$ consists of a family $C_* = \{C_k\}_{k\in\mathbb{Z}}$ of Abelian groups or modules, called *chain groups* along with a collection of homomorphisms $\partial_* = \{\partial_k : C_k \longrightarrow C_{k-1}\}_{k\in\mathbb{Z}}$ called *boundary maps* such that

$$\forall k \in \mathbb{Z}, \qquad \partial_k \circ \partial_{k+1} = 0. \tag{1.1}$$

The elements of C_k are called *k-chains*. A k-chain in the kernel of ∂_k is called a *k-cycle*. A k-chain in the image of ∂_{k+1} is called a *k-boundary*.

The chain complex definition ensures that, for each $k \in \mathbb{Z}$, the kernel of ∂_k contains the image of ∂_{k+1}, that is all k-boundaries are k-cycles.

1.2.2 Definition (k^{th}-Homology)**.** The *k^{th}-homology $H_k(C)$ of the chain complex C* is defined as the quotient space $H_k(C) = \ker \partial_k / \operatorname{im} \partial_{k+1}$. The integer k will be referred to as the *homology degree*.

1.2.3 Definition (Chain morphism)**.** Given two chain complexes $C = (C_*, \partial_*^C)$ and $D = (D_*, \partial_*^D)$, a *chain morphism* is a collection of homomorphisms $f_* = \{C_k \xrightarrow{f_k} D_k\}_{k\in\mathbb{Z}}$ such that $\partial_k^D \circ f_k = f_{k-1} \circ \partial_k^C$, for all integers k. That is all the squares in the following diagram commute.

$$\begin{array}{ccccccccc}
\dots & \xrightarrow{\partial^C_{k+2}} & C_{k+1} & \xrightarrow{\partial^C_{k+1}} & C_k & \xrightarrow{\partial^C_k} & C_{k-1} & \xrightarrow{\partial^C_{k-1}} & \dots \\
 & & \downarrow{\scriptstyle f_{k+1}} & & \downarrow{\scriptstyle f_k} & & \downarrow{\scriptstyle f_{k-1}} & & \\
\dots & \xrightarrow{\partial^D_{k+2}} & D_{k+1} & \xrightarrow{\partial^D_{k+1}} & D_k & \xrightarrow{\partial^D_k} & D_{k-1} & \xrightarrow{\partial^D_{k-1}} & \dots.
\end{array}$$

1.2.4 Remark. *Any chain morphism f induces a linear map f_k^*, for each homology degree:*

$$f : C \longrightarrow D \quad \Rightarrow \quad f^* : H_k(C) \longrightarrow H_k(D)$$

Indeed, the commutativity required in Definition 1.2.3 ensures that cycles in C are mapped to cycles in D, and boundaries to boundaries, so that f_k^* is obtained by moving to the quotient defining homology. However, notice that the possible injectivity or surjectivity of map f is not necessarily preserved when passing to f_k^*. Indeed, on the one hand, cycles might become boundaries and, on the other hand, some cycles in D can have no corresponding cycle in C.

1.2.1 Simplicial chain complexes

We focus on a special class of chain complexes which is obtained from abstract simplicial complexes (Definition 1.1.6). The simplices of the simplicial complex will form a set of generators for chain groups.

In order to build up a chain complex out a simplicial complex, we provide each simplex with an *orientation*. In an abstract simplicial complex S, a simplex $\{v_0, \dots, v_k\}$ is oriented when an

ordering for the vertexes is chosen. Two orderings are equivalent if there exists an even number of permutations changing one into the other, and the classes under this equivalence are called *orientations*. Clearly, only two orientations are possible. An *oriented simplex* is an equivalence class $[v_0,\dots,v_k]$ of its ordering. When the orientation of a simplex is clear, we simply denote it by s. If s is an oriented simplex, $-s$ denotes the same simplex with opposite orientation.

A *simplicial k-chain* is a formal combination of oriented k-simplices with coefficients in some Abelian group G. The chain group $C_k(S;G)$ is the free Abelian group generated by S_k. When the *coefficient group* G is clear from the context, we simply write $C_k(S)$. Boundary maps are obtained by linear extensions from the values on simplices. For each oriented k-simplex $s=[v_0,\dots,v_k]$

$$\partial_k s=\sum_{i=0}^{k}(-1)^k[v_0,\dots,\hat{v}_i,\dots,v_k],$$

where $\hat{v}_i$ means that the vertex v_i has been removed.

It can be easily checked that, under this definition, Condition (1.1) holds.

The so-obtained chain complex is called *simplicial chain complex associated with* S and denoted by $C(S)=(C_*(S),\partial_*^S)$. The *simplicial homology* of S is the homology of $C(S)$. For brevity, the k^{th}-homology group $H_k(C(S))$ will be denoted $H_k(S)$.

Simplicial homology is a topological invariant, meaning that homeomorphic simplicial complexes have the same associated simplicial homology. Simplicial homology is not the only homology Theory one can associates with a simplicial complex with. The notion of *singular homology* [108] can be associated with the underlying space $|S|$. By Theorem 2.27 in [108], the two theories agree on simplicial complexes, implying in particular that simplicial homology does not depend on the chosen orientation for simplices.

Depending on the coefficient group, we get possibly non-isomorphic simplicial homologies. Any Abelian group can be seen as a $\mathbb{Z}$-module. If $G=R$ is a ring, then $H_k(S;R)$ has an R-module structure. By following Section 8.5 in [98], for R-modules the following algebraic decomposition result holds.

1.2.5 Theorem ([98] Structure Theorem)**.** *Any finitely generated R-module M with R a principal ideal domain (PID) admits unique elements $a_1,\dots,a_m$ in R such that*

$$M\cong\bigoplus_{i=1}^{p}R\oplus\bigoplus_{j=1}^{m}R\Big/(a_j)\,. \tag{1.2}$$

We call the summands indexed by i form the torsion part *of the decomposition, while the part indexed by j is called* free part.

Since the number of summands in the free part of the Structure Theorem decomposition does not depend on the chose ring R, we introduce the following invariant for simplicial complexes.

1.2.6 Definition (Betti numbers of a complex)**.** Given a simplicial complex S, the k^{th} *Betti number associated with S with coefficients in R* is the number $\beta^k(S;R)$[1] of summands in the free part of the Structure Theorem 1.2.5 decomposition of $M = H_k(S;R)$, for $k \in \mathbb{Z}$.

If $R = \mathbb{F}$ is a field, then $H_k(S;\mathbb{F})$ has a $\mathbb{F}$-vector space structure. In this case, there is no torsion part in Formula (1.7.14). In the particular case of $R = \mathbb{F}_2$, that is the field with two elements, chains and homology can be easily defined without taking simplex orientation into account.

1.3 Lefschetz complexes

In this section, we introduce the *Lefschetz complex* as an algebraic structure which translates the homology of chain complexes seen at the beginning of Section 1.2 into a combinatorial way which mimics topological relations between cells. In particular, this is useful to express in a common language the chain complex obtained in discrete Morse Theory from a discrete gradient as treated in Section 1.4 and a topological structure such as the ones presented in Section 1.1. We exploit this common language in our contribution of Chapter 5. First, we report only the basic notions. We refer the reader to [131], for more details about Lefschetz complexes, or to [146] for a notation closer to ours, where Lefschetz complexes are called S-complexes. In Section 1.3.1, we introduce the notion of *relative homology* necessary for Chapter 5.

At the base of the homology definition there is the chain complex construction seen in Definition 1.2.1. This latter can be handled in a combinatorial way, in terms of the elements generating each chain group and the incidence relations among them, encoded in the definition of the *incidence function* of a Lefschetz complex. From now on, we fix the coefficient ring to be the field $\mathbb{F}_2$.

1.3.1 Definition (Lefschetz complex)**.** A *Lefschetz complex* (L, κ_L) *over* $\mathbb{F}_2$ (often simply L) consists of a graded finite set $L = \bigsqcup_{k\in\mathbb{Z}} L_k$ along with an *incidence function* $\kappa_L : L \times L \longrightarrow \mathbb{F}_2$ satisfying the following properties:

(i) $\forall a, b \in L$, $\kappa_L(b,a) \neq 0$ implies that $\dim b - \dim a = 1$

(ii) for every two cells $b \in L_{k+2}$ and $a \in L_k$,

$$\sum_{c \in L_{k-1}} \kappa_L(b,c)\kappa_L(c,a) = 0.$$

[1] The superscript k in $\beta^k(S;R)$ should not be confused with the subscript i adopted in Definition 1.7.13 for multigraded Betti numbers. In the former case, k indicates the degree of the homology of a simplicial complex. In the latter case, i indicates the degree of the homology in the Tor functor free resolution. In this latter case, a fixed degree of the homology of a simplicial complex is taken into account.

Analogously to cell complexes, $\dim a = k$ if and only if a belongs to L_k. In this case, a is called a *k-cell* and shortly denoted by a^k.

In order to mimic face relations among cells in a cell complex, we introduce the following notation: if $\kappa_L(b,a) \neq 0$, we call b a *cofacet* of a (resp. a a *facet* of b) and write $b > a$ (resp. $a < b$). These imply, for a single cell, analogous notions of *closure, boundary, coboundary and star*.

We define the *Lefschetz chain complex* $C(L) = (C_*(L), \partial_*^L)$ as the chain complex associated to a Lefschetz complex $L = (L, \kappa_L)$ where, for each integer degree k

- the chain group $C_k(L)$ is the $\mathbb{F}_2$-vector space generated by L_k
- the value of the boundary map ∂_k^L on a basis element b^k is defined by

$$\partial_k^L b = \sum_{a \in L} \kappa_L(b,a) a$$

By the following remark, the above definition is guaranteed to define a chain complex.

1.3.2 Remark. *Under our assumptions, Condition (1.1) is equivalent to Condition (ii) in Definition 1.3.1.*

Indeed, on the one hand Condition (1.1) is clearly stronger. On the other hand, by extending the terms in (1.1) when applied to a specific cell and manipulating the sums, we get that a sum of linearly independent elements gives the null vector. Hence, we deduce that all coefficients are null. Such coefficients corresponds the left hand term in Condition (ii).

The k^{th}-homology group $H_k(C(L))$ in this case is actually an $\mathbb{F}_2$-vector space. For brevity, that space will be denoted $H_k(L)$. Analogously to the simplicial homology case, we indicate by $H_k(L;R)$ when we consider Lefschetz complexes over any ring R rather than the field $\mathbb{F}_2$.

1.3.3 Example (Cell complexes as Lefschetz complexes)**.** *A cell complex X is a Lefschetz complex (L, κ_L) over $\mathbb{F}_2$ with L graded by the cell dimensions, by taking $L_k = X_k$, and κ_L equal to the* cellular incidence function κ_X*, with*

$$\begin{aligned} \kappa_X(b,a) = 1 \quad &\Leftrightarrow \quad b > a \\ \kappa_X(b,a) = 0 \quad &\Leftrightarrow \quad \textit{otherwise} \end{aligned}$$

Indeed, Condition i) in Definition 1.3.1 is straightforward to check. As for Condition ii), in the case of simplicial and cubical complexes, it follows easily since only cells c satisfying $b > c > a$ in X lead to non-null summands. Moreover, in X, if any, there are exactly two such c, thus summing up to zero in $\mathbb{F}_2$. For general cell complexes, Condition ii) follows from Remark 1.3.2 since, as a Lefschetz complex $X = (X, \kappa_X)$ induces a chain complex isomorphic to a standard cellular chain complex [108].

Thanks to the combinatorial mean of Lefschetz complexes, we get a uniform way of dealing with several kinds of homology. Simplicial homology becomes a particular case of cellular homology [108]. For each CW-complex, the underlying space supports the singular homology construction. By Theorem 2.35 in [108] singular and cellular homology agree.

A *Lefschetz subcomplex* (N, κ_N) *of* (L, κ_L) is a Lefschetz complex such that $N_k \subseteq L_k$, for each $k \in \mathbb{Z}$, and κ_N coincides with the restriction of κ_L to $N \times N$. A subset N is *closed* in L if, for all $b \in N$, the condition $\kappa_L(b,a) \neq 0$ for some $a \in L$ implies $a \in N$. It is a known fact (Theorems 3.1, 3.2 in [146]) that a closed subset N in L is a Lefschetz subcomplex with incidence function taken by restriction.

1.3.4 Remark. *A closed set in a simplicial complex is also closed if thought of as a subset in a Lefschetz complex. Moreover, each simplicial subcomplex is necessarily a closed Lefschetz subcomplex. Indeed, a subcomplex is, in particular, a complex and, by Condition (i) in Definition 1.1.1, it implies being closed in the Lefschetz sense.*

1.3.5 Definition (Intersection and union)**.** Given (L, κ_L) and (N, κ_N) Lefschetz subcomplexes in (U, κ_U), we define the *union* $(L \cup N, \kappa_\cup)$ and the *intersection* $(L \cap N, \kappa_\cap)$ by setting $\kappa_\cup$ to be the restriction of κ_U to $(L \cup N) \times (L \cup N)$ and $\kappa_\cap$ to be the restriction to $(L \cap N) \times (L \cap N)$, respectively.

1.3.6 Remark. *If L and N are closed in U, then both the union and the intersection are still closed in U.*

1.3.7 Remark. *Definition 1.3.5 does not apply to the union and the intersection of general Lefschetz complexes. In particular, two Lefschetz complexes may not agree on their intersection*

1.3.1 Relative homology and Mayer-Vietoris pairs

In this section, we define *relative Lefschetz complex pairs* and their *relative homology*. Afterwards, we define *Mayer-Vietoris Lefschetz complex tuples*. Both yield a short exact sequence of chain complexes. In particular, these notions are necessary to prove our contribution of Proposition 5.3.1. We adapt standard notions in Algebraic Topology [108] to be expressed in terms of Lefschetz complexes.

1.3.8 Definition (Relative pair)**.** If (N, κ_N) is a closed Lefschetz subcomplex in (L, κ_L), we write (L,N) and call it a *relative pair* of Lefschetz complexes.

Given a relative pair (L,N), we define the space $C_k(L,N)$ of *relative k-chains* as the elements in the quotient $C_k(L)/C_k(N)$ of $\mathbb{F}_2$-vector spaces. The *relative boundary map* $\partial_k^{\text{rel}} : C_k(L,N) \longrightarrow C_{k-1}(L,N)$ is induced by the boundary map for L by moving to the quotient of vector spaces. The *relative chain complex* associated to a relative pair (L,N) is indicated $C(L,N) = (C_*(L,N), \partial_*^{\text{rel}})$.

1.3.9 Definition (Relative homology)**.** The *relative k^{th}-homology* $H_k(L,N)$ is the k^{th}-homology of the relative chain complex $C(L,N)$.

Relative pairs yield to a short and a long exact sequence which is a standard result in commutative algebra obtained through a technique called diagram chasing (see Theorem 2.16 in [108]).

1.3.10 Theorem. *Each relative pair (L,N) induces a short exact sequence at chain level*

$$0 \longrightarrow C(N) \xrightarrow{\ i\ } C(L) \xrightarrow{\ p\ } C(L,N) \longrightarrow 0 \tag{1.3}$$

with i induced by the inclusion and p being the quotient projection. Moreover, the short exact sequence (1.3) induces a long exact sequence of homology

$$\begin{aligned} \dots \xrightarrow{\epsilon_{k+1}} H_k(N) \xrightarrow{i_k^*} H_k(L) \xrightarrow{p_k^*} H_k(L,N) \xrightarrow{\epsilon_k} \\ \xrightarrow{\epsilon_k} H_{k-1}(N) \xrightarrow{i_{k-1}^*} H_{k-1}(L) \xrightarrow{p_{k-1}^*} H_{k-1}(L,N) \xrightarrow{\epsilon_{k-1}} \dots \end{aligned} \tag{1.4}$$

where the maps ϵ_k are defined over each $[a] \in H_k(L,N)$ by $\epsilon_k([a]) = [\partial_k^L(a)]$, where ∂_k^L is the boundary map in $C(L)$.

1.3.11 Definition (Mayer-Vietoris triple)**.** Given a Lefschetz complex U, a *Mayer-Vietoris triple*, or simply an *MV-triple* is a triple (U,L,N) of Lefschetz complexes such that $U = L \cup N$ and L, N are closed Lefschetz subcomplexes of U.

MV-triples yield to a short and a long exact sequence in the same way relative pairs do.

1.3.12 Theorem. *Each MV-triple (U,L,N) induces a short exact sequence at chain level*

$$0 \longrightarrow C(L \cap N) \xrightarrow{(i,j)} C(L) \oplus C(N) \xrightarrow{h-l} C(N \cup L) \longrightarrow 0 \tag{1.5}$$

with i and j induced by inclusions of $L \cap N$ in L and N respectively, and with h and l induced by the inclusions in $L \cup N$ from L and N, respectively. Moreover, the short exact sequence (1.5) induces the long exact sequence of homology

$$\begin{aligned} \dots \xrightarrow{\delta_{k+1}} H_k(L \cap N) \xrightarrow{(i_k^*, j_k^*)} H_k(L) \oplus H_k(N) \xrightarrow{h_k^* - l_k^*} H_k(L \cup N) \xrightarrow{\delta_k} \\ \xrightarrow{\delta_k} H_{k-1}(L \cap N) \xrightarrow{(i_{k-1}^*, j_{k-1}^*)} H_{k-1}(L) \oplus H_{k-1}(N) \xrightarrow{h_{k-1}^* - l_{k-1}^*} H_{k-1}(L \cup N) \xrightarrow{\delta_{k-1}} \dots \end{aligned} \tag{1.6}$$

where the maps δ_k are defined over each $[a+b] \in H_k(L \cup N)$, with $a \in C_k(L)$ and $b \in C_k(N)$, by $\delta_k([a]) = [\partial_k^L(b)]$, where ∂_k^L is the boundary map from $C_k(L \cap N)$.

1.4 Discrete Morse Theory

In this section, we introduce the main theoretical tools from Forman's discrete Morse Theory [87, 88]. Discrete Morse Theory is the discrete counterpart of a classical branch of Differential Geometry called Morse Theory [142] that studies shape properties from the behavior of special scalar functions defined on the shape, called *Morse functions*. The main notion introduced in this section is that of a *Morse complex* built out from an *original* regular cell complex (Definition 1.1.11). In our work, we are particularly interested in looking at a Morse complex as a *reduction* of the original regular cell complex, in terms of number of cells, that preserves the homology of the original cell complex. In view of the persistence preprocessing algorithms of Chapter 3 and Chapter 4 for the retrieval of a *discrete gradient*, special emphasis is given to defining a Morse complex from its discrete gradient. The notion of a Morse complex will be related to persistent homology in Section 1.5 and Section 1.6. The interplay between discrete Morse Theory and persistent homology is much deeper that what presented in this thesis and we refer the reader to [8] for further details.

In Section 1.4.1, we introduce the basics in discrete Morse Theory. In doing this, we sketch the relationship between *discrete Morse functions* and discrete gradients. In Section 1.4.2, we introduce the notion of a Morse complex and its homology-preserving reduction property with respect to the original complex.

1.4.1 Basic notions in discrete Morse Theory

All along this section, X will be a regular cell complex. As already seen in Example 1.3.3, X can be thought of as a Lefschetz complex whenever necessary.
A *real-valued cell function over* X is a function $f : X \longrightarrow \mathbb{R}$.

1.4.1 Definition (Discrete Morse function)**.** A *discrete Morse function* is a real-valued cell function f over X satisfying, for each $s \in X$:

$$e^+(s) := \#\{t > s \mid f(t) \leqslant f(s)\} \leqslant 1$$
$$e^-(s) := \#\{r < s \mid f(r) \geqslant f(s)\} \leqslant 1.$$

A cell $s \in X$ such that $e^+(s) = e^-(s) = 0$ is a *critical cell*. The dimension of a critical cell is defined as its *critical index* or simply its *index*. Let $m_k(f)$ be the number of critical cells defined by f of index k.

Let $\beta^k(X;\mathbb{F})$ be the k^{th}-Betti number relative to X where $\mathbb{F}$ is any field.

1.4.2 Theorem (Morse inequalities. Corollaries 3.6, 3.7 in [87]). *For all integers k, the* weak Morse inequalities *hold:*

$$m_k(f) \geqslant \beta^k(X;\mathbb{F})$$

and the strong inequality *holds*

$$\sum_{k=0}^{\infty}(-1)^k m_k(f) \geqslant \sum_{k=0}^{\infty}(-1)^k \beta^k(X;\mathbb{F})$$

If in (1.4.2) the equality holds, then f is said to be a *perfect Morse function*. The problem of finding a perfect discrete Morse function over a given regular cell complex is known to be undecidable. Typically, additional assumptions on the cell complex guarantee to decide about the existence of a perfect Morse function.

A discrete Morse function over X such that there exists no other discrete Morse function over X with fewer critical cells (all indexes considered) is called an *optimal discrete Morse function*.

The problem of finding an optimal discrete Morse function over a given cell complex is known to be NP-hard [117]. Indeed, this is inherently due to the undecidability of the problem of finding a perfect Morse function for a general shape.

In discrete Morse Theory, a *(discrete) vector* of X is a pair of cells (s,t) such that $s < t$ and $s,t \in X$.

1.4.3 Definition (Discrete vector field). A *discrete vector field* on a regular cell complex X is any collection of vectors V of X such that each cell is component of at most one vector in V.

1.4.4 Definition. A *V-path* Γ is an ordered sequence of vectors

$$\Gamma = [(s_0,t_0),\ldots,(s_l,t_l)]$$

belonging to V, such that, for all indexes $0 \leqslant i \leqslant l-1$, $s_{i+1} < t_i$ and $s_i \neq s_{i+1}$.

A V-path is said to be *closed* if $s_0 = s_l$ and *trivial* if $l = 0$. Notice that, according to Definition 1.1.4, a V-path $[(s_0^k,t_0^{k+1}),\ldots,(s_l^k,t_l^{k+1})]$ is a k-path from s^k to any face r^k of t_l^{k+1}). Hence, we can say that Γ is from *from s^k to r^k*. The empty set is a valid V-path from each s^k to itself.

1.4.5 Definition (Discrete gradient field). A *discrete gradient field* V is a discrete field with all of its closed V-paths trivial.

From now on, we call a discrete gradient field simply a *discrete gradient*. A discrete Morse function $f : X \longrightarrow \mathbb{R}$ induces a discrete gradient $V(f)$ on X. The following characterization holds.

1.4.6 Theorem (Theorem 3.5 in [88]). *A discrete vector field V is the gradient vector field of a discrete Morse function f if and only if there are no non-trivial closed V-paths.*

The theorem ensures that, by studying discrete gradients over a regular cell complex, we get the same information as studying discrete Morse functions. In the rest of this thesis, we focus on discrete gradients rather than discrete Morse functions. This allows us to avoid possible confusion with the several *filtering functions* that are present in this thesis and are introduced in Section 1.5 in the context of persistent homology. We remark that filtering functions are never required to be discrete Morse functions.

1.4.2 Discrete Morse complex

In this section, we introduce the notion of a *Morse complex* and *Morse reduction*. The Morse complex construction depends on a coefficient ring. Here, we consider only coefficients in the field $\mathbb{Z}_2$ with only two elements. We introduce the following definition.

1.4.7 Definition (Morse set)**.** Given a discrete gradient V the *Morse set* $M(V)$ relative to V is the collection of all critical cells in X with respect to V. When clear from the context, we indicate $M(V)$ simply by M. We denote by M_k the set of critical cells of dimension k.

Given a discrete gradient V, a *separatrix from a critical cell* t^{k+1} *to a critical cell* s^k is a V-path from any face of t^{k+1} to s^k. We can translate incidence relations among critical cells in M into algebraic terms by means of the *Morse incidence function* $\kappa_M : M \times M \longrightarrow \mathbb{F}_2$ defined by

$$\begin{aligned} \kappa_M(t,s) = 1 \quad &\Leftrightarrow \quad \text{the number of different separatrices from } t \text{ to } s \text{ is odd} \\ \kappa_M(t,s) = 0 \quad &\Leftrightarrow \quad \text{the number of different separatrices from } t \text{ to } s \text{ is even} \end{aligned} \tag{1.7}$$

The function κ_M is an actual incidence function in the sense of Definition 1.3.1. Indeed, only critical cells whose dimensions differ by 1 can have non-null value under the incidence function κ_M. In particular, the fact that the function κ_M fulfills condition (ii) follows from Theorem 8.10 in [87] in the particular case of $\mathbb{F}_2$ coefficients and from the Remark 1.3.2.

1.4.8 Definition. The *Morse complex* is the Lefschetz complex (M, κ_M) obtained from a discrete gradient V where:

- $M = M(V)$ is the Morse set with respect to V,
- $\kappa_M = \kappa_M(V)$ is the Morse incidence function obtained from V.

The combination of Forman's Theorems 7.3 and 8.2 [87] proves, in particular, the following result.

1.4.9 Theorem (Morse reduction)**.** *Given a Morse set M over a cell complex X, the inclusion-induced map*

$$\varphi_k : H_k(M) \longrightarrow H_k(X)$$

is an isomorphism for all integers k.

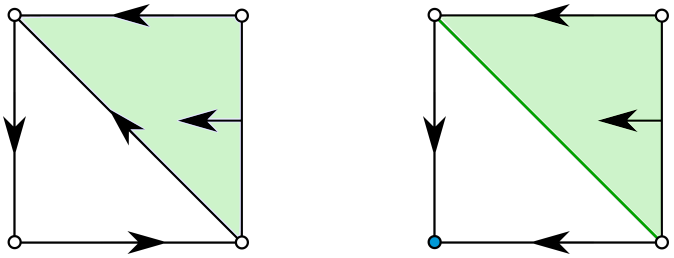


Figure 1.4: two discrete vector fields over the same cell complex. Arrows denote discrete vectors. The first component in discrete vectors correspond to the tail and second component to the head of the arrow. In (a), the discrete field is not a discrete gradient: the arrows around the striped zone form a non-trivial closed V-path. In (b), the discrete field is a discrete gradient. The only critical vertex is depicted in blue. The only critical 1-cell is depicted in green.

1.4.10 Example. *Consider X being the cell complex in Figure 1.4. Its homology is generated by a single connected component in degree 0 and a single 1-cycle. The connected component corresponds to the blue critical vertex v in the Morse set relative to the discrete gradient indicated by the arrows. The loop corresponds to the green critical 1-cell s. Indeed, v and s are both cycles since there are exactly two separatrices linking the two critical cells which implies $\kappa_M(s,v) = 0$.*

1.5 One-parameter persistent homology

In this section, we introduce the necessary background in persistent homology [80, 34]. In order to underline the special character of persistent homology with respect to its multiparameter counterpart treated in Section 1.6, we call it *one-parameter persistent homology*, or simply *one-persistent homology*. Whereas, by *persistent homology*, we mean both one-parameter and multiparameter persistent homology. We warn the reader that the content of Section 1.6 includes the content of this section as a special case. In this section, we refer to the notion of homology of a Lefschetz complex as introduced in Section 1.3. In Section 1.5.1, we introduce the notion of a *one-parameter filtration*. This defines an intermediate step towards multiparameter filtrations introduced in Section 1.6.1. Moreover, this defines the object where to apply the construction of one-persistent homology in Section 1.5.2 that leads to the definition of the *persistence module* in the one-parameter case. The analogous construction for multiparameter persistent homology is given in Section 1.6.2.

1.5.1 One-parameter filtrations

In this section, we introduce the notion of a *one-parameter filtration* of a Lefschetz complex L (Definition 1.3.1). Lefschetz complexes are taken into account to include both cases of regular cell complexes (Definition 1.1.11) and Morse complexes (Definition 1.4.8). Afterwards, we introduce our own notion of a discrete gradient and a Morse complex *compatible with a one-parameter filtration*. These latter is generalized in Section 1.6.1.

1.5.1 Definition (One-parameter filtration)**.** A *one-parameter filtration*, or simply a *one-filtration of* L is a map F taking each $u \in \mathbb{Z}$ to a closed subcomplex L^u of L in such a way that

- $L^0 = \varnothing$;
- there exists a $z \in \mathbb{Z}$ such that $L^z = L$;
- $u \leqslant v$ in $\mathbb{Z}$ implies that L^u is a closed subcomplex of L^v.

Each element $u \in \mathbb{Z}$ is called a *filtration grade*, or simply a *grade*. Each subcomplex L^u is called a *filtration step* or simply a *step*. We call the image $\mathcal{L} = F(\mathbb{Z})$ a *filtered (Lefschetz) complex*. We say that L is *filtered by* F or, indifferently, by $\mathcal{L}$.

We introduce the following property of a discrete gradient over a regular cell complex with respect to a one-filtration.

1.5.2 Definition (Discrete gradient compatible with a one-filtration)**.** Given a regular cell complex X filtered by $\mathcal{X}$, a discrete gradient V over X is said to be *compatible with* $\mathcal{X}$ if

$$\forall (a,b) \in V, \quad \forall u \in \mathbb{Z}, \quad a \in X^u \Leftrightarrow b \in X^u.$$

A Morse complex M obtained from a discrete gradient compatible with a filtered complex $\mathcal{X}$ is said to be *compatible with* $\mathcal{X}$.

1.5.3 Definition (Morse filtered complex)**.** Given a filtered complex $\mathcal{X}$ relative to X with grades in $\mathbb{Z}$ and a Morse complex M over X, we define the *Morse filtered complex* $\mathcal{M}$ relative to M and *compatible with* $\mathcal{X}$ by setting in each filtration step (M^u, κ_M^u), for $u \in \mathbb{Z}$,

$$\begin{aligned} M^u &:= \{a \in M \mid a \in X^u\}, \\ \kappa_M^u &:= \kappa_M \text{ restricted to } M^u \times M^u. \end{aligned}$$

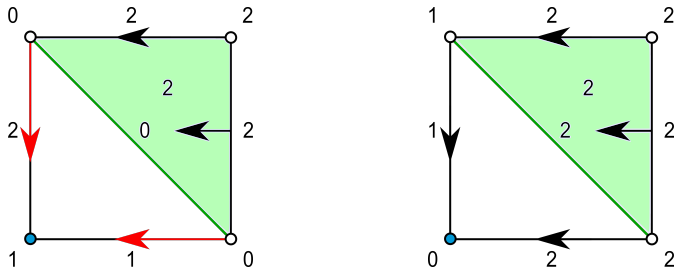


Figure 1.5: a one-filtration indicated by numbers over cells. A discrete gradient which is non-compatible with the one-filtration (a), and another discrete gradient which is compatible with the same one-filtration (b). Arrows denote discrete vectors, red arrows denote non-compatible discrete vectors, colored cells are critical cells. The first component in discrete vectors correspond to the tail and second component to the head of the arrow.

1.5.4 Definition (One-parameter filtering function)**.** A *one-parameter filtering function over* X is a function $\phi : X \longrightarrow \mathbb{Z}$ such that, for all cells $a, b \in X$

$$a < b \quad \Rightarrow \quad \phi(a) \leqslant \phi(b).$$

A filtering function ϕ of (X, κ_S) induces a *sublevel filtration* F^ϕ by setting $F^\phi(u) = (X^u, \kappa_S^u)$ with

$$X^u := \{a \in X \mid \phi(a) \leqslant u\},$$
$$\kappa_S^u := \kappa_S \text{ restricted to } X^u \times X^u.$$

Indeed, each set X^u is closed in X and then in each X^v with $u \leqslant v$. The filtered complex $\mathcal{X}$ obtained through sublevels under ϕ is denoted by $\mathcal{L}^\phi$.

1.5.5 Remark. *A one-parameter filtered complex $\mathcal{X}$ can always be thought of as a $\mathcal{X}^\phi$ for some one-parameter filtering function ϕ. Indeed, it suffices to take, for each cell a in X, $\phi(a)$ equal to the smallest grade u such that $a \in X^u$.*

1.5.2 One-parameter persistence module

Given a filtered complex $\mathcal{L}$, the homology construction $H_k(\cdot; R)$ can be applied to each filtration step. Since all definitions in this section does not depend on R, we skip to indicate it. Each filtration step has its associated chain complex and, by following Remark 1.2.4, each inclusion between successive filtration steps induces a linear map of corresponding homology spaces

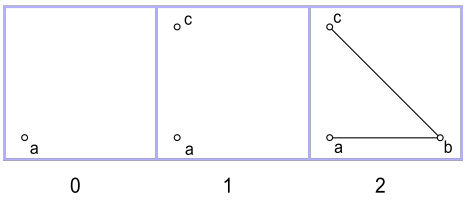
$$L^u \subseteq L^v \quad \Rightarrow \quad \imath^{u,v} : H_k(L^u) \longrightarrow H_k(L^v)$$

As already mentioned, maps $\imath^{u,v}$ are not necessarily injective or surjective. In this sense, a class is *persistent from grade u to grade v* if it is not trivial in $H_k(L^u)$ and still non-trivial in $H_k(L^v)$. This motivates the following definition.

1.5.6 Definition (One-persistent homology)**.** Given a Lefschetz complex L filtered by $\mathcal{L}$ with filtration grades in $\mathbb{Z}$, the *one-persistent k^{th}-homology from grade u to grade v*, with $u \leqslant v$, is the image of the inclusion-induced map $\imath^{u,v}$ of $H_k(L^v)$.

Let us consider a Lefschetz complex L filtered $\mathcal{L}$. The global information of one-persistent homology for all possible filtration grades $u \leqslant v$ is captured by the following definition.

1.5.7 Definition (One-parameter k^{th}-persistence module)**.** The *one-parameter k^{th}-persistence module*, or simply, *one-persistence module* $H_k(\mathcal{L})$ of a filtered complex $\mathcal{L}$ is the family of homology spaces $H_k(L^u)$ with $u \in \mathbb{Z}$ along with all linear maps $\imath^{u,v}$ with $u \leqslant v$ induced by the inclusion of complexes.



$$\mathbb{F}_2\langle \bar{a} \rangle \xrightarrow{\binom{1}{0}} \mathbb{F}_2\langle \bar{a}, \bar{c} \rangle \xrightarrow{(1\ \ 1)} \mathbb{F}_2\langle \bar{a} \rangle$$

(a) (b)

Figure 1.6: In picture (a), we see the two-parameter filtration obtained by considering the one-parameter filtrations in Figure 1.5 together. In the diagram (b), a representation of the corresponding persistence module in degree 0 (connected components) is depicted. For each $\mathbb{F}_2$-vector space in the persistence module, the corresponding reference basis is made explicit, e.g., $\bar{a}$ is the homology class of the vertex a. Each matrix expresses a linear linear map in terms of those bases.

We observe that, for a Lefschetz complex L filtered by $\mathcal{L}$, if $z \in \mathbb{Z}$ is such that $L^z = L$, then $\imath^{z,w}$ is an isomorphism for every $w \geqslant z$.

The Morse reduction result Theorem 1.4.9 has a counterpart for filtered complexes. The result is implied by Theorem 4.3 in [144].

1.5.8 Theorem (One-parameter compatible Morse reduction)**.** *Given a regular cell complex X filtered by a $\mathcal{X}$ with filtration grades in $\mathbb{Z}$ and a Morse complex (M, κ_M) compatible with $\mathcal{X}$, the*

inclusion-induced maps at each grade $u \in \mathbb{Z}$

$$\varphi_k^u : H_k(M^u) \longrightarrow H_k(X^u)$$

are isomorphisms for all integers k *making all the inclusion-induced maps in each filtration commute.*

1.6 Multiparameter persistent homology

In this section, we generalize the background notions given for one-parameter persistent homology in Section 1.5 to the multiparameter case, called *multipersistent homology*. Multipersistent homology constitutes the main focus of this thesis. In this section, we refer to the notion of homology of a Lefschetz complex as introduced in Section 1.3. In Section 1.6.1, we introduce the notion of a *multiparameter filtration*, or simply a *multifiltration*. This defines the object where to apply the construction of multipersistent homology in Section 1.6.2 that leads to the definition of the *persistence module* in the multiparameter case.

In place of a finite total order $(\mathbb{Z}, \leqslant)$, the multiparameter case is based on the partially ordered set $(\mathbb{Z}^n, \preceq)$ with $\preceq$ defined by setting, for any $u = (u_1, \ldots, u_n), v = (v_1, \ldots, v_n) \in \mathbb{Z}^n$,

$$u \preceq v \qquad \Leftrightarrow \qquad \forall i \in \{1, \ldots, n\}, \quad u_i \leqslant v_i.$$

We call $(\mathbb{Z}^n, \preceq)$ the *grade poset*, and *multigrades* its elements. If neither $u \preceq v$ or $v \preceq u$, we call the two multigrades u and v *incomparable*, and *comparable* otherwise. If $u \preceq v$ and $u \neq v$, we shortly write $u \prec v$. The element of $\mathbb{Z}^n$ with all null entries is simply denoted by 0.

1.6.1 Multiparameter filtrations

This section is generalizes concepts from Section 1.5.1 to the multiparameter case.

1.6.1 Definition (Multiparameter filtration)**.** A *multiparameter filtration*, or simply a *multifiltration of* L is a map F associating each $u \in \mathbb{Z}^n$ with a closed subcomplex L^u in L in such a way that

- $L^0 = \varnothing$;
- there exists a grade z such that $L^z = L$,
- $u \preceq v$ implies that L^u is a closed subcomplex of L^v.

Filtration multigrades and *steps* are defined analogously to Definition 1.5.1. We call the image $\mathcal{L} = F(\mathbb{Z}^n)$ a *(multi)filtered (Lefschetz) complex*. We say that L is *(multi)filtered by* F or, indifferently, by $\mathcal{L}$.

In the work of this thesis, we also consider filtrations with filtration multigrades over discrete subsets of $\mathbb{R}^n$ rather than $\mathbb{Z}^n$. In this case, the multigrade set $\mathbb{Z}^n$ can be easily mapped to a discrete subset of $\mathbb{R}^n$. We refer the reader to [44, 141, 52, 132] for more details on filtrations and filtered complexes indexed over general real parameters.

1.6.2 Definition (Discrete gradient compatible with a multifiltration)**.** Given a regular cell complex X multifiltered by $\mathcal{X}$, a discrete gradient V over X is said to be *compatible with* $\mathcal{X}$ if,

$$\forall (a,b) \in V, \quad \forall u \in \mathbb{Z}^n, \quad a \in X^u \Leftrightarrow b \in X^u.$$

In the case of a Morse complex M obtained from a discrete gradient compatible with a filtered complex $\mathcal{X}$, we call M a Morse complex *compatible with* $\mathcal{X}$.

1.6.3 Definition (Morse multifiltered complex)**.** Given a regular cell complex X multifiltered by $\mathcal{X}$, and a Morse complex M over X, we define the *Morse filtered complex* $\mathcal{M}$ relative to M and *compatible with* $\mathcal{L}$ by setting, in each filtration step (M^u, κ_M^u) for $u \in \mathbb{Z}$,

$$\begin{aligned} M^u &:= \{a \in M \mid a \in X^u\}, \\ \kappa_M^u &:= \kappa_M \text{ restricted to } M^u \times M^u. \end{aligned}$$

1.6.4 Definition (Multiparameter filtering function)**.** A *multiparameter filtering function over* X is a function $\phi : X \longrightarrow \mathbb{Z}^n$ such that, for all cells $a, b \in X$

$$a < b \quad \Rightarrow \quad \phi(a) \preceq \phi(b).$$

A filtering function ϕ of (X, κ_S) induces a *sublevel filtration* by setting $F(u) = (X^u, \kappa_S^u)$ with

$$\begin{aligned} X^u &:= \{a \in X \mid \phi(a) \preceq u\}, \\ \kappa_S^u &:= \kappa_S \text{ restricted to } X^u \times X^u. \end{aligned}$$

Indeed, each set X^u is closed in X and then in each X^v with $u \leqslant v$. The multifiltered complex $\mathcal{X}$ obtained through sublevels under ϕ is denoted $\mathcal{L}^\phi$.

1.6.5 Remark. *If $n \geqslant 2$, filtrations induced by functions give a proper subclass of general filtrations. Indeed, filtrations of kind $\mathcal{L}^\phi$ are* one-critical*, which according to [33] , means that the minimal grade $u \in I^n$ a cell in L belong to in $\mathcal{L}(\phi)$ is unique.*

In the case of a Morse complex M compatible with the filtered complex $\mathcal{L}^\phi$ induced by sublevel sets by a filtering function ϕ, we way that M is *compatible with* ϕ.

1.6.2 Multiparameter persistence module

As for the one-parameter case, given a multifiltered Lefschetz complex $\mathcal{L}$, the homology construction $H_k(\cdot)$ can be applied to each filtration step. By Remark 1.2.4, each inclusion between filtration steps induces a linear map of corresponding homology spaces

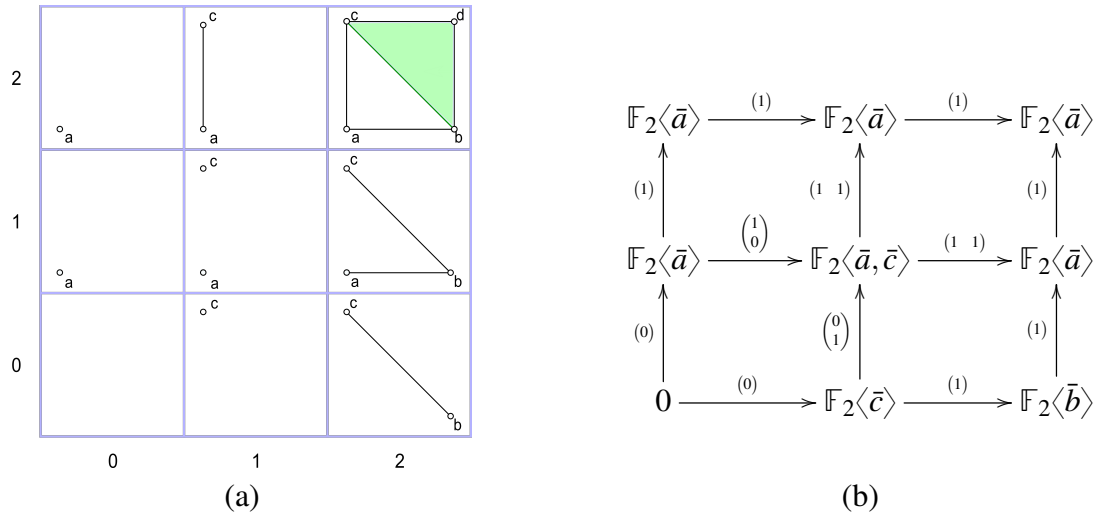
$$L^u \subseteq L^v \quad \Rightarrow \quad \imath^{u,v} : H_k(L^u) \longrightarrow H_k(L^v)$$

The notion of homology class persistent from one step to another is analogous to the one-parameter case. A homology class is *persistent from grade u to grade v* if it is not trivial in $H_k(L^u)$ and still non-trivial in $H_k(L^v)$. This motivates the following definition generalizing Definition 1.5.6.

1.6.6 Definition (Multipersistent homology)**.** Given a Lefschetz complex L multifiltered by $\mathcal{L}$, i.e., with filtration grades in $\mathbb{Z}^n$, the *multipersistent k^{th}-homology from grade u to grade v*, with $u \preceq v$, is the image of the inclusion-induced map $\imath^{u,v}$ of $H_k(L^v)$.

The global information of multipersistent homology for all possible filtration grades $u \preceq v$ is captured by the following definition.

1.6.7 Definition (Multiparameter k^{th}-persistence module)**.** The *multiparameter persistence k^{th}-module* $H_k(\mathcal{L})$ relative to the multifiltered complex $\mathcal{L}$ is the family of homology spaces $H_k(L^u)$ with $u \in \mathbb{Z}^n$ along with all linear maps $\imath^{u,v}$ with $u \preceq v$ induced by the inclusion of complexes.



(a) (b)

Figure 1.7: in picture (a), we see the two-parameter filtration obtained by considering the one-parameter filtrations in Figure 1.5 together. In the diagram (b), a representation of the corresponding persistence module in degree 0 (connected components) is depicted. For each $\mathbb{F}_2$-vector space in the persistence module, the corresponding reference basis is made explicit, e.g., $\bar{a}$ is the homology class of the vertex a. Each matrix expresses a linear linear map in terms of those bases.

We observe that, for a Lefschetz complex L multifiltered by $\mathcal{L}$, if $z \in \mathbb{Z}^n$ is such that $L^z = L$, then $\imath^{z,w}$ is an isomorphism for every $w \geqslant z$.

The one-parameter compatible Morse reduction Theorem 1.5.8 has a counterpart for multiparameter filtered complexes implied by Corollary 2 in [4].

1.6.8 Theorem (Multiparameter compatible Morse reduction)**.** *Given a Lefschetz complex L multifiltered by $\mathcal{L}$ and a Morse complex L compatible with $\mathcal{X}$, the inclusion-induced maps at each grade $u \in \mathbb{Z}^n$*

$$\varphi_k^u : H_k(M^u) \longrightarrow H_k(X^u)$$

are isomorphisms for all integers k making all the inclusion-induced maps in each filtration commute.

1.7 Invariants in persistent homology

The aim of this section is to introduce and discuss the invariants associated with a persistence module that are more relevant in the context of this thesis. We emphasize similarities and dissimilarities between the one-parameter and the multiparameter case.

The section is structured as follows. To begin with, we formalize what we mean by the word "invariant". Moreover, we introduce the *rank invariant* as an example of invariant formally expressed in the same way in both the one and the multiparameter cases. In Section 1.7.1, we introduce the *persistence diagram* as an example of complete invariant available for one-parameter persistence and observe its relation with the rank invariant. In Section 1.7.2, we introduce the *persistence space* as the multipersistence invariant of major interest for this thesis. We see how the persistence space generalizes the persistence diagram. Both persistence spaces and diagrams constitute the necessary background for the experimental results in Section 6.3. Another class of persistence invariants comprises those of algebraic nature. In Section 1.7.3, we introduce the algebraic formulation of the persistence module and the invariants called *multigraded Betti numbers*. This setting is the preferable one to make it explicit the different character when moving from one to multiparameter persistence, so in conclusion, we add some additional remarks.

1.7.1 Definition (Invariant)**.** Given a set E whose elements admits an equivalence relation $\cong$, and a set T, an *invariant on E with values in T* is a function $\mathrm{Inv} : E \longrightarrow T$ such that, for all $e, e' \in E$

$$e \cong e' \qquad \Rightarrow \qquad \mathrm{Inv}(e) = \mathrm{Inv}(e').$$

The function Inv is a *complete invariant* if also the converse holds, that is,

$$\mathrm{Inv}(e) = \mathrm{Inv}(e') \quad \Leftrightarrow \quad e \cong e'.$$

The topic of this section is to define the available invariants on E, when E is the class of persistence modules. Recall that the persistence module (see Definition 1.6.7) depends on a filtration F and the choice of the homology functor in some integer degree k. Recall what a filtration F is from Definition 1.6.1. We indicate by $F(\mathbb{Z}^n)$ when necessary to remark that the filtration parameters run over n copies of a set $\mathbb{Z}$. Recall the definition of the k^{th}-homology functor $H_k(-;\mathbb{F})$, with $\mathbb{F}$ a field, introduced in Definition 1.2.2 and, from now on, indicated by $H_k(-)$. All along this section, we set M to be a persistence module such that $M = H_k \circ F(\mathbb{R}^n)$. In Section 1.7.1, we treat the case $n = 1$. In Section 1.7.1, we treat the general case $n \geqslant 1$.

A first persistence invariant can be defined indifferently in the one and the multiparameter cases. Denote by $\Delta^n_{\preceq}$ the set $\{(u,v) \in \mathbb{R}^n \times \mathbb{R}^n \mid u \preceq v\}$. Moreover, denote by $\Delta^n_{\preceq,\infty}$ the set $\Delta^n \cup \{(u,+\infty) \mid u \in \mathbb{R}^n\}$, where $+\infty$ is an additional element greater than all others in $\mathbb{R}^n$. For each $(u,v) \in \Delta^n_{\preceq,\infty}$ consider the corresponding inclusion-induced map in the persistence module M:

$$\imath_M^{u,v} : M^u \longrightarrow M^v.$$

1.7.2 Definition (Rank invariant)**.** The *rank invariant of* M is the function $\text{rank}_M : \Delta^n_{\preceq,\infty} \longrightarrow \mathbb{N}$ defined by

$$\text{rank}_M(u,v) := \text{rank}\, \imath_M^{u,v}.$$

For each pair of filtration grades $u \preceq v$, the rank invariant counts the number of homology classes persistent through $\imath_M^{u,v}$ from grade u to grade v. In the next sections, we see, in particular, how the rank invariant properties vary according to the character of parameter n.

1.7.1 Invariants in one-persistent homology

In this section, we focus on the case $n = 1$. We introduce the *persistence diagram* as an example of complete invariant for one-parameter persistence. We define the persistence diagram, originally introduced for one-parameter persistence modules with a countable number of filtration steps [75], by following the definition for uncountable filtration steps given in [49]. We focus on this invariant since our experimental tests in Section 6.3 directly involve it in the computations.

The persistence diagram is a representation in the Cartesian plane $\mathbb{R} \times \mathbb{R}$ of a one-parameter persistence module M. Each point in the persistence diagram represents a discontinuity in the rank invariant values which can be given either by a new-born or by a vanishing homology class in M. The rank invariant is defined for points in the Cartesian plane which lay on the diagonal or above. Discontinuities are checked only in the inner part of the rank invariant domain. Denote by $\Delta^1_<$ the set $\{(u,v) \in \mathbb{R} \times \mathbb{R} \mid u < v\}$. Moreover, denote by $\Delta^1_{<,+\infty}$ the set $\Delta^1 \cup \{(u,+\infty) \mid u \in \mathbb{R}\}$, where $+\infty$ is an additional element greater than all others in $\mathbb{R}$. Each pair (u,v) of grades in $\Delta^1_{<,+\infty}$ is given a multiplicity. The multiplicity function detects discontinuities of the rank invariant point-wise.

1.7.3 Definition (Definition 1 in [49])**.** The *multiplicity function* of the one-parameter persistence module M is the function $\mu_M : \Delta^1_{<,+\infty} \longrightarrow \mathbb{N}$ defined by

$$\begin{aligned} \mu_M(u,v) &:= \min_e \Big\{ \operatorname{rank}_M(u+e, v-e) - \operatorname{rank}_M(u-e, v-e) + \\ &\quad - \operatorname{rank}_M(u+e, v+e) + \operatorname{rank}_M(u-e, v+e) \Big\} \\ \mu_M(u,+\infty) &:= \min_e \Big\{ \operatorname{rank}_M(u+e, v) - \operatorname{rank}_M(u-e, v) \Big\}, \end{aligned} \tag{1.8}$$

according to v being in $\mathbb{R}$ or equal to $+\infty$, and with $e > 0$ such that $(u+e, v-e)$ belongs to $\Delta^1_{<,+\infty}$.

1.7.4 Definition (Persistence diagram)**.** Let M be a one-parameter persistence module. The *persistence diagram of* M is the multiset PDgm_M defined by the set $\{(u,v) \in \Delta^1_{<,+\infty} \mid \mu_M(u,v) > 0\}$ whose elements (u,v) are counted with multiplicity $\mu_M(u,v)$.

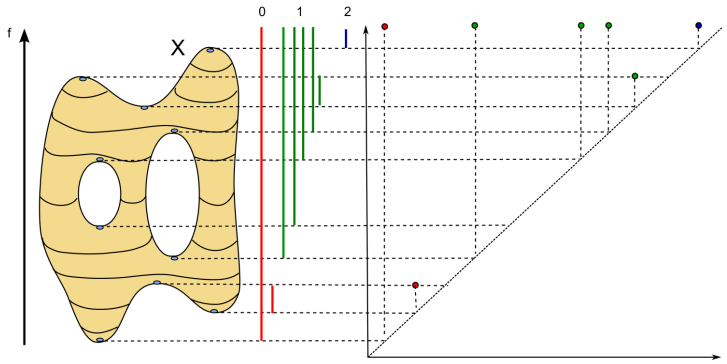


Figure 1.8: an example of persistence diagram along (on the right) and its corresponding barcode (center) with respect to sublevel sets (shape on the left).

Source: image showed under gentle permission of the author, Barbara Di Fabio.

1.7.5 Example. *In Figure 1.8, we see an example of filtration of a shape by sublevel sets with its associated persistence diagram. We see, in red, homology classes in degree 0, in green the degree 1, in blue the degree 2. The bars visualize the barcode corresponding to the persistence diagram.*

We warn the reader that, in the work given as a reference [49], the persistence diagram also includes the diagonal points in the Cartesian plane. This choice is useful for defining the *matching distance* between persistence diagrams. We have chosen to adopt the given definition to better highlight the meaning of points in the persistence diagram.

It follows from Definition 1.7.4 that, by knowing the rank invariant, one can always retrieve the persistence diagram. Conversely, the following result, also known as Triangle Lemma [80], ensures that, in the one-parameter case, the rank invariant is determined by the persistence diagram.

1.7.6 Theorem (Theorem 1 in [49]). *For all* $(u', v') \in \Delta^1_{\leq,+\infty}$, *it holds that*

$$\operatorname{rank}_M(u', v') = \sum_{\substack{u \leqslant u' \\ v' < v}} \mu(u, v) + \sum_{u \leqslant u'} \mu(u, +\infty). \tag{1.9}$$

Hence, the two invariants are equivalent.

Completeness. We postpone the completeness discussion to Section 1.7.3. Here, we motivate completeness by recalling the main steps in the proof. By Theorem 1.7.12, persistence modules admit an algebraic formulation in terms of finitely presented graded modules. The completeness of the persistence space follows by the decomposition given by Theorem 1.7.14 for finitely presented graded modules. Under that decomposition, each pair (u, v) in the persistence diagram with non-null multiplicity $\mu_M(u, v)$ corresponds to $\mu_M(u, v)$ different homology classes in M appearing precisely at filtration grade u and disappering precisely at filtration grade v (never disappearing if $v = +\infty$). Being equivalent to the persistence diagram, the rank invariant also is a complete invariant for one-parameter persistence modules.

1.7.7 Remark. *Often, the notion of a persistence diagram is equivalently expressed by the* barcode. *The barcode* BC_M *consists of a multiset of intervals. Each element* (u, v) *in a persistence diagram corresponds one-to-one to a* persistence bar, *that is an half-open interval* $[u, v)$ *on the real line. The multiplicity of each interval is the same as that in the persistence diagram.*

Invariants representation remarks. We close this section by observing a crucial difference for computational purposes between the persistence diagram and rank invariant in the one-parameter case. The rank invariant is less convenient to be encoded since it requires to represent a natural number for each pair in $\Delta^1_{\leq,+\infty}$. Instead, the persistence space requires to represent only those pairs in $\Delta^1_{<,+\infty}$ with positive persistence. The rank invariant is not discrete, in the sense that, the cardinality of its domain is uncountable when the set of filtration grades is uncountable. On the contrary, the number of points in the persistence diagram depends on the homology features changing along M. That number is finite whenever the filtration considered is a filtration of a finite Lefschetz complex, even with uncountable filtration grades.

1.7.2 Invariants in multipersistent homology

In this section, we focus on the case $n \geqslant 1$. We introduce the *persistence space* which generalizes the persistence diagram to multiparameter persistence modules. We define the persistence space, by following the construction given in [49]. We focus on this invariant since, in our experimental

tests in Section 6.3, we retrieve an approximate version of it. A the end of the section we highlight theoretical dissimilarities with the one-parameter case. Dissimilarities will be clarified into more details in Section 1.7.3.

In order to generalize to multipersistence the persistence diagram, the idea consists in:

- generalizing the multiplicity function of Definition 1.7.3;
- generalizing Theorem 1.7.10 for the generalized multiplicity function.

To address the first step, recall from Definition 1.7.2 that, for $n \geqslant 1$, values of the rank invariant are defined above the diagonal in the space $\mathbb{R}^n \times \mathbb{R}^n$. Denote by $\Delta^n_{\prec}$ the set $\{(u,v) \in \mathbb{R}^n \times \mathbb{R}^n \mid u \prec v\}$ with $u \prec v$ if the strict inequalities $u_i < v_i$ hold for all components. Moreover, denote by $\Delta^n_{\prec,\infty}$ the set $\Delta^n \cup \{(u,+\infty) \mid u \in \mathbb{R}^n\}$, where $+\infty$ is an additional element greater than all others in $\mathbb{R}^n$. The first step is obtained through the following definition.

1.7.8 Definition (Definition 3 in [49])**.** The *multiplicity function* of the multiparameter persistence module M is the function $\mu_M : \Delta^n_{\prec,+\infty} \longrightarrow \mathbb{N}$ defined by

$$\begin{aligned} \mu_M(u,v) := \min_e \quad & \Big\{ \operatorname{rank}_M(u+e, v-e) - \operatorname{rank}_M(u-e, v-e) + \\ & - \operatorname{rank}_M(u+e, v+e) + \operatorname{rank}_M(u-e, v+e) \Big\} \\ \mu_M(u,+\infty) := \min_e \quad & \Big\{ \operatorname{rank}_M(u+e, v) - \operatorname{rank}_M(u-e, v) \Big\}, \end{aligned} \tag{1.10}$$

according to v being in $\mathbb{R}^n$ or equal to $+\infty$, and provided to take $e \in \mathbb{R}^n$ such that $e > 0$ is sufficiently close to 0 with respect to the infinity norm, and such that the pair $(u+e, v-e)$ belongs to $\Delta^n_{\prec,+\infty}$.

A part from the restriction on the vector e being close enough to 0, the multiplicity function directly generalizes Definition 1.7.3. The need of this requirement is explained in Proposition 2 in [49] and it is aimed at preserving monotonicity properties of the rank invariant with respect to component-wise perturbations of (u,v). This leads to the definition of persistence space in the following terms.

1.7.9 Definition (Persistence space)**.** Let M be a multiparameter persistence module. The *persistence space of* M is the multiset PSpc_M defined by the set $\{(u,v) \in \Delta^n_{\prec,+\infty} \mid \mu_M(u,v) > 0\}$ whose elements (u,v) are counted with multiplicity $\mu_M(u,v)$.

As already done in the persistence diagram case, we warn the reader that, in the referenced work [49], the persistence space also includes the diagonal points in $\mathbb{R}^n \times \mathbb{R}^n$. This choice is useful for defining the *multidimensional matching distance* between persistence spaces. We have chosen to adopt the given definition to better highlight the meaning of points in the persistence

diagram. It follows from Definition 1.7.9 that, by knowing the rank invariant, one can always retrieve the persistence space. Conversely, the following result addresses the second step of the desired generalization, thus guaranteeing that the rank invariant is determined by the persistence space.

1.7.10 Theorem (Theorem 2 in [49]). *Let $(u',v') \in \Delta^n_{\leq,+\infty}$. If $e \succ 0$ in $\mathbb{R}^n \times \mathbb{R}^n$ and $\langle e \rangle$ indicates the linear space spanned by e in $\mathbb{R}^n \times \mathbb{R}^n$, then it holds that*

$$\mathrm{rank}_M(u',v') = \sum_{\substack{u \leq u', u'-u \in \langle e \rangle \\ v' \prec v, v-v' \in \langle e \rangle}} \mu(u,v) + \sum_{u \leq u', u'-u \in \langle e \rangle} \mu(u,+\infty). \tag{1.11}$$

A part from the restriction of summands components to be chosen along the direction of $\langle e \rangle$, the result directly generalizes Theorem 1.7.10. The role of $\langle e \rangle$ is explained by the slice-compatibility of the rank invariant stated in Equation (2.1) in Section 2.4.1. Hence, the two invariants are equivalent. Other available invariants are discussed in Section 2.4.1.

Non-completeness. The persistence space is not complete for the class of multiparameter persistence modules. Its non-completeness is first discussed by Carlsson and Zomorodian in [35]. In Section 1.7.3, we explain it into more details by moving to an algebraic formulation of the persistence module. Here, we summarize the most crucial points to motivate it. By Theorem 1.7.12, persistence modules admit an algebraic formulation in terms of finitely presented multigraded modules. The non-completeness of the persistence space follows by the impossibility of generalizing to multigraded modules the decomposition given by Theorem 1.7.14 for finitely presented graded modules. Being equivalent to the persistence space, the rank invariant also is a non-complete invariant for one-parameter persistence modules.

1.7.11 Example. *In Figure 1.9, we see two different persistence modules with the same rank invariant (the example is taken from Example 2.2 in [133]). This shows that the rank invariant is not complete.*

Geometrical meaning. The geometrical meaning of the persistence space is highlighted by its *slice-compatibility* property. For an invariant over persistence modules is the property ensuring that, if restricted to any linear one-persistence submodule called *slice* (Definition 2.4.1), the invariant corresponds to a one-persistence invariant. In Section 2.4.1, we present the slice-compatibility of the persistence space which, over each slice, is equal to a persistence diagram. That property highlights the geometrical meaning of a point in a persistence space. For each pair (u,v) in the persistence space with non-null multiplicity $\mu_M(u,v)$, there exists a slice, and a pair (a,b) in the persistence diagram of the slice, where (a,b) corresponds to $\mu_M(u,v)$ different homology classes in M appearing precisely at filtration grade a and disappearing precisely at filtration grade b (never disappearing if $b = +\infty$).

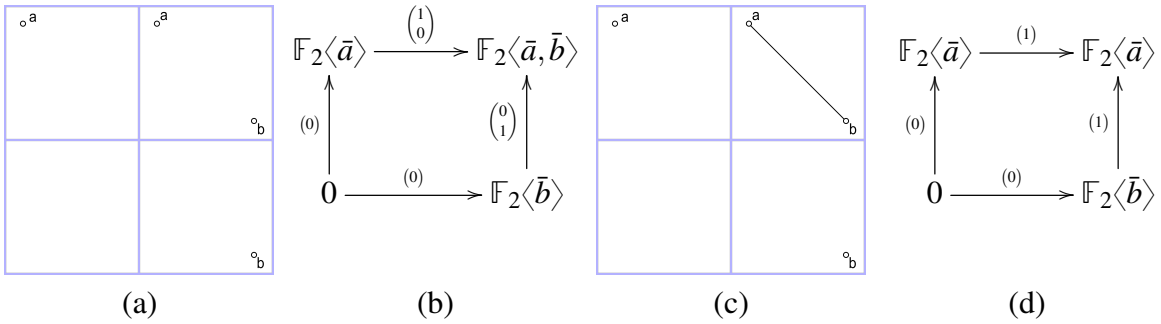


Figure 1.9: Two two-parameter filtrations (a) and (c) with their corresponding persistence modules (b) and (d), respectively.

1.7.3 Algebraic invariants in persistent homology

In this section, we introduce the algebraic formulation of the persistence module. This setting is the preferable one to explain why, in the one-parameter persistence case, we have complete invariants available whereas, in the multipersistence case, we have not. Next, we introduce the invariants called *multigraded Betti numbers* and provide our own perspective on their geometrical meaning.

The algebraic formulation of the persistence module we follow is the one outlined by Carlsson and Zomorodian in [35] and, then refined by Corbet and Kerber in [56]. Consider a filtration F with grades in $\mathbb{Z}^n$ of some Lefschetz complex. Consider the k^{th}-homology construction $H_k(-;A)$ with coefficients in a commutative ring with unity A. Denote by M the persistence module $H_k(-;A) \circ F$ with $M^u = H_k(F(u);A)$, and with $\imath^{u,v}$ denoting the maps induced by the inclusion from grade u to grade v. We want to look at M as a multigraded $A[x_1,\ldots,x_n]$-module $\alpha(M)$. In order to define the *corresponding multigraded module* $\alpha(M)$, we determine, for each $u \in \mathbb{Z}^n$, its homogeneous part $\alpha(M)^u$ and, for each $z = (z_1,\ldots,z_n) \in \mathbb{Z}^n$ the action of the monomials $x^z = x_1^{z_1} \cdots x_n^{z_n}$ over elements in $\alpha(M)$:

$$\begin{aligned} \alpha(M)^u &:= M && \textit{homogeneous part of multigrade } u \\ x^{v-u} \cdot \alpha(M) &:= \imath^{u,v}(\alpha(M)) && A[x_1,\ldots,x_n]\textit{-action of } z = v-u \textit{ over } \alpha(M) \in \alpha(M)^u. \end{aligned}$$

In the following statement, we adapt Theorem 1 in [35] to the refinement for finitely presented modules discussed in Theorem 9 in [56].

1.7.12 Theorem (Correspondence)**.** *The correspondence α defines an equivalence of categories between the category of persistence modules with of finite type with coefficients in A and n parameters, and the category of finitely presented multigraded modules over $A[x_1,\ldots,x_n]$.*

The rank invariant in algebraic terms. Under the correspondence α and by taking $\Delta^n_{\prec,+\infty}$ as in the rank invariant definition (see Definition 1.7.2), we have that, for all $(u,v) \in \Delta^n_{\prec,+\infty}$,

$$\text{rank}_M(u,v) = \text{rank}\left(x^{u-v} \cdot \alpha(M)^u\right),$$

where, in the right-hand term, rank indicates the rank of a module. According to Miller [141], this allows to interpret the rank invariant of a persistence module M as a "persistent version" of the *Hilbert function* [128] of $\alpha(M)$.

1.7.3.1 Multigraded Betti numbers

The correspondence α of Theorem 1.7.12 allows to consider, for a persistence module, multigraded Betti numbers. The latter invariants generalize Betti numbers given by the Structure Theorem 1.2.5 to multigraded modules. To begin with, we follow the construction outlined in [127]. Then, we provide our own point of view on the geometrical meaning of multigraded Betti numbers.

In [127], the author defines multigraded betti numbers in terms of the $\text{Tor}_i(-;\mathbb{F})$ function [83, 128] depending on a non-negative integer i and a field $\mathbb{F}$. The functor acts on the corresponding $\mathbb{F}[x_1,\ldots,x_n]$-module $\alpha(M)$ and returns the multigraded $\mathbb{F}[x_1,\ldots,x_n]$-module $\text{Tor}_i(\alpha(M);\mathbb{F})$. Each homogeneous part $\text{Tor}_i(\alpha(M);\mathbb{F})^u$ is a $\mathbb{F}$-vector field and, since the Lefschetz complex filtered by F is finite, it has finite dimension.

1.7.13 Definition (Multigraded Betti numbers)**.** The *multigraded Betti numbers* associated with a persistence module M consist of a collection of functions $\{\beta_{i,\mathbb{F}} : \mathbb{Z}^n \longrightarrow \mathbb{N})\}_{i\in\mathbb{N}}$ defined by

$$\beta_{i,M}(u) := \dim \text{Tor}_i(\alpha(M),\mathbb{F})^u.$$

From now on, we simply write β_i and omit the module M when clear from the context.

The Hilbert's Syzygy Theorem [128] ensures that, since $\alpha(M)$ is a $\mathbb{F}[x_1,\ldots,x_n]$-module, then only $\beta_0,\ldots,\beta_n$ can be non-null functions. The function β_0 counts, in any multigrade, the number of new homology generators. The function β_{i+1} counts the number of new relations among the relations detected by β_i. Hence, the geometrical meaning is not straightforward. Now, we focus on the geometrical meaning of β_i's by addressing:

- the case $n = 0$ corresponding to static case of homology;
- the case $n = 1$ corresponding to the one-parameter persistence case;
- the case $n > 1$ corresponding to multiparameter persistence.

In the case $n = 0$, we can focus on β_0. There are no grades, so β_0 degenerates into a constant function. The corresponding module $\alpha(M)$ is simply an $\mathbb{F}$-vector space of finite dimension. To $\alpha(M)$, the Structure Theorem 1.2.5 applies and provides the standard decomposition with no torsion part:

$$\alpha(M) \cong \bigoplus_{i=1}^{p} \mathbb{F}.$$

The constant function satisfies $\beta_0 = p$ since it counts the number of generators of $\alpha(M)$.

In the case $n = 1$, we can focus on β_0 and β_1. The corresponding module $\alpha(M)$ is an $\mathbb{F}[x]$-module which is finitely generated since finitely presented implies finitely generated. To $\alpha(M)$, the Graded Structure Theorem 1.2.5, which, by taking advantage of $\mathbb{F}[x]$ being a principal ideal domain (from now on called PID) simply adapts the Structure Theorem to the graded case, applies.

1.7.14 Theorem (Graded Structure Theorem)**.** *Any finitely generated graded $\mathbb{F}[x]$-module $\alpha(M)$ admits, up to ordering, a finite number of unique elements $u^i, u^j, v^j \in \mathbb{Z}^n$ such that*

$$\alpha(M) \cong \bigoplus_{i=1}^{p} \mathbb{F}[x](-u^i) \oplus \bigoplus_{j=1}^{m} \mathbb{F}[x](-u^j) \Big/ (x^{v^j - u^j}),$$

where $\mathbb{F}[x](u)$ denotes the graded $\mathbb{F}[x]$-module $\mathbb{F}[x]$ shifted by grade *u defined, over each homogeneous part $\mathbb{F}[x](u)^v := \mathbb{F}[x]^{u+v}$.*

In [35], authors prove that pairs $(u^i, +\infty)$ and (u^j, v^j) counted as many times as they appear in the decomposition are precisely the elements of the persistence diagram associated with M (see Definition 1.7.4). This result implies that the persistence diagram is a complete invariant for one-persistence modules. Moreover, we get that:

$$p + m = \sum_{u \in \mathbb{Z}^n} \beta_0(u)$$
$$m = \sum_{u \in \mathbb{Z}^n} \beta_1(u).$$

This means that β_0 is non-null where a new homology class appears along the persistence module. In particular, the function β_0 still counts the number of generators in $\alpha(M)$ and, as we will see, this will hold also in the $n > 1$ case. The function β_1 is non-null where a homology class disappear along the persistence module. This perfect correspondence with components of elements in the persistence diagram makes the case $n = 1$ so peculiar. This correspondence will be broken in the $n > 1$ case. However, as shown in Figure 1.10, we observe that multigrade Betti numbers are not a complete invariant for one-parameter persistence modules. Indeed, they detect ending points for life-spans of homology classes without tracking which are the two ending points related to the same class.

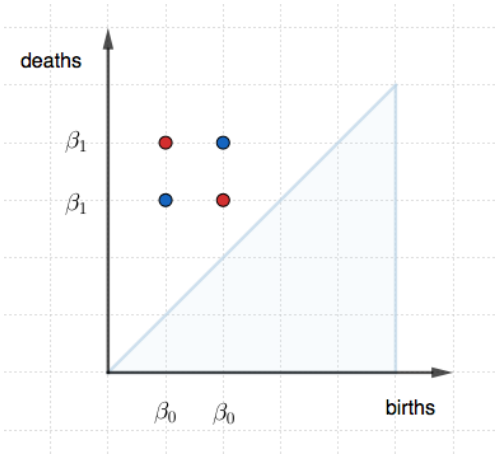


Figure 1.10: two persistence diagrams with the same graded Betti numbers. Different persistence diagrams with the same graded Betti numbers. Red and blue dots belong to different persistence diagrams. The labels β_0, β_1 indicate where graded Betti numbers are non-trivial.

In the case $n > 1$, we can focus on $\beta_0, \ldots, \beta_n$. The corresponding module $\alpha(M)$ is an $\mathbb{F}[x_1, \ldots, x_n]$-module which is finitely presented. Since $\mathbb{F}[x_1, \ldots, x_n]$ is not a PID, the Graded Structure Theorem finds no multigraded counterpart. This is at the heart of non-completeness for multigraded persistence modules invariants. Moreover, non-null values under the function β_1 do not correspond anymore to disappearing homology classes since the notion of relation among generators does not necessarily imply it. Neither non-null values under the function β_2 have that role and the same holds for each function till β_n.

What it is easy to see is that the multigraded Betti numbers and the persistence space or, equivalently, rank invariant are not equivalent. Indeed, the initial differences highlighted in Figure 1.10 for the $n = 1$ case become more and more deep when one miss the link between relations of one order to the other.

2 State of the art

In this chapter, we present the literature of persistence Theory along the pipeline common to all different approaches. The procedure can be simplified as in the figure below.

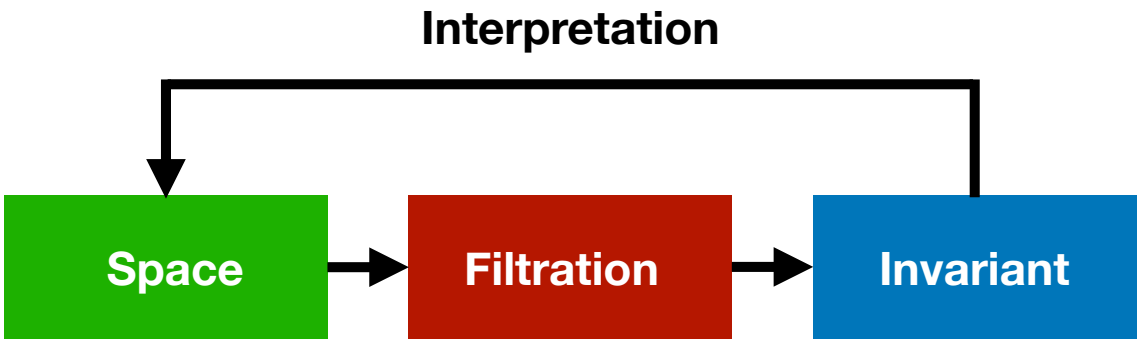


Figure 2.1: persistence pipeline

As Figure 2.1 suggests, we can subdivide the persistence theory pipeline into three main steps that will be reviewed separately treated in this chapter. First, in Section 2.1, we review the main *data structures representing simplicial and cubical complexes* given as the input datum. The data might come with very different encoding. The next step, treated in Section 2.2, is the *filtration*. Computing persistence inherently depends on the way the filtration is built up from data. Thus, filtrations change according to the problem to be addressed and we review the available filtration constructions. The third step consists in the actual *invariant computation* to describe persistence features. This is the step of major interest for the work of this thesis. In this step, we have the major differences between the one-parameter and the multiparameter persistence theories, separately treated in Section 2.3 and Section 2.4, respectively. As a last part, in Section 2.5, we review some literature on the *interpretation* of the retrieved persistence invariants.

2.1 Data structures representing simplicial and cell complexes

The input datum of the persistence pipeline in Figure 2.1 might be of very different nature. More precisely, there might be very different ways to codify it. Here, we consider the most known data structures for cell, simplicial or cubical complexes. In the literature, several topological

data structures encoding a cell complex have been proposed. For sure, the most common one is the Incidence Graph. An *Incidence Graph (IG)* [74] is a topological graph-based data structure representing a cell complex by explicitly encoding all of its cells and their incidence relations with facets. For this reason, even if the domain is discretized through a simplicial complex, the IG can be verbose. In the case of cubical complexes in $\mathbb{R}^d$, the axis-wise distribution of the cubes makes it possible to encode the topology of the complex implicitly. Incidence relations are obtained by index calculations without any overhead.

Simplicial Complexes. The *Simplified Incidence Graph (SIG)* [62] and the *Incidence Simplicial (IS)* [63] data structures specializes the IG to simplicial complexes. The IG, SIG, IS data structures have been all implemented in the public domain software *Mangrove TDS Library* [28, 29]. Mangrove TDS Library is a dimension-independent and extensible framework, targeted to the fast prototyping of topological data structures. However, both these structures behave similarly to the IG when working in high dimensions. Another graph-based data structure is the *Simplex tree* [22, 21] data structure, in which all simplices of the simplicial complex are explicitly stored using trees. The IG, SIG, IS data structures and the Simplex tree share a common trait storing all the simplices in the simplicial complex. In particular, when dealing with data with a huge number of points, or when working in a high dimensional space, a data structure storing all the simplices is unfeasible to use in practice. One could consider data structures encoding only a subset of the simplices. Data structures encoding only top simplices and vertices are, not only, particularly effective in two and three dimensions, but also are the best candidates for extensions to higher dimensions. An example in this direction is represented by the dimension-independent adjacency-based data structure for encoding arbitrary simplicial complexes called *Generalized Indexed data structure with Adjacencies (IA*)* [30].

2.2 Filtrations

In this section, we review the main one-filtration constructions presented in the literature. Multiparameter filtrations are obtained by combining multiple one-parameter filtrations. The most general way of obtaining a one-filtered complex $\mathcal{X}$ from a cell complex X is by means of *sublevel sets* with respect to a scalar *filtering function* ϕ defined on X. We detect two general contexts to produce a one-filtration:

- in the *scalar field* setting, studied in Size Theory [89], one needs to consider smooth spaces together with continuous functions defined on them quantifying some properties one wants to compare. The sublevel sets with respect to the scalar field provide the filtration. The purpose is that of describing the shape through its scalar field in view of applications to shape comparison. The scalar field is described by its persistence features along sublevel sets;

- in the *point cloud* setting, reviewed in [31, 95], one starts from point cloud data; constructs a simplicial complex over it; and filters it according to many possible different criteria. The point cloud is thought of as a sampling over a smooth ideal shape whose homology features should be inferred. Alternatively, the point cloud might represent any geometrical realization of data and the inferred persistence of the filtration might provide topological features as a preprocessing to any statistical data analysis.

Scalar field filtrations. Each smooth topological space is thought together with a continuous *measuring function*. In this context, the cell complex X is thought of as a discrete version of an ideal smooth shape. Thus, at the discrete level, the filtering function is given only on vertexes and a simplex is said to belong to a certain filtration step X^u only if all of its vertexes do. Alternatively, one can consider a piecewise linear function (*PL-function*), obtained by linear interpolation of a function defined on vertexes.
Size Theory is first extended to multiparameters in [90]. The issues for the PL-setting in the multiparameter case are treated in [42].

Point cloud data filtrations. The most common theoretical approach consists in taking the nerve of the covering space given by all radius ϵ neighborhoods of the points in the cloud. This gives in turn a simplicial complex. Then, the filtration is obtained as the parameter ϵ varies. This is known as the *Čech-complex* [108]. In practical situations, other approaches are preferable. One of them is the Vietoris-Rips complex (*VR-complex*) [198] which is simpler in terms of persistence features created. A drawback for VR-complexes is the dimension of the simplicial complex created which exceeds the embedding space dimension. Among the constructions non-exceeding the embedding dimension, α-shapes [81] provide a filtration based on the Delaunay triangulation. Filtrations of *witness complexes*, introduced in [64], allow to reduce the number of vertexes from where to construct the nested sequence of simplicial complexes. Among the optimized constructions, we report *sparse filtrations* [41], recently improved, by Sheedy for α-shapes [170] by means of an output-sensitive algorithm, and by Har-Peled and Mendel in [106] by means of a *hierarchical clustering* technique.

2.3 Computing one-persistent homology

In this section, we review the existing methods for computing one-persistent homology. First, we review the general matrix reduction strategy. Then, we see how differently this strategy can be optimized.

In the one-parameter case, computing persistence, with coefficients in some field, means obtaining any of the equivalent persistence invariants described in Section 1.7 to track the life a homology

feature along the filtration. The computation of the homological information has evolved over the years. At the beginning, persistent homology has been computed by extending techniques developed for retrieving homology. The classical approach to homology computation is based on the *Smith Normal Form (SNF) reduction* [149, 1] applied to the *boundary matrix*, i.e., the matrix representing the boundary maps ∂_k, and is super-cubical in the size of the complex. Optimizations of the SNF algorithm are discussed in [96, 120, 181]. By restricting to field coefficients and ordering the columns in the boundary matrix according to the associated filtration grades before applying the SNF reduction, the persistence diagram can be retrieved. Nowadays, the spread of persistent homology has led to a reversed scenario in which homology is computed through algorithms aimed at the retrieval of persistent homology and, then, specialized to the case of a trivial filtration. An equivalent way of representing the SNF with coefficients in a field is by means of the *Echelon form*. As an advantage, the *standard algorithm* [34, 75], used to get the Echelon form of the boundary matrix, keeps the column order fixed along the reduction. This directly allows to read off the persistence pairs of each homology feature along the filtration. The standard algorithm generalizes to any complex dimension the incremental method for homology proposed in [80, 65].

In the boundary matrix of ∂_k, each column corresponds to a k-cell and each row to a $(k-1)$-cell. Each k-cell comes associated with a filtration grade indicating the first step where the k-cell appears in the filtration. Columns are ordered by increasing values of the associated grades. Non-null column entries indicate the $(k-1)$-facets of the corresponding k-cell. To obtain the persistent pairs in degree k, the standard algorithm performs a left-to-right column reduction over ∂_k and ∂_{k+1}. Whenever two columns are found to have the same non-null entry with highest row index, that row index called *lowest-one*, those two columns are successively summed up till the *reduced matrix* is obtained. Non-null columns in the reduced matrix represents a *persistence pair* formed by the column cell and the cell corresponding to the lowest-one. The *persistence* of the pair is obtained as the difference between the two filtration grades of the cells in the persistence pair. Grades of null-columns in ∂_k represents the birth of a homology class. Such a new homology class can either last all along the filtration or be paired as a row in ∂_{k+1}. The core idea introduced by the incremental method in [80, 65] and exploited in the standard algorithm is that, if cells are added one-by-one along a filtration, then each k-cell either creates a new class in k^{th}-homology (*positive cell*) or destroy an old class in $(k-1)^{\text{th}}$-homology (*negative cell*). Under this interpretation, in the reduced matrix, null-columns represent positive cells and non-null columns represent negative cells. In practical cases, the standard algorithm has a linear complexity in the number of the cells of the complex. Although the methods based on a matrix reduction are theoretically valid in any dimension, their worst-case complexity is super-cubical in the number of cells in the whole complex. This has led to develop other techniques to compute persistent homology. A survey can be found in [75].

We subdivide the methods for retrieving persistence invariants in the one-parameter case into two classes. In Section 2.3.1, we review *integrated* techniques to perform persistence computations. In Section 2.3.2, we review *preprocessings* methods which aim at reducing the size of the input

datum before applying persistence computations. The tools available for computing one-persistent homology are reviewed in Section 2.3.3.

2.3.1 Integrated one-parameter techniques

Practical enhancements in computing one-parameter persistence can be performed in different ways. We subdivide the integrated methods into:

- *direct optimizations* [53, 177, 32, 143, 54] acting on the way the boundary matrix is reduced;
- *distributed optimizations* [75, 135, 11] of the boundary matrix reduction;
- *annotation-based techniques* [66, 21, 26] encoding in a smart way the homology classes along the filtration as an attribute to cells.

Direct optimizations act directly on the standard algorithm by exploiting different aspects.
A first method is the *twist* algorithm [53]. The boundary matrix is reduced from right-to-left. This way, whenever a persistence pair is found, the column with the same index as the row in the persistence pair is set to zero. This reduces the number of column operations to be performed. The twist algorithm outperforms the standard one over filtrations with the number of homology classes higher at the beginning of the filtration and lower towards the end.
The *Row* algorithm [177] (also called dual algorithm) exploits cohomology duality to act on the rows instead of the columns of the boundary matrix. The time complexity is theoretically equivalent to the standard algorithm. However, the row algorithm allows to forget rows once they are paired, thus optimizing the storage cost of the procedure. As a disadvantage explicited by the authors, in most data structures for encoding cell complexes, the boundary retrieval is more efficient than the coboundary retrieval and, explicitly representing a row in the boundary matrix means explicitly retrieving the coboundary of each cell.
In [32] authors introduce a sequential algorithm based on a generalization of persistence to non-necessarily increasing sequences of cell complexes called *zigzag persistence*. Zigzag persistence is exploited in [143] to take advantage of the sparsity of the boundary matrix. The algorithm is a hierarchical variation of the SNF reduction of the boundary matrix that can be obtained in matrix multiplication time, that is $O(n^{2.376})$ in the number of cells. At the moment, this is the lowest worst-case time cost for the boundary matrix reduction.
An output-sensitive algorithm is proposed in [54] where the left-to-right reduction of columns is replaced by successive computations of submatrix ranks. The submatrix ranks are used to describe the persistence diagram through the Formula 1.8. The submatrices are selected according to a threshold parameter δ. Homology classes with persistence lower than δ are not computed. Time complexity depends on the number of persistence pairs found. The worst-case time complexity

depends on the procedure implemented for the submatrix rank computations. Practical advantages are remarkable when a short-living persistence features are not of interest.

Distributed Optimizations take advantage of a divide-and-conquer approach.
Persistent homology is locally computed and then merged through spectral sequences in [75, 135]. The algorithm applies to the *complete boundary matrix* simultaneously encoding all boundary maps ∂_k for all integers k. The complete boundary matrix is subdivided into blocks D_u^v given by intersecting rows with filtration grade equal to u and columns with filtration grade equal to v. Each block is reduced via the standard algorithm. Different blocks are reduced from the diagonal outwards, that is to say, from short to long-living persistence features.
In [11, 12] a similar partition into chunks is exploited to combine the distributed approach to the advantages of the twist algorithm and the elimination of unnecessary unpaired cells inspired by Forman's discrete Morse Theory [87] before merging the blocks contributions.

Annotation-based Techniques have emerged recently in [66, 21] for persistence computations in the simplicial case. Annotations [26] are vectors associated to each simplex to compactly encoding the homological class they belong to. They are often combined with the Simplex Tree data structure [22] instead of using the Incidence Graph as most of the other libraries.

2.3.2 Preprocessing in one-persistent homology

All preprocessing methods in this section share the common task of reducing, while preserving persistence, the input filtration in the number of cells and, possibly, in the number of filtration grades. All the methods adapt similar preprocessings coming from homology computations to the persistence case by imposing compatibility with the filtration structure of the input datum. We subdivide the preprocessing methods into:

- *discrete Morse-based preprocessings* [123, 176, 174, 162, 99, 86, 193, 92];
- *(co)reduction-based preprocessings* [146, 148, 70];
- *acyclic space-based preprocessings* [147, 71].

Discrete Morse-based preprocessings aim at retrieving a discrete gradient compatible with a given filtering function. From [144], implying Theorem 1.5.8, we know that this suffices to obtain a Morse complex with, in general, fewer cells and grades than the original filtration. In general, the discrete gradient is locally found through a divide-and-conquer strategy. We consider methods applying to filtrations whose function attributes are extended from vertexes to higher dimensional cells.
The algorithm described in [123] is the first one to introduce a divide-and-conquer approach which applies to simplicial filtered complexes in any dimension. The main drawback of this

algorithm is introducing many spurious critical simplices with respect to the ones strictly in need for describing the homology changes along the filtration. This drawback grows with the dimension of the simplicial complex.
Two approaches have been defined in [176, 174] for 2D and 3D images respectively. Focusing on a parallel implementation, they provide a substantial speedup in computing the discrete gradient still creating spurious critical simplices. In [162] Robins, Wood and Sheppard introduce a dimension-agnostic algorithm, focused on regular grids, that processes the lower star of each vertex independently. From now on, this algorithm will be called ProcessLowerStars and, since the algorithm presented in Chapter 4 generalizes ProcessLowerStars to the multiparameter case, the algorithm will be discussed in more details in Section 3.2. An efficient implementation of [162] is discussed in [99] while, for simplicial complexes, the same algorithm as been extended to triangle [86] and tetrahedral meshes [193]. The first dimension independent implementation for simplicial complexes of a slightly modified algorithm inspired by [162] is presented in [92].

Reductions and coreductions-based preprocessings are introduced in [146] and they aim at removing pairs of incident cells while preserving homology features. Given a Lefschetz filtered complex $\mathcal{L}$, a *reduction pair* is a pair (a,b) of cells in L such that the coboundary $\mathrm{cb}_L(a) = \{b\}$. a *coreduction pair* is a pair (a,b) of cells in L such that the coboundary $\mathrm{bd}_L(b) = \{a\}$. Under the further constraints of allowing only for pairs of cells in the same level set, the removal of a (co)reduction pair leads to a smaller filtered Lefschetz complex with the persistence preserved [144].
For simplicial complexes, the algorithm is adapted to persistence in [148]. Then, it is extended to general CW-complexes in [70].

Acyclic spaces-based preprocessings aim at detecting a subspace A of the Lefschetz filtered complex $\mathcal{L}$ with the same homology of a single vertex. By excision property of homology, the homology of L is the same as the relative homology of the pair (L,A).
The first algorithm exploiting acyclic subspaces is first introduced in [147]. An acyclic space is found from a seed cell initializing A by successively adding cells while the A keeps being acyclic. The size of A depends on the process ordering of cells.
Acyclic spaces are adapted to the persistence case in [71], where the space A is required to satisfy $A^u := A \cap L^u$ being acyclic for any filtration grade u.

2.3.3 Tools for one-persistent homology

In the recent years, many tools for computing persistence have been developed and distributed. We can classify these software tools as follows:

- tools based on a matrix reduction [184, 190, 145, 13, 160, 9];
- tools based on annotations [138, 67];

- tools exploiting preprocessings [118, 153].

Tools based on matrix-reduction directly work with the boundary matrices of a cell complex and they usually implement the standard algorithm or one of its optimizations. The *javaPlex* library [184] for persistence of simplicial filtrations is based on the first, no longer maintained, open-source tool *jPlex*. The library implements standard, dual and zigzag algorithms. The *GAP persistence* package [190] extends the GAP packages to persistence computations of point cloud data and implements the standard algorithm for simplicial filtraions. The *pHom* library [183] is an R library acting on point cloud data that implements the standard and the dual algorithms. The *Dionysus* C++ based library [145] implements the standard algorithm for simplicial filtrations. The *PHAT* C++ library [10] is the first one to support a parallel implementation of the reduction algorithm for persistence. The toolbox accepts in input directly boundary matrices, thus allowing for computations over general cell complexes, and implements several algorithms: the standard algorithm for persistence [80] and four optimizations, namely the twist algorithm [53], the chunk algorithm [11], the spectral sequence algorithm [75] and the row algorithm [177]. It also supports a Python package version available from the PyPi repository at https://pypi.python.org/pypi/phat. The spin-off version of the tool *DIPHA* [160] supports a distributed implementation. The *Ripser* C++ package [9] implements the twist and dual algorithm in a optimized way for VR-coplexes.

Tools based on annotations often combine the Simplex Tree data structure [22] instead of using the Incidence Graph as most of other libraries. The *Gudhi* C++ library [138] implements the twist algorithm and the multifield algorithm for simultaneous computation of persistence over several coefficient fields [23]. Gudhi supports a Python module available at http://gudhi.gforge.inria.fr/python/latest/index.html. The *SimpPers* [67] library implements the simplicial maps algorithm [66].

Tools based on preprocessings are motivated by the constant growth of the size of the data. *Perseus* [153] is a very flexible C++ library which handles general simplicial, cubical filtrations. The tool reduces the size of the input complex via discrete Morse Theory and then retrieves persistent homology by exploiting the standard algorithm. The *RedHom* C++ library [118] is based on reduction and coreduction operators for the simplification step and it uses PHAT for the persistent homology computation.

2.3.4 Discussion

The first thing to be noticed is the rich variety of techniques and optimizations proposed in the recent years. This holds both from the algorithms and from the tools points of view. Nowadays, challenges in persistence are more related to the next step of the pipeline, namely the feasibility of statistics methods over persistence results.

From the computational point of view, challenges are, on the one hand, directly inherited from

the static case of simple homology, especially regarding the time efficiency, and, on the other hand, they are related to the representation of the multiple spaces within a filtration rather than a single space necessary in simple homology. In the former case, techniques and optimizations have managed to make persistence feasible over real-sized data with the same efficiency as for simple homology. In the latter case, the adopted strategies for dealing with the storage costs are the preprocessing strategies and the parallel and distributed approaches.
In a first phase, preprocessing optimizations had led the way in time and storage costs efficiency. In the work of this thesis, our main contribution consists in proposing a preprocessing optimization which generalizes those preprocessings to the multipersistence case. Nowadays, software applying different techniques are compared in [155] and performances vary according to the kind of filtration involved. For the sake of the work of this thesis, we remark that practical tests have shown that, in the case of one-parameter persistence, libraries implementing integrated optimizations outperform those implementing any preprocessing. The measured time and storage cost difference in performances is remarkable. We believe this is in part due to the fact that, where possible, parallel and distributed techniques are more efficient than preprocessing ones. Moreover, best performing tools, such as Ripser or DIPHA integrate a parallel or distributed implementation to the reduction power of the Morse complex directly on the boundary matrix process. For instance, the clear and compress strategy exploited in the chunk divide-and-conquer approach [11] is inherently connected to discrete Morse Theory and both the libraries implement it at the level of matrix reduction, directly. This considerations may lead to think that preprocessings are not the right way of addressing computational issues in the multipersistence case. However, in the next section we present the related works in multipersistence computations and, within that context, we discuss the advantages of preprocessing techniques.

From the theoretical point of view, we focus on existing Morse-based preprocessing for the one-parameter case. It is remarkable that, for 3D grey-scale images (and, thus for 3-dimensional cubical complexes) the algorithm proposed in [162] is able to find a minimal number of critical cells. However, this minimality property has no formal definition as a property for the discrete gradient. In Section 5.1, we formalize the corresponding property for a discrete gradient. This will allow us to express the optimality of [162] into more general terms including also the multiparameter case, and, finally, to prove that the algorithm of Chapter 4 we have proposed is optimal in the so-defined sense.

2.4 Computing multipersistent homology

In this section, we review the existing methods for computing multiparameter persistence. Unlike the one-parameter case, computing multiparameter persistent homology may indicate the retrieval of different invariants. Indeed, as shown by Carlsson and Zomorodian in [35] and reviewed in Section 1.7.2, the algebraic formulation of the persistence module theoretically prevents a complete descriptor to be discrete or independent from the coefficient field chosen for homology.

This theoretical limitation opens a wide research area in finding multipersistence representations in a suitable way for computations. So, differently for what we have done in Section 2.3, in Section 2.4.1, we briefly report the state of the art for the existing multipersistence invariants. After that, we subdivide the methods for multipersistence computations into two classes. In Section 2.4.2, we review the methods for *computing representations of multipersistence*. In Section 2.4.3, we review the *preprocessings* methods which reduce the input filtration size and preserve multipersistence features. In Section 2.4.4, we discuss the provided overview in relation to the contribution of this thesis.

2.4.1 Invariants in multipersistent homology

In this section, we classify the state-of-the-art invariants available for summarizing multipersistence. Indeed, as opposed to one-parameter persistence, representing multipersistence features has still to find the best trade-off between amount of captured information and computability. This motivates the seek for, not simply new computational methods, but also new invariants for multipersistence. Apart from the complete information of the persistence module itself, which has no canonical mathematical representations, there are several non-complete invariants associated to multipersistence. We classify the existing invariants into the following classes:

- slice-compatible invariants [35, 49, 132];
- algebraic invariants [126, 141, 107];
- noise-oriented invariants [169].

Slice-compatible invariants. We classify an invariant as slice-compatible if, restricted to a slice, it corresponds to a one-persistence invariant. Given a multiparameter filtration F with filtration grades in $\mathbb{R}^n$, one can always consider a linear injective map $l : \mathbb{R} \longrightarrow \mathbb{R}^n$.

2.4.1 Definition (Slice)**.** The *slice of F associated to l* is the one-parameter filtration given by

$$F^l := F \circ l.$$

If $u, v \in \mathbb{R}^n$ are in the image of l, we say that l *is through* (u, v).

Let M be equal to $H_k \circ F$, where H_k is the k^{th}-homology functor. The *rank invariant of M*, introduced in [35] falls into this category. Denote by $\Delta^n_{\preceq}$ the set $\{(u, v) \in \mathbb{R}^n \times \mathbb{R}^n \mid u \preceq v\}$. Moreover, denote by $\Delta^n_{\preceq,\infty}$ the set $\Delta^n \cup \{(u, +\infty) \mid u \in \mathbb{R}^n\}$, where $+\infty$ is an additional element greater than all others in $\mathbb{R}^n$. For each $(u, v) \in \Delta^n_{\preceq,\infty}$ consider the corresponding inclusion-induced map

$$\iota_M^{u,v} : M^u \longrightarrow M^v.$$

2.4.2 Definition (Rank invariant)**.** The *rank invariant of M* is the function $\mathrm{rank}_M : \Delta^n_{\leq,\infty} \longrightarrow \mathbb{N}$ defined by

$$\mathrm{rank}_M(u,v) := \mathrm{rank}\, \iota_M^{u,v}.$$

For each pair of filtration grades $u \preceq v$, the rank invariant counts the number of homology classes persistent through $\iota_M^{u,v}$ from grade u to grade v. The rank invariant is slice-compatible in the sense that, for every $(u,v) \in \Delta^n_{\leq,\infty}$, the slice F^l with l through (u,v) defines a unique pair $(a,b) \in \Delta^1_{\leq,\infty}$ such that, if $M^l = M \circ l$

$$\mathrm{rank}_{M^l}(a,b) = \mathrm{rank}_{M)}(u,v) \tag{2.1}$$

Equality (2.1) was proved for sublevel filtrations of continuous functions, first for 0^{th}-homology in [17], then for general homology degrees in [27].
The *persistence space* [49] generalizes the persistence diagram of Definition 1.7.4 to multipersistence. Denote by $\Delta^n_{\prec}$ the set $\{(u,v) \in \mathbb{R}^n \times \mathbb{R}^n \mid u \prec v\}$ with $u \prec v$ if the strict inequalities $u_i < v_i$ hold for all components. Moreover, denote by $\Delta^n_{\prec,\infty}$ the set $\Delta^n \cup \{(u,+\infty) \mid u \in \mathbb{R}^n\}$, where $+\infty$ is an additional element greater than all others in $\mathbb{R}^n$. Each pair (u,v) of grades in $\Delta^n_{\prec,\infty}$ is given a multiplicity defined by

$$\begin{array}{rcl} \mu_M(u,v) & := & \lim_{|\epsilon|\to 0} \big(\mathrm{rank}_M(u+\epsilon, v-\epsilon) - \mathrm{rank}_M(u-\epsilon, v-\epsilon) + \\ & & -\mathrm{rank}_M(u+\epsilon, v+\epsilon) + \mathrm{rank}_M(u-\epsilon, v+\epsilon)\big) \\ \mu_M(u,+\infty) & := & \lim_{|\epsilon|\to 0} \big(\mathrm{rank}_M(u+\epsilon, v) - \mathrm{rank}_M(u-\epsilon, v)\big), \end{array} \tag{2.2}$$

according to v being in $\mathbb{R}^n$ or admitting infinite components and with $\epsilon \in \mathbb{R}^n$.

2.4.3 Definition (Persistence space)**.** The *persistence space* PSpc_M *of* the persistence module M is the multiset defined by the set $\{(u,v) \in \Delta^n_{\prec,\infty} \mid \mu_M(u,v) > 0\}$ whose elements (u,v) are counted with multiplicity $\mu_M(u,v)$.

By the Representation Theorem in [49], the persistence space is theoretically equivalent to the rank invariant. Hence, by Equation (2.1), each point in the persistence space corresponds to a point in a unique slice l of F. Denoted by $\mathrm{Adm}(\mathrm{PSpc}_M)$ the set of linear injective maps l through (u,v) with $(u,v) \in \mathrm{PSpc}_M$, we get that

$$\mathrm{PSpc}_M \cong \bigsqcup_{l \in \mathrm{Adm}(\mathrm{PSpc}_M)} \mathrm{PDgm}_{F^l(X)}.$$

First introduced in [133], the *fibered barcode* generalizes the barcode to multipersistence. We denoted by $\mathrm{Adm}(\Delta^n_{\leq,\infty})$ the set of linear injective maps l through (u,v) with $(u,v) \in \Delta^n_{\leq,\infty}$. To highlight the analogy with the persistence space, we define the barcode in a slightly different way from the one originally proposed in [133].

2.4.4 Definition (Fibered barcode)**.** The *fibered barcode* FBC_M of a persistence module M is defined by

$$\mathrm{FBC}_M := \bigsqcup_{l \in \mathrm{Adm}(\Delta^n_{\leq,\infty})} \mathrm{BC}_{F^l(X)}\,.$$

The fibered barcode is slice-compatible by definition. By Remark 1.7.7, the barcode and the persistence diagram are equivalent over each slice. Observe the different kinds of considered slices for the persistence space and the fibered barcode. In the former case, the multiparameter perspective detects a number of slices which depends on the homology features of the filtration. In the latter case, the number of slices is taken a priori as all possible slices in the filtration grade space with no relation to where homology features are. This subtle aspect is not of much interest when the two invariants are either retrieved in an approximate way or supported by a filtration preprocessing. However, it should be taken into account in the formalism.

Algebraic invariants. We classify in this class all multipersistence invariants linked to the algebraic formulation, as a multigraded $\mathbb{F}[x_1,\ldots,x_n]$-module M of the persistence module under the Correspondence Theorem (Theorem 1.7.12).
In [35], a natural-valued family of functions β_i defined over the the multigrades of M and indexed over the integers is associated to multiperistence for the first time. This family corresponds to the classical algebraic invariant called *multigraded Betti numbers* [83, 128]. By Hilbert's Syzygy Theorem , we have non-trivial functions β_i only when $1 \leqslant i \leqslant n$. Multigraded Betti numbers capture the relationships among generators of M. In geometrical terms, non-null values of $\beta_0(u)$ detect where, in the filtration, new homology classes appear. The multigraded Betti numbers $\beta_1,\ldots,\beta_n$ register the appearance of higher order relations among homology generators. Further insights on the geometrical interpretation of β_i's are investigated in [127] where they are related to spectral sequences and Hypertor functors.
Other classical definitions such as the Hilbert function and the Hilbert polynomial are first associated to multipersistence by Miller in [141]. Moreover, the author defines *QR-codes* generalizing barcodes to the multiparameter case. QR-codes are algebraically determined by the two notions of modules generators and cogenerators. In geometrical terms, a generator captures the idea of a homology class birth. Indeed, it corresponds to β_0. Instead, a cogenerator captures a vanishing homology class from some multigrade on. Unlike one-parameter persistence, *elder morphisms* have to be specified before defining QR-codes. Elder morphisms generalize the elder rule for one-parameter persistence which conventionally determines that elder homology classes survive over younger ones when merging along a filtration. At the moment, there is no algorithm for computing QR codes.
A similar perspective is given by Harrington et al. in [107] where the suggestion is to investigate the stratified structure of the module M obtained by *associated prime ideals*. By tensoring through local primes, one may inspect M and, possibly, distinguishing among different kinds of *components*, i.e., homology features, according to their persistence along several parameter directions

in the filtration: *transient components* are definitely vanishing in any parameter direction, *persistent components* have some never vanishing parameter direction, *fully persistent components* persist along any parameter direction. Again, there is still no computational counterpart for this persistence module representation.

Noise-oriented invariants. In the one-parameter case, the persistence of a homology class is a well-defined concept. Noise is implicitly assumed to correspond to short-living persistence features. Metrics between invariants (barcodes and persistence diagrams) implicitly depend on that convention. Indeed, any two invariants which differ only for short-living features are typically declared close. The *feature counting map* for a one-parameter filtration is a map which, according to an increasing value for persistence t, counts the number of homology classes with persistence higher than t. The feature counting map behavior reveals the relative contribution of long-living homology features with respect to noise (short-living features). By taking advantage of a Category Theory formalism, authors in [169] define the *feature counting invariant* as the invariant associating with each persistence module its own feature counting map depending on a chosen *noise system*. The noise system is defined to provide a generalization of persistence for homology classes in a multiparameter persistence module. As authors explain, in the multiparameter case, there are several possible ways of defining what noise is. A noise system is introduced as a suitable family of functors inducing a metric on a subclass of the persistence module class. Several instances of noise systems are proposed. In particular, the *standard noise* is the noise system determined by the choice of a specific growth direction in the space of filtration grades. Long-living homology classes are the long-lasting ones along that direction. It is possible to compute the feature counting map with respect to standard noise of a multiparameter filtration through the open-source Topcat library [154].

2.4.2 Computing multipersistent descriptors

In the recent years, methods for computing multipersistence, limitedly to simplicial complex filtrations, have been proposed. Proposing new algorithms for multipersistence retrieval is a very active research branch for the computational topology community. Indeed, only some of the methods we present in this overview were available before we started the work of this thesis. As reported in Section 1.7, the invariants or, more in general, the representations available for multipersistence are not equivalent one another. So, we classify the available methods according to the invariant or the representation they provide:

- persistence module retrieval [33, 94];
- slice-compatible invariants retrieval [17, 27, 133];
- other invariants retrieval [94, 33, 133].

Persistence module retrieval. The first algorithm for the persistence module retrieval of a simplicial filtered complex $\mathcal{S}$ of S is proposed in [33], where the three tasks of computing the k-boundaries, k-cycles and their quotients at each grade u are translated into *submodule membership problems* in computational commutative algebra. As drawbacks, the algorithm introduces an artefact dependency on the chosen basis for expressing the chain complex and implies high computational costs in terms of time: $O(|S|^4 n^3)$, where n is the number of independent parameters in the filtration.
The actual persistence module can be retrieved by means of the algorithm proposed in [94]. The algorithm has been implemented in the Topcat library [154] and distributed in public domain. See Section 6.2 for a preliminary evaluation of our method as a preprocessing to the Topcat library performances. The algorithm acts on the filtration $\mathcal{S}$ at chain level rather than at homology level. First, k-cycles and k-boundaries are expressed in terms of the same basis along the filtration at the chain level. Then, the Smith Normal Form reduction is applied at each step u in the filtration leading to a worst time complexity of $O(|S|^3 \bar{u}^n)$, where $\bar{u}$ is the maximum cardinality among the components u_i in the poset of filtration grades, and n the number of dimensions in the filtration.

Slice-compatible invariants retrieval. In the time line, the first approach in this class is the *foliation method* introduced by Biasotti et al. in [17] for the retrieval of the persistence space limitedly to the case of 0^{th}-homology. Then, in [27], the foliation method is extended to higher homology degrees. The foliation method is not properly an algorithm but rather a framework, particularly suited for shape comparison purposes in the context of Size Theory [89] but not strictly limited to it. The foliation method reduces the computations to multiple iterations of one-persistence computations over each possible filtration slice. The cost of the foliation method is the cost of each persistence computation times the number of possible slices. Encoding the persistence space retrieved by the foliation method has the cost of a single persistence diagram for each possible slice in the set of filtration grades. In Section 6.3, we evaluate the impact on our implementation of the foliation method of the preprocessing described in Chapter 4.

Here, the filtration is induced by sublevel sets with respect to a continuous filtering function $\phi : Y \longrightarrow \mathbb{R}^n$ with Y a topological space. Each injective linear map $l : \mathbb{R} \longrightarrow \mathbb{R}^n$ determining the filtration slice $F^l(X)$ is uniquely determined by two parameters

- $m \in \mathbb{R}^n$ such that $\sum_{i=1}^n m_i = 1$ and $m_i > 0$, for all $1 \leqslant i \leqslant n$;
- $b \in \mathbb{R}^n$ such that $\sum_{i=1}^n m_i = 0$.

Indeed, if $\mathbb{R}$ is parametrized by $a \in \mathbb{R}$, then we define $l : \mathbb{R} \longrightarrow \mathbb{R}^n$ by

$$l(a) := a \cdot m + b. \tag{2.3}$$

2.4.5 Proposition. The filtration slice $F^l(Y)$ is the sublevel set filtration induced by the function

$\phi^l : Y \longrightarrow \mathbb{R}$ such that, for each point $x \in Y$,

$$\phi^l(x) := \min_{i=1,\dots,n} m_i \cdot \max_{i=1,\dots,n} \frac{\phi_i(x) - b_i}{m_i}.$$

By Proposition 1 in [17], each pair (u,v) such that $u \prec v$ or (u,∞) with ∞ greater than all other elements in $\mathbb{R}^n$ uniquely determines a linearization l through (u,v). Hence, there is a unique pair $(a,b) \in \mathbb{R}^n \times \mathbb{R}^n$ such that $l(a) = u$ and $l(b) = v$. Hence, Theorem 3 in [17], written equivalently in terms of rank invariant rather than *size functions*, ensures that, if $M = H_k \circ F$ and $M^l = M \circ l$, then

$$\text{rank}_M(u,v) = \text{rank}_{M^l}(a,b). \tag{2.4}$$

Thus, Equation (2.4) ensures that one-parameter persistence can be performed independently over each slice. Moreover, this can be done by means of any available algorithm or tool.
In the context of Size Theory, the main purpose is that of comparing shapes. Existing algorithms implementing the foliation method have not as final goal that of retrieving the entire persistence space. The persistence space is a mean towards the computation of the matching distance between persistence spaces which quantifies the (dis)similarity between shapes. To this purpose, an approximate version of the persistence space is retrieved by the algorithm proposed in [47]. The space of all available filtration slices is uniformly sampled. Authors quantify the density of the sampling which is necessary for determining the matching distance up to a fixed threshold error. The same procedure is specialized and optimized for two-parameter filtrations in [16] in a hierarchical way. This method finds applications for shape comparison in the PHOG library [15] where authors use the approximate persistence space to deal with photometric attributes. The tool handles graph domains with multiple filtering functions. The foliation method is applied and then the persistence diagram of the graph is computed by applying the optimization by Δ^*-reduction introduced in [57].

For the two-parameter case, i.e., for $n = 2$, a complete representation of a multipersistence slice-compatible invariant is the one proposed in [133] for the retrieval of the fibered barcode of $M = H_k \circ F(S)$ with S a simplicial complex. The algorithm is a variation of the foliation method where slices are parametrized to exploit suitable duality properties for $n = 2$. The algorithm is implemented in the Rivet visualization tool [133] which is available online at [197].
Rivet consists of three main steps:

- a preprocessing by multigraded Betti numbers to reduce the number of filtration grades;
- introduction of the *augmented arrangement data structure* for handling a barcode template;
- successive and efficient updatings of barcodes in the template from one slice to another.

In the preprocessing step, Rivet locates λ_i interesting grades along each component in the grade poset via bigraded Betti numbers [127, 83]. This procedure is optimized for the $n = 2$ case and requires time $O(|S|^3\lambda)$, where λ is the product of $\lambda_1\lambda_2$.
Secondly, the support of the bigraded Betti numbers is crucial for Rivet to introduce a specific data structure, called *augmented arrangement* based on the duality line-to-point for lines in the plane connecting points determined by grades in the support of bigraded Betti numbers. The barcode template is obtained in time $O(|S|^3\lambda + (|S| + \log\lambda)\lambda^2)$.
In the final step, the barcode retrieval is successively updated over each entity in the augmented arrangement. The final step is linear in $|S|$ and the implementation is optimized to decide whether to modify the barcode from adjacent entities in the augmented arrangement or to compute it from scratch. Though limitedly to the two-parameter case, this strategy sensibly increases the size of the inputs which is possible to handle and, depending on the kinds of filtration, it has led to computations over simplicial complexes of up to 15M simplices. This is not comparable to one-parameter persistence standards but, up to now, it is way the highest size available for multipersistence.

Other invariants retrieval. The *rank invariant* value over a single pair (u,v) can be easily derived from a persistence module representation by an algorithm discussed in [33]. However, the full rank invariant computation requires the iteration of this simple procedure for all possible grades satisfying $u \preceq v$ in the filtration which multiplies the complexity by approximately $\frac{1}{2}|F|^2$, where $|F|$ is the, typically very large, cardinality of filtration grades. The Topcat library [154] implements the function for the rank invariant computation starting from the persistence module representation obtained through [94]. The implemented function computes the rank invariant in each (u,v) but not the full rank invariant.
The algorithm in [94] is not limited to the persitence module retrieval but it also allows for the retrieval of the *feature counting map*.
The preprocessing performed by Rivet allows to compute the *multigraded Betti numbers*, limitedly to the two-parameter persistence case.

2.4.3 Preprocessing in multipersistent homology

By preprocessings for multipersistence, we mean methods that aim at reducing the size of the input filtration, in terms of filtration grades or number of cells, before applying any possible multipersistence computation.

A filtration-oriented preprocessing in multipersistence. As already described, by detecting multigraded Betti numbers, the algorithm in [132], and implemented by the tool Rivet [197], manages to reduce the number of filtration grades where to apply persistence computation. Thus,

we classify part of the procedure implemented in Rivet as a preprocessing oriented to the reduction of filtration grades.

Morse-based preprocessings in multipersistence reduce both the number of filtration grades and cells in the input pipeline for multipersistence. They share the common trait of implicitly referring to discrete Morse Theory by generalizing similar algorithms from one-parameter persistence discussed in Section 2.3.2. However, we remark that discrete Morse Theory has no multiparamenter counterpart. For filtered graphs X, rather than d-dimensional cell complexes, the algorithm proposed in [48] reduces the size of X without affecting the persistence module. Clearly, the procedure is limited to the 0^{th}-homology study. Among the Morse-inspired methods, the first attempt for extending the concept of discrete gradient to the multiparameter case, can be found in [2] as a multiparameter generalization of [123]. Afterwards, in [4] a first generalization of [162] is proposed. Being of particular interest for the description of our algorithm presented in Chapter 4, we dedicate Section 3.3 to a more detailed description of algorithm [4], from now on called Matching. In Section 4.5, the two algorithms are proved to be equivalent, thus guaranteeing correctness of our algorithm. The main drawback of the algorithm is in the runtime storage cost which prevents it to be applied over large datasets. In Section 6.1, we evaluate the time and space efficiency of the Matching algorithm compared to ours, introduced in Chapter 4.

2.4.4 Discussion

Computations in multiparameter persistent homology suffer from high time and space costs which, at the moment, prevent the application to real-world data. Moreover, integrated optimizations from one-parameter persistent homology have not found yet a counterpart in the multiparameter case, to the best of our knowledge. This motivates the interest in finding new preprocessings for the reduction of the input filtration size both in terms of number of cells and number of relevant filtration grades.

2.5 Interpreting invariants in persistent homology

In this section, we review the last step of the persistence Theory pipeline, that is how to interpret persistence invariants once computed. In Section 2.5.1, we introduce the main (pseudo)metrics for persistence invariants. In Section 2.5.2, we report the main results concerning stability properties of those distances. In Section 2.5.3, we review the methods for applying statistics to persistence invariants.

2.5.1 Distances between invariants

We first address the one-parameter case and then, we move to the multiparameter one.

One-persistence distances. In one-persistent homology case, the persistence diagram (Definition 1.7.4) is the preferable invariant for quantifying distances. Persistence diagrams are multiset of points in the plane whose coordinates represent the birth and the death time of each persistent homology class. Distances among persistence diagrams are given by possible distances among finite sets of points in the plane based on the ∞-norm $\|\cdot\|_\infty$ in $\mathbb{R}^2$.

2.5.1 Definition. Given X and Y finite sets of points in the metric space defined by the real plane with the infinity norm, the p^{th}-*Wasserstein distance* between X and Y corresponds to:

$$W_p(X,Y) := \inf_{\phi} \left(\sum_{x \in X} \|x - \phi(x)\|_\infty{}^p \right)^{\frac{1}{p}},$$

if $p \in [1,\infty)$, else if $p = \infty$, it corresponds to:

$$W_\infty(X,Y) := \inf_{\phi} \left(\sup_{x \in X} \|x - \phi(x)\|_\infty \right).$$

The most commonly used distance among persistence diagrams is W_∞ which is also called *matching distance* in the Size Theory context or *bottleneck distance*in [58, 55].

Another less frequently used distance is the *Hausdorff distance*.

2.5.2 Definition. Given X and Y finite sets of points in the metric space defined by the real plane with the infinity norm, the *Hausdorff distance* between X and Y corresponds to:

$$d_H(X,Y) := \max \left\{ \sup_{x \in X} \inf_{y \in Y} \|x - y\|_\infty, \sup_{y \in Y} \inf_{x \in X} \|x - y\|_\infty \right\}.$$

On the same X and Y, the matching distance is in general greater than or equal to the Hausdorff distance. Among the persistence tools reviewed so far, only Dionysus [145] computes the Wasserstein distance of persistence diagrams. An optimized version of the Wasserstein distance computation is implemented in the library *Hera* [122]. Other distances are computable through the library *TDA Package* [84] based on GUDHI, PHAT and Dyonisus.

Multipersistence distances. In the multiparameter setting, one wants to quantify distances among persistence spaces (Definition 1.7.9). According to what already reviewed in Section 2.4.1 the persistence space is a slice-compatible invariant. The foliation method reviewed in Section 2.4.2 allows to generalize the matching distance to multiple parameters, through the notion of *multidimensional matching distance* [27]. The distance between two persistence spaces is defined in terms of the matching distance over slices.

2.5.3 Definition. Let $\overline{X}, \overline{Y}$ be two persistence spaces. If the line l and m are as in Formula (2.3), we indicate by X_l the persistence diagram of the slice corresponding to the restriction to the line l. The *multidimensional matching distance* between $\overline{X}$ and $\overline{Y}$ is

$$D(\overline{X},\overline{Y}) := \sup_{l} \min_{i=1,\dots,n} m_i \cdot W_\infty(X_l, Y_l).$$

2.5.2 Stability

In order to apply statistics to the invariant computations, stability is the main property to seek. The leading results for the stability of the matching distance are provided, in the context of Size Theory, by D'Amico, Frosini and Landi [58] for continuous functions in the 0^{th}-homology, and then extended to any homology degree in [44]. In the context of homology inference, the same result is proved by Cohen-Steiner, Edelsbrunner and Harer [55] for *tame* functions in any homology degree. For the multiparameter case, with respect to multidimensional matching distance, stability of the *Persistent Betti numbers*, equivalent to the rank invariant, is proved in [44]. The multidimensional matching distance finds application in the retrieval of textured 3D objects by mixing photometric and geometrical features in [15]. With respect to *Hausdorff distance*, the stability of persistence space is proved in [49]. In [161], theoretical stability with respect to Wasserstein distance is proved under suitable hypothesis.

Algebraic stability is introduced by Chazal et al. [52] with respect to a pseudometric on persistence modules. The pseudometric is constructed by means of *interleaving distance* of persistence modules, that is, relating objects at an algebraic level. The purpose is that of allowing for comparisons among, not simply several filtrations of the same space, but also different spaces. A multiparameter formulation of multidimensional matching distance is proved to be stable by Micheal Lesnick in his Phd thesis [132]. The visualization tool *RIVET*, released in public domain by Lesnick and Wrights in 2016 [197], applies this algebraic formulation of distance and compute it.

2.5.3 Towards Statistics

Since any notions of addition or mean is unfeasible for persistence diagrams, some authors propose larger environments to allow for such operations. For instance, *persistence landscapes* are theoretically introduced in [25] with the aim of allowing for such operations for statistical purposes by means of classical results such as a law of large numbers and a central limit theorem. The new environment is exploited in [161], where a multiscale kernel approach is applied to persistence landscapes. Persistence landscapes can be computed through the *Persistence Landscape Toolbox* [69].

2.6 Discussion

Computational and theoretical issues, in our state-of-the-art review, highlight the discrepancy between one-parameter persistence maturity and its multiparameter counterpart. From a computational point of view, the amount of methods available for the multiparameter persistence is not comparable to those for the one-parameter case. Qualitatively, if we compare the persistence pipeline above to the multiparameter analogue, the difficulty in interpreting results is evident as well as the lack of algorithms able to compute descriptors in feasible time. From the theoretical point of view, invariants and representations to be computed are still to be well-assessed in multipersistence and this has led to a wide variety of possibilities to be evaluated. In particular, in the case of preprocessing methods, which is more tightly related to the purposes of this thesis, there is no available notion to declare a preprocessing optimal in reducing the size of the input filtration. We give our contribution to this problem in Chapter 5. Moreover, there is no available tool which is able to do the entire pipeline. We give our contribution to this problem in Chapter 6.

3 Preliminaries on Morse-based preprocessings for persistence

In this chapter, we focus on two algorithms in the state of the art which are necessary preliminaries to the algorithm introduced in Chapter 4. The first algorithm is called ProcessLowerStars and was first introduced in [162]. The second algorithm is called Matching and was first introduced in [4]. They share the common trait of providing with a *discrete gradient compatible with a filtering function* (Definition 1.6.2) and, thus they can be thought of as preprocessings for persistence computations. Another similarity is that both the algorithms construct the output discrete gradient by independently processing certain *lower stars* (Definition 3.1.3). The final discrete gradient in output is simply the disjoint union of the outputs of an auxiliary function called HomotopyExpansion, introduced in [162].

Dissimilarities between the two algorithms are of two kinds: in their generality and in the computational efficiency. From the generality point of view, the algorithm ProcessLowerStars is limited to scalar-valued filtering functions, thus it can be thought of as a preprocessing for one-parameter persistence only. Moreover, authors formally present the algorithm when the filtering function is defined over a *cubical complex* (Definition 1.1.8). Whereas, the algorithm Matching admits vector-valued filtering functions, thus generalizing the lower star partition method to a suitable preprocessing for multipersistence computations. Moreover, authors prove the algorithm formal correctness when the filtering function is defined over a *simplicial complex* (Definition 1.1.1). From the computational efficiency point of view, saying that both algorithms run HomotopyExpansion "independently" over a partition into lower stars assumes different meanings for the two algorithms here presented. We call *local* the algorithm ProcessLowerStars since the algorithm, not simply runs HomotopyExpansion over a partition into lower stars but the entire process can be performed in parallel. We call *global* the algorithm Matching since the algorithm, although running HomotopyExpansion over a partition into lower stars, requires a global queue to select which lower stars where to run the auxiliary function. Hence, it cannot be broken into a parallel process.

The (dis)similarities observed in this chapter allow us to present the algorithm of Chapter 4 as another Morse-based persistence preprocessing by lower stars which is: suitable for multipersistence, local, and valid for both simplicial and cubical complexes. In Section 3.1, we review the auxiliary function HomotopyExpansion able to find a discrete gradient within a single lower star. In Section 3.2, we review the algorithm ProcessLowerStars. In Section 3.3, we review the algorithm Matching.

3.1 Finding a discrete gradient over a single lower star: HomotopyExpansion

In this section, we review an auxiliary function introduced by Robins, Wood and Sheppard in [162] and we call it HomotopyExpansion since, as the authors highlight, it finds discrete vectors by applying successive homotopy expansions. Since HomotopyExpansion is at the base of the three algorithms described in Section 3.2, Section 3.3 and Chapter 4, we elaborate the formalism introduced by the authors in order to get a common description.

The section is structured as follows. In Section 3.1.1, we make the input assumptions explicit and show some useful properties of the output and the input elements. In Section 3.1.2, we describe the auxiliary function. In Section 3.1.3, we remark some complexity aspects related to the auxiliary function which will be useful when considering next algorithms.

3.1.1 Input and output formalization

In this section, we make the assumptions on the input for the auxiliary function HomotopyExpansion explicit. By presenting our assumptions on the input, we show some useful properties holding. At the end of the section, we state the properties of the output retrieved. As already stated, the assumptions slightly differ from the ones proposed by the authors in [162]. To begin with, in the referred work, the auxiliary function receives in input a gray scale digital image, formalized as a cubical complex. Here, we present HomotopyExpansion acting over a *regular cell complex X with the intersection property* in order to capture both the cubical and the simplicial complex cases. For convenience, we report Definition 1.1.12 adapted to regular cell complexes below.

3.1.1 Definition. A *regular cell complex X with the intersection property* is a regular cell complex such that, for every two cells in X, the corresponding topological closures have either empty intersection or their intersection is a unique cell in X.

As a further difference, in [162], the auxiliary function receives as input the lower star of a vertex with respect to a scalar-valued filtering function. Here, we present HomotopyExpansion receiving in input a lower star of a general cell with respect to a vector-valued filtering function. The filtering function is obtained by a vector-valued function defined over the vertexes of the complex through the following construction.

3.1.2 Definition (Vertex-induced filtering function)**.** Given a function $f : X_0 \longrightarrow \mathbb{R}^n$ over the vertexes of a cell complex X, the *vertex-induced filtering function from* $\tilde{f}$ is the function $\tilde{f} : X \longrightarrow \mathbb{R}^n$ defined by setting, for each cell $a \in X$, and each $1 \leqslant i \leqslant n$,

$$\tilde{f}_i(a) := \max_{\substack{v \in X_0 \\ v \ll a}} f_i(v).$$

For each cell a, the filtering function $\tilde{f}$ defines the following subset of the star of a.

3.1.3 Definition (Lower star)**.** The *lower star of a with respect to f* is the set

$$\mathrm{Low}_f(a) := \{t \in \mathrm{Star}_X(a) \mid \tilde{f}(b) \preceq \tilde{f}(a)\}.$$

In Figure 3.1, we show an example of a lower star with respect to e one-parameter filtration.

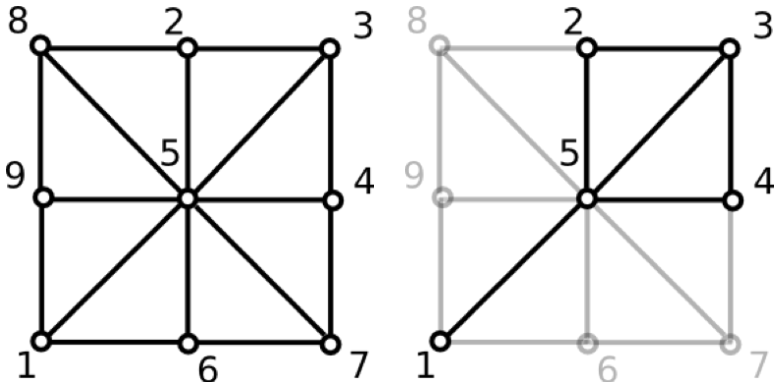


Figure 3.1: a triangulation (a) of the plane filtered by a single parameter over vertexes. In (b), the lower star of the vertex 5.

As a first property of lower stars, we have the following one.

3.1.4 Proposition. Lower stars of cells in X form a covering for X.

Proof. It follows from each cell belonging, at least, to its own lower star. □

Lower stars with respect to functions extended from vertexes satisfy the following important property with respect to level sets under $\tilde{f}$.

3.1.5 Proposition. For every $a \in X$, there exists one, and only one, level set $\mathrm{Lset} = \tilde{f}^{-1}(\tilde{f}(a))$ such that.

$$\mathrm{Low}_f(a) \subseteq \mathrm{Lset}\,.$$

Proof. By construction of the extension $\tilde{f}$ from the function f, $\tilde{f}(b) \succeq \tilde{f}(a)$, for any $b \in \mathrm{Low}_f(a)$. By assumption, if $b \in \mathrm{Low}_f(a)$, then $\tilde{f}(b) \preceq \tilde{f}(a)$ and this concludes the proof. □

As a final difference, in [162], the auxiliary function depends on a special ordering over cells in the lower star induced by a suitable lexicographic order. Here, we present HomotopyExpansion receiving in input an *indexing* i over $\mathrm{Low}_f(a)$ compatible with the coface partial order relation $\ll$ in X. This more general form is compatible with our usage of the auxiliary function in Section 3.3.

3.1.6 Definition (Indexing)**.** Any injective function $i : A \to \mathrm{Tot}$ with Tot a totally ordered set is an *indexing over* A. If A admits a partially ordered relation $\trianglelefteq$, then i is *compatible with* $\trianglelefteq$ if, for all $a, b \in A$,

$$a \trianglelefteq b \qquad \Rightarrow \qquad i(a) \leqslant i(b)$$

To summarize, in this work HomotopyExpansion receives in input:

- X : a regular cell complex with intersection property;
- $\mathrm{Low}_f(a)$: the lower star of a cell $a \in X$ with respect to the vertex-induced filtering function $\tilde{f}$;
- i : an indexing over $\mathrm{Low}_f(a)$ compatible with the coface partial order $\ll$ on X.

As regards the computed output properties, we refer the reader to Definition 1.4.5 for the notion of discrete gradient, and to Definition 1.4.7 for the notion of Morse set relative to a discrete gradient.

$\mathrm{HomotopyExpansion}(X, \mathrm{Low}_f(a), i)$ returns:

- V : a discrete gradient over X with cells in $\mathrm{Low}_f(a)$;
- M : the Morse set relative to V, i.e., $M = M(V)$ restricted to $\mathrm{Low}_f(a)$.

3.1.2 Description

In this section, we describe the auxiliary function HomotopyExpansion. We refer to the pseudocode in Algorithm 1. HomotopyExpansion exploits the following auxiliary data structures:

- declared : a length $|\mathrm{Low}_f(a)|$ array of Booleans;
- Ord0 and Ord1 : two ordered lists[1] with ordering given by increasing values of elements under the indexing i;

[1] Ord0 and Ord1 are equivalently described as priority queues in [162] but we present them as ordered lists since, actually cells entering before others have no priority. Only the indexing imposes the priority as it happens in an ordered list. Moreover, the algorithm requires to perform element removals which are not standard operations over a priority queue.

and the following auxiliary functions:

- num_undeclared_facets(b, Low$_f(a)$) which counts the number of facets c of b in Low$_f(a)$ such that declared$[i(c)]$ =false ;
- unpaired_facet(b, Low$_f(a)$) returns the only facet c of b such that undeclared$[i(c)]$ = false;
- add_cofacets(c, Low$_f(a)$,Ord1) adds to Ord1 all cofacets d in Low$_f(a)$ of c such that num_undeclared_facets(c, Low$_f(a)$)$= 1$.

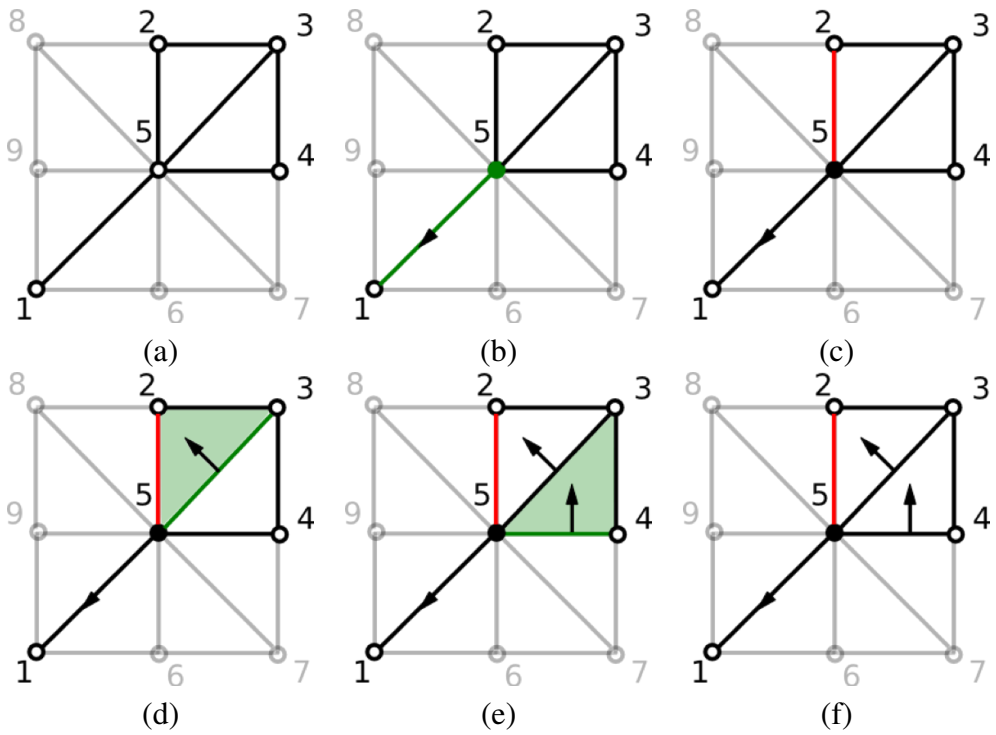


Figure 3.2: step-by-step action of HomotopyExpansion over a the lower star of vertex 5 with respect to a scalar function.

We describe HomotopyExpansion by following the example in Figure 3.2.

The array declared keeps track of the cells in Low$_f(a)$ already classified either as part of the list M or as part of a pair in the list V. At the beginning, declared is initialized with all entries set to false. In Figure 3.2(a), we show the lower star of vertex 5 with respect to the one-filtration indicated by filtering values over vertexes. The darker cells are part of the lower star of 5. The two ordered lists Ord0 and Ord1 assign priorities to the multiple possible pairing of cells.

In the first for loop at line a in Algorithm 1, $\mathrm{Ord0}$ and $\mathrm{Ord1}$ are initialized. The function $\mathrm{num\_undeclared\_facets}(b, \mathrm{Low}_f(a))$ counts the number of facets c of b in $\mathrm{Low}_f(a)$ with $\mathrm{declared}[i(c)] = \text{false}$. A cell enters $\mathrm{Ord0}$ when all of its facets in $\mathrm{Low}_f(a)$ are already declared. Hence, in this for loop, only a enters $\mathrm{Ord0}$. Cells in $\mathrm{Ord0}$ are suitable either to form a pair in V with a higher-dimensional cell or to be added to M. In the example of Figure 3.2 vertex 5 enters $\mathrm{Ord0}$ since it has no facets in its lower star. A cell b enters $\mathrm{Ord1}$ when b has only one facet c, pointed by $\mathrm{unpaired\_facet}(b)$, still to be declared. In this for loop, all cofacets of a in $\mathrm{Low}_f(a)$ are added to $\mathrm{Ord1}$. The cells in $\mathrm{Ord1}$ are suitable to form a pair in V with a lower-dimensional cell. In the example of Figure 3.2 the edges $[5,1],[5,2],[5,3],[5,4]$ enters $\mathrm{Ord1}$ since they have only vertex 5 as their cofacets in the lower star of 5. The indexing i ensures that, in the two ordered lists, faces take priority over cofaces.

In the two nested while loops at lines 12 and 13, the first cell b in $\mathrm{Ord1}$ is set to form a pair in V with the cell c pointed by $\mathrm{unpaired\_facet}(b)$. Both c and b are updated as true in $\mathrm{declared}$ and all of their cofacets suitable to enter $\mathrm{Ord1}$ are inserted in such ordered list by function $\mathrm{add\_cofacets}$. In the example of Figure 3.2(b), vertex 5 is paired with the edge $[5,1]$ having maximum priority in $\mathrm{Ord1}$. The inner while loop ends when there is no cell to be extracted from $\mathrm{Ord1}$. At that point, the first cell in $\mathrm{Ord0}$ is declared critical and all of its cofacets suitable of entering $\mathrm{Ord1}$ are inserted in such ordered list. In the example of Figure 3.2(c), we see the red edge $[5,2]$ declared critical since the list $\mathrm{Ord1}$ is empty and, thus, no pairing is available for the cells $[5,2],[5,3],[5,4]$, currently in $\mathrm{Ord0}$. This allows to update both $\mathrm{Ord1}$, with the triangle $[5,3,2]$, paired with $[5,3]$ in Figure 3.2(d). In the example, the procedure terminates with no additional critical cells declared. In Proposition 4 in [162], authors show that the outer while loop breaks if and only if all cells have been declared.

Algorithm 1 HomotopyExpansion($X, \mathrm{Low}_f(a), i$)

Input: X cell complex # regular, with intersection property
Input: $\mathrm{Low}_f(a)$ list of cells in X # $\mathrm{Low}_f(a)$ is the lower star of a in X with respect to $\tilde{f}$
Input: i indexing over the cells of $\mathrm{Low}_f(a)$ # i compatible with incidence relations in X restricted to $\mathrm{Low}_f(a)$
Output: V list of pairs of cells # discrete gradient over $\mathrm{Low}_f(a)$
Output: M list of cells # Morse set relative to V

set V, M to be empty lists
set Ord0,Ord1 to be empty ordered lists # values of nodes in Ord1,Ord0 are ordered by increasing values under i
set declared to be an array of length $|\mathrm{Low}_f(a)|$ with Boolean values equal to false # in declared entries are ordered according to increasing values of i
for all b in $\mathrm{Low}_f(a)$ **do**
    # $\mathrm{num\_undeclared\_facets}(b, \mathrm{Low}_f(a))$ which counts the number of facets c of b in $\mathrm{Low}_f(a)$ such that declared$[i(c)]$ =false
    if $\mathrm{num\_undeclared\_facets}(b, \mathrm{Low}_f(a)) = 0$ **then**
        insert b into Ord0
    else if $\mathrm{num\_undeclared\_facets}(b, \mathrm{Low}_f(a)) = 1$ **then**
        insert b into Ord1
    end if
end for
while Ord1$\neq \varnothing$ or Ord0$\neq \varnothing$ **do**
    while Ord1$\neq \varnothing$ **do**
        $b \leftarrow$ the first element in Ord1
        remove b from Ord1
        if $\mathrm{num\_undeclared\_facets}(b, \mathrm{Low}_f(a)) = 0$ **then**
            insert b into Ord1
        else
            $c \leftarrow \mathrm{unpaired\_facet}(b, \mathrm{Low}_f(a))$ # $\mathrm{unpaired\_facet}(b, \mathrm{Low}_f(a))$ returns the only facet c of b such that undeclared$[i(c)]$ = false
            add (c, b) to V
            declared$[i(c)] \leftarrow$ true
            declared$[i(b)] \leftarrow$ true
            remove c from Ord0
            $\mathrm{add\_cofacets}(c, \mathrm{Low}_f(a), \mathrm{Ord1})$ # $\mathrm{add\_cofacets}(c, \mathrm{Low}_f(a), \mathrm{Ord1})$ adds to Ord1 all cofacets d of c in $\mathrm{Low}_f(a)$ such that $\mathrm{num\_undeclared\_facets}(c, \mathrm{Low}_f(a)) = 1$
            $\mathrm{add\_cofacets}(b, \mathrm{Low}_f(a), \mathrm{Ord1})$
        end if
    end while
    if Ord0$\neq \varnothing$ **then**
        $b \leftarrow$ the first element in Ord0
        remove b from Ord0
        append b to M
        declared$[i(b)] \leftarrow$ true
        $\mathrm{add\_cofacets}(b, \mathrm{Low}_f(a), \mathrm{Ord1})$
    end if
end while
return (V, M)

3.1.3 Complexity remarks

In this section, we discuss the complexity of HomotopyExpansion and highlight a difference in time complexity analysis between the simplicial and the cubical case.

We indicate with Star, the star of a in X, as defined in Section 1.1. The cardinality of Star is an upper bound for the cardinality of the input $\mathrm{Low}_f(a)$.

To describe the complexity of the auxiliary function HomotopyExpansion, we assume two facts:

- Star is stored so that visiting the (co)facets of any cell in Star takes linear time in the number of (co)facets[2];
- the ordered lists Ord0, Ord1 are implemented as self-balancing binary search trees, so that inserting, and removing an element from the tree is logarithmic in the length of the tree. The storage cost of each tree is linear in the length of the tree.

The number of operations performed by HomotopyExpansion is obtained by counting the repetitions of the following simple operations:

- visiting the cofacets of $b \in \mathrm{Low}_f(a)$ is linear in the number of cofacets of b in $\mathrm{Low}_f(a)$. All cofacets c of b are visited to update $\mathrm{num\_undeclared\_facets}(c)$, and to possibly activate the function $\mathrm{unpaired\_facet}(c)$. This happens whenever b is classified. Each cell is visited as a cofacet once per each of its facets. Since each cell c has a fixed number of cofacets in $\mathrm{Low}_f(a)$ there are order $O(|\mathrm{Low}_f(a)|)$ cells to visit in $\mathrm{Low}_f(a)$. The time contribution is $O(|\mathrm{Low}_f(a)|)$;
- updating (inserting to and removing from) an ordered list is logarithmic in the length of the list. Each cell in $\mathrm{Low}_f(a)$ enters each list at most once. So, at most $4|\mathrm{Low}_f(a)|$ elements updatings are performed. Since the maximal size of each list is $|\mathrm{Low}_f(a)|$, the contribution is $O(|\mathrm{Low}_f(a)| \cdot \log|\mathrm{Low}_f(a)|)$;
- searching for the first element in an ordered list is logarithmic in the length of the list. Not all cells are selected as the first element in an ordered list, but only the cells found critical and the cells declared in a discrete vector with a lower dimensional cell. So, at most $|\mathrm{Low}_f(a)|$ first element searches are performed. The contribution is $O(|\mathrm{Low}_f(a)| \cdot \log|\mathrm{Low}_f(a)|)$.

The time complexity of HomotopyExpansion is, thus, dominated by the contribution of the ordered list operations and we get

$$O(|\mathrm{Low}_f(a)| \cdot \log|\mathrm{Low}_f(a)|).$$

[2]For instance, the representation of a complex described in Section 6.1 meets this requirement.

The space required by the operations performed does not exceed the space necessary to represent $\text{Low}_f(a)$.

In the following remark, we highlight a property of the auxiliary function treated in this section when compared to functions acting on the entire cell complex X. The remark will be useful for analyzing the complexity of the algorithms in Section 3.2, Section 3.3 and Chapter 4.

3.1.7 Remark. *In the case of X being a cubical complex, each cube a has $|\text{Star}|$ bounded by a constant $const(d)$ depending on the dimension d of the complex. Hence, for low dimensions d* HomotopyExpansion *can be considered a constant operation. On the contrary, if X is a simplicial complex, there is no upper bound for the cardinality of the star of a simplex. Hence,* HomotopyExpansion *cannot be considered a constant operation, independently from the dimension.*

3.2 A local preprocessing algorithm for one-persistent homology: ProcessLowerStars

In this section, we present the algorithm ProcessLowerStars introduced by Robins, Wood and Sheppard in [162]. ProcessLowerStars retrieves a discrete gradient compatible with a scalar-valued filtering function. Hence, it can be thought of as a preprocessing for one-parameter persistence. Details for ProcessLowerStars are provided since the algorithm of Chapter 4 generalizes the same procedure to vector-valued filtering functions. For convenience in comparing the procedure to other algorithms, we do not describe the algorithm in [162] as it is done by the authors. We describe ProcessLowerStars by means of the auxiliary function HomotopyExpansion described in Section 3.1.

The section is structured as follows. In Section 3.2.1, we make the input assumptions explicit and show some useful properties of the output and the input elements. In Section 3.2.2, ProcessLowerStars is described. In Section 3.2.3, we remark some complexity aspects related to the ProcessLowerStars which will be useful in comparison to the algorithm described in Chapter 4.

3.2.1 Input and output formalization

In this section, we make the assumptions on the input for the algorithm ProcessLowerStars explicit. By presenting our assumptions on the input, we show some useful properties by highlighting the local nature of the algorithm. At the end of the section, we state the properties of the output retrieved.

According to the authors, ProcessLowerStars requires in input a cubical complex Q as in Defini-

tion 1.1.8 with no restriction on the embedding dimension. By Remark 1.1.13, a cubical complex is a special case of a regular cell complex with intersection property.

Furthermore, ProcessLowerStars requires in input a scalar-valued function f defined over the vertexes of Q. For the correctness of the procedure f is assumed to be injective. A general scalar-valued filtering function can be made into an injective one, with a controlled difference with respect to the infinity norm, by *simulation of simplicity* [82].

There are no other input requirements. However, in our presentation, we make it explicit a further implicit feature which affects the output retrieved by the algorithm. We remark this aspect in view of a different choice implemented by the algorithm we describe in Section 3.3. In ProcessLowerStars, each k-cell q has its vertexes $[q] := [v_1, \ldots, v_{2^k}]$ represented in decreasing order of values under f, i.e., in such a way that $f(v_1) > \cdots > f(v_{2^k})$. This implicit representation is clearly possible whenever a cell is completely determined by its vertexes which, by Proposition 4.7.13 in [20] holds, more generally, for regular cell complexes with intersection property. A particular class of indexings over the cells (Definition 3.1.6) is the following one.

3.2.1 Definition (Lexicographic indexing)**.** For each cell a in a a regular cell complex with intersection property X, let $[a]$ be the tuple of vertexes of a ordered by decreasing values under f. We denote by $\leqslant_{\text{Lex}}$ the lexicographic order over tuples $[a]$ with $a \in X$. A *lexicographic indexing* over X is an indexing such that, for all cells $a, b \in X$

$$[a] \leqslant_{\text{Lex}} [b] \qquad \Leftrightarrow \qquad i(\leqslant_{\text{Lex}})(a) \leqslant i(\leqslant_{\text{Lex}})(b).$$

To summarize, ProcessLowerStars receives in input:

- Q : a cubical complex;
- f : a real-valued injective function on the vertexes of Q,

and implement:

- $i(\leqslant_{\text{Lex}})$: an implicit lexicographic indexing on Q.

Under the input assumptions for ProcessLowerStars, we focus on two properties of the lower stars of vertexes, defined according to Definition 3.1.3 when the function f defined over the vertexes is extended to higher-dimensional cells by $\tilde{f}$ (Definition 3.1.2). The two of them follow rather clearly and are both used in [162]. The first result makes it explicit the local nature of the algorithm and it ensures that the algorithm can be possibly parallelized. We provide our own proof of this fact.

3.2.2 Proposition. Under the input assumptions for ProcessLowerStars, lower stars of vertexes in Q form a partition for Q.

Proof. It follows from the definition of vertex-induced filtering function. Indeed, in the special case of $\tilde{f}$ being a scalar-valued function, each cell has the same value as one of its vertexes. Moreover, being f injective, no cell belongs to two different vertex lower stars. □

Our proof makes explicit use of the filtering function being a scalar function. Otherwise, vertex lower stars do not cover the entire cubical complex. The second property relate lower stars to level sets. We do not give formal proof of it since the proof of Proposition 3.3.1 can be easily adapted to this case.

3.2.3 Proposition. Under the input assumptions for ProcessLowerStars, for every $v \in Q_0$, the level set $\text{Lset} = \tilde{f}^{-1}(\tilde{f}(v))$ satisfies

$$\text{Low}_f(v) = \text{Lset}\,.$$

As regards the output retrieved by ProcessLowerStars, we refer to Definition 1.4.5 for the notion of discrete gradient, and to Definition 1.4.7 for the notion of Morse set relative to a discrete gradient. According to Definition 3.1.2, let $\tilde{f}$ be the vertex-induced filtering function from f. Then, $\text{ProcessLowerStars}(Q, f)$ returns:

- V : a discrete gradient over Q compatible with $\tilde{f}$;
- M : the Morse set relative to V, i.e., $M = M(V)$.

The output V is guaranteed to be a discrete gradient by Proposition 5 in [162]. The compatibility with function $\tilde{f}$, as defined in Definition 1.5.2, follows directly from the construction since all discrete vectors are found within a single level set.

A discrete gradient compatible with a function $\tilde{f}$ is also compatible with the filtration $\mathcal{Q}^{\tilde{f}}$ induced on Q by $\tilde{f}$. Hence, the retrieved output determines a filtered Morse complex $\mathcal{M}^{\tilde{f}}$ which is a compatible Morse reduction of $\mathcal{Q}^{\tilde{f}}$ in the sense of Theorem 1.5.8. This ensures that ProcessLowerStars is a valid preprocessing for computing one-parameter persistence in any possible way.

For the case of cubical complexes embedded in a 3D space, Theorem 11 in [162] (explicitly reported in this thesis as Theorem 5.1.1) relates the elements in the Morse set M to homology changes along the filtration induced by $\tilde{f}$. The idea is that, up to embedding dimension equal to 3, there are no spurious critical cells detected by ProcessLowerStar. Theorem 5.1.1 is crucial for the purposes of this thesis. Indeed, it inspires Section 5.1 to formalize an optimality property of a discrete gradient compatible with a filtration, and Section 5.2 to generalize the introduced optimality property to the multiparameter setting.

3.2.2 Description

In this section, we describe the algorithm ProcessLowerStars. We refer to the pseudocode in Algorithm 2. ProcessLowerStars runs the following auxiliary functions:

- ComputeLowerStar for extracting the lower star of a vertex;
- HomotopyExpansion for finding discrete vectors and critical cells in a vertex lower star (see Section 3.1).

Algorithm 2 ProcessLowerStars(Q, f)

Input: Q cubical complex
Input: $f : Q_0 \longrightarrow \mathbb{R}$ function # f is injective
Output: V list of cell pairs # V is a discrete gradient compatible with the max-extension of f
Output: M list of cells # M is the Morse set of V
1: V, M are empty lists
2: **for all** v in Q_0 **do** # independently from the order
3: $\text{Low}_f(v) \leftarrow \text{ComputeLowerStar}(v, Q)$ # $\text{Low}_f(v)$ is the list of cells in the lower star of v
4: $(V_{\text{Low}_f(v)}, M_{\text{Low}_f(v)}) \leftarrow \text{HomotopyExpansion}(Q, \text{Low}_f(v), f)$ # classifies all cells in $\text{Low}_f(v)$ as either part of a pair in $V_{\text{Low}_f(v)}$ or a cell in $M_{\text{Low}_f(v)}$
5: append $V_{\text{Low}_f(v)}$ to V
6: append $M_{\text{Low}_f(v)}$ to M
7: **end for**
8: **return** (V, M)

The algorithm processes all vertexes in Q in a for-cycle (lines 2-7 in Algorithm 2). By Proposition 3.2.2, there is no preference in the vertex process and the for-cycle can be possibly broken into a parallel or distributed approach.

For each vertex v, the auxiliary function ComputeLowerStar first extracts $\text{Star}_Q(v)$, i.e., the star of v. Then, it visits $\text{Star}_Q(v)$ to select those cells a satisfying condition $\tilde{f}(a) \leqslant f(v)$. Finally, $\text{ComputeLowerStar}(v, Q)$ returns the list $\text{Low}_f(v)$ of so-selected cells to be given in input to the auxiliary function HomotopyExpansion. According to Section 3.1.1 the latter returns a discrete gradient $V_{\text{Low}_f(v)}$ over $\text{Low}_f(v)$ which updates the output list V, and the corresponding Morse set $M_{\text{Low}_f(v)}$ which updates the output list M.

3.2.3 Complexity remarks

In this section, we discuss the complexity of ProcessLowerStars. First, we analyze each single auxiliary function. Then, we adapt the analysis to the simplicial case. This latter part is of special interest for the complexity analysis of the algorithm reported in Section 3.3 and the one presented in Chapter 4.

To describe the complexity of the algorithm, we focus on the two auxiliary functions: ComputeLowerStar and HomotopyExpansion. We assume the input cubical complex to have dimension d. We indicate with Star the vertex star with maximal cardinality in X.

ComputeLowerStar. In a cubical complex representation, faces and cofaces are implicitly indicated by manipulating cell indexes. Thus, the star of a vertex needs not to be explicitly represented. Visiting the star takes linear time in the cardinality of the star. Comparing two floats is considered to take constant time, so that a single lower star extraction is performed in linear time in the cardinality of the star.
The procedure is repeated for every vertex in the cubical complex. We can overestimate the number of visited cells in the following way:

$$\sum_{v \in Q_0} |\,\mathrm{Star}(v)\,| \leqslant |\,\mathrm{Star}\,||Q_0|. \tag{3.1}$$

Recall that in a cubical complex $|\,\mathrm{Star}\,|$ is bounded by a constant depending on d. Hence, the overall time cost is $O(|Q_0|)$ with no extra space required.

HomotopyExpansion We observe that, for cubical complexes, the working assumptions introduced in Section 3.1 to analyze the complexity of this auxiliary function are fulfilled. As already observed in Remark 3.1.7, running the auxiliary function HomotopyExpansion has a time cost bounded by a constant depending on d. Moreover, the space required does not exceed the space needed to represent each vertex lower star. This motivates the authors of [162] to consider each call of the auxiliary function to have constant cost. The iteration of the auxiliary function for each vertex in the cubical complex makes the overall contribution of HomotopyExpansion linear in the vertexes.

The time complexity of ProcessLowerStars is given by two contributions that are linear in the vertexes of the cubical complexes, thus providing an overall time complexity of

$$O(|Q_0|). \tag{3.2}$$

The space required by the operations performed does not exceed the space necessary to represent Q.

The simplicial case. Now, we emphasize two facts about ProcessLowerStars applied to simplicial complexes that are of interest for this thesis. A first fact is about the validity of the algorithm. Indeed, there is no theoretical obstacle to apply the same procedure to a simplicial complex S rather than to a cubical complex Q. Indeed, Proposition 3.2.2 holds also for simplicial complexes and, as seen in Section in 3.1, the auxiliary function HomotopyExpansion applies to regular cell complexes with intersection property, including the simplicial case by Remark 1.1.13.

As the second fact, the complexity analysis is not the same and it is important to remark those differences since, the first generalization of ProcessLowerStars to the multiparameter case, discussed in Section 3.3, is presented for simplicial complexes. Thus, differences in the complexity analysis cannot be explained as depending on the multiparameter case. Most of them, instead, are given by the simplicial setting.

Indeed, for ComputeLowerStar, we express the time cost in the simplicial case linearly in the number of simplices in Section 4.4.1 through Formula (4.2). Indeed, for simplicial complexes, there is no fixed upper bound for the cardinality of the star of a vertex. For the same reason, when analyzing HomotopyExansion, as remarked in Section 3.1.3, the time complexity of each call of HomotopyExpansion over a vertex v is not in general bounded by a constant and its contribution is given by $O(|\mathrm{Low}_f(v)| \cdot \log |\mathrm{Low}_f(v)|)$. We have a call for each vertex $v \in S$ and thus

$$\sum_{v \in S_0} |\mathrm{Low}_f(v)| \cdot \log |\mathrm{Low}_f(v)|. \tag{3.3}$$

The logarithmic term can be overestimated by $\log |\mathrm{Star}|$ which makes it independent from the vertex v. We take the logarithmic term out of the sum. By Proposition 3.2.2, lower stars form a partition of S. Hence, globally, the cost of ProcessLowerStar is dominated by the contribution of HomotopyExpansion and thus, in the simplicial case, we get

$$O(|S| \log |\mathrm{Star}|). \tag{3.4}$$

By comparing Formula (3.2) to Formula (3.4), we have that the number of vertexes have been replaced by the number of cells and, moreover, we cannot get rid of the parameter $|\mathrm{Star}|$. Indeed, this latter, in the worst-case, can possibly reach the same order of magnitude as $|S|$.

3.3 A global preprocessing algorithm for multipersistent homology: Matching

In this section, we present the algorithm Matching introduced by Allili, Kaczynski, Landi, and Masoni in [4]. Matching retrieves a discrete gradient compatible with a vector-valued filtering

function generalizing the algorithm treated in Section 3.2. The algorithm can be thought of as a preprocessing for multiparameter persistence. Details for Matching are provided since the algorithm is proved to be equivalent to the one treated in Chapter 4. For convenience in comparison to other algorithms, we do not describe the algorithm in [4] as it is done by the authors. Rather, we describe Matching by means of the auxiliary function HomotopyExpansion we have introduced in Section 3.1.

The section is structured as follows. In Section 3.3.1, we make the input assumptions and the output properties explicit by following what the authors outlined. Additionally, we explain some crucial dissimilarities with respect to the algorithm of Section 3.2 generalized by Matching. This emphasizes the *global nature* of the algorithm as opposed to a local nature. In Section 3.3.2, Matching is described. In Section 3.3.3, we remark some complexity aspects related to Matching which will be useful in comparison to the algorithm described in Chapter 4.

3.3.1 Input and output formalization

In this section, we make the input assumptions for Matching explicit. Afterwards, we state the properties of the output retrieved. At the end of the section, we show some properties of lower stars of general cells holding under those assumptions which provides preliminaries for Chapter 4 and highlight the global nature of the algorithm.

According to [4], Matching requires in input an abstract simplicial complex S as in Definition 1.1.6 with no constraints on the dimension of the complex. By Remark 1.1.13, an abstract simplicial complex is a special case of a regular cell complex with intersection property.

As a second input, Matching requires a vector-valued function $\tilde{f} = (\tilde{f}_1, \dots, \tilde{f}_n)$ vertex-induced (Definition 3.1.2) by a function f defined over the vertexes of S with no restriction on the number of function components n. For the correctness of the procedure f is assumed to be component-wise injective, i.e., injective on each function component f_i. A general vector-valued filtering function can be made into a component-wise injective one, with a controlled difference with respect to the infinity norm, by *simulation of simplicity* [82] applied to each component.

We state a property which is already implied by Proposition 3.6 in [3] and which paves the way to the correctness of the algorithm presented in Chapter 4. We give our proof on how the correspondence between simplex lower stars and level sets of Proposition 3.2.3 generalizes to multiparameter case. In view of the algorithm in Chapter 4, we express the result in terms of regular cell complexes X with intersection property, as in Definition 3.1.1.

3.3.1 Proposition. For any component-wise injective function $f : X_0 \longrightarrow \mathbb{R}^n$ defined over the vertexes of a regular cell complex with the intersection property and any non-empty level set Lset of $\tilde{f}$ there exists a unique cell $a \in X$ such that

$$\text{Lset} = \text{Low}_f(a)$$

and $\text{Lset} = \tilde{f}^{-1}(\tilde{f}(a))$.

Proof. Let $u \in \mathbb{R}^n$ be such that $\text{Lset} := \tilde{f}^{-1}(u)$ is non-empty. In order to prove uniqueness, suppose there are two cells $a, a' \in X$ such that $\text{Low}_f(a) = \text{Lset} = \text{Low}_f(a')$. Notice that, $a, a' \in \text{Lset}$, since any cell belongs to its own lower star. Moreover, condition $\text{Low}_f(a') = \text{Low}_f(a)$ implies that $a' \in \text{Low}_f(a)$ and $a \in \text{Low}_f(a')$ at the same time. By definition of lower star, we get in particular $a' \ll a$ and $a \ll a'$, that is $a' = a$. Now, we show that there exists $a \in X$ such that $\text{Lset} \subseteq \text{Low}_f(a)$. Since f is component-wise injective on vertexes, for each component f_i, there exists a unique vertex $v_i \in X$ such that $f_i(v_i) = u_i$. By assumption, for each $b \in \text{Lset}$, $\tilde{f}_i(b) = f_i(v_i)$. By construction of $\tilde{f}$ from f, we have that v_i is a vertex of all cells in Lset. By Proposition 4.7.13 (i) in [20], in a regular cell complex with intersection property, any cell is completely determined by its vertexes. Hence, the vertexes $v_1, \ldots, v_n$ determine a unique cell a in X. By construction, $\tilde{f}(a) = u$, implying $a \in \text{Lset}$. By Proposition 4.7.13 (ii) in [20], if the vertexes of a are also vertexes of another cell b, then $a \ll b$. Thus, for each $b \in \text{Lset}$, $b \gg a$ which proves $\text{Lset} \subseteq \text{Low}_f(a)$. The opposite inclusion is given by Proposition 3.1.5. Hence, we conclude that $\text{Lset} = \text{Low}_f(a)$ which concludes our proof. □

This means that there is an injective but not necessarily surjective map between non-empty level sets and cells in X. Let P be the set of cells in X corresponding to non-empty level sets. Since the non-empty level sets form a partition for X, we get the following result.

3.3.2 Proposition. Given a regular cell complex with the intersection property X and a vector-valued filtering function $\tilde{f}$ which is vertex-induced from a component-wise injective function f on vertexes of X, the lower stars $\text{Low}_f(a)$, for $a \in P$, form a partition of X.

There is an additional input required. The third input is, according to Definition 3.1.6, an indexing J on S. The indexing has to be compatible both to the coface partial order $\ll$ among simplices and to the values under $\tilde{f}$. Explicitly, J has to satisfy, the following property

$$\forall s \neq t \in S, s \ll t \text{ or } \tilde{f}(s) \precnsim \tilde{f}(t) \quad \Rightarrow \quad J(s) < J(t). \tag{3.5}$$

The role of the indexing J is twofold. On the one hand, J locally determines the priority for the retrieval of discrete gradient in the same way as a lexicographic indexing does for the algorithm in Section 3.2. On the other hand, J determines the global process ordering for simplices in the input complex. In this sense, as opposed to the algorithm already treated in Section 3.2, the approach is global and cannot be parallelized. The introduction of a global indexing on simplices is motivated by the multiparameter case. Indeed, under the input assumptions for Matching, authors show how differently from the scalar-valued case vertex lower stars (Definition 3.1.3) subdivide the input simplicial complexes. The analogue for Proposition 3.2.2 does not hold, in general, for vector-valued filtering functions $\tilde{f}$. Consider, for instance, the edge in Figure 1.10(c) having incomparable vertex values of function components. Nonetheless, from Proposition 3.1.4, we

know the lower stars to cover S when all the simplices, and not simply the vertexes are involved. However, this by no means is a partition since simplex lower stars overlap as it happens for the lower star of the edge and the ones of the vertexes in Figure 1.7(a).

To summarize, Matching receives in input:

- S : an abstract simplicial complex;
- $\tilde{f}$: a vector-valued filtering function which is vertex-induced from a component-wise injective function f on the vertexes of S;
- J : an indexing over S.

As regards the output retrieved by Matching, we refer to Definition 1.4.5 for the notion of discrete gradient, and to Definition 1.4.7 for the notion of Morse set relative to a discrete gradient. Then, $\text{Matching}(S, \tilde{f}, J)$ returns:

- V : a discrete gradient over S compatible with $\tilde{f}$;
- M : the Morse set relative to V, i.e., $M = M(V)$.

The output V is guaranteed to be a discrete gradient by Theorem 3.8 in [3]. The compatibility with function $\tilde{f}$ follows directly from Proposition 3.7 in [3].

A discrete gradient compatible with a function $\tilde{f}$ is also compatible with the filtration $\mathcal{S}^{\tilde{f}}$ induced on S by $\tilde{f}$. Hence, the retrieved output determines a filtered Morse complex $\mathcal{M}^{\tilde{f}}$ which is a compatible Morse reduction of $\mathcal{S}^{\tilde{f}}$ in the sense of Theorem 1.6.8. This ensures that Matching is a valid preprocessing for computing multipersistence in any possible way.

3.3.2 Description

In this section, we describe the algorithm Matching. We refer to the pseudocode in Algorithm 3. Matching runs the following auxiliary functions:

- ComputeLowerStar for extracting the lower star of simplex. It corresponds to the simplicial version of auxiliary function in Section 3.2 with the same name;
- HomotopyExpansion (see Section 3.1) for finding discrete vectors and critical simplices in a lower star of a simplex,

and exploits the auxiliary data structure

Algorithm 3 Matching($S, \tilde{f}, J$)

Input: S simplicial complex
Input: $\tilde{f} : S \longrightarrow \mathbb{R}^n$ function # extended from a component-wise injective function $\tilde{f}$ on the vertexes
Input: $J : S \longrightarrow \mathbb{R}$ indexing # J compatible with $\tilde{f}$ and $\ll$
Output: V list of cell pairs # discrete gradient compatible with $\tilde{f}$
Output: M list of cells # M is the Morse set of V

1: set V, M to empty lists
2: set classified to an array of length $|S|$ with false Boolean values # in classified entries are ordered according to increasing values of J
3: **for all** s in S **do** # simplex processing in ascending order of $J(s)$
4: **if** classified$[J(s)]$ =false **then**
5: $\mathrm{Low}_f(s) \leftarrow \mathrm{ComputeLowerStar}(s, S)$ # $\mathrm{Low}_f(s)$ contains the simplices in the lower star of s w.r.t. $\tilde{f}$
6: $(V_{\mathrm{Low}_f(s)}, M_{\mathrm{Low}_f(s)}) \leftarrow \mathrm{HomotopyExpansion}(S, \mathrm{Low}_f(s), J)$ # classifies all cells in $\mathrm{Low}_f(s)$ as either part of a pair in $V_{\mathrm{Low}_f(s)}$ or a cell in $M_{\mathrm{Low}_f(s)}$
7: append $V_{\mathrm{Low}_f(s)}$ to V
8: append $M_{\mathrm{Low}_f(s)}$ to M
9: **for all** t in $\mathrm{Low}_f(s)$ **do**
10: classified$[J(t)] \leftarrow$ true
11: **end for**
12: **end if**
13: **end for**
14: **return** (V, M)

- classified, a length $|S|$ array of Booleans.

The algorithm processes all simplices in S in a for-cycle. As observed in the previous section, lower stars of simplices do not form a partition of the input simplicial complex. Hence, simplices have to be processed according to increasing values of the indexing J. This implies that, as opposed to the local algorithm of Section 3.2, Matching cannot be broken into a parallel or distributed approach.

The auxiliary data structure classified is initialized with all entries set to false. For each simplex s, at line 4 in Algorithm 3, the algorithm checks whether s is classified. An already classified simplex is not processed. A classified simplex s is passed to the auxiliary function ComputeLowerStar. The auxiliary function ComputeLowerStar extracts $\mathrm{Star}_S(s)$, i.e., the star of s. Then, the auxiliary

function visits $\mathrm{Star}_S(s)$ to select each simplex t satisfying condition $\tilde{f}(t) \preceq \tilde{f}(s)$. The so-obtained list $\mathrm{Low}_f(s)$ of simplices is given in input to the auxiliary function $\mathrm{HomotopyExpansion}$. According to Section 3.1.1 the latter returns a discrete gradient $V_{\mathrm{Low}_f(s)}$ over $\mathrm{Low}_f(s)$ which is used to update the output list V, and the corresponding Morse set $M_{\mathrm{Low}_f(s)}$ which is used to update the output list M. As a simplex is classified either as an element in a discrete vector in V or as a critical simplex in M, the algorithm updates the corresponding entry in $\mathrm{classified}$ with value true.

3.3.3 Complexity remarks

In this section, we provide our complexity analysis of $\mathrm{Matching}$. First, we make our assumptions explicit. The assumptions are satisfied by the implementation of the algorithm we propose in Section 6.1 where we compare the algorithm performances to the performances of the algorithm of Chapter 4. Afterwards, we focus on the auxiliary structures and their impact on the overall cost. Then, we analyze each single auxiliary function in a parallel way with respect to what we did in Section 3.2.3. Finally, we compare our analysis to the one provided by the authors and the one related to the algorithm in Section 3.2.3. In doing this, we discriminate between differences due to the simplicial case and differences due to the vector-valued input filtering function.

Assumptions. We assume the input simplicial complex to have dimension d. We indicate with Star the simplex star with maximal cardinality in S. Recall, from Remark 3.1.7 that, in a simplicial complex, $|\mathrm{Star}|$ is not bounded by a constant and it is possibly as large as $|S|$. Moreover, we assume some information about the simplicial complex to be computed and stored to simplify the discussion.

- for each simplex $s \in S$, the value $\tilde{f}(s)$ can be retrieved in linear time in the number of the vertices of s from the offline-stored values of f. The corresponding extra space required is considered to be constant;
- for each cell $s \in S$, we assume that the boundary and coboundary can be retrieved in linear time in their size. Visiting the boundary or coboundary of s takes linear time in the size of the boundary or coboundary, respectively;
- for the auxiliary function $\mathrm{HomotopyExpansion}$, we make the same assumptions of Section 3.1.3. Hence, each call over $\mathrm{Low}_f(s)$ takes time $O(|\mathrm{Low}_f(s)| \log |\mathrm{Low}_f(s)|)$ and requires an additional space which is negligible with respect to the storage cost of the input representation.

Auxiliary structures. The auxiliary structure $\mathrm{classified}$ globally keeps track of the classified simplices all along the algorithm. This requires an $O(|S|)$ extra space. Moreover, all simplices in

S need to be processed in the for-cycle. This requires to add the extra space of a global queue of size $O(|S|)$. Both the auxiliary structure extra spaces are not negligible with respect to the cost of representing the algorithm inputs.

ComputeLowerStar. In a simplicial complex, the cardinality of Star is not bounded by a constant. Visiting the star takes linear time in the cardinality of the star. Comparing two floats is considered to take constant time, so that a single lower star extraction is performed in linear time in the cardinality of the star. According to Proposition 3.3.1, the auxiliary function is repeated for every lower star corresponding to a non-empty level set and P indicates the set of simplices whose lower stars are non-empty level sets in a such a way that $|P|$ is the number of non-empty level sets under the filtering function.
We estimate the time cost in the following way:

$$\sum_{s\in P} |\operatorname{Star}_S(s)|. \tag{3.6}$$

Depending on the input, we have

$$|S_0| \leqslant |P| \leqslant |S|.$$

Hence, in Formula (3.6), each j-simplex can be counted from $j+1$ times (the number of its vertexes) to $\sum_{k=0}^{j+1}\binom{j+1}{k+1}$ times (the number of its faces). The time cost Time of the auxiliary function satisfies the following inequalities:

$$\sum_{j=0}^{d}(j+1)|S_j| \leqslant \text{ Time } \leqslant \sum_{j=0}^{d}|S_j|\sum_{k=0}^{j+1}\binom{j+1}{k+1}. \tag{3.7}$$

In both extremal cases, we have linearity in $|S|$ but under very different proportionality coefficients. For the extra space required, we recall that, for simplicial complexes, implicit (co)boundary representations are not possible in general. By assumption, the extra space is linear in the size of the star, and linear in $|\operatorname{Star}|$ in the worst-case. Hence, it does not exceed the space required for representing the algorithm input.

HomotopyExpansion. By assumptions, each call of the auxiliary function $\mathrm{HomotopyExpansion}$ over the lower star $\operatorname{Low}_f(s)$ of a simplex s takes time $O(|\operatorname{Low}_f(s)|\cdot\log|\operatorname{Low}_f(s)|)$ and requires an additional space which is negligible with respect to the storage cost of the input representation. We have a call for each simplex $s\in P$ and thus

$$\sum_{s\in P}|\operatorname{Low}_f(s)|\cdot\log|\operatorname{Low}_f(s)|. \tag{3.8}$$

The logarithmic term can be overestimated by $\log|\operatorname{Star}|$ which makes it independent from the vertex v. We take the logarithmic term out of the sum. By Proposition 3.3.2, lower stars over simplices in P form a partition of S, so we have that $|S| = \sum_{s\in P}|\operatorname{Low}_f(s)|$.

Thus, globally, the cost of Matching is dominated, in time, by the contribution of HomotopyExpansion and thus, we get

$$O(|S| \cdot \log |\,\mathrm{Star}\,|), \tag{3.9}$$

and in runtime space, by the contributions of classified and global queue of the for-cycle

$$O(|S|).$$

Comparison to other analyses. The time cost analysis slightly differs from the one proposed by the algorithm authors in [4] where the given worst-case complexity is expressed by $O(|S| \cdot |\,\mathrm{Star}\,| \cdot \log |\,\mathrm{Star}\,|)$. We obtained a lower worst-case estimation by making the set P explicit. Indeed, our analysis takes into account that only lower stars of simplices in P are processed and, only there HomotopyExpansion is called. Thus, we do not confute the complexity analysis given by authors but we provide an input-dependent one. In Section 4.4, we see how the same worst-case analysis applies to the algorithm of Chapter 4, this time, independently from the input.

In the simplicial case, we compare the time cost analysis provided, in Section 3.2, for ProcessLowerStars and, in this section, for Matching. Formula (3.4) is equivalent to Formula (3.9). This means that the filtering function being vector-valued rather than scalar-valued does not affect the worst-time complexity. In fact, a vector-valued filtering function is more likely to have more level sets and, thus a value $|P|$ closer to $|S|$ than to $|S_0|$. Indirectly, this affects the average-case of both the contributions of HomotopyExpansion and ComputeLowerStar.

In conclusion, as seen at the beginning of this section, the global nature of Matching affects directly the runtime storage cost. We will see in Chapter 4 an equivalent algorithm preserving, in the case of vector-valued filtering functions, the local nature of the algorithm in Section 3.2. From the time cost point of view, the global nature of algorithm Matching preserves the same worst-case holding for scalar-valued filtering functions. A relevant difference in between local and global approaches is highlighted by the contribution of the auxiliary function ComputeLowerStar. The experimental tests in Section 6.1 make it clear the advantages of the local algorithm of Chapter 4 over Matching. Indeed, the former algorithm will be shown to keep, through a local approach, the time complexity of ComputeLowerStar in the best case of Formula 3.7 independently from the kind of input. Matching instead, depending on the input, can possibly have a ComputeLowerStar contribution closer to the worst case of Formula 3.7. The more components the filtering function has, the more level sets are likely to be close to $|S|$ in number.

4 A local Morse-based preprocessing algorithm for multipersistent homology: ComputeDiscreteGradient

In this chapter, we present the algorithm ComputedDiscreteGradient we introduced in [115]. ComputeDiscreteGradient retrieves a discrete gradient compatible with a vector-valued filtering function generalizing to multiparameter filtrations the *local* nature of the algorithm treated in Section 3.2, as opposed to the algorithm in Section 3.3 which generalizes to multiparameter filtrations the same algorithm by means of a *global* approach. The algorithm can be thought of as a preprocessing for multiparameter persistence.

The chapter is structured as follows. In Section 4.1, we make the input assumptions and the output properties explicit. In Section 4.2, we formalize the main novelty of the proposed algorithm, that is the retrieval of a *well-extensible* indexing on the vertexes of the input complex. We show how this guarantees a local nature to ComputeDiscreteGradient. In Section 4.3, ComputeDiscreteGradient is described. In Section 4.4, we provide our complexity analysis of ComputeDiscreteGradient with special emphasis on comparisons to algorithms in Section 3.2 and Section 3.3. In Section 4.5, we prove the ComputeDiscreteGradient correctness by showing equivalence to the algorithm reviewed in Section 3.3. In Section 4.6, we describe the algorithm presented in [93] to retrieve an actual filtered Morse complex out of the discrete gradient output by ComputeDiscreteGradient.

4.1 Input and output formalization

In this section, we make the input assumptions for ComputeDiscreteGradient explicit.

As a first input, the algorithm requires a *regular cell complex with the intersection property* X (Definition 3.1.1). This requirement is the same we state in our review of the algorithm in Section 3.1. By Remark 1.1.13, this includes both cubical complexes, required by the algorithm in Section 3.2, and simplicial complexes, required by the algorithm in Section 3.3.

As a second input, the algorithm ComputeDiscreteGradient requires a vector-valued function $f = (f_1, \dots, f_n) : X_0 \longrightarrow \mathbb{R}^n$ defined over the vertexes of the input cell complex. As already done for the algorithm of Section 3.3, for the correctness of the procedure f is assumed to be component-wise injective, i.e., injective on each function component f_i. A general vector-valued

filtering function can be made into a component-wise injective one, with a controlled difference with respect to the infinity norm, by *simulation of simplicity* [82] applied to each component. As opposed to what is done for the global algorithm in Section 3.3, the input is the function f and not function $\tilde{f}$, that is the corresponding vertex-induced function (Definition 3.1.2). This is coherent to the requirements for the local algorithm in Section 3.2. The algorithm is valid for any number n of parameters in the multifiltered complex. Furthermore, the algorithm is agnostic in the dimension d of the input complex.

To summarize, $\mathrm{ComputeDiscreteGradient}$ receives in input:

- X : a regular cell complex with intersection property;
- f : a vector-valued function on the vertexes of S which is component-wise injective function.

As regards the output retrieved by $\mathrm{ComputeDiscreteGradient}$, we refer to Definition 1.4.5 for the notion of discrete gradient, and to Definition 1.4.7 for the notion of Morse set relative to a discrete gradient. Then, $\mathrm{ComputeDiscreteGradient}(X, f)$ returns:

- V : a discrete gradient over X compatible with $\tilde{f}$, i.e., the multiparameter filtering function which is vertex-induced by f;
- M : the Morse set relative to V, i.e., $M = M(V)$.

The output V is guaranteed to be a discrete gradient compatible with the function $\tilde{f}$ by the equivalence to algorithm in Section 3.3 proved in Section 4.5. A discrete gradient compatible with a function $\tilde{f}$ is also compatible with the filtration $\mathcal{X}^{\tilde{f}}$ induced on X by $\tilde{f}$. Hence, the retrieved output determines a filtered Morse complex $\mathcal{M}^{\tilde{f}}$ which is a compatible Morse reduction of $\mathcal{X}^{\tilde{f}}$ in the sense of Theorem 1.6.8. The algorithm described in Section 4.6 allows to retrieve a $\mathcal{M}^{\tilde{f}}$ from a discrete gradient given by $\mathrm{ComputeDiscreteGradient}$. This ensures that $\mathrm{ComputeDiscreteGradient}$ is a valid preprocessing for computing multipersistence in any possible way. This persistence-preserving reduction in the number of cells and non-trivial filtration steps is proved to satisfy an optimality property we have introduced and which is fully treated in Chapter 5.

4.2 Well-extensible indexing

In this section, we formalize the main novelty of the proposed algorithm, that is the retrieval of a *well-extensible* indexing on the vertexes of the input complex. We show how this guarantees a local nature to $\mathrm{ComputeDiscreteGradient}$. The results of this section are in particular useful for the complexity analysis provided in Section 4.4.

With reference to the formalization of the input provided in the previous section, let X be a regular cell complex with the intersection property and $f = (f_1, \dots, f_n) : X_0 \longrightarrow \mathbb{R}^n$ a component-wise injective function defined over the vertexes of X. The algorithms presented in Section 3.2 and Section 3.3 share the common trait of acting over a partition into *lower stars* (Definition 3.1.3) where to independently build the output discrete gradient by means of the auxiliary function HomotopyExpansion we describe in Section 3.1. The algorithm in Section 3.2, detects the partition into lower stars by processing all vertexes in X and, for each of those vertexes, computing its lower star. The algorithm in Section 3.2, detect the partition into lower stars by processing all cells in X and, for each of those cells, deciding whether or not to compute its lower star. To do so, an indexing over all the cells in X with suitable properties is required in input. The partition into lower stars where to run HomotopyExpansion is thus constructed runtime and not a priori defined. Instead, ComputeDiscreteGradient produces its own *well-extensible indexing* I, this time restricted to vertexes of X to a priori define the partition where to act. Let $\tilde{I} : X \longrightarrow \mathbb{R}$ be the indexing over X vertex-induced by I defined over the vertexes. Explicitly, we have

$$\forall i \in \{1, \dots, n\}, \qquad \tilde{f}_i(a) := \max_{\substack{v \in X_0 \\ v \ll a}} f_i(v) \qquad \text{and} \qquad \tilde{I}(a) := \max_{\substack{v \in X_0 \\ v \ll a}} I(v).$$

4.2.1 Definition (Well-extensible indexing)**.** Let $f : X_0 \longrightarrow \mathbb{R}^n$ be a function defined over the vertexes of a cell complex X. A *well-extensible indexing with respect to* f is an indexing I over X_0 such that, for all $a, b \in X$,

$$\tilde{f}(a) \preceq \tilde{f}(b) \qquad \Rightarrow \qquad \tilde{I}(a) \leqslant \tilde{I}(b)$$

The cell complex X is subdivided into *index-based lower stars*.

4.2.2 Definition (Index-based lower star)**.** For each vertex v in a cell complex X, the *index-based lower star of* v is the set

$$\mathrm{Low}_I(v) := \{a \in \mathrm{Star}_X(v) \mid \tilde{I}(a) \leqslant I(v)\}.$$

Clearly index-based lower stars correspond to lower stars of vertexes with respect to a scalar-valued filtering function that is vertex-induced from an injective one. Hence, they form a partition by Proposition 3.2.2 even in the case of non-well-extensible indexings I. However, cells in the same index-based lower star might have different values under $\tilde{f}$. We subdivide the index-based lower star of each vertex v into level sets

$$K(v) := \mathrm{Low}_I(v)/_{\sim}, \quad \text{with} \quad a \sim b \Leftrightarrow \tilde{f}(a) = \tilde{f}(b).$$

The well-extensible assumption guarantees the following compatibility between index-based lower stars and level sets of $\tilde{f}$.

4.2.3 Proposition (Index-based partition compatibility with level sets)**.** Let X be a regular cell complex with the intersection property, and $f : X_0 \longrightarrow \mathbb{R}^n$ a component-wise injective function on the vertexes of X. Let I be a well-extensible indexing with respect to f. Then, for any non-empty level set $\mathrm{Lset} = \tilde{f}^{-1}(\tilde{f}(a))$ with $a \in X$, there exists one, and only one, vertex $v \ll a$ such that

$$\mathrm{Lset} \subseteq \mathrm{Low}_I(v).$$

Proof. By Proposition 3.3.1, for a component-wise injective function f over the vertexes of X, each level set $\tilde{f}^{-1}(\tilde{f}(a))$ is equal to the lower star $\mathrm{Low}_f(a)$. By Proposition 3.2.2, the cell a belongs to the index-based lower star $\mathrm{Low}_I(v)$ of a unique vertex $v \in X$, so that by Proposition 3.1.5, we get $\tilde{I}(a) = \tilde{I}(v)$. Moreover, since I is injective on vertexes, we have that $v \ll a$. Now consider a cell $b \in \mathrm{Low}_f(a)$ which, by Proposition 3.1.5, implies $\tilde{f}(a) = \tilde{f}(b)$, that is both $\tilde{f}(a) \preceq \tilde{f}(b)$ and $\tilde{f}(a) \succeq \tilde{f}(b)$. By I being well extensible, we get $\tilde{I}(b) = \tilde{I}(a)$ which gives us $\mathrm{Low}_f(a) \subseteq \mathrm{Low}_I(v)$ and concludes the proof. □

The Proposition 4.2.3 ensures that each non-empty level set of $\tilde{f}$ is entirely contained in one, and only one, index-based lower star. As a corollary of Proposition 4.2.3, we get the following set equality.

$$X = \bigsqcup_{v \in X_0} \bigsqcup_{\mathrm{Lset} \in K(v)} \mathrm{Lset}\,. \tag{4.1}$$

4.3 Algorithm description

In this section, we describe the algorithm ComputeDiscreteGradient. We refer to the pseudocode in Algorithm 4 and the notation introduced in Section 4.1, for the input and output formalism, and in Section 4.2 for the auxiliary structures. The example in Figure 4.1 presents an overview of the main steps of the divide-and-conquer approach of the algorithm. The algorithm is described by means of the following auxiliary functions:

- ComputeIndexing computes the well-extensible indexing I (see Figure 4.1(a)-(b));
- ComputeIndexLowerStar extracts the list $\mathrm{Low}_I(v)$ containing the index-based lower star of the vertex v (see differently colored parts in Figure 4.1b);
- SplitIndexLowerStar computes the list $K(v)$ of all lists of level sets with respect to $\tilde{f}$ in $\mathrm{Low}_I(v)$ (see Figure 4.1c);

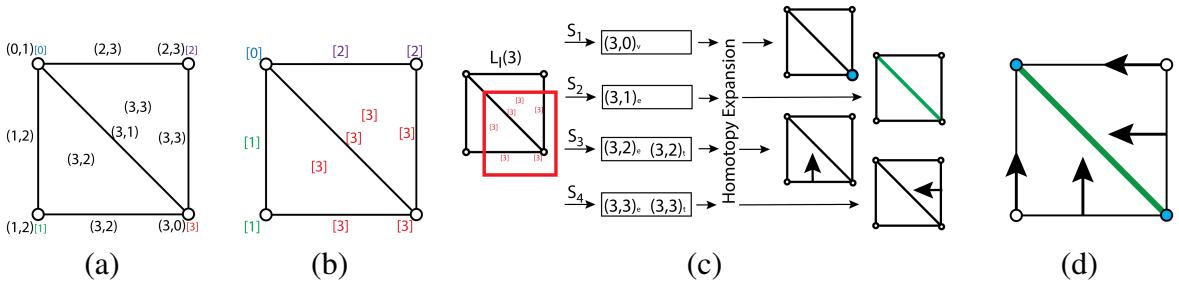


Figure 4.1: (a) the filtering function values are depicted as pairs. The vertex indexing I is depicted within square brackets. (b) $\tilde{I}$ is propagated to all the simplices. Colors indicate simplices belonging to the same index-based lower star. (c) For each lower star, the level sets Lset_i are built collecting simplices having the same function value $\tilde{f}$. Pairs and critical simplices are created via homotopy expansions within each Lset_i. (d) The discrete gradient is obtained as the union of all the pairs and critical simplices classified in the lower stars. Critical vertices are depicted in blue and the critical edge is depicted in green.

- HomotopyExpansion computes the lists $V_{\mathrm{Low}_I(v)}$, with the discrete vectors, and $M_{\mathrm{Low}_I(v)}$ with the corresponding Morse set $M_{\mathrm{Low}_I(v)} = M_{\mathrm{Low}_I(v)}(V_{\mathrm{Low}_I(v)})$ restricted to the index-based lower star $\mathrm{Low}_I(v)$ (see Figure 4.1d). HomotopyExpansion is the auxiliary function is introduced in Section 3.1.

ComputeDiscreteGradient implements:

- $i(\leqslant_{\mathrm{Lex}})$: an implicit lexicographic indexing on X. Each k-cell a has its vertexes $[a] := [v_1, \ldots, v_{|a|}]$ represented in decreasing order of values under I, i.e., in such a way that $I(v_1) > \cdots > I(v_{|a|})$, with $|a|$ the number of vertexes of a. Cells are implicitly totally ordered by
$$[a] \leqslant_{\mathrm{Lex}} [b] \qquad \Leftrightarrow \qquad i(\leqslant_{\mathrm{Lex}})(a) \leqslant i(\leqslant_{\mathrm{Lex}})(b),$$
where $\leqslant_{\mathrm{Lex}}$ is the lexicographic order between tuples of kind $[a]$.

ComputeDiscreteGradient. The algorithm retrieves a well-extensible indexing I, stored in a length $|X_0|$ array, by means of the auxiliary function ComputeIndexing, and then, it processes all vertices, one by one, in a for-cycle according to increasing values of I. However, the body of that for-cycle can be also implemented in parallel since index-based lower stars of vertexes form a partition for X as remarked in Section 4.2. For each vertex v, at line 4 in Algorithm 4, the algorithm calls the auxiliary function $\mathrm{ComputeIndexLowerStar}(v, I, X)$ which returns the list $\mathrm{Low}_I(v)$ of cells in the index-based lower star of v. At line 5 in Algorithm 4, the list is passed to SplitLowerStar to obtain a list $K(v)$ with all lists of level sets of $\tilde{f}$ relative to the vertex v. For each list $\mathrm{Lset} \in K(v)$, at line 7 in Algorithm 4, the auxiliary function HomotopyExpansion returns the discrete vector list V_{Lset}, and it corresponding critical cell list M_{Lset} with respect to the implicit lexicographic order $i(\leqslant_{\mathrm{Lex}})$. The lists V_{Lset} and M_{Lset} are appended to the final output lists V and M to be returned.

Algorithm 4 ComputeDiscreteGradient(X, f)

Input: X regular cell complex with intersection property
Input: $f : X_0 \longrightarrow \mathbb{R}^n$ component-wise injective function
Output: V list of cell pairs # discrete gradient compatible with $\tilde{f}$
Output: M list of cells # Morse set of V

V, M are empty lists
$I \leftarrow \mathrm{ComputeIndexing}(X_0, f)$ # I is a well extensible indexing with respect to f
for all v in X_0 **do** # independently from the order
  $\mathrm{Low}_I(v) \leftarrow \mathrm{ComputeIndexLowerStar}(v, I, X)$
  # $\mathrm{Low}_I(v)$ is a min-heap with the cells in the index-based lower star of v partially ordered by increasing values of $\tilde{f}$
  $K(v) \leftarrow \mathrm{SplitIndexLowerStar}(f, \mathrm{Low}_I(v))$
  # $K(v)$ contains the lists of level sets in $\mathrm{Low}_I(v)$
  for all Lset in K_v **do** # independently from the order
    $(V_{\mathrm{Lset}}, M_{\mathrm{Lset}}) \leftarrow \mathrm{HomotopyExpansion}(X, \mathrm{Lset}, i(\leqslant_{\mathrm{Lex}}))$ # classifies all cells in Lset as either part of a pair in V_{Lset} or a cell in M_{Lset}
    append V_{Lset} to V
    append M_{Lset} to M
  end for
end for
return (V, M)

4.3.1 Auxiliary functions

In this section, we describe in more detail the auxiliary functions of ComputeDiscreteGradient.

ComputeIndexing There are different ways to obtain a well-extensible indexing I. A well-extensible indexing guarantees correctness. In our approach, ComputeIndexing is a heap sort algorithm which produces I by ordering all vertexes v according to the values $f_1(v)$.

4.3.1 Proposition. The indexing I computed by ComputeIndexing is well-extensible.

Proof. ComputeIndexing computes an indexing I since f_1 is injective, and, by definition, for any two vertexes v and w in X, $f_1(v) \preceq f(w)$ is equivalent to $I(v) \leqslant J(w)$. To see that I is well-extensible, we notice that the condition $\tilde{f}(a) \preceq \tilde{f}(b)$ implies $\tilde{f}_1(a) \preceq \tilde{f}_1(b)$. Thus, a vertex $v \in b$ exists such that $\tilde{f}_i(v) \geqslant \tilde{f}_i(w)$ for every vertex $w \ll a$. This implies $I(v) \geqslant I(w)$, for every vertex $w \ll a$. Thus, we conclude that $\tilde{I}(a) \leqslant \tilde{I}(b)$. □

ComputeIndexLowerStar**.** ComputeIndexLowerStar extracts the list Star(v) with elements all cells in the star of v in X. The cells in Star(v) are processed and, for each cell a, if the first entry in $[a]$ equals to v, then a is added to the index-based lower star $\mathrm{Low}_I(v)$. The auxiliary function returns the list of cells $\mathrm{Low}_I(v)$.

SplitIndexLowerStar. The values under f are associated with vertexes only. The auxiliary function SplitIndexLowerStar initializes an associative array to tag each level set in $\mathrm{Low}_I(v)$ by its value under $\tilde{f}$. The associative array is implemented so that a tag appears after all the tags preceding it. The level set with tag equal to $\tilde{f}^{-1}(\tilde{f}(v))$ is initialized with the element v. The auxiliary function SplitIndexLowerStar processes all elements in $\mathrm{Low}_I(v)$ but v with order given by $i(\leqslant_{\mathrm{Lex}})$, so that each processed cell has only few vertexes more than some already processed cell (one vertex more in the simplicial case). For each $a \in \mathrm{Low}_I(v)$, the value $\tilde{f}(a)$ is computed through the maximum rule defining vertex-induced filtering functions (Definition 3.1.2) for each component. If the value is not in the associative array, then a new value is added and a new level set list is initialized with the element a. Otherwise, the cell is added to the level set list corresponding to the value tag to be found in the associative array of tags. The auxiliary function returns the list K_v containing all level set lists.

HomotopyExpansion. The last step consists in the actual pairing of cells, within each level set Lset in $K(v)$, performed by the subroutine HomopotyExpansion. The auxiliary function HomotopyExpansion is the one presented in Section 3.1 and used for describing both by the algorithm of Section 3.2 and that of Section 3.3. HomotopyExpansion requires a regular cell complex with intersection property, a lower star of some cell along with an indexing of the lower star cells compatible with the incidence cell relations. The requirements for HomotopyExpansionare satisfied. Indeed, the complex X is a suitable input. By Proposition 3.3.1, each level set Lset in $K(v)$ is a lower star for some cell. The indexing $i(\leqslant_{\mathrm{Lex}})$ is a lexicographic indexing and, thus it is compatible with the coface relation $\ll$ over X. The main idea of HomotopyExpansion is that, two cells, say k-cell a and $(k+1)$-cell b, are paired via homotopy expansion when a has no unpaired faces and b has only one unpaired face (i.e. a). This is achieved by means of two lists whose elements are ordered according to increasing values of $i(\leqslant_{\mathrm{Lex}})$: Ord0 and Ord1. The ordered lists are used for selecting which cells in Lset are paired respectively with a higher or with a lower dimensional cell. It is proved in Proposition 3.6 in [3] that, in HomotopyExpansion, all cells in Lset are either classified as critical or paired with another cell and this happens exactly once per cell. This produces the two lists: V_{Lset} containing all pairs found in Lset, and M_{Lset} with all the remaining cells. Finally, the lists V containing all independent contributions of V_{Lset} and M_{Lset} is output by ComputeDiscreteGradient.

4.4 Complexity

This section is devoted to the complexity analysis of ComputeFormanGradient whose pseudocode is reported in Algorithm 4. First, we make our notation and assumptions explicit. In Section 4.4.1, we provide our analysis of ComputeDiscreteGradient. In Section 4.4.2, we discuss the analysis provided and we compare our analysis to those provided for the algorithms in Section 3.2 and Section 3.3.

To fix the notation, the input complex is assumed to be a d-dimensional cell complex X. Parameters involved in the complexity analysis are expressed in terms of cardinality $|\cdot|$ of sets used in Section 4.3. Moreover, we indicate with $\mathrm{Star}(a)$ the star of a cell $a \in X$, and, with Star, the cell star with maximal cardinality in X. Recall, from Remark 3.1.7 that, in a simplicial complex, $|\mathrm{Star}|$ is not bounded by a constant and it is possibly as large as $|X|$. Whereas, in a cubical complex $|\mathrm{Star}|$ is bounded by a constant.

In our analysis, we assume some information about our complex to be computed and stored; this will simplify the discussion by focusing on the complexity of the introduced algorithm only.

- for each cell in X the number of its vertexes is assumed to be constant.
- for each $\mathrm{Star}(v)$, the value $\tilde{f}(a)$, for all cells $a \in \mathrm{Star}(v)$ can be retrieved in linear time in the number of the vertices of cells in $\mathrm{Star}(v)$ from the offline-stored values of f;
- for each cell $a \in X$, we assume that the boundary and coboundary can be retrieved in linear time in their size and co(boundary) is stored off-line. Visiting the boundary or coboundary of a takes linear time in the number of cells in the boundary or coboundary, respectively;
- for the auxiliary function HomotopyExpansion, we make the same assumptions we made explicit in Section 3.1, so that to apply the same complexity analysis.

All these assumptions are satisfied by the implementation of ComputeDiscreteGradient described in Chapter 6 to perform our experimental tests.

4.4.1 Analysis

In this section we provide our time and space complexity analysis of ComputeDiscreteGradient. Here, we analyze separately the contribution of each auxiliary function involved in ComputeDiscreteGradient. For each auxiliary function, we make its time and storage cost contribution to the overall process explicit.

ComputeIndexing. Here, the vertices are sorted according to a single component of the input function. Thus, it requires $O(|X_0| \cdot \log |X_0|)$ operations. The extra space required is $O(|X_0|)$.

ComputeIndexLowerStar requires the extraction of the star of each vertex v which can be performed in constant time by assumption. Then, all cells in the star need to be visited in order to check whether their first vertex is v which takes $O(|\operatorname{Star}(v)|)$ time. The procedure is repeated for all vertexes in the input complex. Since, by assumption, the number of vertexes of any cell is constant, each cell is counted a constant number of times as part of a vertex star. Hence, the overall time cost is $O(|X|)$. In particular, since for a simplicial d-complex, each j-cell is extracted as part of $j+1$ different stars of a vertex, we get that the overall number of visited cells is

$$\sum_{v \in X_0} |\operatorname{Star}(v)| = \sum_{j=0}^{d} (j+1)|X_j|. \tag{4.2}$$

Since, for a cubical d-complex, each j-cell is extracted as part of 2^j different stars of a vertex, we get that the overall number of visited cells is

$$\sum_{v \in X_0} |\operatorname{Star}(v)| = \sum_{j=0}^{d} 2^j |X_j|. \tag{4.3}$$

This implies in both cases a time complexity contribution of $O(|X|)$. By assumption, there is no extra space requirement.

SplitIndexLowerStar requires visiting all cells in the vertex index-based lower star $\operatorname{Low}_I(v)$. By assumption, the cost of retrieving all values of $\tilde{f}$ for cells in $\operatorname{Low}_I(v)$ is linear in the number of vertexes of cells in $\operatorname{Low}_I(v)$. By assumption, the number of vertexes per cell is constant, hence the number of vertexes of cells in $\operatorname{Low}_I(v)$ has the same order of $|\operatorname{Low}_I(v)|$. In particular, in the cubical case the cost of computing $\tilde{f}$ is constant since the number of cells in the star is bounded by a constant depending on the complex dimension, and it is linear in $|\operatorname{Low}_I(v)|$, in the simplicial case. The cost of determining whether or not a cell in $\operatorname{Low}_I(v)$ is in a new level set is $O(\log |K_v|)$ since it requires to process, at most, a length $|K_v|$ associative array implemented to tag the level set values whose tags are ordered compatibly with the values of $\tilde{f}$. The overall cost of each call is given by the sum of the cost of computing values of $\tilde{f}$, which is $O(|\operatorname{Low}_I(v)|)$ and that of creating the level set lists, which is $O(|\operatorname{Low}_I(v)| \cdot \log |K_v|)$, this latter dominating the complexity. For general v, the worst-case of $|K_v|$ is bounded by the cardinality of the worst case index-based lower star and, thus ultimately, by the worst-case star, indicated by Star, so that we can overestimate the logarithmic term by $\log |\operatorname{Star}|$. Since Proposition 3.2.2 applies also to regular cell complexes with the intersection property, index-based lower stars form a partition of X, we get that the overall time cost contribution of the auxiliary function is

$$O(|X| \cdot \log |\operatorname{Star}|).$$

In particular, in a cubical complex the time cost becomes linear in $|X|$. The extra space required is $O(|\operatorname{Star}|)$ since the filtering function $\tilde{f}$ needs to be represented for each cell in $\operatorname{Low}_I(v)$.

HomotopyExpansion works over each level set $\text{Lset} \in K_v$ within a lower star $\text{Low}_I(v)$ of a vertex v. By the complexity analysis in Section 3.1, here taken by assumption, each call of the auxiliary function HomotopyExpansion over the level set Lset takes time $O(|\text{Lset}| \cdot \log |\text{Lset}|)$ and requires an additional space which is negligible with respect to $O(|\text{Lset}|)$. By Formula (4.1), we have a call of HomotopyExpansion for each $\text{Lset} \in K_v$ and each vertex $v \in X_0$ and, thus the overall number of operations is given by

$$\sum_{v \in X_0} \sum_{\text{Lset} \in K_v} |\text{Lset}| \log |\text{Lset}|. \tag{4.4}$$

The logarithmic term can be overestimated by $\log |\text{Star}|$ which makes it independent from both the level set in K_v and the vertex v in X_0. We take the logarithmic term out of the sum. Again by Formula (4.1), we have that $|X| = \sum_{v \in X_0} \sum_{\text{Lset} \in K_v} |\text{Lset}|$. Thus, the overall time contribution of the auxiliary function HomotopyExpansion is

$$O(|X| \cdot \log |\text{Star}|), \tag{4.5}$$

with a negligible extra space contribution.

ComputeDiscreteGradient From the analysis of each single auxiliary function contribution, it follows that the main time contributions are given by the ordering of the vertexes performed by ComputeIndexing, the level set splitting performed by SplitIndexLowerStar, and the classification of the cells performed by HomotopyExpansion. The global time computational cost is

$$O(|X_0| \log |X_0| + |X| \log |\text{Star}|). \tag{4.6}$$

The extra space required is given by the ordering of the vertexes performed by ComputeIndexing $O(|X_0|)$, and the level set splitting performed by SplitIndexLowerStar $O(|\text{Star}|)$. The global extra storage cost is

$$O(|X_0| + |\text{Star}|). \tag{4.7}$$

Case $O(|\text{Star}|) = O(|X|)$. It is possible that a single star has the same order of magnitude as the whole complex X. This happens, for instance, in the case of a cell complex X with a single cell of high dimension. In that case, we get an overall time complexity of $O(|X_0| \log |X_0| + |X| \log |X|)$, and an extra space required of $O(|X_0| + |X|)$.

Case $O(|\text{Star}|) = O(1)$. This happens when the number of cells incident to a single cell is bounded by a constant value which is negligible with respect to the parameters $|X_0|$ and $|X|$. In that case, we get a time complexity of $O(|X_0| \log |X_0| + |X|)$, and an extra space required of $O(|X_0|)$.

4.4.2 Discussion

We first discuss a key aspect of the algorithm ComputeDiscreteGradient relative to its runtime storage cost. Afterwards, we compare the time cost analysis provided to, first, the analysis of the algorithm presented in [4], we call Matching, and we review in Section 3.3, secondly, to the one-parameter case analogue of our algorithm introduced in [162], we cal ProcessLowerStars and we review in Section 3.2.

Runtime storage cost. Under our assumptions, the key aspect of the space complexity analysis of algorithm ComputeDiscreteGradient(X, f) is that, unless $O(|\,\mathrm{Star}\,|) = O(|X|)$, the runtime storage cost does not exceed the space required for representing the input and the output. The entities that are assumed to be stored off-line are: the simplicial complex X, the function f, the star of each vertex, the boundary of each cell, the discrete gradient V, and the Morse set M. The entities that need to be stored runtime are: the indexing I, the ordered lists Ord0 and Ord1, and the local values of filtering function $\tilde{f}$ over each index-based lower star. ComputeIndexing produces an indexing over the vertexes and the call of the function requires an extra storage cost which is proportional to the number of vertexes in X. So, the space required is negligible with respect to the space required by the representation of the whole complex X. The space required by the other entities is, in the worst-case, proportional to the number of cells in the star of a vertex. In this case also, the contribution is dominated by the contribution of the cell complex representation.

Comparison to the algorithm Matching in Section 3.3. The time complexity analysis provided for Matchingin [3] excludes the construction of the indexing J over the cells of the input complex which is directly taken as input. As reviewed in Section 3.3, Matching applies HomotopyExpansion to exactly the same level sets as ComputeDiscreteGradient does. In [3], authors express the overall cost differently from ours but the input-dependent worst-case time analysis of Formula (3.9) applies:

$$O(|X|\log|\,\mathrm{Star}\,|). \tag{4.8}$$

Nevertheless, there is a relevant time cost difference between the two algorithms which is relative to the different number of stars which need to be visited in order to apply HomotopyExpansion. Indeed, Matching visits a whole star through its auxiliary function ComputeLowerStar to produce a lower star, for each level set of $\tilde{f}$. Instead, in ComputeDiscreteGradient, a whole star is visited by function ComputeIndexLowerStar only for each vertex in the input complex. This means that, in our case, Formula (4.2) gives the global exact number of cells visited by ComputeIndexLowerStar. Instead, the number of cells visited by ComputeLowerStar in Matching depends on the input-dependent number $|P|$ of level sets, satisfying

$$|X_0| \leqslant |P| \leqslant |X|.$$

In the case of $|P| = |X_0|$, the two algorithms visit the same number of cells. In the case of $|P| = |X|$, it means that each cell belongs to a different level set. Since each j-cell has $\binom{j+1}{k+1}$

different k-dimensional faces, each j-cell belongs to exactly $\binom{j+1}{k+1}$ cell stars. In that case, the amount of visited cells is given by

$$\sum_{a \in X} |\operatorname{Star}(a)| = \sum_{j=0}^{d} |X_j| \sum_{k=0}^{j+1} \binom{j+1}{k+1}. \tag{4.9}$$

By comparison of Formula (4.9) to Formula (4.2), we get that, in both cases, the number of visited cells is proportional to the number of cells in the input complex. This justifies the same asymptotical complexity analysis. However, the linearity is satisfied under very different proportionality coefficients. Those coefficients increase the closer $|P|$ is to $|X|$ and the higher the dimension d is. We made this aspects explicit with practical tests discussed in Section 6.1.

From the storage cost point of view, the biggest difference between the two algorithms is given to the local nature of ComputeDiscreteGradient as opposed to the global nature of Matching. Indeed, at runtime and even by imposing $O(|\operatorname{Star}| << O(|X|))$, Matching might exceed the cost of the input and output representation since it needs to keep track of the classified cells all along the computations globally and it needs to implement a global queue with length $|X|$. This, not simply affects the runtime storage cost, but it also prevents a parallel implementation. Comparison tests under the same implementation are treated in Section 6.1.

Comparison to ProcessLowerStars **in Section 3.2.** The complexity analysis of ProcessLowerStars applies to the case when X is a cubical complex and the function f takes scalar-values. The algorithm results linear in the number of vertexes in X. We look at it as a special case of our analysis. Indeed, the restriction to scalar-valued functions, eliminates the need of introducing the indexing I for the vertexes since the role of the indexing is directly played by the values of the input function f. So, there is no need of considering the contribution of ComputeIndexing. Moreover, the level sets correspond to lower stars of vertexes rather than general cells. So, HomotopyExpansion is repeated $|X_0|$ times. Finally, in an embedded cubical complex, the number of cells incident to a vertex is constant for each vertex. So, the time cost of HomotopyExpansion is considered to be constant constant. Under the same bounds on the input assumptions, our complexity analysis of ComputeDiscreteGradient is coherent to that provided by author for ProcessLowerStars.

Final remarks on the complexity of ComputeDiscreteGradient. In Section 3.3, we remarked how the input-dependent worst-case time complexity of Matching in Formula (3.9) preserves, for vector-valued input functions with n components, the same worst-case asymptotical analysis as ProcessLowerStars, valid for the case $n = 1$, when applied to regular cell complexes with the intersection property rather than to cubical complexes. The analysis provided in Section 4.4.1 shows that the local nature of ComputeDiscreteGradient provides the same time complexity without being input-dependent. However, in fact, the parameter n acts indirectly on the average-case by increasing the number $|P|$ of level sets. The number $|P|$ affects the number of calls of

the auxiliary function HomotopyExpansion and the effective time contribution of the auxiliary function SplitIndexLowerStar. In this section, we also saw how additional bounds on the maximal number $|\text{Star}|$ of cells in the star of a single cell in the input complex make the global time linear. These bounds are satisfied by the case of cubical complexes in low dimensions and by the datasets we use to perform our tests in Chapter 6. From the point of view of the runtime storage cost, the local nature of ComputeDiscreteGradient requires, in practical cases an extra space smaller than the necessary footprint required for representing the input and the output. A non-negligible extra space is required only if $O(|\text{Star}|) = O(|X|)$. Instead, the global nature of the algorithm Matching requires an extra space of the same order as the representation of the input complex. Compared to the algorithm ProcessLowerStars, ComputeDiscreteGradient needs an extra space linear in the number of vertexes to sort the vertexes of the complex. This procedure is not needed in the case $n = 1$.

4.5 Correctness

In this section, we provide a formal proof of correctness for algorithm ComputeDiscreteGradient. We formalize the correctness statement as follows:

> "The vector field returned by algorithm ComputeDiscreteGradient with input (X, f) is a discrete gradient field V and the corresponding Morse set M is compatible with the filtered complex $\mathcal{X}^{\tilde{f}}$."

As already mentioned, the discrete gradient retrieved by the algorithm ComputeDiscreteGradient is independently given by the outputs of the auxiliary function HomotopyExpansion (for details, see Section 3.1). Each call of the auxiliary function HomotopyExpansion runs over a single part in the partition into level sets according to the input function. The strategy to prove correctness consists in showing equivalence to the Matching algorithm reviewed in Section 3.3 and introduced in [4], fully proved to be correct in [3] and described in Section 3.3.

Observe that the algorithm Matching is introduced by authors to act over simplicial complexes (Definition 1.1.1) whereas the algorithm ComputeDiscreteGradient acts, more generally, on regular cell complexes with intersection property (Definition 3.1.1). However, the correctness for the algorithm Matching easily extends to regular cell complexes with intersection property. Indeed by Proposition 3.3.2, the algorithm Matching outputs a list of discrete vectors found independently over a partition of the input cell complex. Discrete vectors are locally found by the auxiliary function HomotopyExpansion and, Proposition 5 in [162] implies that, over each lower star, the output is a valid discrete gradient. The union of all the independent discrete gradients returned by the auxiliary function HomotopyExpansion form a discrete gradient by Theorem 3.8 in [3]. The Theorem is proved for simplicial complexes, but the only assumption related to simplicial complexes used by the authors to prove the result is the intersection property. Hence,

the proof is easy to adapt to regular cell complexes with intersection property since, by definition, intersecting cells determines a single cell. The compatibility with the input filtered complex is guaranteed by Theorem 3.7 in [3].

First, we adapt the correctness result in [3] for the algorithm Matching in order to highlight the partition the algorithms acts on, and to show that the partition is the same for the algorithm ComputeDiscreteGradient. This allows us to say that the two algorithms act on the same partition of the input complex. Finally, we show precisely the conditions guaranteeing the same outputs for the two algorithms. This ensures full equivalence between the two algorithms and then correctness for the algorithm ComputeDiscreteGradient.

4.5.1 Corresponding partitions

In order to find a discrete gradient compatible with a filtered complex $\mathcal{X}^{\tilde{f}}$ algorithm Matching requires in input three objects $(X, \tilde{f}, J)$, where the additional part with respect to ComputeDiscreteGradient is the indexing J over X. The indexing J is required to satisfy the following condition of compatibility with the coface relation and the filtering function values:

$$\forall a \neq b \in X, a \ll b \text{ or } \tilde{f}(a) \precnsim \tilde{f}(b) \quad \Rightarrow \quad J(a) < J(b). \tag{4.10}$$

We refer to the pseudocode of Matching reported in Algorithm 3. The algorithm Matching processes all cells in X in a for-cycle. The indexing J defines the order for visiting the cells of X. For each cell a, if the cell is found unclassified (line 4 in Algorithm 3), the algorithm extracts the *lower star* of the cell $\mathrm{Low}_f(a)$

$$\mathrm{Low}_f(a) := \{c \in X \mid c \gg a \wedge \tilde{f}(c) \preceq \tilde{f}(a)\}.$$

Then, the algorithm calls $\mathrm{HomotopyExpansion}(X, \mathrm{Low}_f(a), J)$. By Proposition 3.6 in [3], discrete vectors and critical cells are found as part of a single lower star of a cell. The union of all lower stars of cells in X forms a covering of X but not necessarily a partition. The global tracking of cells performed by the array classified allows to skip lower stars of cells already considered as part of the lower star of another cell. This prevents to classify a single cell more than once. At line 4, the if-condition ensures to call HomotopyExpansion only for the lower stars of cells found unclassified. All along this section, we indicate by P the set of the cells found unclassified at line 4 by Matching. Then, Proposition 3.6 in [3] directly implies the following property of the lower stars actually processed by Matching.

4.5.1 Lemma. Let $(X, \tilde{f}, J)$ be a suitable input for Matching. Then, the lower stars $\mathrm{Low}_f(a)$ for $a \in P$ form a partition of X.

Hence, both algorithms build their output by running HomotopyExpansion on a partition of the input complex X:

- Matching finds the discrete gradient independently over each lower star $\mathrm{Low}_f(a)$ with $a \in P$,
- ComputeDiscreteGradient finds the discrete gradient independently over each level set $\mathrm{Lset} \in K(v)$ with $v \in X_0$.

We show in the following lemma that the two partitions are the same.

4.5.2 Lemma. Let $f : X_0 \longrightarrow \mathbb{R}^n$ be a component-wise injective function and P the set of cells in X found unclassified by the algorithm Matching at line 4 in Algorithm 3. Then, for any $a \in P$, the lower star $\mathrm{Low}_f(a)$ is a level set under $\tilde{f}$.

Proof. Let $(X, \tilde{f}, J)$ be a suitable input for the Matching algorithm. Let a be a cell in P, that is a is found unclassified at line 4 of the Matching Algorithm 3. This means that a has not been processed as part of the lower star of some other cell $c \in X$ with $J(c) < J(a)$. Let Lset be the level set under $\tilde{f}$ containing a. By Proposition 3.3.1, there exists a cell $a' \in \mathrm{Lset}$ such that $\mathrm{Lset} = \mathrm{Low}_f(a')$. By input assumptions on J, all cells $b \neq a'$ in $\mathrm{Low}_f(a')$ satisfy $J(b) > J(a')$. That is to say that b is found classified at line 4 in the Algorithm 3 or, equivalently, $b \notin P$. So, the only possible cell in the level set Lset belonging to P is a' which implies $a' = a$. □

4.5.2 ComputeDiscreteGradient and Matching are equivalent

4.5.3 Proposition. For any input (X, f) for ComputeDiscreteGradient, there exists a suitable input indexing $J : X \longrightarrow \mathbb{R}$ for Matching such that the output of $\mathrm{Matching}(X, \tilde{f}, J)$ equals that of $\mathrm{ComputeDiscreteGradient}(X, f)$.

Proof. By Lemma 4.5.2 along with Proposition 3.3.1, the two algorithms run the auxiliary function HomotopyExpansion over the level sets Lset under the function $\tilde{f}$. For ComputeDiscreteGradient, the global output is independently given by the outputs of each call of $\mathrm{HomotopyExpansion}(X, \mathrm{Lset}, i)$. From input (X, f) for ComputeDiscreteGradient, we want to construct a suitable indexing J over all the cells in X for the input of Matching such that each call of $\mathrm{HomotopyExpansion}(X, \mathrm{Lset}, J)$ gives the same output. Let I be the well-extensible indexing on vertexes returned by $\mathrm{ComputeDiscreteGradient}(X, f)$. We know from Proposition 3.3.1 that, any level set Lset under $\tilde{f}$ satisfies $\mathrm{Lset} = \mathrm{Low}_f(a)$ for some unique $a \in \mathrm{Lset}$. Let P be as in Lemma 4.5.2. By Lemma 4.5.2, we know that $a \in P$. For each $a \in P$, we indicate by J_a the inclusion-compatible indexing $i(\leqslant_{\mathrm{Lex}})$ over Lset providing the order in the ordered lists Ord0, Ord1 in $\mathrm{HomotopyExpansion}(X, \mathrm{Low}_f(a), i(\leqslant_{\mathrm{Lex}}))$ Let g be any injective function $P \xrightarrow{g} \mathbb{R}$ which is compatible with $\tilde{f}$, i.e., $\tilde{f}(a) \preceq \tilde{f}(b)$ implies $g(a) \leqslant g(b)$ (obtained, for example, by topological sorting). Clearly, the level sets under $\tilde{f}$ form a partition of X. Thus, by Lemma 4.5.2, it follows

that for every cell $b \in X$ there exists a unique cell $a_b \in P$ such that $b \in \mathrm{Low}_f(a_b)$. This implies that we can extend g to a map $G : X \longrightarrow \mathbb{R}$ still compatible to $\tilde{f}$ by setting $G(b) := g(a_b)$. Thus, for any simplex b, we get associated a pair of real numbers $GJ(b) = (G(b), J_{a_b}(b))$. Choose $J : X \longrightarrow \mathbb{R}$ to be an injective map imposing a total order equivalent to the lexicographic order over pairs of the form $GJ(b)$.

$$GJ(a) \leqslant_{\mathrm{Lex}} GJ(b) \quad \Leftrightarrow \quad J(a) \leqslant J(b).$$

We need to show that J satisfies the input requirements for algorithm Matching. In other words, J has to satisfy:

$$\forall a \neq b \in X, a \ll b \text{ or } \tilde{f}(a) \lesssim \tilde{f}(b) \quad \Rightarrow \quad J(a) < J(b). \tag{4.11}$$

By definition of $\tilde{f}$, the relation $a \ll b$ with $a \neq b$ implies $\tilde{f}(a) \leq \tilde{f}(b)$. By compatibility of G with $\tilde{f}$, it follows that $\tilde{f}(a) \leq \tilde{f}(b)$ implies $G(a) \leqslant G(b)$. In the case $G(a) < G(b)$, by equivalence of J with the lexicographic order over GJ, we get $J(a) < J(b)$. In the case $G(a) = G(b)$, by injectivity of g, it follows that a, b belong to the same lower star $\mathrm{Low}_f(\bar{a})$, where $\bar{a} \in P$. Being $J_{\bar{a}}$ compatible with the incidence relation $\ll$, we get $J(a) < J(b)$. Otherwise, suppose that $\tilde{f}(a) \lesssim \tilde{f}(b)$. This implies necessarily $G(a) \leqslant G(b)$ and, thus, $J(a) < J(b)$. By Lemma 4.5.2, the two algorithms call HomotopyExpansion with the same input. Hence, they produce the same output. □

As a corollary, we get the correctness of ComputeDiscreteGradient.

4.5.4 Corollary. Algorithm ComputeDiscreteGradient with input (X, f) returns a discrete gradient V compatible with the filtered complex $\mathcal{X}^{\tilde{f}}$.

Proof. Each input (X, f) determines the filtered complex $\mathcal{X}^{\tilde{f}}$. By Proposition 4.5.3, the discrete gradient V is the same as the one retrieved by Matching with a suitable input representing the same filtered complex $\mathcal{X}^{\tilde{f}}$. By Proposition 3.6 in [3], Algorithm Matching returns a discrete gradient V with its relative Morse set M which is compatible with the filtered complex $\mathcal{X}^{\tilde{f}}$. □

The following proposition completes the equivalence between the algorithms ComputeDiscreteGradient and Matching. In particular, it states that all possible gradients retrieved by Matching can be obtained by the divide-and-conquer strategy of ComputeDiscreteGradient.

4.5.5 Proposition. Let $(X, \tilde{f}, J)$ be a suitable input for Matching. Let $f : X_0 \longrightarrow \mathbb{R}^n$ be the restriction of $\tilde{f}$ to the vertexes. Then, the output of $\mathrm{Matching}(X, \tilde{f}, J)$ equals that of $\mathrm{ComputeDiscreteGradient}(X, f)$, provided that, for each level set Lset under $\tilde{f}$, the function HomotopyExpansion is given (X, Lset, J) as input.

Proof. Analogously to the proof of Proposition 4.5.3, by Lemma 4.5.2 along with Proposition 3.3.1, the two algorithms run the auxiliary function HomotopyExpansion on the level sets Lset of the function $\tilde{f}$. The global output is independently given by the output of each call

of HomotopyExpansion(X, Lset, i), with i a general indexing over X compatible with the coface relations among cells. The output depends on i in assigning the order to the ordered lists Ord0, Ord1. In general, ComputeDiscreteGradient sets $i = i(\leqslant_{\text{Lex}})$ where $\leqslant_{\text{Lex}}$ follows from the well-extensible indexing returned by ComputeIndexing. Instead, Matching sets $i = J$. Hence, ComputeDiscreteGradient finds exactly the same discrete gradient as Matching, provided that (X, Lset, J) is given to HomotopyExpansion as input. □

This last statement provides not simply correctness, but it also states that ComputeDiscreteGradient is as general as Matching. This means that the introduction of the well-extensible indexing over the vertexes which is needed in algorithm ComputeDiscreteGradient can be performed for every possible input.

4.6 From a discrete gradient to the Morse complex: BoundaryMaps

In this section, we describe the algorithm BoundaryMaps introduced in [93] to retrieve the Morse complex associated to a discrete gradient either on a simplicial complex or on a cubical complex. Hence, it is a suitable post-processing to the algorithms in Section 3.2, Section 3.3 and ComputeDiscreteGradient. The algorithm is applied as a post-processing of ComputeDiscreteGradient in the experimental tests in Section 6.2, Section 6.3, and Section 6.4. First, we make the input assumptions and the output properties explicit. Then, we review the algorithm BoundaryMaps in the simplicial case. Finally, we adapt the same algorithm to the cubical case.

4.6.1 Input and output formalization

As a first input, the algorithm requires a cell complex X which is either a simplicial (Definition 1.1.1), or a cubical complex (Definition 1.1.8). As a second input, the algorithm requires a discrete gradient V on X (Definition 1.4.5). As a last input, the algorithm requires a critical cell t, that is t belongs to the Morse set M relative to the discrete gradient V, i.e., $M = M(V)$ (Definition 1.4.7).

To summarize, BoundaryMaps receives in input:

- X : either a simplicial or a cubical complex;
- V : a discrete gradient on X;
- t : a critical cell in the Morse set relative to V.

As regards the output retrieved by BoundaryMaps, we refer to Definition 1.2.1 for the notion boundary map $\partial(t)$ in a chain complex, and we refer to Formula (1.7) for the definition of incidence Morse function $\kappa_M : M \times M \longrightarrow \mathbb{Z}_2$. Then, BoundaryMaps$(X,V,t)$ returns:

- $B(t)$: the boundary map $\partial(a)$ of $s \in M$. The boundary map is represented by the list $B(t)$ where each element is a value identifying a cell s. Since we only consider incidence functions with $\mathbb{Z}_2$ coefficients, if the number of occurrences of values identifying s is even, then $\kappa_M(t,s) = 0$, otherwise $\kappa_M(t,s) = 1$.

By Definition 1.4.8, the list of boundary maps $B(t)$ for any $t \in M$ defines the Morse complex $(M(V), \kappa_M)$ induced by the discrete gradient V on X.

As a remark on the output retrieved by BoundaryMaps, we recall from Definition 1.6.3 that, if the input discrete gradient is compatible with a (multi)filtered complex $\mathcal{X}$, then the output Morse complex gets induced a filtered structure $\mathcal{M}$ compatible with $\mathcal{X}$. This makes BoundaryMaps a valid post-processing to the algorithms in Section 3.2, Section 3.3 and ComputeDiscreteGradient to get the corresponding Morse complex which preserves the compatibility with a filtration.

4.6.2 Description in the simplicial case

In this section, we focus on the case of X being a simplicial complex to describe the algorithm BoundaryMaps as done in [93]. The algorithm is implemented for the experimental results in Chapter 6. For the input and the output of the algorithm, we take the notation of the previous section. The pseudocode of BoundaryMaps is reported in Algorithm 5. First, we make the representation of entities in the simplicial case explicit. Then, we describe the auxiliary functions called by BoundaryMaps. Afterwards, we describe the procedure of the algorithm. Finally, we provide some computational remarks on the simplicial case.

For representing the simplicial complex X, we assume a representation such that the top-simplices incident to a given vertex can be retrieved in linear time with respect to the number of top-simplices incident to the vertex. Moreover, we assume to have constant access to the vertexes of a cell. This property is met by representations such as the adjacency-based IA* data structure [30] and the incidence-based implementation for triangle meshes described in Section 6.1.2.

For representing the discrete gradient, our description assumes the representation introduced in [86] for triangular meshes and in [193] for tethrahedral meshes. The discrete gradient V is represented by a bit-array as an attribute for each top-simplex t in X. In the bit-array, each entry defines a possible discrete vector between faces of t. Discrete vectors are given by valued-1 entries in the array.

The algorithm BoundaryMaps calls the following auxiliary functions:

Algorithm 5 BoundaryMaps(X,V,t)

Input: X, d-dimensional regular cell complex
Input: V, discrete gradient on X
Input: t, critical cell
Output: $B(t)$, boundary maps as collections of arcs

```
1:  Q ← ∅
2:  Q.enqueue(t)                          # Q is an empty queue
3:  while Q ≠ ∅ do
4:     t_0 ← Q.dequeue()
5:     for s_1 ∈getFacets(t_0, X) do      # s_1 is a facet of t_1 in X
6:        if isPaired(s_1, V) then        # s_1 is not critical
7:           t_1 ←getPairComponent(s_1, V)   # (s_1, t_1) ∈ V
8:           Q.enqueue(t_1)
9:        else                            # s_1 is critical
10:          add(B(t), t_0, s_1)          # the value identifying s_1 is appended to B(t)
11:       end if
12:    end for
13: end while
14: return B(t)
```

- $\mathrm{getFacets}(t,X)$ returns a list of facets of the simplex t in X. This is done in linear time in the dimension of the simplex t, since each facets is obtained by removing a vertex from those defining t;

- $\mathrm{isPaired}(s,V)$ returns the value *True* if s belongs to a discrete vector (s,t) in V, and the value *False* otherwise. In the former case it also returns the paired cell. This is done by examining all the top simplices in the star of a vertex w of s. For each top-simplex its bit-vector is visited limitedly to the entries defining suitable pairs (s,t). The operation requires $O(|t_w|)$, where t_w denotes the set of top simplices incident in a vertex w of a.

- $\mathrm{getPairComponent}(s,V)$ is used to return, in the pseudocode, the cofacet t of s returned by $\mathrm{isPair}(s,V)$.

Starting from a critical cell t of dimension k, a breadth-first traversal is performed by navigating from t to its adjacent k-cells passing through their shared $(k-1)$-cells (by intersection property, adjacent k-cells share one, and only one, facet). The breadth-first traversal is supported by a queue Q. Given a k-cell t_0 extracted from the queue Q (row 8), we examine all of its facets in X, returned by the auxiliary function $\mathrm{getFacets}$ (row 11). For each $(k-1)$-cell s_1, if s_1 is part of a discrete vector together with a k-cell t_1 returned by function isPair (row 12), t_1 is added to the queue (rows 15 and 16). If s_1 is a critical cell, then the value identifying s_1 is appended to $B(t)$. When the

while-loop terminates, the list $B(t)$ is returned.

Algorithm 5 is executed for a critical k-simplex t in the Morse set $M(V)$ relative to a discrete gradient V. A cell t_0 is popped out from Q up to k times, with k the number of facets of t_0. For each facet s_1 of t_0 we check whether it is paired or not which takes, by assumption at most time $O(|t_w|)$. Then, we can conclude that each iteration of the while loop takes $O(k \cdot |t|)$, where t is the maximal-sized set of top-simplices incident in w. Hence, the algorithm has $O(q \cdot k \cdot |t|)$ worst-case time complexity, where q is the number (counted with multiplicity) of k-simplices of X inserted in the queue Q. In practical cases, time complexity is dominated by the factor q. Concerning the retrieval of the boundary map for all critical cells, we can build degenerate cases where each critical cell is connected, through a V-path, to all the cells of the same dimension in X. This raises the complexity to be cubic in the number of cells. In practice, each cell belongs to the descending path of a limited number of critical cells (possibly zero). In Section 6.4.3, we see the measured timings for BoundaryMaps applied to some entire Morse sets of large size.

4.6.3 Description in the cubical case

In this section, we focus on the case of X being a cubical complex embedded in a d-dimensional space to describe the algorithm BoundaryMaps. The algorithm is implemented for the experimental results in Section 6.4. Our description adapts the presentation of the algorithm in [93] to the cubical case. For the input and the output of the algorithm, we take the notation of the previous section. The procedure and pseudocode of BoundaryMaps in the cubical case is the same as for the simplicial case and it is reported in Algorithm 5. First, we make the representation of entities in the cubical case explicit. Then, we describe dissimilarities with respect to the simplicial case of the auxiliary functions called by BoundaryMaps. Afterwards, we describe the procedure of the algorithm. Finally, we provide some computational remarks on the cubical case.

A cubical complex embedded in a d-dimensional space is represented as a regular grid. A regular grid is defined by a graph $G = (N, A)$ where N is the set of cells forming the regular grid and A is the set of incidence relations between the cells in N. Representing a regular grid is a much easier task than representing a simplicial complex. The regular distribution and connectivity of the cells in G makes it possible to encode the topology of the complex implicitly. By enumerating the cells in N, we can extract any relation in A using index calculations without any overhead. Facets and cofacets of a given cell are retrieved in linear time with respect to the number of facets or cofacets, respectively. The number of facets of a k-cell is at most $2k$, and the number of cofacets is at most $2k$.

In the case of a regular grid, the discrete gradient is encoded by assigning a Boolean value to each arc α in A, where the two cells connected by α are paired in V. Then, the discrete gradient is encoded as an array of bits of length $|A|$.

The algorithm BoundaryMaps calls the same auxiliary functions as in the simplicial case with

the following differences:

- $\mathrm{getFacets}(t, X)$ is performed by index calculations in linear time in the number of facets of t;
- $\mathrm{isPaired}(s, V)$ is performed by index calculation in linear time in the number of cofacets of s.

From the computational point of view, we focus on the dissimilarities with the simplicial case. A cell t_0 is popped out from Q up to $2k$ times equal to the number of facets of t rather than k, but this provides a linear contribution in the same way. For each facet s_1 of t_0 we check whether it is paired or not which takes at most time $O(k)$ instead of depending on the number of top-simplices incident in a vertex. Then, we can conclude that each iteration of the while loop takes $O(k^2)$. Hence, the algorithm has $O(q \cdot k^2)$ worst-case time complexity, where q is the number (counted with multiplicity) of k-cells of X inserted in the queue Q. Again, in practical cases, time complexity is dominated by the factor q. Differently from the simplicial case, parameter k can be considered constant. Degenerate case with $O(q) = O(|X_k|)$ exist also in the cubical case and parameter q dominates the final time cost. In Section 6.4.3, we see the measured timings for $\mathrm{BoundaryMaps}$ applied to some entire Morse sets of regular grids of large size and notice the advantage of the cubical case over the simplicial one.

5 Optimality by relative homology

As exposed in Section 1.4, a notion of optimality is commonly associated with a complex through a discrete Morse function defined on it. Intuitively, in this case, optimality means that each critical cell corresponds to a change in topology when moving from one filtration step, induced by sublevel sets with respect to the Morse function, to the other. In this chapter, we deal with a notion of optimality that applies to a complex already equipped with a filtration. In doing this, we express optimality as a property of a discrete gradient compatible with a filtered complex instead of a discrete Morse function.

The chapter is structured as follows. First, we discuss the case of one-parameter filtered complexes. In particular, we formalize, into more general terms, the optimality notion expressed in [162] for the specific case of regular grids under a scalar-valued filtering function. Under this formalization of optimality, the algorithm proposed in [162], and generalized by our algorithm described in Chapter 4, retrieves an optimal discrete gradient, for cubical complexes realized in a 3D space, an so, the algorithm itself is called optimal. We discuss why this one-parameter notion of optimality for a discrete gradient does not generalize to the multiparameter case. As a first contribution, we translate the one-parameter notion of optimal discrete gradient equivalently in terms of relative homology. Afterwards, we move to the multiparameter case and provide our second contribution about optimality. This consists in generalizing the optimality definition via relative homology to the multiparameter case. Successively, as a third contribution, we prove that a Morse complex compatible with a multiparameter filtered complex induced by sublevel sets of a filtering function preserves the relative homologies involved in the optimality definition. This fact is used in the final part to guarantee that the one-parameter optimality of the algorithm in [162] holding for 3D cubical complexes is preserved in the multiparameter case by our algorithm described in Chapter 4. Finally, we show an additional case where optimality of the algorithm described in Chapter 4 holds, that is for abstract 2-dimensional simplicial complexes.

All along this chapter, we assume to be given a Lefschetz complex with intersection property X, a filtration $\mathcal{X}$ of X (either one or multiparameter), a discrete gradient V compatible with $\mathcal{X}$, its Morse set $M = M(V)$, and the corresponding filtered Morse complex $\mathcal{M}$.

5.1 One-parameter optimality in terms of relative homology

In this section, we focus on the case of one-parameter filtrations. The purpose is that of formalizing a notion of optimality for V with respect to $\mathcal{X}$. We take inspiration from the property proved for

the algorithm ProcessLowerStars [162] reviewed in this thesis in Section 2.3.

5.1.1 Theorem (Theorem 11 in [162]). *Given a grayscale digital image, $g : D \longrightarrow \mathbb{R}$ and its associated cubical complex Q, let x_u be the u^{th} voxel in the ordering of the greyscale values, Q^u be the sublevel set for the value $g(x_u)$, and $\mathrm{Low}_g(x_u)$ the lower star of x_u. Then, for each critical k-cell $\gamma \in \mathrm{Low}_g(x_u)$ identified by the algorithm* ProcessLowerStars *either*

i) γ *is a* positive cell, *i.e.,* γ *creates a new k-cycle in $H_k(Q^u)$, or*

ii) γ *is a* negative cell, *i.e.,* γ *fills in a $(k-1)$-cycle from $H_{k-1}(Q^{u-1})$.*

In order to look at the property in Theorem 5.1.1, into more general terms, we consider a filtered complex $\mathcal{X}$, we take two successive filtration grades $u-1$ and $u \in \mathbb{Z}$ and the corresponding inclusion-induced maps of homology

$$\imath_k^{u-1,u} : H_k(X^{u-1}) \longrightarrow H_k(X^u),$$

for all integers k. With respect to these family of inclusion-induced maps, we observe two facts:

a) a *positive* k-cell creates a non-trivial homology class not existing before, so it generates a non-trivial homology class in $\operatorname{coker} \imath_k^{u-1,u}$;

b) a *negative* k-cell either fill in an independent loop or make two independent loops become dependent. So, it generates a non-trivial homology class in $\ker \imath_{k-1}^{u-1,u}$.

All along this section, we denote by $m_k(u)$ the number of new critical k-cells in $M = M(V)$ at grade $u \in \mathbb{Z}$, that is

$$m_k(u) := |M_k^u \backslash M_k^{u-1}|. \tag{5.1}$$

5.1.2 Definition (One-parameter optimality). A discrete gradient V compatible with a one-parameter filtered complex $\mathcal{X}$ is *optimal with respect to* $\mathcal{X}$ if, for all $u \in \mathbb{Z}$ and integers k,

$$m_k(u) = \dim \operatorname{coker} \imath_k^{u-1,u} + \dim \ker \imath_{k-1}^{u-1,u}.$$

We can express the property for the algorithm ProcessLowerStars expressed in Theorem 5.1.1 as the property of finding a discrete gradient satisfying Definition 5.1.6. In order to show the coherence between Definition 5.1.2 and Theorem 5.1.1, we notice that we can totally order all identifications of a pair or a critical cell performed by ProcessLowerStars in a single $\mathrm{Low}_g(x_u)$. Let $t(u)$ be the number of identifications occurred in $\mathrm{Low}_g(x_u)$. For each l from 1 to $t(u)$, define $Q(l)$ as the cubical subcomplex of Q^u with cells identified either as critical or in a discrete vector

when, or before, the l^{th} identification is performed by ProcessLowerStars. This allows us to consider the filtered complex

$$Q^{u-1} =: Q(0) \subseteq \cdots \subseteq Q(t(u)) := Q^u. \tag{5.2}$$

We recall the following useful fact from Section 4 in [162].

(*) All discrete vectors preserve homology along the filtration (5.2): the homology changes from $Q(0)$ to $Q(t(u))$ only occur when a critical cell is added.

The following propositions ensure that, we can express the property for the algorithm ProcessLowerStars expressed in Theorem 5.1.1 as the property of finding a discrete gradient satisfying Definition 5.1.6.

5.1.3 Proposition. Under the assumptions of Theorem 5.1.1. If one denotes by $[c_\gamma]$ the homology class in $H_k(Q^u)$ created by the positive k-cell γ in $\mathrm{Low}_g(x_u)$, then the set

$$P_k^u := \{[c_\gamma] \mid \gamma \text{ positive cell in } \mathrm{Low}_g(x_u)\}$$

is a basis for $\operatorname{coker} \imath_k^{u-1,u}$.

Proof. Elements in P_k^u are non-trivial classes in $\operatorname{coker} \imath_k^{u-1,u}$ by fact a) and are linearly independent by definition. The set P_k^u is a set of generators for $\operatorname{coker} \imath_k^{u-1,u}$ by fact (*). □

5.1.4 Proposition. Under the assumptions of Theorem 5.1.1. If one denotes by $[c_\gamma]$ the homology class in $H_{k-1}(Q^{u-1})$ destroyed by the negative k-cell γ in $\mathrm{Low}_g(x_u)$, then the set

$$N_{k-1}^u := \{[c_\gamma] \mid \gamma \text{ negative cell in } \mathrm{Low}_g(x_u)\}$$

is a basis for $\ker \imath_{k-1}^{u-1,u}$.

Proof. Elements in N_{k-1}^u are non-trivial classes in $\ker \imath_{k-1}^{u-1,u}$ by fact b) and are linearly independent by definition. The set N_{k-1}^u is a set of generators for $\ker \imath_{k-1}^{u-1,u}$ by fact (*). □

The key point in the formalization of the optimality notion in Definition 5.1.2 is to relate the number of critical cells to precise algebraic terms. As an example, in Figure 5.1, we see how the just-provided definition of optimal discrete gradient depends on the considered filtered complex. Indeed, a discrete gradient might be, with respect to a given filtration 5.1(a), optimal, and, with respect to another filtration 5.1(b), non-optimal.

The translation of optimality into the multiparameter case is not straightforward as explained by the following remark.

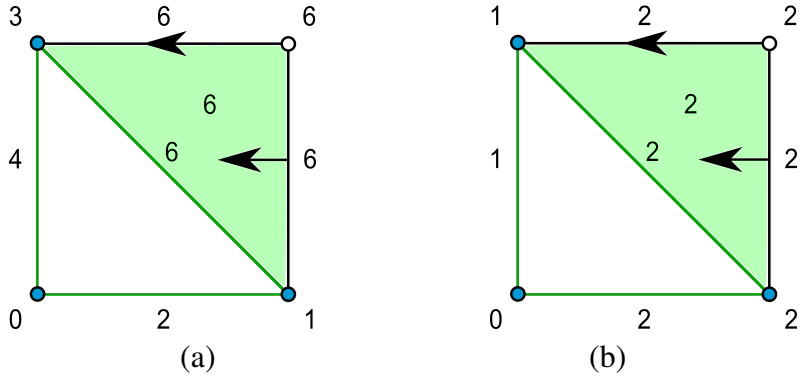


Figure 5.1: in (a) an optimal discrete gradient with respect to the filtered complex obtained by sublevel sets under the filtering function values depicted for each cell. In (b), the same discrete gradient with a different filtering function.

5.1.5 Remark. *The fact that multiparameter filtration are defined using a partial order among grades instead of a total order makes two peculiar facts possible. When considering the homology of the union of two steps with incomparable grades, on the one hand, a new homology class can be created even if no new cell is added. On the other hand, at a single step, a new cell can destroy some homology class existing only in the union of filtration steps and not in a single filtration step. As an example, see Figure 5.2. Thus, the idea of positive and negative cells might be ineffective in the multiparameter case.*

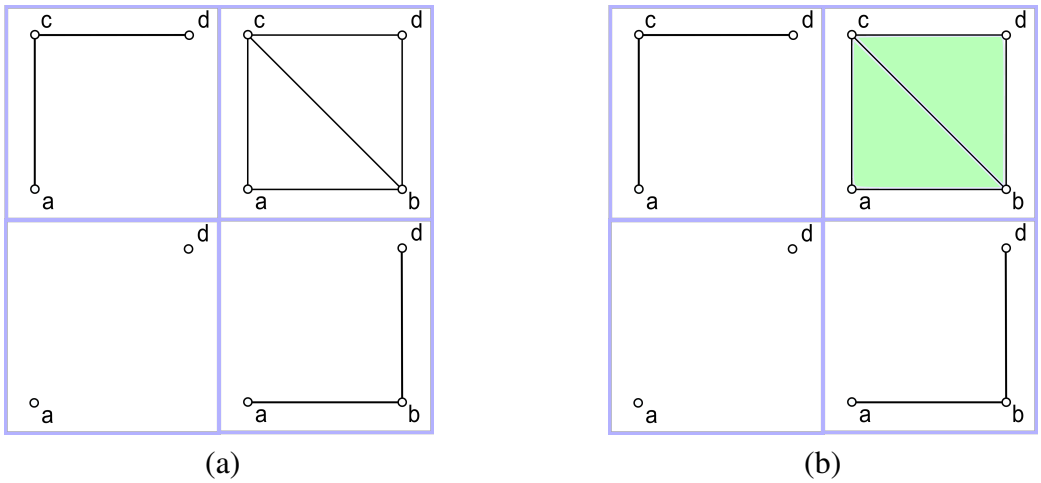


Figure 5.2: in the top-right corner of (a), we have a new homology class with no associated critical cell. In the top-right corner of (b), we have a critical cell killing a homology class obtained by union of incomparable steps and never-existing along the filtration.

This means that the simple idea of capturing homology changes between single filtration steps is

not sufficient to establish whether a critical cell is necessary or not. This prompts us in finding an equivalent way of expressing the same property in a generalizable way. To this purpose, we exploit the following well-known consequence of long exact sequences of relative pairs, holding for every homology degree k and every filtration grade $u \in \mathbb{Z}$:

$$\operatorname{coker} \iota_k^{u-1,u} \oplus \ker \iota_{k-1}^{u-1,u} = H_k(X^u, X^{u-1}). \tag{5.3}$$

This leads us to the restate Definition 5.1.2 equivalently as follows.

5.1.6 Definition (Optimality for one-filtrations)**.** A discrete gradient V compatible with a filtered complex $\mathcal{X}$ is *optimal with respect to* $\mathcal{X}$ if, for all $u \in \mathbb{Z}$ and integers k,

$$m_k(u) = \dim H_k(X^u, X^{u-1}).$$

This last definition captures in a single algebraic term the two necessary characteristics for critical cells to reflect the filtered structure of a one-parameter filtration, that is being either positive or negative. In the next section, we present a generalization of Definition 5.1.6 to the multiparameter case.

We close this section by giving an example to clarify that not all filtered complex admit an optimal discrete gradient.

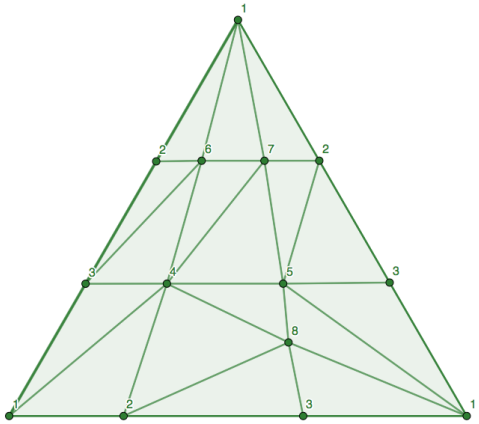
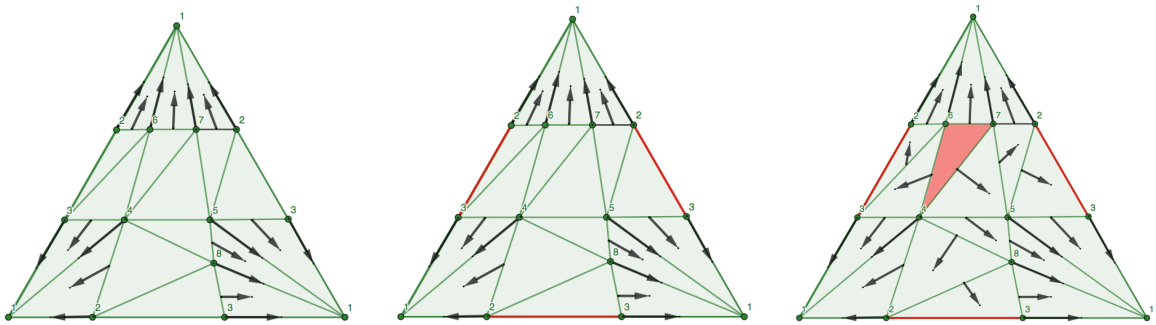


Figure 5.3: a triangulation of a duncehat filtered by vertex indexing.

5.1.7 Example. *Consider the one-filtration of a cone over the triangulation of the* duncehat *showed in Figure 5.3. We imagine a ninth vertex, indexed by 9, connected to all vertexes of the dunce hat, with all the additional tetrahedra, triangles and edges built on top of the simplices*

of the dunce hat. The considered filtering function is the vertex-induced one. We focus on the filtration grade $u = 9$, whose corresponding filtration step is the lower star of the vertex 9. We know the dunce hat to be homotopy equivalent to a point, so we deduce that $H_k(X^u, X^{u-1})$ has dimension 0. Hence, an optimal gradient should have no critical cells in the lower star of vertex 9. However, since the dunce hat does not admit a perfect Morse function *(see Section 1.4) and all possible discrete vectors in the lower star of 9 but a unique discrete vector containing vertex 9 are in one-to-one correspondence with discrete vectors on the dunce hat, we have provided an example of one-filtration with no possible optimal discrete gradient.*

The example above is compatible with our optimality results for algorithm ComputeDiscreteGradient proved later on in Section 5.4. Indeed, the simplicial complex proposed has dimension 3. More specifically, we think that the crucial fact is that the simplicial complex proposed admits lower stars not embeddable in $\mathbb{R}^3$. In Figure 5.4, we see how the algorithm ComputeDiscreteGradient necessarily finds more critical cells than what expected from optimality in Definition 5.1.6. First, the vertex 9 is paired to edge $[9, 1]$. This latter corresponding to point 1 in



(a) (b) (c)

Figure 5.4: ComputeDiscreteGradient acting on the lower star of vertex 9 in the cone of Figure 5.3.

the base representation in Figure 5.4(a)-(b)-(c). In Figure 5.4(a), all discrete vectors found by the algorithm with no need of classifying any critical cell. In Figure 5.4(b), the triangle corresponding to the red edge is classified as critical since it belongs to Ord0 but its possible pairings with higher-dimensional cells have not yet entered Ord1. In Figure 5.4(c), all cells in the cone are classified. The last tetrahedron, corresponding to the red triangle in (c) is classified critical and it balances the homology class created by the red edge.

5.2 Multiparameter optimality

In this section, we generalize to multiparameter filtrations the notion of optimality by relative homology introduced in the previous section in Definition 5.1.6. The idea consists in replacing

the single filtration grade $u-1$ preceding u in Definition 5.1.6 by all filtration grades preceding u when it belongs to $\mathbb{Z}^n$ with a general integer $n \geqslant 1$. Thus, we consider the union of all filtration steps included in X^u and we call that union of cell complexes the *union preceding* u. The union preceding u can be expressed as a finite union by taking only those steps preceding X^u which are maximal. Those, if multiple, are necessarily incomparable. All along this chapter, for i from 1 to n, we set grades $u^i = u - e^i$ with $e^i \in \mathbb{Z}^n$ such that the i^{th} entry equal to 1 and all other entries equal to 0. The definition of Formula (5.1) given in the previous section for $m_k(u)$ is generalized by the following one

$$m_k(u) := |M_k^u \backslash \bigcup_{i=1}^{n} M_k^{u^i}|.$$

The following definition generalizes Definition 5.1.6 to the multiparameter case

5.2.1 Definition (Optimality)**.** For any integer n, a discrete gradient V compatible with a multiparameter filtered complex $\mathcal{X}$ is *optimal with respect to* $\mathcal{X}$ if, for all $u \in \mathbb{Z}^n$ and integers k,

$$m_k(u) = \dim H_k\left(X^u, X^{u^1} \cup \cdots \cup X^{u^n}\right).$$

5.2.2 Remark. *Differently from the one-parameter case, the union preceding a single grade does not necessarily correspond to an actual step in the filtration. See, for instance, Figure 5.2.*

The given definition is clearly a generalization of Definition 5.1.6. However, Remark 5.2.2 highlights a crucial difference which needs to be captured by our generalized definition. Indeed, the relative homology considered in Definition 5.2.1 is not related anymore to the inclusion-induced maps $\imath_k$ belonging to the filtration. Instead, the analogue of Formula (5.3) still holds but for the following family of inclusion-induced maps:

$$j_k^u : H_k\left(X^{u^1} \cup \cdots \cup X^{u^n}\right) \longrightarrow H_k(X^u).$$

This means that Formula (5.3) generalizes to the multiparameter case as:

$$\operatorname{coker} j_k^u \oplus \ker j_{k-1}^u = H_k\left(X^u, X^{u^1} \cup \cdots \cup X^{u^n}\right).$$

This means that we have moved the homology changes to be captured by the optimality property, from those occurring from one step to another, to those occurring from what we already expect by the union of previous incomparable steps and what actually happens at a step. Coming back to Remark 5.1.5, we see how this point of view interprets the two cases exposed therein. On the one hand, a new homology class can be created even if no new cell is added. This has no corresponding relative homology class. Indeed, there is no critical cell to be captured. On the other hand, at a single step, a new cell can destroy some homology class existing only in the union of filtration steps and not in a single filtration step. This gives a homology class in $\ker j_k$.

5.3 Filtered Morse complexes preserve relative homology

The aim of this section is to show that the relative homology of each filtration step with respect to the union preceding the same step is preserved when moving from a one-critical filtered complex $\mathcal{X}$ to a filtered Morse complex $\mathcal{M}$ compatible with $\mathcal{X}$.

In this section, we assume that the filtered complex $\mathcal{X}$ is one-critical. For example, it might be obtained by sublevel sets with respect to a filtering function $\phi : X \longrightarrow \mathbb{Z}^n$ as observed in Remark 1.6.5.

5.3.1 Proposition. For each integer $n \geqslant 1$, each filtration grade $u \in \mathbb{Z}^n$, and each homology degree k,

$$H_k\left(X^u, X^{u^1} \cup \cdots \cup X^{u^n}\right) \cong H_k\left(M^u, M^{u^1} \cup \cdots \cup M^{u^n}\right).$$

Proof. We claim that, for all $1 \leqslant i \leqslant n$, $H_k(X^{u^1} \cup \cdots \cup X^{u^i}) \cong H_k(M^{u^1} \cup \cdots \cup M^{u^i})$. For simplifying the notation, we denote by i the grade u^i, by $X^{\cup i}$ the union of Lefschetz complexes $X^1 \cup \cdots \cup X^i$, and by $M^{\cup i}$ the union of Lefschetz complexes $M^1 \cup \cdots \cup M^i$.
We proceed inductively on i. The base step is provided by Theorem 1.6.8 which implies that the inclusion-induced maps from $H_k(M^i)$ to $H_k(X^i)$ are isomorphisms for all integers k. Now, consider the MV-triples $(X^{\cup i+1}, X^{\cup i}, X^{i+1})$ and $(M^{\cup i+1}, M^{\cup i}, M^{i+1})$. Inclusion-induced maps connect the chain complexes of the components in $(M^{\cup i+1}, M^{\cup i}, M^{i+1})$ to those in $(X^{\cup i+1}, X^{\cup i}, X^{i+1})$ and commute with the maps in the two short exact sequences. From Theorem 1.3.12, we obtain the corresponding long exact Mayer-Vietoris sequences, depicted by rows in the following diagram:

$$\begin{array}{ccccccc}
\ldots \longrightarrow & H_k(X^{\cup i}) \oplus H_k(X^{i+1}) & \longrightarrow & H_k(X^{\cup i} \cup X^{i+1}) & \longrightarrow & H_{k-1}(X^{\cup i} \cap X^{i+1}) & \longrightarrow \ldots \\
& \downarrow{\scriptstyle \varphi_k^1} & & \downarrow{\scriptstyle \varphi_k} & & \downarrow{\scriptstyle \varphi_{k-1}^2} & \\
\ldots \longrightarrow & H_k(M^{\cup i}) \oplus H_k(M^{i+1}) & \longrightarrow & H_k(M^{\cup i} \cup M^{i+1}) & \longrightarrow & H_{k-1}(M^{\cup i} \cap M^{i+1}) & \longrightarrow \ldots
\end{array}$$

where, by Theorem 6.13 in [164] maps φ_k^1, φ_k, φ_{k-1}^2 are given by naturality property from the corresponding chain complex maps and, thus, make the diagram commute. By the Five Lemma, if all φ_k^2, φ_k^1 are isomorphisms, then φ_k is an isomorphism.
Each map φ_k^1 is given as direct sum of two maps. The first map is an isomorphism by inductive assumption. The second one is the isomorphism implied by Theorem 1.6.8. For the maps φ_k^2, we notice that, once indicated by $u-1$ the grade with i^{th}-entry defined by $u_i - 1$, then $X^{u-1} = X^{\cup i} \cap X^{i+1}$ and $M^{u-1} = M^{\cup i} \cap M^{i+1}$. Indeed, the filtrations are one-critical. This means that there is no cell in the intersection $X^{\cup i} \cap X^{i+1}$ that might be outside X^{u-1}, and the converse inclusion is obvious. Thus, the maps φ_k^2 are the isomorphisms implied by Theorem 1.6.8. Hence, the Five Lemma applies and, for all i from 1 to $n-1$, $H_k(X^1 \cup \cdots \cup X^{i+1}) \cong H_k(M^1 \cup \cdots \cup M^{i+1})$ which proves our claim.

Consider now the inclusions of $X^{\cup n}$ into X^u and of $M^{\cup n}$ into M^u. This allows us to consider the relative pairs $(X^u, X^{\cup n})$ and $(M^u, M^{\cup n})$.

By Theorem 1.3.10, the two relative pairs induce each a long exact sequence of homology corresponding to the rows in the following diagram:

$$\begin{array}{ccccccccc}
\ldots & \longrightarrow & H_k(X^u) & \longrightarrow & H_k(X^u, X^{\cup n}) & \longrightarrow & H_{k-1}(X^{\cup n}) & \longrightarrow & \ldots \\
 & & \downarrow{\scriptstyle \psi_k^1} & & \downarrow{\scriptstyle \psi_k} & & \downarrow{\scriptstyle \psi_{k-1}^2} & & \\
\ldots & \longrightarrow & H_k(M^u) & \longrightarrow & H_k(M^u, M^{\cup n}) & \longrightarrow & H_{k-1}(M^{\cup n}), & \longrightarrow & \ldots
\end{array}$$

where, maps ψ_k^1, ψ_k, ψ_k^2 are inclusion-induced and make the diagram commute by applying the same argument as for maps φ_k^1, φ_k^2. The maps ψ_k^1 are the isomorphisms implied by Theorem 1.6.8. The maps ψ_k^2 are the isomorphisms of our claim. Hence, by the Five Lemma, we get that ψ_k are isomorphisms. Thus, $H_k(X^u, X^{\cup n}) \cong H_k(M^u, M^{\cup n})$ which concludes our proof. □

A crucial application of Proposition 5.3.1 is that optimality of V compatible with $\mathcal{X}$ can be directly checked on the associated filtered Morse complex $\mathcal{M}$. Moreover, the following Lemma gives a sufficient condition to check, independently for each filtration grade u, that the number $m_k(u)$ of k-cells in any level set $M^u \setminus \left(M^{u^1} \cup \cdots \cup M^{u^n}\right)$ is the one expected by considering relative homology.

5.3.2 Lemma. Let $u \in \mathbb{Z}^n$ be any filtration grade of a filtered Morse complex $\mathcal{M}$, and ∂^{rel} the collection of boundary maps for the relative pair $(M^u, M^{u^1} \cup \cdots \cup M^{u^n})$. If, for all integers k and all k-cells $a \in M^u \setminus \left(M^{u^1} \cup \cdots \cup M^{u^n}\right)$, it holds that $\partial_k^{\text{rel}} a = 0$, then

$$\forall k \in \mathbb{Z}, \quad \dim H_k\left(M^u, M^{u^1} \cup \cdots \cup M^{u^n}\right) = m_k(u).$$

Proof. Fix an integer k. Notice that $m_k(u)$ is the cardinality of $M_k^u \setminus (M_k^{u^1} \cup \cdots \cup M_k^{u^n})$ and thus, the dimension of the space of relative k-chains in the desired relative pair $(M^u, M^{u^1} \cup \cdots \cup M^{u^n})$. If, for all $a \in M_k^u \setminus (M_k^{u^1} \cup \cdots \cup M_k^{u^n})$, we have that $\partial_k^{\text{rel}} a = 0$, then the dimension of the space of relative k-cycles is the same as that of relative k-chains, that is $m_k(u)$. If, for all $a \in M_{k+1}^u \setminus (M_{k+1}^{u^1} \cup \cdots \cup M_{k+1}^{u^n})$, we have that $\partial_{k+1}^{\text{rel}} a = 0$, then the dimension of the space of relative k-boundaries is 0. Hence, the dimension of the relative k^{th}-homology of the desired relative pair is $m_k(u)$. This concludes our proof. □

5.4 Algorithm ComputeDiscreteGradient is optimal

As anticipated at the beginning of this chapter, an algorithm capable of finding an optimal discrete gradient is said to be optimal itself. In this section, we detect two cases which guarantee our algorithm ComputeDiscreteGradient, described in Chapter 4, to be optimal. In both cases, we do not require further assumptions on the input filtering function. Additional requirements has to be satisfied by the input cell complex.

The first additional requirement for the input cell complex is that of being a cubical complex embedded in $\mathbb{R}^3$. This generalizes the optimality result of Theorem 11 in [162] to the multiparameter case. The second additional requirement is that of being an abstract 2-dimensional simplicial complex. Recall that a suitable input for ComputeDiscreteGradient is a pair (X, f) where X is a regular cell complex with intersection property, and $f : X_0 \longrightarrow \mathbb{R}^n$ is a component-wise injective function. This kind of input represents the filtered complex $\mathcal{X}^{\tilde{f}}$ denoting the filtered complex induced by the function $\tilde{f}$ vertex-induced from f as in Defition 3.1.2. The output retrieved by ComputeDiscreteGradient applied to (X, f) is a pair (V, M) where V is a discrete gradient and $M = M(V)$ is the corresponding Morse set. The output pair represents a filtered Morse complex $\mathcal{M}^{\tilde{f}}$ compatible with $\mathcal{X}^{\tilde{f}}$.

5.4.1 Optimality for 3D regular grids

The following theorem generalizes Theorem 11 in [162] to the multiparameter filtrations.

5.4.1 Theorem. *Let (Q, f) be a suitable input for algorithm* ComputeDiscreteGradient *and let (V, M) be the output retrieved. If Q is cubical complex with vertexes in a 3D space, then V is optimal with respect to the filtered complex $\mathcal{Q}^{\tilde{f}}$.*

Proof. By Proposition 5.3.1, it suffices to show that, for each filtration grades $u \in \mathbb{Z}^n$ and each integer k, we have that $m_k(u) = \dim H_k(M^u, M^{u^1} \cup \cdots \cup M^{u^n})$. By Lemma 5.3.2, it is sufficient to prove that each critical k-cube q in M^u satisfies $\partial_k^{\text{rel}} q = 0$, with ∂^{rel} the family of relative boundary maps with respect to the relative pair $(M^u, M^{u^1} \cup \cdots \cup M^{u^n})$. This is guaranteed by Lemmas 8, 9 and 10 in [162]. □

5.4.2 Optimality for abstract simplicial 2-complexes

Here, we consider (S, f) a suitable input for ComputeDiscreteGradient with S a simplicial 2-dimensional complex. Before stating our optimality results, we need to state some local properties holding for simplicial complexes of any dimension. In fact, we focus our attention on each level set Lset of S under the extended function $\tilde{f}$. The output retrieved by ComputeDiscreteGradient

applied to (S,f) is a pair (V,M) where V is a discrete gradient and $M = M(V)$ is the corresponding Morse set. The output pair represents a filtered Morse complex $\mathcal{M}^{\tilde{f}}$ compatible with $\mathcal{X}^{\tilde{f}}$. Recall from Chapter 4, that we have a call of the subroutine HomotopyExpansion for each level set Lset. According to Theorem 3.3.1, we define z to be the unique simplex in S such that $\mathrm{Low}_f(z) = \mathrm{Lset}$. All along this section, we introduce the notation $s^{(+p)}$ to indicate a simplex $s \in \mathrm{Low}_f(z)$ such that $\dim s - \dim z = p$, along with the obvious extensions of the notions of $(+p)$-paths and $(+p)$-connected components in $\mathrm{Low}_f(z)$.

5.4.2 Remark. *Let z be a simplex in S. Each $(+p)$-simplex $s \in \mathrm{Low}_f(z)$ has exactly p faces in $\mathrm{Low}_f(z)$. Indeed, if s and z differ in dimension by p, then s has p faces incident to z. None of those faces can have values under $\tilde{f}$ higher than z, otherwise s would not be part of the lower star.*

5.4.3 Lemma. For each level set Lset, if K is a $(+1)$-connected component in the lower star $\mathrm{Low}_f(z) = \mathrm{Lset}$ and $c^{(+1)}$ is the first simplex in K to be classified as critical by HomotopyExpansion. Then $c^{(+1)}$ is the only possible critical $(+1)$-simplex in K belonging to M^u.

Proof. Let $c^{(+1)}$ be the first $(+1)$-simplex in K to be classified as critical by HomotopyExpansion in $\mathrm{Low}_f(z)$. Assume by contradiction that there is a simplex $s^{(+1)}$ in K other than $c^{(+1)}$ to be classified as critical. The subroutine HomotopyExpansion declares a simplex in Ord0 critical when the ordered list Ord1 is found empty. We focus on the moment when $c^{(+1)}$ is already classified and $s^{(+1)}$ is in Ord0 and Ord1 is empty. Since K is $(+1)$-connected, there exists a $(+1)$-path Γ from $s^{(+1)}$ to $c^{(+1)}$ in K whose simplices satisfy $s_r^{(+1)} < t_r^{(+2)} > s_{r-1}^{(+1)} < t_{r-1}^{(+2)} > \cdots > s_1^{(+1)} < t_1^{(+2)} > s_0^{(+1)}$ such that $s_r^{(+1)} = s^{(+1)}$ and $s_0^{(+1)} = c^{(+1)}$. Let the index i, with $1 \leqslant i \leqslant r$, be the smallest index such that $s_{i+1}^{(+1)}$ is not yet classified. Since $s_i^{(+1)}$ is already classified and $(+2)$-simplices have exactly two faces inside $\mathrm{Low}_f(z)$, the simplex $t_{i+1}^{(+2)}$ has entered Ord1. Since $s_{i+1}^{(+1)}$ is not yet classified, the simplex $t_{i+1}^{(+2)}$ cannot be critical. Indeed, a simplex is made critical if all of its faces have been classified. Secondly, the pair $(s_i^{(+1)}, t_{i+1}^{(+2)})$ cannot be in V since the last face of $t_{i+1}^{(+2)}$ to be classified is $s_{i+1}^{(+1)}$. Moreover, there cannot be a pair $(t_{i+1}^{(+2)}, w^{(+3)})$ in V since this requires $t_{i+1}^{(+2)}$ inside Ord0, that is all faces of $t_{i+1}^{(+2)}$ to be already classified. Hence, the simplex $t_{i+1}^{(+2)}$ is still in Ord1 which contradicts Ord1 being empty. Since $c^{(+1)}$ remains the only possible critical $(+1)$-simplex in K, this concludes the proof. □

5.4.4 Lemma. For each level set Lset, if K is a $(+1)$-connected component in the lower star $\mathrm{Low}_f(z) = \mathrm{Lset}$, then

(i) there exists one and only one simplex $c^{(+1)}$ in K such that, for any simplex $s^{(+1)} \in K$, there is a V-path Γ from $s^{(+1)}$ to $c^{(+1)}$ in K;

(ii) when HomotopyExpansion classifies as critical a simplex $a^{(+2)}$ in K, any $s^{(+1)} \in K$ has already been classified.

Proof. Let $c^{(+1)}$ be the first $(+1)$-simplex in K classified by HomotopyExpansion as part of $\mathrm{Low}_f(z)$.
For proving (i), we construct recursively the V-path Γ. We assume a $(+1)$-simplex s_i and a V-path $\Gamma_i = \{(s_0,t_0),\dots,(s_{i-1},t_{i-1})\}$ (empty in case $i=0$) from s_0 to $s_i < t_{i-1}$ with $s_0 = s^{(+1)}$ and $s_i \neq s_j$ for every $i \neq j$. If $s_i = c^{(+1)}$, then $\Gamma = \Gamma_i$. If $s_i \neq c^{(+1)}$, by Lemma 5.4.3, s_i cannot be critical. Moreover, it cannot be that $(z,s_i) \in V$ since that would have been the first classification in Lset, even before $c^{(+1)}$. Since, by Proposition 3.6 in [3], all simplices in $\mathrm{Low}_f(z)$ are eventually classified and being K $(+1)$-connected, a simplex $t_i^{(+2)}$ exists in K such that $(s_i,t_i) \in V$ and this classification is performed after that of $c^{(+1)}$. By Remark 5.4.2, there is a unique face $s_{i+1}^{(+1)}$ of t_i in $\mathrm{Low}_f(z)$ different from s_i. In particular, s_{i+1} is classified before t_i because $(s_i,t_i) \in V$ implies that the other faces of t_i are classified before t_i. Now, we set $\Gamma_{i+1} = \Gamma_i \cup \{(s_i,t_i)\}$ and we update the input of the recursive argument by setting $s_i = s_{i+1}$ and $\Gamma_i = \Gamma_{i+1}$. Eventually, the procedure will reach $s_i = c^{(+1)}$, by finiteness of K and by observing that each pair (s_{i+1},t_{i+1}) is classified before (s_i,t_i) while $c^{(+1)}$ is the first $(+1)$-simplex classified by assumption, and all s_i's are different.
For proving the uniqueness of $c^{(+1)}$, consider a simplex $\bar{c}^{(+1)}$ in K satisfying condition (i) and different from $c^{(+1)}$. Then, there exists a V-path from $\bar{c}^{(+1)}$ to $c^{(+1)}$. If $\bar{c}^{(+1)}$ satisfies point (i) then, in particular, there is a V-path from $c^{(+1)}$ to $\bar{c}^{(+1)}$. This gives a cyclic V-path which, by Theorem 3.8 in [3], is a contradiction.
To prove (ii), let $a^{(+2)}$ be a critical simplex in K. The simplex $c^{(+1)}$ is classified before $a^{(+2)}$ by definition of $c^{(+1)}$. By point (i), each simplex $s^{(+1)}$ in K admits a V-path

$$\{(s_0^{(+1)},t_0^{(+2)}),\dots,(s_l^{(+1)},t_l^{(+2)})\}$$

from $s^{(+1)}$ to $c^{(+1)}$. The pair $(s_l^{(+1)},t_l^{(+2)})$ is available for classification as $c^{(+1)}$ is already classified. Notice that, by Remark 5.4.2, each $(+2)$-simplex has exactly two faces in $\mathrm{Low}_f(z)$. Thus, as each pair $(s_i^{(+1)},t_i^{(+2)})$, with $1 \leqslant i \leqslant l$, is classified, the pair $(s_{i-1}^{(+1)},t_{i-1}^{(+2)})$ becomes available for classification and this happens for all $(+1)$-simplices in K. Hence, if there is no pair in K available for classification, then all $(+1)$-simplices in K have already been classified which concludes our proof. □

5.4.5 Definition (Branching V-path)**.** A V-path $\Gamma = \{(s_i,t_i)\}_{0\leqslant i\leqslant l}$ *branches in* $W \subseteq V$ if and only if there exists another V-path $\overline{\Gamma} = \{(\bar{s}_j,\bar{t}_j)\}_{0\leqslant j\leqslant r}$ in W such that $(s_i,t_i) = (\bar{s}_j,\bar{t}_j)$ for some indexes $i \leqslant l-1$ and $j \leqslant r-1$, and $(s_{i+1},t_{i+1}) \notin \overline{\Gamma}$ and $(\bar{s}_{j+1},\bar{t}_{j+1}) \notin \Gamma$.

5.4.6 Lemma. For each level set Lset, let V_{Lset} be the subset of discrete vectors in V whose cells are in Lset. Then, there is no V-path from any $(+1)$-simplex to any $(+1)$-simplex that can branch in V_{Lset}.

Proof. Let $\Gamma = \{(s_i^{(+1)},t_i^{(+2)})\}_{0\leqslant i\leqslant l}$ be a V-path in V_{Lset}. Assume by contradiction that Γ branches inside V_{Lset}. This means, there exists in V_{Lset} a V-path $\overline{\Gamma} = \{(\bar{s}_j^{(+1)},\bar{t}_j^{(+2)})\}_{0\leqslant j\leqslant r}$ such that $(s_i,t_i) = (\bar{s}_j,\bar{t}_j)$ for some indexes $i \leqslant l-1$ and $j \leqslant r-1$ and $(s_{i+1},t_{i+1}) \notin \overline{\Gamma}$ and $(\bar{s}_{j+1},\bar{t}_{j+1}) \notin \Gamma$. However,

this implies $s_{i+1} \neq \bar{s}_{j+1}$, that is t_i is a $(+2)$-simplex with three different faces s_i, s_{i+1} and $\bar{s}_{j+1}$ inside $\mathrm{Low}_f(z)$ which is contradiction by Remark 5.4.2. □

The previous lemmas, valid for HomotopyExpansion on each level set, are used to prove the following result for ComputeDiscreteGradient.

5.4.7 Theorem. *Let (S,f) be a suitable input for algorithm* ComputeDiscreteGradient *and let (V,M) be the output retrieved. If S is an abstract simplicial d-complex with $d \leqslant 2$, then V is optimal with respect to the filtered complex $\mathcal{S}^{\tilde{f}}$.*

Proof. By Proposition 5.3.1, it suffices to show that, for each filtration grades $u \in \mathbb{Z}^n$ and each integer k, we have that $m_k(u) = \dim H_k(M^u, M^{u^1} \cup \cdots \cup M^{u^n})$. By Lemma 5.3.2, it is sufficient to prove that each critical k-simplex s in M^u satisfies $\partial^{\mathrm{rel}}_k s = 0$, with ∂^{rel} the family of relative boundary maps with respect to the relative pair $(M^u, M^{u^1} \cup \cdots \cup M^{u^n})$. Let Lset^u be $M^u \backslash (M^{u^1} \cup \cdots \cup M^{u^n})$. Observe that the subroutine HomotopyExpansion builds the output (V,M) independently over each Low^u. According to Theorem 3.3.1, we define z to be the unique simplex in S such that $\mathrm{Low}_f(z) = \mathrm{Low}^u$. Let $a^{(+p)}$ be a critical simplex in $\mathrm{Low}_f(z)$. We separately treat the cases $p = 0,1,2$. A simplex $a^{(+0)}$ is critical in $\mathrm{Low}_f(z)$ when $a^{(+0)} = z$. Since each separatrix links two different critical simplices, each possible separatrix ends outside $\mathrm{Low}_f(z)$, so that we have $\partial^{\mathrm{rel}} a^{(+0)} = 0$.
A simplex $a^{(+1)}$ is critical in $\mathrm{Low}_f(z)$ when its only face z inside $\mathrm{Low}_f(z)$, has been matched to another simplex. Hence, we get $\partial^{\mathrm{rel}} a^{(+1)} = 0$.
For a simplex $a^{(+2)}$, we prove that $\partial^{\mathrm{rel}} a^{(+2)} = 0$ by analyzing all the V-paths from $a^{(+2)}$ to some critical $(+1)$-simplex in $\mathrm{Low}_f(z)$. Let K be the $(+1)$-connected component of $\mathrm{Low}_f(z)$ such that $a^{(+2)}$ belongs to K. Notice that no V-path from $a^{(+2)}$ can reach simplices outside K. By Lemma 5.4.4(i), the two faces of $a^{(+2)}$ admit each a V-path to the same simplex $c^{(+1)} \in K$. If $c^{(+1)}$ is not critical, then there is no V-path from $a^{(+2)}$ to a critical $(+1)$-simplex in $\mathrm{Low}_f(z)$. Hence, $\partial^{\mathrm{rel}} a^{(+2)} = 0$. Suppose on the contrary that $c^{(+1)}$ is critical. By Lemma 5.4.6, a V-path starting from $a^{(+2)}$ cannot branch inside $\mathrm{Low}_f(z)$, in particular inside K. This means that precisely two V-paths connect $a^{(+2)}$ to $c^{(+1)}$. By definition of incidence coefficient (not only for coefficients in $\mathbb{Z}_2$ as assumed here), we get $\partial^{\mathrm{rel}} a^{(+2)} = 0$. □

6 Experimental analysis and evaluation

This chapter reports the details and the results of the experimental tests we performed to evaluate the performances of the new algorithm ComputeDiscreteGradient described in Chapter 4. In Section 6.1, we evaluate the efficiency of our divide-and-conquer approach compared to the global approach of the algorithm proposed in [4] and reviewed in our preliminaries in Section 3.3 whose pseudocode is reported in Algorithm 3. In Section 4.5, we proved the two algorithms to be equivalent and, in this chapter, we see the practical advantages of our algorithm. As a second issue, we evaluate the impact of ComputeDiscreteGradient as a preprocessing for computing multipersistence invariants. Indeed, the discrete gradient retrieved by our algorithm can be used to obtain a filtered Morse complex which preserves all multipersistence features. In general, the so-obtained filtered Morse complex is smaller in terms of cells and filtration grades than the original one. We evaluate the impact in time and storage cost performances of this filtration reduction when applied to multipersistence computations. To the best of our knowledge, this is the first experimental evaluation of a filtration reduction for enhancing multipersistence computations. This second issue will be addressed both in Section 6.2 and Section 6.3. The former applies the filtration reduction to the computation of the complete information of persistence, that is the *persistence module* of a multiparameter filtration (Definition 1.6.7). The latter applies the filtration reduction to the computation of an approximate version of the *persistence space* (recall Definition 1.7.9) through the method of foliations first introduced in [17]. Finally, in Section 6.4, we examine the experimental tests performed over large simulation data to evaluate the feasibility of our method in the practical analysis of real-size data.

6.1 Comparison to the Matching Algorithm

In this section, we compare the time and storage cost performances of our new algorithm ComputeDiscreteGradient described in Chapter 4 to the algorithm Matching, introduced in [4] and described in our state-of-the-art review in Section 2.4. As discussed in Section 4.5, the two algorithms are equivalent. Same inputs correspond to same outputs.

6.1.1 Method

The two algorithms are tested on a single thread implementation and compared over three triangulations of shapes whose vertex coordinates belong to $\mathbb{R}^3$: a torus, a sphere, and a Gorilla.

Each triangulation is subdivided by recursively applying the Catmull-Clark algorithm [40] adapted to triangle meshes. This way we obtain three triangle meshes with a different number of cells for the same shape. This provides us with nine different triangle meshes to test. Each dataset to be tested is composed by one of the nine triangle meshes and, the two-parameter filtering function given by the x and y coordinates of the vertexes. The datasets are described in Table 6.1. For each dataset, we are indicating the number of parameters in the filtration (column *Parameters*), the total number of simplices (column *Original*), total number of critical simplices in the resulting discrete gradient (column *Critical*) and the resulting compression factor (column *Original/Critical*).

Dataset	Parameters	Cells		Compression factor
		Original	Critical	Original/Critical
Torus	2	1.3M	0.035M	37.9
		5.3M	0.11M	45.3
		21.5M	0.77M	27.7
Sphere	2	2.9M	0.28M	10.2
		11.7M	0.11M	10.2
		47.1M	0.46M	10.1
Gorilla	2	3.8M	0.4M	9.5
		15.2M	1.6M	9.4
		60.9M	6.4M	9.4

Table 6.1: Datasets used for the experiments. For each of them, we indicate the number of parameters in the filtration (column *Parameters*), the number of simplices in the original dataset (column *Original*), number of critical simplices retrieved by ComputeDiscreteGradient and Matching (column *Critical*) and the compression factor (column *Original/Critical*).

In order to guarantee a fair comparison, the same representations of the datasets have been used for implementing both algorithms. We measure time and storage performances along with scalability properties of the two algorithms with respect to the number of cells in the dataset.

6.1.2 Implementation

Both algorithms have been implemented in C++ based on the MDG library [85]. The latter provides an efficient encoding for both a 2-dimensional or 3-dimensional simplicial complex as well as a compact encoding for the discrete gradient.

Triangle mesh representation. A triangle mesh X is a simplicial 2-complex formed by vertices, edges, and triangles. To compactly encode the relations among these simplices we are using an incidence-based data structure. The incidence-based data structure encodes vertices and triangles in the triangle mesh which gives $|X_0| + |X_2|$ entities to be stored explicitly, plus some additional relations. For each triangle, we encode 3 arcs giving a reference to its vertexes. For each vertex instead, we encode the list of triangles incident in it, given by the same 3 arcs in the opposite direction. The total footprint required for encoding X is $7|X_2| + |X_0|$. This kind

of representation fulfills the assumptions for the description and the complexity analysis of the algorithm ComputeDiscreteGradient of Chapter 4, and the description of the algorithm BoundaryMpas in the simplicial case provided in Section 4.6.2.

Filtering function representation. The two-parameter filtering function determined by the x and y components is stored for vertexes, only. For each vertex and each parameter, a floating point is stored. The filtering values of the extension to simplices of higher dimensions s is retrieved only locally, over each vertex star. It can be retrieved in linear time in the number of the vertices of s.

Discrete gradient representation. The discrete gradient is here encoded by adopting the representation described in [86]. The latter focuses on encoding all the gradient pairs locally to a triangle s in X. The faces of dimension 1 of s are three edges, each one having two faces of dimension 0 (vertices). Thus, there are 9 possible gradient pairs internally to a triangle. If we consider also the possible pairs between ad edge of s and an adjacent triangle we get 12 possible gradient pairs and thus $2^{12} = 4096$ possible combinations. However, according to [193], the restrictions imposed by the discrete gradient (i.e. that each simplex can be involved in at most one pairing) imply that we have only 97 valid cases for a triangle. We can encode all these cases using only 1 byte per triangle and, thus, encoding the gradient only requires $|X_2|$ bytes. In this case, the auxiliary function isPaired in Algorithm 5 acts in the following way. Given a cell s in the triangle mesh, we can find the corresponding cell t such that $(s,t) \in V$ by considering all the triangles in the star of a vertex w of s and visiting the gradient encoding of those triangles incident to s. This procedure requires, in the worst case, $O(|t_w|)$ operations, where t_w is the set of triangles incident to some vertex w of s. This cost satisfies the assumptions in the review of the algorithm in Section 4.6 for the retrieval of the boundary maps.

6.1.3 Results

The experiments have been performed on a dual Intel Xeon E5-2630 v4 CPU at 2.20Ghz with 64GB of RAM.

Timings are shown in Figure 6.1. ComputeDiscreteGradient takes between 0.89 seconds and 4.8 minutes to finish depending on the dataset and it is generally 7 times faster than our implementation of Matching.

Other than time efficiency, our divide-and-conquer strategy also requires a limited use of memory. The memory consumption is shown in Figure 6.2 where we are reporting the maximum peak of memory used by the two algorithms. ComputeDiscreteGradient uses up to 10 gigabytes of memory and it is twice more compact than the our implementation of Matching.

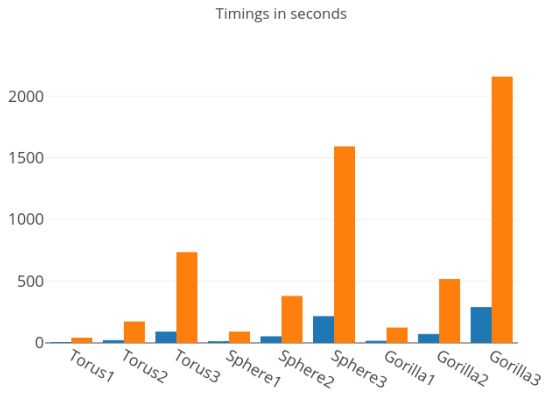


Figure 6.1: Timings required by ComputeDiscreteGradient (in blue) and Matching (in orange).

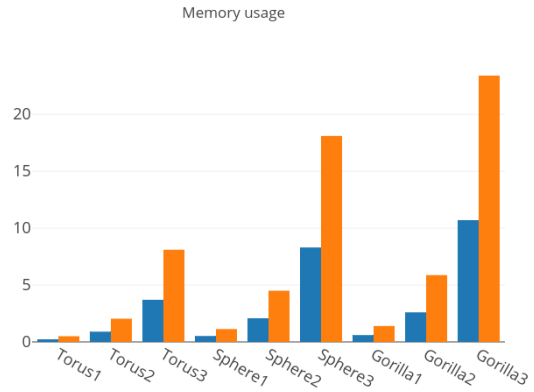


Figure 6.2: Maximum peaks of memory (in gigabytes) required by ComputeDiscreteGradient (in blue) and Matching (in orange).

6.1.4 Discussion

Time performances show the practical efficiency of ComputeDiscreteGradient compared to Matching. As seen in Section 4.4, the expected asymptotical complexity over a simplicial complex X with worst case-sized vertex star Star is $O(|X|\log|\text{Star}|)$ for both algorithms. The datasets considered in these tests satisfy $O(|\text{Star}|) = O(1)$. Thus, we expected linearity in the number of simplices which is confirmed by our tests. In Figure 6.3, we show the trends as the number of simplices increases. The linearity is confirmed for both algorithms but under very different slope coefficients. The latter is determined by the different number of stars to be retrieved and visited. In the case of ComputeDiscreteGradient, we need to process each vertex star. In the case of Matching, we need to process a star for each level set determined by the dataset.

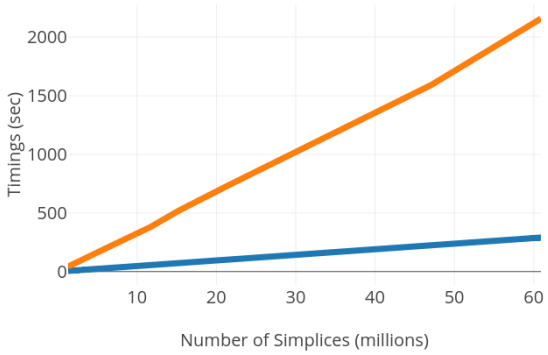


Figure 6.3: Trend in the timings for ComputeDiscreteGradient (in blue) and Matching (orange).

Storage cost results show the advantage of the divide-and-conquer approach of ComputeDiscrete-Gradient over Matching. The storage cost difference between the two implementations grows linearly in the number of simplices in the datasets. As seen in Section 4.4.2, the advantage is due to the different runtime storage costs. In particular, the runtime storage cost for Matching is affected by the global queue over all the simplices in the dataset and the need of tracking all classified cells. Both of them are performed locally, over each level set, by ComputeDiscreteGradient.

6.2 Impact on the computation of the persistence module

In this section, we apply ComputeDiscreteGradient in a pipeline for computing multipersistence. Indeed, our algorithm accepts as input a representation of a multiparameter filtration. The discrete gradient retrieved by our algorithm can be used to obtain a filtered Morse complex which preserves all multipersistence features. In general, the so-obtained filtered Morse complex is smaller in terms of cells and filtration grades than the original one (see Table 6.2). We evaluate the impact in time and storage cost performances of this filtration reduction. To the best of our knowledge, this is the first experimental evaluation of a filtration reduction for enhancing multipersistence computations. In this test, we evaluate, in particular, the impact in computing the complete information of persistence, that is the persistence module of a multiparameter filtration. In the next section, we evaluate our impact with respect to another invariant for multipersistence, namely the persistence space. We compare

We perform the persistence module computation by the open-source library Topcat [154]. Topcat is currently the only open-source library for retrieving a representation of the persistence module. However at the moment, Topcat has strong limitations in terms of time and memory costs.

6.2.1 Method

The datasets we consider correspond to six small triangulations of a torus and a sphere with vertexes in $\mathbb{R}^3$ synthetically produced.

	Original		Reduced		Original/Reduced	
Dataset	Cells	Grades	Cells	Grades	Cells	Grades
Sphere	38	8x8	4	5x5	9.5	2.56
	242	42x42	20	10x10	12.1	17.64
	2882	482x482	278	92x89	10.36	29.36
Torus	96	16x16	8	9x9	12	3.16
	4608	768x768	128	65x66	36	137.49
	7200	1200x1200	156	70x80	46.15	257.14

Table 6.2: Dataset sizes subdivided into original and reduced datasets. Each one reports the number of cells (column *Cells*) and the number of grades along each parameter (column *Grades*). The compression factor (columns *Original/Reduced*) is reported in terms of cells and global number of grades.

Each triangulation has a different number of simplices so to consider several resolutions of the same shape. This provides us with six different triangle meshes to test. Each dataset to be tested consists of one of the six triangle meshes and the two-parameter filtering function given by the x and y coordinates of the vertexes. We call these datasets the *original* datasets. Subcolumn *Cells* in column *Original* in Table 6.2 reports the number of cells for each triangle mesh. We keep the number of simplices small due to the strong limitations of Topcat. The number of values for x coordinates and y coordinates is reported in subcolumn *Grades* in column *Original* in Table 6.2.

Each original dataset is given as input to ComputeDiscreteGradient to retrieve the corresponding discrete gradient. Starting from the discrete gradient, we compute the boundary maps of the corresponding filtered Morse complex as described in [92] through the algorithm of Section 4.6. This can be done by retrieving the connections between pairs of critical cells, using the V-paths. Once the boundary maps have been represented, the filtering function values of the original dataset make the representation of the filtered Morse complex complete. The computed filtered Morse complex is called the *reduced* dataset. Subcolumn *Cells* in column *Reduced* in Table 6.2 reports the number of critical cells for each filtered Morse complex. Filtration steps with no corresponding critical cell are not considered in the filtered Morse complex. The number of remaining filtration grades for x coordinates and y coordinates is reported in subcolumn *Grades* in column *Reduced* in Table 6.2.

For each original and each reduced dataset, the Topcat library algorithm for computing the persistence module is applied. We measure time cost of the reduced dataset retrieval along with time and storage costs for each persistence module retrieval performed by the Topcat algorithm.

6.2.2 Implementation

The triangle meshes, the filtering functions and the discrete gradient used in this test are represented as in Section 6.1.2. The algorithm ComputeDiscreteGradient is implemented in C++ based on the MDG library [85] as discussed in detail in Section 6.1.2. Here the implementation is single threaded only. We have modified the Topcat library to adapt it to support one-critical filtrations of general cell complexes rather than only simplicial complexes in order to support a representation of the filtered Morse complex.

Datasets, original and reduced, are represented as text files indicating, for each cell, its first grade of appearance in the filtration, and all of its facets. In the case of reduced datasets, they are produced by applying Algorithm 5 to the output produced by ComputeDiscreteGradient.

6.2.3 Results

The experiments have been performed on a dual Intel Xeon E5-2630 v4 CPU at 2.20Ghz with 64GB of RAM.

	Original				Reduced				Reduction Time
			Persistence Module				Persistence Module		
Dataset	Cells	Grades	Time	Memory	Cells	Grades	Time	Memory	
Sphere	38	8x8	0.3	0.24	4	5x5	0.18	0.1	0.0264
	242	42x42	4.4	0.86	20	10x10	0.28	0.2	0.0244
	2882	482x482	-	-	278	92x89	24.3	1.5	0.0473
Torus	96	16x16	0.5	0.1	8	9x9	0.25	0.2	0.0255
	4608	768x768	-	-	128	65x66	7.96	2.4	0.0643
	7200	1200x1200	-	-	156	70x80	12.05	3.0	0.0815

Table 6.3: Timings (in seconds) and storage costs (in gigabytes) for the persistence module retrieval over the original (columns *Original*) and the corresponding reduced (columns *Reduced*) datasets. Timings (in seconds) for the getting the reduced dataset from the original one are reported in column *Reduction Time*. Columns *Cells* and *Grades* reports the number of cell in the dataset and the number of grades in along each parameter, respectively. Where no entry is visible, the computation is failed.

We combine time contributions of the running of ComputeDiscreteGradient applied to each original dataset together with the retrieval of the corresponding filtered Morse complex. Column *Reduction Time* in Table 6.3 reports the timings for the sum of the two timings. We measure time and storage costs for each persistence module retrieval performed by the Topcat algorithm. Columns *Time* in Table 6.3 reports the measured timings for the running of Topcat in the original and reduced cases. Columns *Mem.* in Table 6.3 reports the measured maximal memory peaks for the running of Topcat in the original and reduced cases.

Table 6.3 shows the results obtained by computing the persistence module of different datasets in the original and reduced cases. Where no result is reported, the storage space required exceeds

that of the machine used for the tests. Where possible to compare, we see that the persistence module retrieval takes approximately half of the time required in the original case for the two smallest datasets (row 1 and row 4 in Table 6.3). Where not possible to compare, timings over reduced datasets are affordable and range from 7.96 to 24.3 seconds. All maximal peaks measured for the memory usage are significantly below the machine RAM limits of 64GB. This suggests a very fast increasing rate of storage costs in the ones where the computations have failed. The measured reduction timings, ranging from 0.0244 to 0.0815 seconds are negligible with respect to all persistence module timings and not significantly affected by the different number of cells in the (tiny) considered datasets.

6.2.4 Discussion

The tests performed suggest an effective impact of the proposed preprocessing. The preprocessing makes all the datasets persistence module retrievable with reasonable time and space costs. The preprocessing reduces both the number of cells and the number of grades in the filtration considered and this leads to remarkable differences in time and space costs. Moreover, the preprocessing time cost is negligible with respect to the persistence module retrieval. As reviewed in Chapter 2, the asymptotical time complexity of the algorithm implemented in Topcat is given by $O(|F||X|^3)$ with $|F|$ the number of filtration grades in the multiparameter filtration and X the cell complex filtered by the filtration of the dataset. Our datasets are too small to highlight the cubical dependence in the number of cells. On our datasets, the biggest contribution to the results comes from the liner dependence in the number of filtration grades since it increases way faster and it makes computation fail even for not very big datasets.

However, the failure of tests over, for instance, the Sphere dataset with 2882 cells and 482x482 filtration grades, suggests that a filtered Morse complex of the same size would make the persistence module retrieval fail. This does not give an exact perception of the size of an hypothetical original dataset having such a corresponding reduced dataset. However, it is enough to deduce that our preprocessing does not make the persistence module computation feasible over reasonably large datasets. The bottleneck is given by the storage costs.

In conclusion, our preprocessing provides effective speed-ups in computing the persistence module. Further enhancements are in need in order to reach feasibility over large datasets. Smaller representations of multipersistence through invariants, such as the persistence space, are preferable to perform computations. In the next section, we show our tests for the persistence space retrieval.

6.3 Impact on the Foliation Method

In this section, we apply ComputeDiscreteGradient in a pipeline for computing multipersistence. As in the previous section, our algorithm accepts as input a representation of a multiparameter filtration. The discrete gradient retrieved by our algorithm can be used to obtain a filtered Morse complex which preserves all multipersistence features. In general, the so-obtained filtered Morse complex is smaller in terms of cells and filtration grades than the original one. The goal of this section is to evaluate our impact with respect to the retrieval of an approximate version of the persistence space associated, as an invariant, to multipersistence. In particular, we focus on the foliation method [17] we described and reviewed in Section 2.4.2. The foliation method is a way of determining the persistence space (Definition 1.7.9) by applying several one-parameter persistent homology computations over different one-parameter linearization of a multiparameter set of filtration grades, called *slices*. The totality of the possible linearization would give the persistence space. In practice, the greater the number of slices considered, the more accurate is the approximation of the persistence space.

The purpose is not that of evaluating our reduction impact over an efficient implementation of the foliation method but giving evidence of that impact independently from the implementations or motivations for applying the foliation method. Indeed, the foliation method is not to be thought of as a final goal. Rather, it is a mean to encode some of the multipersistence features in a longer pipeline. For instance, in the original work [17], the final motivation is that of comparing shapes by means of the matching distance between two approximated persistence spaces.

6.3.1 Method

In our tests, we only consider the case of two-parameter filtrations, i.e., $n = 2$. The datasets we consider correspond to four triangulations from the Princeton Shape Benchmark [171] whose original size is reported in Table 6.4 column *Cells*.

		Cells		Compression factor
Dataset	Parameters	Original	Reduced	Original/Critical
Shark	2	9.49K	1.11K	8.54
Turtle	2	10.86K	1.20K	9.07
Gun	2	27.82K	3.14K	8.85
Piano	2	119.08K	10.95K	10.87

Table 6.4: Datasets used for the experiments. For each of them, we indicate the number of parameters in the filtration (column *Parameters*), the number of thousands of simplices in the original dataset (column *Original*), number of critical simplices retrieved by ComputeDiscreteGradient (column *Reduced*) and the compression factor (column *Original/Reduced*).

Each dataset to be tested is composed by one of the four triangle meshes X and, by the two-parameter filtering function given by the x and y coordinates of the vertexes, i.e., by the function $\phi : X \longrightarrow \mathbb{R}^2$ defined by $\phi(x,y,z) = (x,y)$. We call these datasets the *original* datasets.

Then, we apply the following:

a) *Reduction Phase:* original datasets are processed, as described in Section 6.2, to retrieve corresponding *reduced datasets*.

Timings for this phase are reported in column *Reduction Time* in Table 6.5. Datasets, original and reduced, are represented as text files indicating, for each cell, its first multigrade of appearance in the filtration, and all of its facets. The text files are given as input for the second phase of our experiments, depending on a fixed integer parameter ω from 2 to 10:

b) *Line Extraction:* uniformly selecting ω^2 lines of non-negative slope in the Euclidean plane (from 4 to 100 lines);

c) *Foliation Phase:* computing the persistence space of the original and reduced datasets over the same selected ω^2 slices corresponding to the lines extracted at the previous point.

These two phases are performed by a Python script based on the Pandas Data Analysis Library.

Line Extraction A number of ω^2 lines are extracted depending on the processed original dataset. Each original dataset comes associated with its filtering function $\phi : X \longrightarrow \mathbb{R}^2$ defined over vertexes by $\phi(x,y,z) = (x,y)$. For each component $i = 1,2$, we detect parameters $C_i := \max_{x \in X} \phi_i(x)$ and $c_i := \min_{x \in X} \phi_i(x)$. Each line is determined by two parameters. The first is the slope coefficient λ ranging from 0 to $\frac{\pi}{2}$ independently from the dataset, and, the second one is a *base point* b ranging in an interval of the bisector of the II and IV quadrant dependening on the processed original dataset. We uniformly select ω values for λ. For each λ value, points (c_1, C_2) and (C_1, c_2) are projected over the bisector along direction λ, this way delimiting the interval range for the base points. We uniformly select ω values for b in the so-obtained interval. All possible values for λ and b are combined to represent the ω^2 possible lines. The line l is the line of unit vector $m = (\cos(\lambda), \sin(\lambda))$ passing though b and, thus indicated by $l = (m,b)$. This phase is applied once per original dataset. The extracted lines determine slices for both the original and reduced dataset computations in the next phase. Timings, for the case of 100 lines are reported in column *Line Extraction* in Table 6.5.

Foliation phase Each extracted line $l = (m,b)$ is processed in a for-cycle. However, computations over each line are one another independent. In order to highlight, in our test, the parts depending on Python script and those based on the routines in PHAT for computing PH, we subdivide the foliation phase into three steps applied to both the original and reduced datasets:

1) *building persistence input:* in this step, each cell s in the original dataset is represented by a unique id number. The id's are preserved in the reduced dataset. A dictionary D assigns to each id a single-parameter filtration grade obtained via:

$$\Phi^l(s) := \min_{i=1,2} m_i \cdot \max_{i=1,2} \frac{\phi_i(s) - b_i}{m_i}.$$

Cells in the reduced and in the original datasets are sorted by increasing values under Φ^l and, where equal, we preserve the id order. Facets, for each cell, are updated according to the new ordering via D to form the input boundary matrix required by PHAT routine to compute PH. For the case of 100 slices, timings for this step, are reported in column *Building Pers. Input* in Table 6.5;

2) *computing persistence:* in this step, the boundary matrix built in the previous step is passed to the PHAT routine to compute PH. The computations are performed via five different algorithms: the standard algorithm for persistence [80] and four optimizations, namely the twist algorithm [53], the chunk algorithm [11], the spectral sequence algorithm [75] and the row algorithm [177]. For the case of 100 slices and the standard algorithm, timings for this step, are reported in column *Computing Persistence* in Table 6.5. Timings for the other algorithms are reported in Figure 6.5;

3) *reindexing persistence output:* PHAT returns the list of *persistence pairs* (i, j) relative to the current slice. PHAT always assumes the step-by-step filtration induced by the column order of the input boundary matrix. Let C be the indexing of the columns. Hence, we get $i < j$ with $i, j \in C$. This way, a returned persistence pair (i, j) may or may not have positive persistence, i.e., the corresponding values under Φ^l may or may not be different. In this step, we verify whether or not (i, j) has posixtive persistence. For positive (i, j), we find the corresponding cell id through the dictionary D defined in step 1). For the case of 100 slices, timings for this step, are reported in column *Reindexing Pers. Output* in Table 6.5.

6.3.2 Implementation

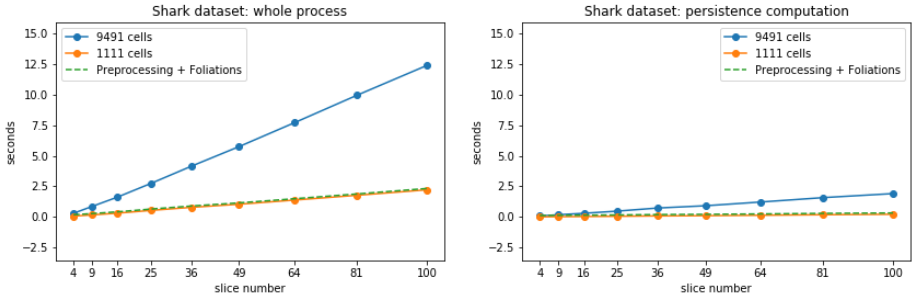
The triangle meshes, the filtering functions and the discrete gradient are represented as described in Section 6.1.2. The algorithm $\mathrm{ComputeDiscreteGradient}$ is implemented in C++ based on the MDG library [85] as discussed in detail in Section 6.1.2. Here the implementation is single threaded only. The origninal and reduced datasets are represented as text files indicating, for each cell, its first grade of appearance in the filtration, and all of its facets. In the case of reduced datasets, they are produced by applying Algorithm 5 to the output produced by $\mathrm{ComputeDiscreteGradient}$. The line extraction and slice restriction phases are performed by a Python script based on the Pandas Data Analysis Library. For each line l, the slice is obtained by sorting all cells in the dataset according to increasing values of Φ^l. We keep track of the original indexing of cells as read

from the input text file through a dictionary. For any of the five algorithms implemented in PHAT, the corresponding PHAT function is called within the Python script. The algorithms implemented in PHAT accepts the filtration of the data as implicitly given by the order of appearance of cells provided in input. Then, PHAT retrieves persistence pairs with respect to that step-by-step filtration so that the PHAT output depends on the ordered given in input. The dictionary allows us to translate PHAT outputs with respect to the two-parameter filtration in the input text file for the entire dataset. Moreover, we can distinguish persistence pairs with positive persistence from the others by comparing filtering function values for the two cells in the same pair.

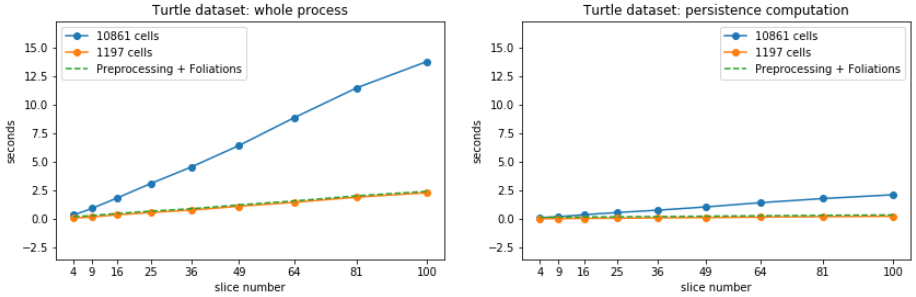
6.3.3 Results

Tests are performed by an Intel Core i7 CPU 940 at 3.20GHz with 16GB of RAM.

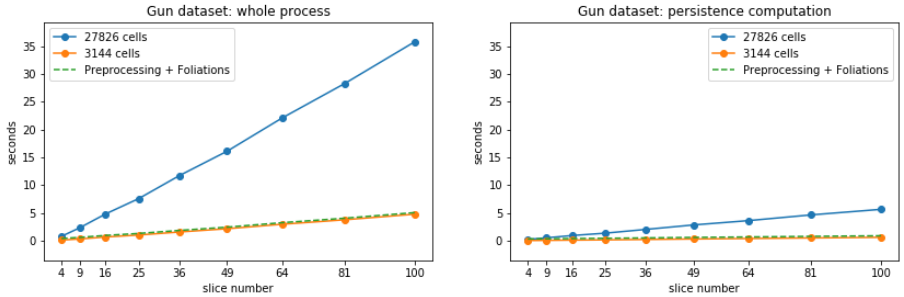
To address our first task, in Figures 6.4, we compare time performances with slices ranging from 4 to 100, and the one-parameter persistence over each slice computed by the standard algorithm implemented in PHAT. For each dataset, we show, on the left, the global timings for the foliation phase b), on the right, the focus on the persistence computation of step 2). In both cases, the analyzed datasets highlight the linear dependency of the foliation method on the number of slices considered. For reduced datasets (orange line), the slope coefficient is smaller than for the original datasets (blue line) more evidently for global timings rather than for the persistence focus. Moreover, we juxtapose (dashed green line) the sum of the foliation timings for reduced dataset (orange line) to the timings of phase a) for obtaining the reduced dataset. For global timings, the dashed green line is always below the blue line meaning that, in our tests, a preprocessing reduction is preferable independently from the number of considered slices. Instead, for the persistence focus, only the case of 4 slices shows the blue line just below the green dashed line.



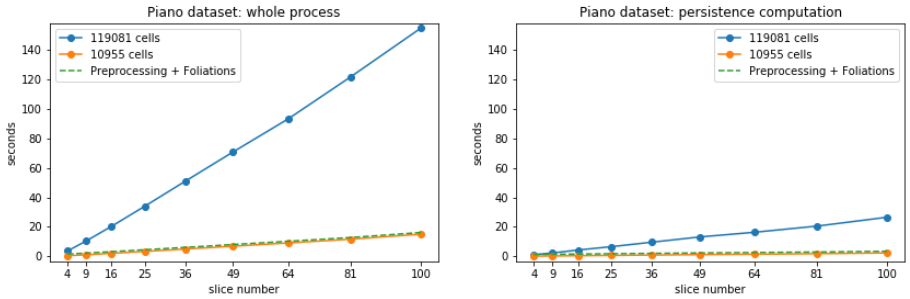
(a)



(b)



(c)



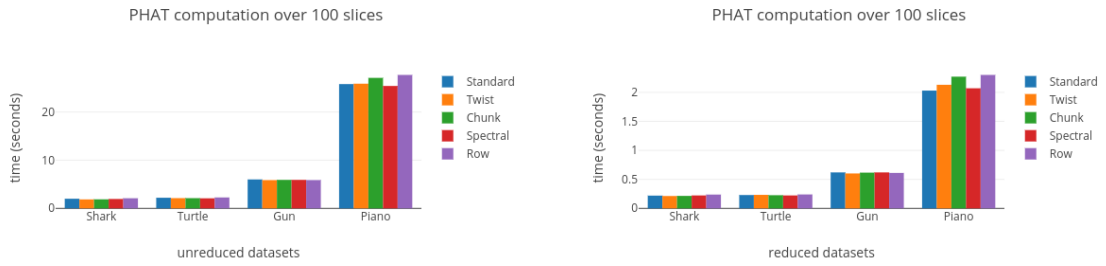
(d)

Figure 6.4: time performances plotted with respect to a number of slices varying form 4 to 100 over the same dataset. Datasets considered are triangle meshes: (a) Shark, (b) Turtle, (c) Gun, and (d) Piano. In all the figures, on the left, performances are indicated in blue for the original dataset and in orange for the corresponding reduced dataset. On the right, we show the same plotting with respect to step 2) in the foliation phase only.

Dataset	Cells	Pairs		Reduction Time	Line Extraction	Foliations Time: Building Pers. input	Foliations Time: Computing Persistence	Foliations Time: Reindexing Pers. output	Foliations Time: Foliations Total
Shark	9491	4744	(81.4)	0.11	0.86	9.04	1.91	1.45	12.42
	1111	554				1.15	0.21	0.84	2.22
Turtle	10861	5426	(8.8)	0.12	0.63	10.21	2.11	1.53	13.87
	1197	594				1.22	0.22	0.84	2.29
Gun	27826	13873	(10.2)	0.28	0.65	27.49	5.65	2.69	35.85
	3144	1532				3.18	0.60	0.99	4.77
Piano	119081	59349	(79.5)	1.14	0.85	118.14	26.56	10.33	155.91
	10955	5286				11.09	2.26	1.65	15.01

Table 6.5: Timings (in seconds) required for computing the persistence pairs on 100 uniformly sampled slices. Datasets are reported by rows. For each triangle mesh, the first row is for the original dataset and the second one for the reduced dataset considered over the same 100 slices. Column *Cells* reports the number of cells in the filtration. Column *Pairs* reports the average number of persistence pairs found per slice. In parantheses, the number of pairs with positive persistence (equal for original and reduced filtrations). Reported timings are subdivided into phases a), for the reduced dataset retrieval (column *Reduction Time*), b), for the retrieval of the 100 slices (column *Line Extraction*), and c), for the foliation method (column *Foliations Time*). The latter subdivided into step 1), for preparing the 100 inputs for computing PH (column *Building Pers. input*) , step 2), for computing PH via the standard algorithm (column *Computing Persistence*), and 3), for detecting positive pairs and reindexing the cell id's in the PH output (column *Reindexing Pers. output*).

In order to show partial results in the different phases of our test, we arbitrarily fix the number of considered slices to 100. In Table 6.5, we report the results obtained when the one-parameter persistence is computed by the standard algorithm implementation of PHAT. We notice that, by reducing the number of cells of approximately one order, we get a one-order reduction on all phases. Results in column *Reduction Time*, phase a), are independent from the number of slices considered. As expected, greater original datasets have higher timings, from 0.11 to 1.14 seconds. Column *Line Extraction* results, phase b), depend on the 100 selected slices with negligible differences between datasets. Both columns are to be added to column *Foliation Total*, phase c), in order to get the results for the entire experiment pipeline. For phase c), total timings for a reduced dataset range from the minimum of 2.22 (Shark triangle mesh) to the maximum of 15.01 seconds (Piano triangle mesh). Whereas, total timings for an original dataset ranges from 12.42 (Shark triangle mesh) to 155.91 seconds (Piano triangle mesh). Partial results in column *Foliations Time* highlight the sum (over the 100 slices) of the partial contributions of steps 1), 2), and 3). The highest contribution is showed in column *Building Pers. Input* for step 1). This is due to our implementation of the sorting and reindexing of facets. We need it to set up the input for computing PH over a slice. Instead, step 2) and 3) give smaller contributions. For step 1) and step 2), results for the reduced dataset reflect the one-order reduction in the number of cells. For step 3), the ratio between reduced and original results is higher due to the different percentage of positive pairs (necessarily the same for both original and reduced datasets). Indeed, in columns *Pairs* we show the average amount of persistence pairs found per slice. We notice that the percentage of positive persistence pairs (in parentheses) over all retrieved persistence pairs (on the left) is more significant for the reduced dataset (second row), thus affecting results for step 3).



(a) (b)

Figure 6.5: timings for 100 slice computations via the five algorithms for persistence implemented in the PHAT library. Original datasets (a) are compared to reduced datasets (b).

However, this also means that our reduction avoids to consider many spurious persistence pairs and this affects the size of the output representation.

In order to address our second issue, in Figure 6.5, we report results limited to step 3) in the foliation phase. Here, we compare the equivalent five algorithms for computing PH applied to the 100 slices case. Timings measured for original datasets (a) are compared to those for reduced datasets (b). Different colors for the bars correspond to the different five algorithms implemented in PHAT. Differences among bars in (a) are negligible with respect to the gap in height between bars in (a) and in (b). Moreover, different datasets have a different favorite PHAT algorithm. This confirms that in the foliation method, the number of slices impacts time performances way more strongly than optimizations over the single slice. Moreover, our tests suggest that a multiparameter reduction strategy such as the one tested as a preprocessing in these experiments, is preferable than focusing on one-parameter persistence optimizations only. Finally, we notice that, in our test, performances of the standard algorithm are comparable to the considered optimizations. This was not expected according to [11]. Our choice of triangle meshes datasets, that is low geometric dimensions with cell stars limited in size, may explain that results.

6.3.4 Discussion

As anticipated at the beginning of the section, the purpose of these experiments was not that of evaluating our reduction impact over an efficient implementation of the foliation method, but rather giving evidence of that impact independently from the implementations or motivations for applying the foliation method. Our tests confirm that the time complexity in the foliation methods is dominated by the number of slices to be considered. Our reduction effectively impacts the kind of linear dependency on the number of slices by simply considering the reduction in terms of number of cells. In particular, our tests show that the proposed preprocessing is effective also for a small number of considered slices, from 4 on. For 100, arbitrarily fixed as the number

of slices considered, our tests confirm that the reduction in terms of number of cells, directly implies advantages in the foliation method computations. Moreover, our tests suggest that a multiparameter reduction strategy such as the one tested as a preprocessing in these experiments, is preferable than focusing on one-parameter persistence optimizations only.

In these tests, we have compared the original and the reduced datasets over exactly the same slices. As a future task, we wish to highlight our preprocessing properties in reducing the number of grades in a multiparameter filtration as highlighted in Section 6.2 for the computation of the persistence module. This would restrict the possible slices to choose and the impact on the foliation method of that reduction in terms of grade is still to be evaluated.

6.4 Evaluation on Real-sized data

In this section, we test time performances of our new algorithm ComputeDiscreteGradient, described in Chapter 4, when applied to simulation data of real size.

6.4.1 Method

Simulation Datasets. The simulation datasets considered for these tests correspond to the seven rows reported in Table 6.6. The first three datasets are triangle meshes, each having three scalar fields defined on the vertices laying on $\mathbb{R}^3$ (see [45] Section 6.2 - Db2 for details on the functions). The last four datasets are collections of time-varying scalar fields defined on a regular grid. For our experimental analysis, we have coupled a subset of the original scalar fields, on a single time step. The Hurricane Isabel WRF Model Data describes the simulation of the Hurricane Isabel. The first dataset $Hurricane_{pt}$ is created by coupling the scalar fields describing pressure (weight of the atmosphere above a grid point) and temperature. The second dataset $Hurricane_{rit}$ still combines the temperature measurement with two scalar fields describing the density of clouds in the atmosphere distinguishing between rain clouds and ice clouds. The turbulent combustion simulation instead is a simulation of temporally-evolving plane jet flames. $Combustion_{vo}$ presents two scalar fields describing vorticity and combustion-generated OH. $Combustion_{hm}$ instead combines the scalar field describing mixture fraction and the one for chi distribution. Hence, the datasets considered are both simplicial and cubical complexes. The number of parameters in the filtering functions of the datasets vary from 2 to 3.

We measure time performances for the discrete gradient retrieval via ComputeDiscreteGradient over a single and a parallel implementation. Starting from the discrete gradient, we compute the filtered Morse complex. This is done by expliciting the boundary maps as described in [92] and discussed in more details in Section 6.2. Moreover, we measure timings for the retrieval of the filtered Morse complex.

6.4.2 Implementation

Algorithm implementation The algorithm ComputeDiscreteGradient has been implemented in C++ based on the MDG library [85]. The latter provides an efficient encoding for both triangle meshes and regular grids, as well as a compact encoding for the discrete gradient. The parallel implementation version is based on OpenMP.

Triangle meshes and filtrations are represented as described in Section 6.1.2.

Regular Grids. A regular grid is a special case of a 3-dimensional cubical complex. It is represented as a graph $G = (N, A)$ where N is the set of cells forming the regular grid and A is the set of incidence relations between the cells in N. Representing a regular grid is a much easier task than representing unstructured meshes (such as triangle meshes). The regular distribution and connectivity of the cells in G makes it possible to encode the topology of the complex implicitly. By enumerating the cells in N, we can extract any relation in A using index calculations without any overhead.

Discrete gradient representation. In the case of triangle meshes, the discrete gradient is encoded in the same way as described in Section 6.1.2. In the case of a regular grid, the discrete gradient is encoded by assigning a Boolean value to each arc α in A, where the two cells connected by α are paired in V. Then, the discrete gradient is encoded as an array of bits of length $|A|$. As discussed in Section 4.6.3, this ensures that the time for accessing to the discrete vector (s,t) given s is linear in the number of cofacets of s. Since the number of cofacets of a single cell, in a cubical complex of low dimension, is considered constant, we consider to have constant access to (s,t).

6.4.3 Results

Experiments have been performed on a MacBook Pro with a 2.8GHz quad-core processor and 16GB of memory. We can see that the compression factor obtained goes from 40x to 7.6x in the worst case. The efficiency provided by our algorithm in computing the discrete gradient is remarkable. We have compared a single threaded implementation with a parallel implementation based on OpenMP. On average, the parallel implementation achieves a 2.7x speedup when working on triangle meshes and a 3.6x speedup when working with regular grids.

Dataset	Size	Fields	$\lvert M \rvert$	Time G.		Time M.
				Single	Parallel	
Neptune	12M simpl	3	1.4M (8x)	2.1m	0.8m (2.5x)	7.3m
Statue	30M simpl	3	2.1M (15x)	6.1m	2.1m (2.9x)	12.1m
Lucy	84M simpl	3	5.6M (15x)	15.4m	5.6m (2.7x)	22.9m
Hurricane$_{pt}$	$[500 \times 500 \times 100]$	2	12.2M (16x)	46.1m	11.2m (4.1x)	13.4m
Hurricane$_{rst}$	$[500 \times 500 \times 100]$	3	5.7M (40x)	47.8m	12.2m (3.9x)	17.0m
Combustion$_{vo}$	$[480 \times 720 \times 120]$	2	43.1M (7.6x)	58.2m	18m (3.2x)	23.7m
Combustion$_{hm}$	$[480 \times 720 \times 120]$	2	45.9M (7.3x)	67.2m	18.6m (3.5x)	20.1m

Table 6.6: Results obtained computing the discrete gradient and the boundary maps on three triangle meshes and four volumetric images. For each dataset, we indicate its size (number of cells for triangle meshes or resolution of the volume datasets), number of scalar values for each point (Fields), number of critical cells obtained ($|M|$), and timings required for computing the discrete gradient (Time G.) and the boundary maps (Time M.).

6.4.4 Discussion

The algorithm is easy to parallelize and the actual performances have been tested on real-sized datasets with realistic simulation filtering values. Our tests suggest that the pipeline from a dataset with a vector-valued function and the filtered Morse complex is feasible for data of interesting sizes. Our reduction in terms of cells might be a first step towards an efficient computation of multidimensional persistent homology. This evidences along with the results obtained in Section 6.3 suggest a strong practical advantage in exploiting ComputeDiscreteGradient as a preprocessing for multipersistence computations via the foliation method. However, the tests of this section compared to the results in Section 6.2 show an important gap between what can be reduced by ComputeDiscreteGradient and what can be exploited in the computation of the persistence module. As last remark, we observe that the *Time M.* column reports, generally, higher timings than column *Single* in *Time G.*. This suggests that, optimizations in the retrieval of the Morse complex when a discrete gradient is already given, would lead to even better performances.

As a conclusion to this chapter, the practical advantage of considering our algorithm as a preprocessing for multipersistence is confirmed by all tests. The proposed method scales well with dataset sizes. However, the practical problems of performing multipersistence computations, mainly depend on the invariant to be computed. There are good perspectives for the foliation method but still very limited perspective for the persistence module retrieval.

In the next chapter, we discuss properties of algorithm ComputeDiscreteGradient, not simply as a preprocessing for multipersistence, but as a visualization mean.

7 A discrete approach to multivariate data visualization

In this chapter, we present our contribution in the context of data visualization of multivariate fields. Our contribution of this part is, at the moment, disconnected from the previous parts of the thesis. However, in our future works section, we will see how the two parts suggest promising research topics for the future. Both theoretical and experimental aspects for this part of the thesis have been developed in collaboration with Federico Iuricich (University of Maryland (MD), USA). The main focus of this part consists in showing the visualization perspectives related to a discrete gradient compatible with a multiparameter filtering function such as the one retrieved by the algorithm of Chapter 4. In Section 7.1, we review some preliminary notions on Morse Theory applied to data visualization. In this part, we see the multiplicity of the notions in moving from the univariate to the multivariate setting and we provide the basis for the classification of state-of-the-art methods in Section 7.2. In Section 7.3, we present our contribution. In Section 7.4, we discuss limitations of our visualization by critical cells and we report our observations on the compatibility of critical cells and Pareto-related notions from the discrete, the smooth, and the PL settings.

7.1 Preliminary notions in topology-based visualization

In this section, we review the necessary definitions to generalize to the multivariate setting the notion of a critical point in the smooth case. This provides the necessary background for describing the literature in topology-based visual analytics of multivariate data in Section 7.2 and discussing our contribution in Section 7.3 and Section 7.4. In doing this, it is necessary to move to the smooth counterpart of discrete Morse Theory. In Section 7.1.1, we quickly describe some preliminaries on critical point notions in the univariate case. The relation between Morse Theory and, simple homology and persistent homology motivates our point of view. This part it is not supposed to be an actual background, rather to provide motivations for our on-going research in the field.
In Section 7.1.2, we introduce the main notions which apply to the multivariate case, namely the *Jacobi* and the *Pareto critical point* definitions. For each of them, we review the main results for relating those notions one another and to the notion of homology. The definitions reported in this section refer, in their content, to the work of Smale [179]. The terminology which is adopted slightly differ from Smale's ones to be consistent with the state-of-the-art in multivariate data visualization.

7.1.1 Critical points in the univariate setting

The main tool provided by topology in the context of data visualization is Morse Theory [142, 139], a branch of Mathematics in between Topology and Differential Geometry which studies differential manifolds on the base of properties of smooth scalar functions defined over it. For both Morse Theory and its discrete counterpart called discrete Morse Theory (see Section 1.4) which constitutes the main focus of this thesis, there is no comprehensive generalization to the multivariate case.

Morse Theory focuses on the subclass of smooth functions whose *critical points* (points were the gradient is null) are isolated and non-degenerate, that is the Hessian matrix in that point has maximal rank. Those functions are called *Morse functions*. For each critical point, the number of negative eigenvalues of the Hessian matrix defines the *critical index*. The index is the analogue for the dimension of a critical cell in discrete Morse Theory [87]. *Weak* and *strong Morse inequalities* (smooth analogues of Formulas (1.4.2)) relate critical points of Morse functions to the homology of the shape.

Morse-based subdivisions. By visualizing critical points, and their reciprocal connections through integral lines and regions of influence, Morse Theory is at the base of topological consistent segmentation techniques and compact field representations we review in Section 7.2. The level sets of a Morse function $\phi : D \longrightarrow \mathbb{R}$ provides a way to subdivide a topological space D. The abstract notion of a *Reeb graph* is defined in [159] with respect to a Morse function, then first introduced in the context of Computer Graphics in [172].

7.1.1 Definition. Given a a Morse function $\phi : D \longrightarrow \mathbb{R}$, the *Reeb graph* $R(\phi)$ is the quotient space of D obtained by considering one equivalence class -one node- for each connected component in a level set with respect to ϕ. The topology of $R(\phi)$ is the quotient topology inherited from D. Thus, the graph structure of $R(\phi)$ is given by setting an arc between any two nodes whose respective connected components are adjacent in D.

Nodes from a level set merge or separate in correspondence of critical level sets.

Another subdivision is obtained by following the discrete field associated to a Morse function. The paradigm consists in simplifying the representation of the scalar field under study by considering as equal all points in the domain with the same behavior with respect to the gradient of the scalar field of interest. Each point admits an integral line through it. Separatrices and integral lines of the gradient field are interpreted as *ascending paths* when traversed according to the gradient field orientation, and *descending paths*, otherwise.

7.1.2 Definition. The *ascending (descending) Morse complex* of a Morse function $\phi : D \longrightarrow \mathbb{R}$ over a smooth manifold of dimension d is the d-dimensional cell complex where each cell is defined as follows. The points in D whose ascending (descending) path ends in the same critical

point p of index k are considered part of the same $k(d-k)$-cell associated with p. The cell corresponding to a critical point p is a face of the cell corresponding a critical point p', if the gradient field of ϕ admits a separatrix from p' to p.

By subdividing a domain D into its cells in the ascending or descending Morse complex, we get a Morse-based segmentation. As a remark, all along this thesis we have dealt with the Morse complex of Definition 1.4.8 which is a discrete counterpart of a descending Morse complex. If each non-empty intersection of a descending and an ascending cell is transversal, their connected components define the *Morse-Smale MS-complex*, which encodes both dual constructions, ascending or descending. In the MS complex, two points are equal if their respective integral lines share the same critical extrema points in their support.

With respect to the gradient field of a Morse function, critical points are equivalently:

- null points *(singular points)*;
- ending points of ascending/descending paths *(stationary points)*.

We will see in Section 7.1.2 how these two properties separate in the multivariate case.

Morse-based simplifications. Additionally, there is a strong interplay in between one-parameter persistent homology and Morse Theory. In the preceding parts of the work of this thesis, we have looked at Morse Theory as a way of decrease the computational persistent homology costs. Here, we focus on how persistence supports Morse-based visualization techniques. Indeed, by considering the filtration obtained by sublevel sets with respect to a Morse function, one obtains that each critical point of index k from 0 to the dimension d of the differential manifold, corresponds to, either the birth of a k^{th}-homology class, or the death of a $(k-1)^{\text{th}}$-homology class in the first sublevel set where the critical point appears. Critical points corresponding to birth or death of short-living topological features are removed from the Morse-based subdivision. Hence, one-parameter persistent homology provides a criterion to simplify, in a topologically consistent way, any Morse-based subdivision.

This simplification paradigm has no straightforward generalization to the multivariate case since there is no (or there are too many) notions of persistence to measure the lifespan of a homology feature along a multiparameter filtration. Moreover, as seen in Remark 5.1.5 for the discrete case, homology changes are not necessarily given by a new cell entering a multiparameter filtration, so the relation between critical cells and multiparameter persistence need further studies. In that direction, we gave our contribution by proposing the optimality notion of Chapter 5 which can help in relation critical cells and homology changes along a multiparameter filtration.

7.1.2 Critical points in the multivariate setting

In this section, we introduce two possible generalizations of critical points of a scalar field to a multivariate setting. In the last part of this section, we see in Theorem 7.1.6 how to relate one of those critical point generalization to multiparameter persistent homology (see Section 1.6).

We recall from Section 7.1.1, that a critical point is both *singular* and *stationary* for the gradient field of a scalar field. For the definitions in this section, we follow Smale's work [179] as adapted in [136]. In the following, we set $(\phi_1, \ldots, \phi_n) = \phi : D \subseteq \mathbb{R}^d \longrightarrow \mathbb{R}^n$ to be a continuous function defined over a smooth manifold D, with $n \leqslant d$. To simplify the review, we assume D to be embedded in $\mathbb{R}^d$ so that points in D are expressed by global coordinates $x = (x_1, \ldots, x_d)$. In this section, we denote by $\nabla\phi_i(x)$ the gradient of ϕ_i in $x \in D$, for i from 1 to n.

The property of a critical point of being *singular* for a vector field is generalized to the multivariate setting by the following definition.

7.1.3 Definition (Jacobi point)**.** Let $\phi : D \subseteq \mathbb{R}^d \longrightarrow \mathbb{R}^n$ with $n \leqslant d$ be a C^2-function on a smooth manifold D. A point $x \in D$ is a *Jacobi point* if

$$\exists \lambda_1, \ldots, \lambda_n \in \mathbb{R} \text{ such that } \sum_i \lambda_i \nabla\phi_i(x) = 0.$$

In the univariate case, the fact that critical points were singular was at the base of merging and splitting of nodes in a Reeb graph (Definition 7.1.1). We will see in Section 7.2.2 how Reeb graphs generalize to *Reeb space* [78].

The property of a critical point of being *stationary* for a gradient field, i.e., the critical point is an extremum point of ascending/descending paths, is generalized to the multivariate setting by the following definition implied by Proposition 3 in [136].

7.1.4 Definition (Pareto critical point)**.** Let $\phi : D \subseteq \mathbb{R}^d \longrightarrow \mathbb{R}^n$ with $n \leqslant d$ be a C^2-function on a smooth manifold D. A point $x \in D$ is a *Pareto critical point* if

$$\exists \lambda_1 > 0, \ldots, \lambda_n > 0 \in \mathbb{R} \text{ such that } \sum_i \lambda_i \nabla\phi_i(x) = 0.$$

In the univariate case, the fact that critical points were stationary was at the base of ascending and descending Morse complex subdivisions (Definition 7.1.2). In the multivariate case, there is no counterpart to the ascending or descending Morse complex. We will see in Section 7.2 that finding subdivisions based on Pareto notions is an active research field.

Smale in [179] distinguishes points satisfying Definition 7.1.4 from what is commonly meant in Game Theory or Economics contexts by *local Pareto optimum*. The meaning of a local

Pareto optimum $p \in D$ is that there is no possibility of moving from p and increasing all scalar components of ϕ at the same time.

7.1.5 Definition (Pareto optimum)**.** Let $\phi : D \subseteq \mathbb{R}^d \longrightarrow \mathbb{R}^n$ be continuous function on a smooth manifold. We say that $x \in D$ *dominates* $y \in D$ *through* ϕ if $\phi(x) \succeq \phi(y)$ (dually, y is dominated by x through ϕ). A point $x \in D$ is called *local Pareto optimum* if it admits a neighborhood whose points do not dominate x.

This corresponds to an optimality notion valid in an *ascending framework*. Reciprocally, one can consider x as a local Pareto optimum if it admits a neighborhood with no points dominated by x and this provides the analogue for the *descending framework*. The formal relation between *local Pareto optima* and *Pareto critical points* of Definition 7.1.4 is investigated in the work of Smale [179] for the smooth setting, and in the work of Lovison [136], for the PL setting. It is shown that, under certain transversality and genericity conditions, Pareto critical points include local Pareto optima from both the ascending and the descending frameworks and many more points.

As we can see in Figure 7.1, in the case $n = 2$, the Definition 7.1.4 capture a special case of Jacobi point which is satisfied when the gradients of ϕ_1 and ϕ_2 are collinear but point in opposite directions. Indeed, for points on the red curves, values of the two scalar functions increase in opposite directions, whereas, for points on the green curves, values of the scalar functions increase along the same direction. If we, change the sign of the first component ϕ_1 measuring the x coordinate, we get that green arcs become Pareto critical points. Moreover, we notice that the two red arcs in the outer circle in Figure 7.1 have a different local character. Indeed, the right-upper arc collects Pareto optima points which are also Pareto optima with respect to the ascending framework. Whereas, the left-bottom arc collects Pareto optima points with respect to the *descending framework*. The converse holds for the inner red arcs. In the same example we notice how critical points of the single scalar fields bound both the sets of Pareto critical points and Jacobi points. This is not always the case. What holds in general is that both definitions of Pareto critical point and Jacobi point specialize to critical point in the univariate case.

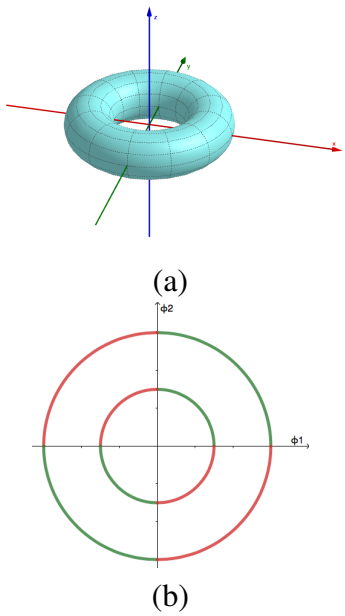


Figure 7.1: (a) a smooth torus embedded in the 3D Euclidean space. (b) the 2D projection of the torus on the xy-plane delimited by red and green curves. In green, Pareto critical points with respect to the function $\phi_1(x,y,z) = x$ and $\phi_2(x,y,z) = y$. Both green and red points are Jacobi points with respect to the same functions.

In the univariate setting, Pareto critical and Jacobi points are equivalent notions. In general, Pareto critical points are no longer equivalent to Jacobi points. As a weaker property, we have that a Pareto critical point is a Jacobi point. Moreover, we can retrieve Jacobi points from Pareto critical points. We take ϕ as in Definition 7.1.4. Then, we take $\phi^{(i,-)}$ the function whose components are those of ϕ but the signs of the first i components is the opposite one, i.e., $\phi_k^{(i,-)} = -\phi_k$, with $k = 1, \ldots, i$. We denote by $P^{(i,-)}$ the set of all Pareto critical points with respect to $\phi^{(i,-)}$ with i from 0 to $n-1$. We denote by J the set of all Jacobi points of any of the functions $\phi^{(i,-)}$ with i from 0 to $n-1$. We get that

$$J = \bigcup_{i=0}^{n-1} P^{(i,-)}.$$

This means that Jacobi points can be retrieved from Pareto critical points.

Recently, Cerri and Frosini [46] have related some multipersistence properties to Pareto critical points. Here, we briefly report the main result for our purposes and, in doing this, we warn the reader that we change the terminology used in [46] for the sake of coherence with the rest of the thesis. In [46], the rank invariant is called *multidimensional size function* since it is defined in a context of Size Theory. Moreover, a Pareto critical point is called *pseudo-critical point*. Theorem 2.13 in [46] can be stated in the following terms. Recall from Definition 1.7.2 the definition of the rank invariant and its source set $\Delta^n_{<,\infty}$.

7.1.6 Theorem. *Let $\phi : D \subseteq \mathbb{R}^d \longrightarrow \mathbb{R}^n$ be a continuous function on a smooth manifold D. Let M be the multiparameter filtration obtained by the sublevel sets of ϕ and $(u,v) \in \Delta^n_{<,\infty}$. If (u,v) is a discontinuity point for the rank invariant* rank_M *then*

- *either u is a discontinuity point for* $\mathrm{rank}_M(\cdot, v)$*;*
- *or v is a discontinuity point for* $\mathrm{rank}_M(u, \cdot)$.

Moreover, in the former case, a projection $\pi : \mathbb{R}^n \longrightarrow \mathbb{R}^p$ exists such that u is either a Pareto critical point or a non-differentiable point of $\pi \circ \phi$. In the latter case, a projection $\pi : \mathbb{R}^n \longrightarrow \mathbb{R}^p$ exists such that v is either a Pareto critical point or a non-differentible point of $\pi \circ \phi$.

This result is of crucial interest for further developments of the work of this thesis. Indeed, the result relates the notion of Pareto critical point, which generalizes univariate critical points to the multivariate setting, to a property of an invariant for multipersistence. As already discussed, there is no multivariate counterpart to discrete Morse Theory. In our opinion, multipersistence might help in connecting the Pareto properties of the smooth setting to the discrete gradient compatible with multiparameter filtrations.

7.2 State of the art in topology-based visualization

Visualization aims at capturing and rendering relevant features in data. Challenging tasks in Visualization comprise the integration of both scientific and information visualization, and the detection of relevant features [116]. In particular, the demand for meaningful concise representations interest different visualization areas such as the analysis of spatio-temporal, multi-model, multi-modal, vector-fields data and, for what is our concern, multi-variate data [121]. Other comprehensive surveys in visualization techniques can be found in [195, 91]. Topology provides tools which aim at capturing the significant information of data in a simplified object and in a robust way with respect to noise. A recent survey on topological methods for the visualization context is [109]. First, we review topology-based methods for univariate scalar fields. In other words, our setting is that of a scalar field data over a shape, that is a domain with geometrical attributes. Afterwards, we address topology methods for analyzing multivariate data, that is many scalar fields at a time with possibly heterogeneous values.

7.2.1 Univariate data visualization

In the univariate case, that is when a single scalar field is taken into account, topology has already revealed its advantages. In particular, it provides simplified and robust tools for feature-tracking, isosurface traversal and segmentation of the domain based on Morse Theory. We classify the methods in univariate topology-based data visualization into two main paradigms:

- *value-driven* are those methods referring to the notion of *Reeb graph* [159] (Definition 7.1.1). The retrieval of the Reeb graph provides a compact representation of the scalar field based on its level sets. For this reason, we call the topological methods related directly or indirectly to the Reeb graph as *value-driven*;
- *monotony-driven* are those method referring to the notion of an ascending/descending *Morse complex* (Definition 7.1.2). The paradigm consists in simplifying the representation of the scalar field under study by considering as equal all points in the domain with the same behavior with respect to the gradient of the scalar field of interest. For this reason, we call the topological methods related directly or indirectly to the Morse complex as *monotony-driven*.

7.2.1.1 Value-driven Paradigm

Most of value-driven applications refer to the notion of *contour tree*, which is a Reeb graph in the special situation of a simply connected domain D. Contour trees [129] are easier to compute and their retrieval can be performed in any dimension of the domain [38]. Moreover, a method

that makes a Reeb graph into a contour tree for volumetric meshes only is proposed in [189]. Among the possible applications to visualization, we report those for isosurface traversal [129], volumetric rendering simplification [39], transfer function design for volume rendering [192], and similarity estimation among different isosurfaces [186, 110]. Comprehensive surveys of the Reeb graph algorithms applied to visualization can be found in [19, 157]. We subdivide value-driven techniques into:

- *vertex-based* [172];
- *approximate* [18, 110, 196];
- *critical-based* [158, 72]

Vertex-based. The first combinatorial algorithm for 2D meshes and PL scalar functions for the Reeb graph retrieval is proposed in [172]. It considers a level set for each vertex in the domain and then use adjacency relation on the domain to define the arcs in the Reeb graph.

Approximate. Approximate algorithms for the Reeb graph retrieval are proposed in [18, 110, 196] for PL domains. These methods speed up the computations. Their result is not precise but depends on a tolerance parameter set by the user which defines how far apart to sample values on the target space. Then, connected components of the respective level sets are computed and then connected according to adjacency relation in the domain D.

Critical-based. The topology of one level set can only change when changing the scalar value by moving across some critical value. Algorithms exploiting the role of critical points has shown to be more accurate and fast [158, 72]. They compute, for each critical point, the connected component of the level set passing through it. Next, it suffices to consider the adjacencies in the segmentation induced by the critical level sets cuttings in the domain to retrieve adjacency among nodes.

7.2.1.2 Monotony-driven Paradigm

The second abstract structure exploited in topology-based approaches is the *Morse complex* in either the *ascending* or *descending* forms. We subdivide monotony-driven techniques into:

- *ascending or descending PL methods* [60, 59, 137, 105, 97, 68, 140, 191, 163] where scalar fields over vertexes are extended to higher dimensional cells linearly with respect to the barycentric coordinates of the cell. These techniques refer to the notion of PL critical cell [6, 7, 79];

- *Morse-Smale complex PL methods* [182, 79, 77, 24, 156, 14, 165, 166, 5] where scalar field is PL-extended from vertexes and the purpose is that obtaining a Morse-Smale domain decomposition which combines ascending and descending decompositions [79, 77];

- *discrete methods* [43, 123, 101, 162, 102, 99, 173, 175, 193] where scalar fields over vertexes are extended to higher dimensional cells by the maximum-rule in Definition 3.1.2. These techniques refer to discrete Morse Theory [87].

Ascending or descending PL methods. Methods for PL functions refer to the definition of critical point according to Banchoff [6, 7], then, extended to higher dimensional spaces by Edelsbrunner et al.[79]. A comprehensive survey for Morse-based segmentation techniques can be found in [61]. Algorithms based on piecewise linear Morse Theory, and compute the ascending (or descending) Morse complex by growing the top cells, here called regions, from seeds located at the minima (or maxima). [60, 59, 137, 105]. The construction of another approach, called stable flow complex [97], can be obtained through a region-growing approach [68]. The watershed transform is an alternative framework to Morse Theory analogue to the ascending Morse complex which segment the domain according to the so-called topographic distance. It has been first defined for grey-scale images. [140, 191]. The watershed transform has also been defined for C^2-differentiable functions over a connected domain, having the property that the gradient is non-null everywhere except possibly at some isolated points. This includes Morse functions.[140, 163].

Morse-Smale complex PL methods If each non-empty intersection of a descending and an ascending cell is transversal, their connected components define the Morse-Smale (MS) complex, which encodes both dual constructions, ascending or descending. In the MS complex, two points are equal if their respective integral lines share the same critical extrema points in their support. The notion of Morse-Smale complex has found applications to different fields [130, 103, 100, 104] such as fluid dynamics and distance field definition. However, the Morse-Smale complex construction is feasible only up to dimension 3 in the domain with some limitations in feasibility over real-size data. The notion of *Quasi Morse-Smale complex* as the PL counterpart to the MS complex is introduced in [79, 77] for PL domains. Algorithms for the retrieval of the Quasi Morse-Smale complex can be subdivided according to the kind of domain. For triangular and tetrahedral meshes, we have [182, 79, 77, 24, 156, 14]. For regular grids, we have algorithm computing derivatives numerically [5] or analitically [165, 166].

Discrete methods. Methods for discrete functions refer to the definition of critical point according to Forman's discrete Morse Theory [87]. Methods from discrete Morse Theory allows to retrieve all the Morse and Morse-Smale cells in a derivative-free way. The subdivision is obtained by traversing the V-paths, that is discrete analogues for integral lines available when the discrete

gradient is computed. The algorithms of this kind [43, 123, 101, 162, 102, 99, 173, 175, 193] start from a discrete scalar function f defined over the vertices of a cell complex, and aim at constructing a discrete gradient that best fits function f. Discrete methods are dimension-agnostic and they have shown practical advantages in computational feasibility over real-size data with respect to PL-methods. Our method, described in Section 7.3.2, constitutes a multivariate counterpart of those algorithms.

7.2.2 Multivariate data visualization

Recently, multivariate data have become a more and more challenging task for Data Analysis. The need for simplified representations and the search for significant features has got more and more demanding. Likewise for the univariate case, the aim of topology-based approaches is to offer robust and qualitative tools for rendering, segmenting and analyzing multivariate data. A possible approach consists in considering juxtaposed modalities of multiple linked views [194]. This way however, the relationships between the different scalar fields are not taken into account or left to the user to explore. Hence, for topology-based techniques, the challenge becomes to provide features which, not simply belong independently to each single scalar field but somehow capture their relationships.

A first topological method developed for the two scalar fields over a 3D domain can be found in [168] where the method called *comparison of largest contour* is first set and then, generalized to multifields in [167]. In the bivariate case, the two field's contour trees are independently computed and simplified into a largest contour segmentation. By area measure of spatial overlaps, contour similarities of different fields are computed and represented as an edge-weighted bipartite graph. The drawback is that the selection of the contours to analyze as well as the active study of relationships among the original fields is left to the user.

We review how the two univariate paradigms, namely the *valued-driven* and the *monotony-driven* finds generalization to the multivariate case to address the components relation-aware task.

7.2.2.1 Value-driven Paradigm

We classify in this class those techniques which refer directly or indirectly to the topological changes along level sets in the multivariate data. The main topological structure these methods refer to is the Reeb space, that is the multivariate analogue of a Reeb graph. The definition of the Reeb space is proposed in [78] for PL functions. The Reeb space $R(f)$ of a *PL* function f (generic according to [78]) defined over a d-manifold D and taking values in $\mathbb{R}^n$ is the quotient space $D/\sim$ with $p \sim q$ if and only if p and q belong to the same connected component of a single level set of f. The topology of $R(f)$ is the quotient topology inherited from D. This no longer leads to a graph structure but to a cell complex structure whose dimension depends on n and

two cells are adjacent if their corresponding connected components in the level set are. Many different structures have been proposed so far. Analogously to what already done for univariate visualization, we subdivide the value-driven techniques into:

- *approximate methods* [178, 36, 73, 185, 125] to retrieve the Reeb space;
- *Reeb-related methods* [76, 37, 124] as those methods retrieving auxiliary attributes of the Reeb space;
- *critical-based methods* as those methods exploiting PL Jacobi cells as critical cells for computing the Reeb-related representations [50, 51] or as a mean for numerical approaches [152, 151, 150].

Approximate methods. A simplicial complex called *Mapper* is introduced in [178] in the context of Topological Data Analysis and address the problem of multiple different entities of the Reeb space mapped to the same region in the target space $\mathbb{R}^n$ and provide a partial clustering technique for multivariate data. Each vertex corresponds to a connected component in the function target space pulled back on the domain. A simplex is defined when the vertices have corresponding intersecting components. This procedure can be performed at different resolution, that is according to the coarseness of the starting coverings to discriminate between persistent and artificial features. The Joint Contour Net [36] is proposed as an approximate version of the Reeb space which depends on a fixed tolerance in an analogous way to univariate approximate approaches. Joint Contour Nets are applied in [73] to nuclear scission processes and in [185] to Lattice Quantum Dynamics, a physical approach to model strong nuclear force.

Reeb-related methods. Fiber surfaces [37] generalize the notion of isosurfaces and they were initially introduced as a valid feature towards visualization via multifield transfer functions [125]. An algorithm for their exact retrieval is proposed in [124], where the method allows for a user-guided selection of values of interest. In analogy with the univariate case critical-based approaches, other methods are based on a generalization of critical points to the multivariate case called *Jacobi set*. The Jacobi set of n real-valued Morse functions defined on a common d-manifold is the set of all points where the function's gradients are linearly dependent, which is directly related to the rank of the Jacobian matrix. It follows that when $n > d$ all points become trivially Jacobi, so the interesting setting requires $n \leqslant d$. Alternatively, the Jacobi set is the closure of the set of all critical points of one scalar field restricted to all preimages of the remaining scalar fields. In analogy with the univariate case, nodes in the Reeb space determine a topological change among the level sets when they belong to the Jacobi set. The Jacobi set is defined for PL functions in [76] together with the main reference algorithm for the Jacobi set retrieval.

Critical-based methods. The PL definition of the Jacobi set is adapted to the Fiber surface retrieval in [187]. The algorithm returns a Reeb Space segmentation, simplified by adapting the algorithm in [124] to the extraction of Jacobi Fiber surfaces, that is fiber surfaces containing Jacobi points. The method is used to perform interactive clutter reduction. In [50] the Jacobi structure is introduced as the Jacobi set mapped to the target space of f. An approximate version by means Joint Contour Nets [51] allows to define a topology consistent simplification scheme based on Reeb Skeleton, the dual of the decomposition via Jacobi structure, and Range and Geometric Measures to impose a total order among such components. Jacobi inspires also numerical techniques aimed at rendering a single volume function out of the multiple scalar fields in a way compatible with the relationships among scalar fields. These methods are based on the local and global comparative measures introduced in [76] for a domain whose dimension is less than the number of fields involved and share the common trend of assigning scalar value 0 to points in the Jacobi set, when the definition is applicable. In [150], the comparative measure is used to define a variation density function which is applied to isosurface selection. Then, in [152], the same idea is reformulated for any number of fields into a gradient-based measure for the evaluation of the local coherence among the different scalar fields. The global comparison measure defined in [76] is also applied to define a simplification of Jacobi sets robust to noise in [151].

7.2.2.2 Monotony-driven techniques.

Monotony-driven techniques have not found an exact transposition of the subdivision by ascending/descending Morse complex (Definition 7.1.2) in the multivariate case since monotony loses meaning in that case. However, in analogy with the univariate case, critical points can be described as stationary points with respect to a single vector field which replaces and agrees with the multiple gradient fields given by the data components. The aim is to subdivide the domain into regular regions, where it is possible to describe the behavior of the different scalar fields as regions of equal behavior with respect to a single vector-field, and critical regions where this cannot be done. All monotony-driven techniques are based on Pareto-related notions. In our review, we classify methods according to the Pareto notion they are related to:

- *Pareto critical-based methods* [136] are those methods referring to the definition of *Pareto critical point* (Definition 7.1.4);
- *Pareto optima-based methods* [180, 113, 114, 112] are those methods referring to the definition of *local Pareto optimum* (either in the ascending or descending, or both frameworks of Definition 7.1.5) .

Pareto critical-based methods. In the PL setting, Pareto critical points are detected in [136] by checking directly the linearity condition in Definition 7.1.4. Pareto critical points have

the advantage of admitting, under certain generic and transversality conditions for ϕ [179], a multivariate analogue for the critical index (in the univariate case, the critical index defines a critical point as a maximum, minimum or a saddle). Algorithms cannot yet fully capture and exploit this stratification of the set of Pareto critical point performed by the critical index. Whereas local Pareto optima have the advantage of being decidable by analyzing the function values within a neighborhood of the inspected point. The discrete approach we describe in Section 7.3 is empirically found to be closer to Pareto critical-based methods than Pareto optima-based methods. However, the classification of our method needs further insights.

Pareto optima-based methods. The notion of local Pareto optimum is exploited in the barrier tree (a forest in the multi-variate case) generalization [180] to poset-valued landscapes. That technique is limited to graphs but it provides the only fully discrete monotony-driven subdivision for multivariate data. The function values are attributes of the vertexes and they are never extended to edges in the graph. The neighborhood of a vertex is formed by the adjacent vertexes. Basins, which are the leaves in the barrier tree, are generated by *local Pareto minima* that is vertex not strictly dominating vertexes in their neighborhood. In this sense, leaves in the barrier tree are generated by local Pareto optima points in the descending framework in the sense of Definition 7.1.5. This can be thought of as a generalization to the multivariate setting, for graph domains, of an ascending segmentation like the ones described in Section 7.2.1.2. Saddle points are found in a watershed-like approach as points where two basins merge. However, theoretically saddles should not be single points but set. So, only one representative element is extracted. Saddles and basins are arranged in a dendrogram representation called barrier tree. The discrete approach of barrier trees inspires the PL notion of Pareto cell introduced in [113] where the domain is a simplicial complex embedded in a d-dimensional space and the vector-valued function ϕ takes values in $\mathbb{R}^n$ with $n < d$. In [113], a point is a *local minimal point* if it admits a neighborhood with no dominated points. A point is a *local maximal point* if it admits a neighborhood with no dominating points. A point is a *local optimal point* if it is both minimal and maximal local Pareto point. A cell is a *minimal/maximal/optimal Pareto cell* if the same character holds for all points in its interior where, for PL functions, the character can be checked over a single interior point. Those definitions are applied in [114] to the retrieval of a Reachability graph, which decomposes the domain into, regular regions, subdivided into either ascendent-sets or descendent-sets, and Pareto extrema regions, as collections of minimal, maximal and optimal cells. This is the only example of PL-subdivision in the monotony-driven paradigm. We remark that, maximal and minimal Pareto cells in [113] have, in their interior, local Pareto optima points in the sense of Definition 7.1.5 in the ascending framework or in the descending one, respectively. Whereas optimal Pareto cells in [113] are those cells with interior points that are local Pareto optima points in both frameworks. In this sense, they, not simply transpose the Barrier tree construction for higher-dimensional domains in the PL setting, but they also capture both the ascending and the descending frameworks. There is no proved relation between Pareto minimal/maximal or optimal cells in the PL setting presented in [113] and smooth Pareto critical points as defined

in [136]. However, Pareto cells, as defined in [113], are shown in [111] to be also Jacopi point as defined in the PL setting in [76]. This allows Huettenberg et al. in [114] to remove noise from the reachability graph according to an adaptation of the simplified techniques for Jacobi sets in [151]. This method is applied to quality control in car manifacturing in [111].

The *TTK: Topology Toolkit* [188] is a publicly available platform, integrated with the Visualization tool Paraview. TTK implements many of the visualization instruments here reviewed, such as: critical points, integral lines, persistence diagrams, persistence curves, merge trees, contour trees, Morse-Smale complexes, fiber surfaces, Jacobi sets, Reeb spaces.

7.3 Critical features in the multivariate discrete case

In this section, we discuss our results in the context of multivariate data visualization. In Section 7.1.1, we have recalled the relationship between Morse Theory and univariate data visualization. Here, we present our results, in [115], obtained by applying the algorithm ComputeDiscreteGradient presented in Chapter 4 to visualization purposes. As opposed to the preliminary notions presented in Section 7.1, our setting is fully discrete. First, we show a toy example of a discrete gradient retrieved by the algorithm ComputeDiscreteGradient to motivate our approach. Then, we present our experimental results to test the significance of visualization.

7.3.1 A motivating example

As already seen in Section 7.1.1, persistent homology plays a crucial role in univariate data analysis. Indeed, it provides a criterion to simplify critical cells in a topological consistent way. In this section, we see, through an example, how this paradigm fails when moving from the univariate and the multivariate case. In doing this, we motivate the definitions in the following section and the experiments discussed in Section 7.3.2.

In Figure 7.2, we consider two scalar fields f_1 and f_2, each with a compatible gradient depicted by arrow, for indicating discrete vectors, and a symbol "x", for indicating critical cells. We focus on the red function f_2. In the univariate case, it is the persistence descriptor (the persistence diagram or the persistence barcode) obtained by studying the sublevel sets under f, to suggest the threshold value for persistence in order to discriminate the signal from the noise. Each critical cell corresponds to an extremal value for the life-span of a homology class. By applying the elder rule [80], we can pair critical cells so that each pair bounds the life of a homology class. In the example, the persistence pairs are given by the critical vertex with number 3 paired to the critical edge with number 6, and the critical vertex with number 2 paired to the critical edge with number 7. Removing noise from function f_2 according to the persistence of pairs means successively canceling a persistence pair and updating the remaining incidence structure in the Morse complex

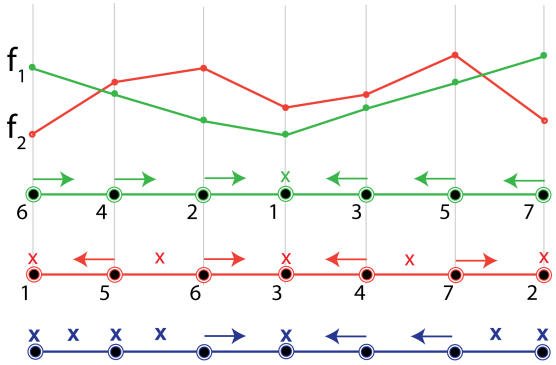


Figure 7.2: at the top, a discrete gradient over a discrete line compatible with function f_1 (in green) and function f_2 (in red). At the bottom, in blue, a discrete gradient compatible with the two functions together.

(see Equation (1.7) for the definition of incidence function). For the two pairs, the persistence is given by the difference between function values of the components: we get persistence equal to 3, for the pair $(3,6)$ and equal to 5 for the pair $(2,7)$. In this sense, the persistence feature indicated by the pair $(3,6)$ is less informative, from the topological point of view, than the one indicated by $(2,7)$.

As anticipated in Section 7.1.1, this paradigm has no straightforward generalization to the multivariate case since there is no (or there are too many) notions of persistence to measure the lifespan of a homology feature along a multiparameter filtration. Moreover, as seen in Remark 5.1.5, there is no counterpart for persistence pairs since homology changes are not necessarily given by a new cell entering a multiparameter filtration.

In the multivariate case, critical cells seem to highlight the reciprocal behavior of multiple function components in the following sense. In Figure 7.2, the blue line shows a discrete gradient compatible with both functions at the same time. We notice that this multi-compatible discrete gradient needs to declare critical, necessarily, all critical cells with respect to each function component, moreover, all cells where the two discrete gradients disagree. In this sense, critical cells of a discrete gradient compatible with multivariate data highlight where in the domain the multiple scalar data have different monotony. When moving to higher dimensional domains, the meaning of critical cells needs further insights. Empirical tests, like the one in Figure 7.3, have shown that critical cells tend to gather into clusters in correspondence of regions where the multiple discrete gradients of the scalar components disagree. To highlight this, in the next section, we propose a way of collecting critical cells into clusters. The cluster definition addresses, on the one hand, the necessity of compactly describing where multivariate data grow in disagreement one another, on the other hand, the necessity to provide simplification criteria in the multivariate setting.

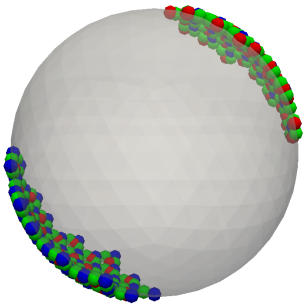


Figure 7.3: a triangulation of a 2-dimensional sphere embedded in the 3D Euclidean space. Colored points indicate critical cells found according to the function $\phi(x,y,z) = (x,y)$. In blue, critical vertexes, in green, critical edges, and in red, critical triangles.

7.3.2 Visualization through critical clusters

Here, we show some experimental results presented in [115]. In the experiments, we visualize cubical datasets of the Hurricane Isabel simulation and the Turbulent combustion simulation described in Section 6.4 and depicted in Figure 7.4.

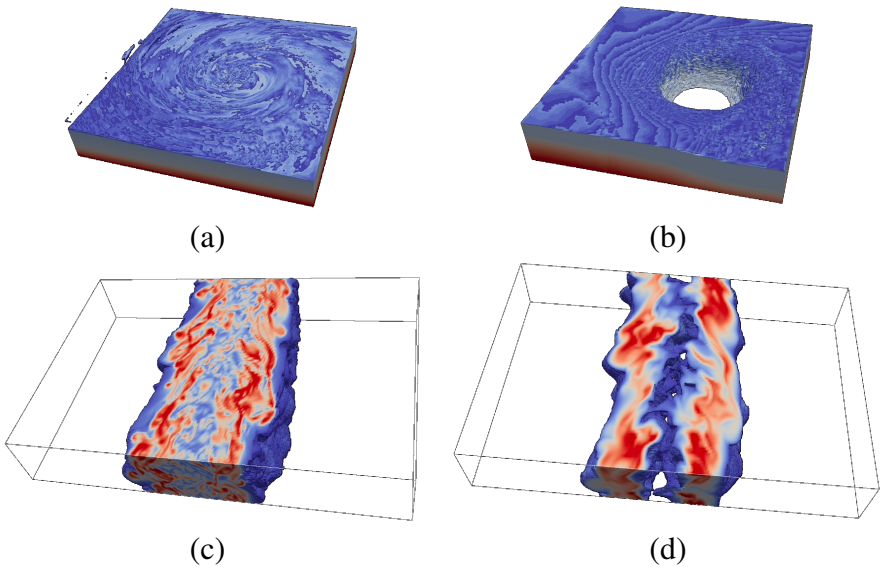


Figure 7.4: a visualization of the temperature (a) and the pressure (b) scalar fields in the hurricane Isabel dataset. The vorticity (c) and the dioxide concentration (d) scalar fields in the Turbulent combustion dataset. Values increase from blue to red.

We look at regular grids through their associated Hasse diagram, that is the incidence graph whose nodes are the cells in the regular grids and undirected arcs link facets to cofacets. We recall Definition 1.6.3 for the notion of Morse complex compatible with a multivariate function. Starting

from the discrete gradient we collect the sets of connected critical cells in the Hasse diagram of the regular grid. We call *minima-clusters* the collection of critical 0-cells connected by critical 1-cells. They can be easily computed using an union-find data structure defined on the critical 0-cells and by linearly processing all the critical 1-cells. Each 0-cell initially forms a minima-cluster on its own. Sweeping on the critical 1-cells, we consider only those having two critical 0-cells v_1,v_2 on their boundary. If v_1,v_2 belong to different minima-clusters, they are merged. Dually, *maxima-clusters* are defined as collections of critical d-cells connected by critical $(d-1)$-cells and they are computed in a similar fashion. Maxima- and minima-clusters are easily encoded as segmentations defined on the vertices/d-cells of the dataset (i.e. an integer label for each element). For visualization purposes, we do not directly show minima- or maxima-clusters. Rather, we show the area where minima- and maxima-clusters overlap. Minima- and maxima-clusters overlap in those areas where the function is more chaotic. By considering the nodes corresponding to cells in a minima-cluster as belonging to the dual regular grid, we can intersect minima- and maxima-cluster to get *critical clusters*. While the extremal-clusters are not a rigorous estimator for evaluating differences among multiple functions, they are fast to compute and, more importantly, they provide an interactive framework to help the user to eliminate noise. As a preliminary way of removing noise compatibly with multivariate data, we propose to filter out critical clusters according to their size in number of cells. In Figure 7.5, we visualize critical clusters relative to

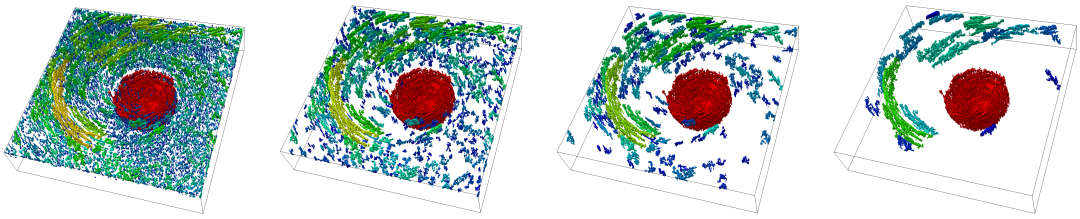


Figure 7.5: critical clusters relative to the hurricane Isabel dataset colored according to increasing number of critical cells in the same cluster, from cold to warm colors. Only clusters above a threshold size are depicted. From left to right the threshold value is: 10, 100, 400, 2000.

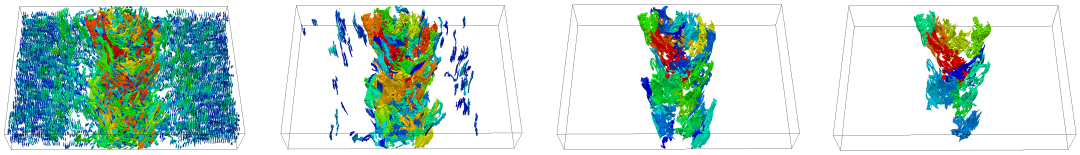
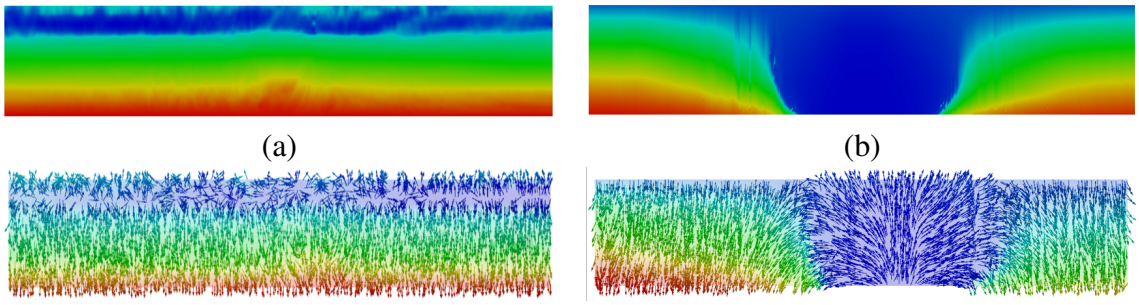


Figure 7.6: critical clusters relative to the Turbulent combustion dataset colored according to increasing number of critical cells in the same cluster, from cold to warm colors. Only clusters above a threshold size are depicted. From left to right the threshold value is: 50, 1000, 5000, 15000.

the Hurricane Isabel dataset. In Figure 7.6, we visualize critical clusters relative to the Turbulent combustion dataset. Clusters are filtered out according to the number of cells they contain form left to right. The right-most picture shows that the discrete gradient has its largest critical cluster in

correspondence of the eye of the hurricane. For the Hurricane Isabel dataset, since the gradients of the temperature and the pressure scalar fields do not agree in the eye of the hurricane, this is where biggest critical clusters are located. In Figure 7.7, we see the behavior of the discrete gradient of



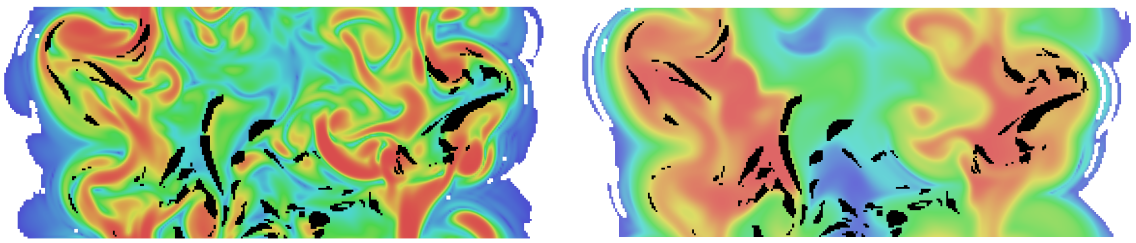
(a) (b)

(c) (d)

Figure 7.7: at the top, a visualization of a 2D slice of the scalar fields of the Hurricane Isabel: temperature field (a) and pressure field (b). At the bottom, the univariate discrete gradient associated to the temperature field (c) and the pressure field (d).

temperature and pressure fields on two different 2D slices of the domain, taken transversaly.

For the Turbulent combustion dataset, the visualization is more chaotic. In Figure 7.8, we extract two 2D transversal slices form the domain. There, we can see the role of critical clusters depicted in black. By looking at values of the scalar fields in the neighborhood of critical clusters, we see that critical clusters highlight region where, for instance, vorticity grows when sweeping through a critical cluster from right to left whereas dioxide concentration does the converse. In other words, the two scalar fields have opposite behavior through critical clusters.



(a) (b)

Figure 7.8: a visualization of a 2D slice of the scalar fields of the Turbulent combustion simulation: vorticity (a) and dioxide concentration (b). Juxtaposed in black, the critical clusters.

7.4 Towards a new definition of Pareto critical cells

In this section, we report our observations on the correspondence between critical cells and Pareto-related notions from the discrete, the smooth, and the PL settings. In doing this, we discuss some limitations of our visualization by critical cells proposed in this thesis in Section 7.3. This section is motivated by the difficulties we have found in knowing what to expect when experimentally comparing several Pareto-related notions present in the literature. We have decided to shed some light in reciprocal meaning of different Pareto-related notions chosen in the literature for the discrete, the smooth, or the PL setting. This is a preliminary study aimed at obtaining a motivated definition of *(discrete) Pareto critical cell* coherently compared to other existing Pareto-related notions.

The first Pareto-related notions is treated in Section 7.4.1 and it is the notion of *local Pareto minima* defined in the barrier tree approach [180] as the only other method, in the literature of multivariate topology-based subdivisions, which is fully discrete. As a second Pareto-related notion, in Section 7.4.2, we treat the smooth case of *Pareto critical points*. As a last Pareto-related notion, in Section 7.4.3, we treat the piecewise linear (PL) Pareto cells as defined in [114].

7.4.1 Discrete critical cells and discrete local Pareto minima

In this section, we compare (discrete) critical cells relative to a discrete gradient compatible with a multiparameter filtration to the notion of local Pareto minima in [180]. We first recall the definition of local Pareto minimum and then we present our tests on simple datasets to support the comparison.

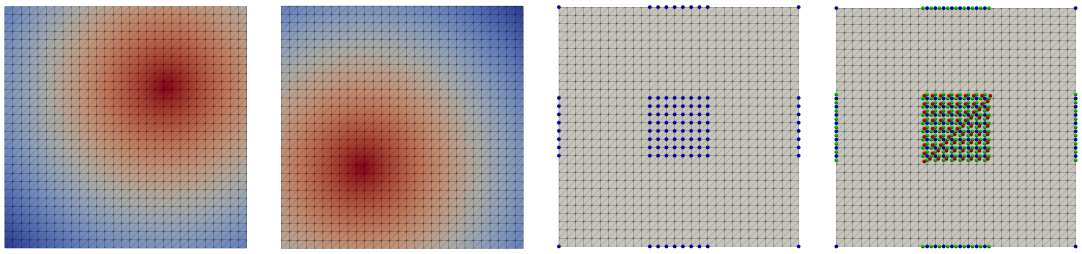
As mentioned in Section 7.2, the local Pareto minima in the barrier tree approach presented in [180] applies to graph domains. we have a function $f = (f_1, \dots, f_n)$ defined over the nodes N of the graph $G = (N, A)$. Nodes adjacent to a given node $v \in N$ define the set $N(v)$. A node v forms a basin if

$$\forall v' \in N(v), \quad f(v') \preceq f(v) \Rightarrow \quad v' = v.$$

By considering, the notion of *lower star of a vertex* with respect to f as in the Definition 3.1.3, we get that basins are generated by nodes v in the graph satisfying the following equivalent condition

$$\mathrm{Low}_f(v) = \{v\}.$$

Since all discrete gradient compatible with f have, as critical, only those vertexes that are "alone" in their lower star, we expect correspondence between critical vertexes and local Pareto minima as defined in [180]. In the example of Figure 7.9, we have compared critical cells found by ComputeDiscreteGradient and the basins originated by our implementation of the barrier tree method over a triangulation of the plane. In Figure 7.9(c), blue dots in the interior of the triangulation show the basins found in correspondence of the region where the two scalar fields



(a) (b) (c) (d)

Figure 7.9: (a)-(b) two scalar fields defined over a square grid with increasing values from colder to warmer colors. (c) blue dots indicate basins through the barrier tree method. (d) colored dots indicate critical cells obtained through ComputeDiscreteGradient: blue for vertexes, green for edges, red for triangles.

of Figure 7.9(a)-(b) do not agree. In Figure 7.9(d), blue dots indicate critical vertex found by ComputeDiscreteGradient over the same dataset. This correspondence suggests that critical cells of a discrete gradient compatible with a multivariate filtration is a possible generalization to simplicial complexes of discrete local Pareto minima as defined in [180]. However, for higher-dimensional cells, the fact of being found critical it is not necessarily related to being alone in the lower star. Hence, if we call *strongly critical* those critical cells $a \in X$ such that

$$\mathrm{Low}_f(a) = \{a\},$$

we would be interested in understanding whether strongly critical cells capture interesting properties. Strongly critical cells are candidate to define discrete Pareto critical cells.

7.4.2 Discrete critical cells and smooth Pareto critical points

In this section, we compare (discrete) critical cells relative to a discrete gradient compatible with a multiparameter filtration to the notion of *Pareto critical point* with respect to a multivariate function (Definition 7.1.4).

We have empirically noticed a *correspondence* like the one in Figure 7.10 between discrete critical cells and Pareto critical points. Indeed, by comparing Figure 7.10(a) to Figure 7.1(b), we notice a surprising similarity which deserves further insights. We plan to perform experimental comparisons in this sense which are not part of the work of this thesis.

A first obstacle to that correspondence, at the moment, is that we know that this correspondence can be clearer or more obscure depending on *component-wise perturbation* performed via *simulation of simplicity* [82] in order to apply ComputeDiscreteGradient. In Figure 7.10(c), we see how the Pareto correspondence is affected by that.

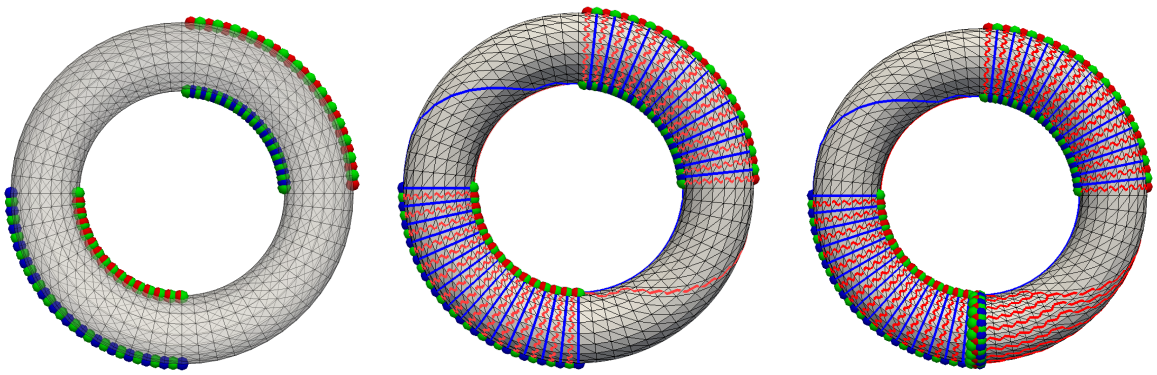
There is another obstacle to perfect correspondence with Pareto critical points, namely the strong *dependence on the triangulation* and on the data perturbation we empirically noticed in our

experiments. For the case of the triangulation, the results obtained on the torus of Figure 7.10(a) is not as similar to what we expect from the smooth case as the results obtained for the sphere in Figure 7.3. We suggest that a central role in that is played by the triangulation. The closeness to the smooth Pareto critical notion varies a lot by changing triangulation and, in the case of the torus, we can conjecture that the triangulation is sufficiently well-suited with respect to the x and y coordinates taken as scalar fields. For the role played by the data perturbation performed by simulation of simplicity before the retrieval of the discrete gradient, we notice the difference between Figure 7.10(a) and Figure 7.10(b).

Empirical tests give some hints about the *definition of critical cluster*. We have noticed that our definition of critical cluster in the previous section might be changed in order to capture the cluster suggested by Figure 7.1. Indeed, critical clusters in Figure 7.1 corresponds either to minima- or maxima-clusters from the previous section. However, they are not given by intersecting those. Hence, we do not detect the four clusters suggested in the figure since the intersection of maxima- and minima-clusters is always empty.

We are currently working on a definition of *discrete Pareto critical cell* to form a new notion of critical Pareto cluster able to capture the four of the clusters in Figure 7.1. One can consider all critical cells to be discrete Pareto critical cells but, that choice, as already noticed in Section 7.3.2, might leave too many critical cells to visualize. Alternatively, for the specific example of the torus in Figure 7.1, it would be enough to define discrete Pareto critical cells as *strongly critical cells* defined in the previous section. These two are the extremum possibilities. After that, discrete Pareto critical cells might be arranged into clusters, called *Pareto clusters* by detecting connected components in the Hasse diagram as done for maxima-/minima-clusters in Section 7.3.2. Facets relations among Pareto clusters might be defined by following the discrete gradient, so that to obtain a *Pareto graph* whose nodes are Pareto clusters and whose arcs connect two Pareto clusters connected by gradient paths. This graph structure would be the only discrete counterpart to the Reachability graph presented in [114].

Finding formal relationships between the smooth and the discrete settings is a challenging task which has its roots in the univariate case. We suggest that *homology might be a contact point* between smooth and discrete setting. We know the critical cells retrieved by ComputeDiscreteGradient to form a Morse complex which preserves the multipersistence information. In Theorem 7.1.6, authors have found a relation between Pareto critical points and discontinuity points of the multipersistence invariant called *rank invariant* (Definition 1.7.2). A possibility would be that of studying the discrete counterpart for discontinuities in the rank invariant, so that to define a suitable notion of discrete Pareto critical cell such that Theorem 7.1.6 translates in those terms.



(a) (b) (c)

Figure 7.10: critical cells over a triangulation of a thorus (a) embedded in $\mathbb{R}^3$ with scalar valued functions given by the x and y coordinates of vertexes. (b) gradient paths connecting critical cells from one cluster to another. In blue paths from edges to vertexes, in red, from triangles to edges. (c) a perturbation of vertex coordinate.

7.4.3 Issues in comparing discrete critical cells to PL Pareto cells

In this section, we consider the issues in comparing (discrete) critical cells relative to a discrete gradient compatible with a multiparameter filtration to the notion of *Pareto cell* as defined in [113] for the PL setting.

As discussed previously in Section 7.2.2.2, for multivariate functions d-dimensional simplicial complex with $d > n$, authors define in [113] *Pareto extrema cells* by locally analyzing the level sets of the PL extension in the star of each cell. Pareto extrema are subdivided into *Pareto minima*, that is cells without dominating cells in their star, *Pareto maxima*, that is cells without dominated cells in their star, and *Pareto optima*, that is cells without both dominating and dominated cells in their star. We have not yet performed comparison tests of the two methods.

Finding formal relationships between the discrete and the PL settings is a challenging task which has its roots in the univariate case (we refer to [134] for insights on the relation between critical cells in the PL or discrete settings). Our future plan is to proceed empirically to separate discrete-PL issues from the multivariate specific ones. Up to now, we have considered simple cases, such that of Figure 7.10(a), we expect the two method to agree in the following sense. Triangles incident to critical cells should be declared as Pareto optimal. For the clusters with critical triangles and edges (red and green), the algorithm proposed in [113] should find two Pareto maximal cells bounding the cluster. For the clusters with critical vertexes and edges (blue and green), the algorithm proposed in [113] should find two Pareto minimal cells bounding the cluster. As a first thing, we notice that there is no direct correspondence between dimensions of critical cells and the character of Pareto extrema. As a second thing, this simple case should suggest a more global correspondence. However, that correspondence should become more subtle when considering examples like the one in Figure 7.10(c) or, even worse, in Figure 7.3. Hence, we plan

to better investigate the comparison between the two methods.

Differently from what we plan to do for the comparison with smooth Pareto critical point discussed in the previous section, we cannot rely on an implicit comparison through multipersistent homology. Indeed, we are not aware of any relation between Pareto cell in the PL setting as described in [113] and multipersistence of sublevel sets. Moreover, according to authors in [42], the standard PL extension does not preserve homotopy retracts of sublevel sets with respect to the approximate smooth function, thus affecting multipersistence compatibility between the smooth and the PL settings. That motivates author to introduce an axis-wise linear extension which is proved to preserve the desired compatibility under retracts. Further developments in this sense will shed some light in the theoretical comparison of our discrete approach and that PL notion of Pareto extrema cells.

8 Concluding Remarks

In the work of this thesis, we have mainly dealt with theoretical and computational issues in multipersistent homology computation. In the last part, we have also given our contribution in the context of discrete approaches for multivariate data visualization. Here, we recall the main issues we have addressed. Then, in Section 8.1, we review the results we have discussed along the thesis with respect to the detected issues. Afterwards, in Section 8.2, we introduce our current works. Finally, in Section 8.3, we report the main open perspectives on our topic. From the state of the art review in Chapter 2 for computational issues in Persistent Homology (PH), we addressed the following issues:

1) computating Multiparameter Persistent Homology (MPH) is not feasible over any reasonably sized datasets;

2) the only available MPH computational optimizations were preprocessing algorithms based either on multigraded Betti numbers (Definition 1.7.13), which was not available when we began our work, or discrete gradients from Discrete Morse Theory (DMT) (see Section 1.4);

3) no preprocessing algorithm had impact evaluation over MPH computations;

4) there was no notion of optimality for a discrete gradient compatible with a filtration, either with one or multiple parameters. There was, instead, a preprocessing algorithm (see Section 3.2) for the one-parameter persistence case that is optimal for cubical complexes embedded in $\mathbb{R}^3$. The multiparameter counterpart of that optimality was not straightforward to define;

5) the algorithm treated in Section 3.2 was already generalized to the multiparameter case by the algorithm we treated in Section 3.3, but the multiparameter algorithm required in input a simplicial rather than a cubical complex, the algorithm was not parallelizable and it was not feasible over reasonably-sized data.

Additionally, from our state of the art review in topology-based data visualization in Section 7.2, we addressed the following issues:

a) discrete approaches in univariate data visualization outperform other approaches in terms of scalability in the size of the domain and dimension-independence but discrete approaches in monotony-driven multivariate data visualizations were limited to graph domains;

b) multivariate data visualization methods have no counterpart to PH as a topological mean to perform noise removal;

c) monotony-driven multivariate methods refer to multiple Pareto-related notions from the smooth, the PL, and the discrete case which demand for further insights on their reciprocal relationships.

8.1 Results

Here, we highlight the contributions of the work of this thesis by referring to the issues listed in the previous section and by following the chapter structure of this thesis.

In Chapter 3, we provided a *unified description of the preprocessing algorithm* of Section 3.2 for one-persistent homology (1-PH) and the preprocessing algorithm of Section 3.3 for multiparameter persistent homology (MPH). These are the two algorithms of point 5) and our presentation allows to analyze the algorithm complexity in terms of space and time by discriminating between dissimilarities either due to simplicial/cubical complexes or due the changes from 1-PH to MPH. We have highlighted the local character of the 1-PH preprocessing as opposed to the global character of the MPH preprocessing. Moreover, we have provided an input-dependent worst-time estimation of the MPH preprocessing. This analysis makes it clear that the two preprocessing algorithm might have the same worst-case time complexity depending on the input. In fact, we have discussed how the presence of multiparameter in place of one parameter affect the average case rather than the worst-case time. Our complexity analysis explains that the global character of the MPH preprocessing limits its performances over any reasonably-sized datasets, especially due to its runtime storage cost which, possibly exceeds the footprint required for the input and output representation.

In Chapter 4, motivated by issue 2) and, in order to address issue 1), we have proposed an MPH preprocessing algorithm. The algorithm retrieves a discrete gradient compatible with a multiparameter filtration. From the discrete gradient retrieved, we can apply a *Morse reduction which is compatible with the given multifiltration*. MPH computations of any kind can be applied to the Morse reduced object rather than the original one. The same output is guaranteed but the Morse reduced object is likely to be smaller in terms of filtration and cell complex sizes. We have formally proved the equivalence of the proposed algorithm with the one treated in Section 3.3, thus proving its correctness. We have provided our complexity analysis of the proposed algorithm which highlights its local character as opposed to the global character of the MPH preprocessing of Section 3.3. The local character of the proposed MPH preprocessing allows for parallel and distributed implementations. Moreover, the local character shows the theoretical advantages of the proposed algorithm over the only available equivalent MPH preprocessing. The worst-case time complexity is the same as in the equivalent algorithm but the estimation is not input-dependent anymore. We have discussed in detail how the storage managing of the two equivalent MPH preprocessing is different. In particular, the runtime storage cost of our local MPH preprocessing does not exceed the footprint required for the input and output representation.

In Chapter 5, we have addressed issue 4) by restating the optimality proved for the 1-PH preprocessing algorithm (Section 5.1) over cubical complexes embedded in $\mathbb{R}^3$ into more general terms as a property of a discrete gradient compatible with a one-parameter filtration. This *optimality notion relates relative homology to the number of necessary critical cells* to capture the MPH properties of the filtration. The proposed optimality notion admits a multiparameter generalization we have introduced in this thesis. We have showed that a Morse reduction compatible with the filtration preserves the properties of relative homology in the optimality definition. This has led us to formally prove that our MPH preprocessing algorithm of Chapter 4 is optimal in the case of cubical complexes embedded in $\mathbb{R}^3$ (like in the 1-PH case where it was already known) and in the case of abstract simplicial complexes of dimension 2.

In Chapter 6, we have *evaluated the local MPH preprocessing* introduced in Chapter 4 through four experiments.
The first experiment was meant to evaluate the *advantages of our proposed local approach over its equivalent global approach* to be tested on the same implementation. Our theoretical complexity analysis has been confirmed by that test. Our local approach outperforms the global one in both terms of space and time costs. From the time cost point of view, both algorithms shows a linear dependency in the number of cells belonging to the input cell complex. However, the proportionality coefficient defining the linear dependency is much lower in the case of the local approach. From the storage cost point of view, the contributions of the runtime storage cost is much lower in the local case as expected from the analysis of Section 4.4 and Section 3.3.3.
The second and third experiments address issue 3) by proposing an *impact evaluation of our local approach as a preprocessing for MPH*: in the former case, applied to the persistence module retrieval (Definition 1.6.7), that is to the complete information of MPH, in the latter case, applied to the persistence space retrieval through the foliation method. Computing both the persistence module and the persistence space suffers of high time and storage costs as underlined in issue 1) and in all considered datasets the applied Morse reduction reduced the size of the original dataset. In the case of the *persistence module retrieval*, our local MPH preprocessing increases of up to about 50 times the size of the input complex that can be treated, and up to about 250 times the size of the filtration that can be treated. In all considered datasets (rather small), the the Morse reduction allows to complete the pipeline. Some non-reduced datasets have failed for running out of memory. These failures for rather small datasets suggest that, at the moment, our Morse reduction is not enough to make the persistence module computation feasible over real-size data. In particular, we detected memory costs as a bottleneck for current persistence module computational methods. Optimizations of current algorithms require better handling of memory usage in terms of size of the filtration and number of cells in the input complex. In the case of the *persistence space retrieval*, our local MPH preprocessing shows its advantages in all considered datasets. The foliation methods allows to retrieve the persistence space by multiple iterations of 1-PH computations. One goal was that of evaluating the tradeoff between the number of iterations

and the advantages of the MPH preprocessing. We found that in all considered datasets, the Morse reduced datasets outperforms the corresponding original dataset, regardless of the number of iterations applied. Moreover, we have found that our MPH preprocessing is preferable over all considered 1-PH optimized algorithm to be iterated.
The fourth experiment addresses the issue 1) by proposing our Morse reduction compatible with a multiparameter filtration over simulation datasets with both simplicial and cubical complexes of up to about 80M cells, and up to 3 filtration parameters. In the worst case, timings are about 18 minutes in the discrete gradient retrieval and about 23 minutes for the Morse reduced dataset retrieval. The reduction factor in the number of cells is up to 40x. These promising results confirm the scalability of our MPH preprocessing over larger datasets. However, at the moment, MPH computation are not made feasible over datasets of reasonable size by our MPH preprocessing. Indeed, only the visualization tool RIVET [197] can handle Morse reduced datasets of millions of cells to retrieve an equivalent version of the persistence space. However, the visualization tool requires in input a simplicial complex whereas our Morse reduced datasets are not necessarily simplicial complexes but rather cell complexes. For the persistence module computation, no available method can handle Morse reduced datasets of millions of cells. Yet the limitations we have found are around 2K cells with 10K filtration grades.

In Chapter 7, we have presented our preliminary results in the context of multivariate data visualization via a discrete Morse-based approach. We have studied the properties of a *discrete gradient V compatible with a multiparameter filtration* (Definition 1.6.2) in the context of multivariate visualization. We have proposed a visualization method based on *critical clusters* to address issue b). The tests show that critical clusters capture region where scalar field components have opposite growing direction. By filtering out critical clusters according to their size in terms of number of cells, we focus on the biggest regions where opposite growing directions occur. We have classified our method as a *discrete* and *monotony-driven* approach to multivariate data visualization. With respect to other methods, ours does not propose yet an actual monotony-driven subdivision but only the critical points defining it. Under the assumptions for applying algorithm of Chapter 4, we have compared the critical cells of V to the Pareto-related notions of issue c). To address issue a), we have found that critical cells of V generalize the Pareto-related notion underlying the only other discrete method among the monotony-driven ones that was limited to graph domains. Issue c) suggests to compare our critical cluster method to other smooth or PL monotony-driven techniques. We have detected the notions of *Pareto critical cell* (Definition 7.1.4) and that of *Pareto optima* (Definition 7.1.5) as the notions to compare to. For the smooth case, we have empirically showed a correspondence between critical cells of V and Pareto critical points. Moreover, we have suggested further insights in the role of the triangulation and the perturbation adopted to formalize that relation. Additionally, we have detected MPH itself as a mean to relate smooth Pareto-related notions and critical cells of V. For the PL case, we have found that correspondences might be more subtle to prove than in the smooth case due to the lack of MPH relation. Moreover, correspondences between critical cells and critical PL-cells are

known to be subtle also in the univariate case. Empirical tests might shed some light on the topic that was too large to be investigated in this thesis and constitute current research.

8.2 Current work

In this section, we list the current works in developing the work of this thesis. We refer to the issues numbering of the beginning of this chapter. We are currently working on three main fronts.

Developing of issue 4): we are working on proving the optimality of the algorithm we proposed in Chapter 4 also for simplicial complexes embedded in $\mathbb{R}^3$.

Developing of issue 4): we are working on finding relations in between the optimality notion proposed in Chapter 5 and the notion of *multigraded Betti numbers* (Definition 1.7.13).

Developing of issue a): as anticipated in Section 7.4.2, we are working in defining *(discrete) Pareto cells* with respect to a multivariate function so that to obtain a discrete characterization through MPH by means of a discrete version of Theorem 7.1.6. Pareto cells would be at the base of a notion of *Pareto cluster* to overcome the critical cluster visualization method proposed in this thesis. Finally, we are working on proposing a *Pareto graph* as a compact representation of a multivariate field able to capture the role of the discrete gradient connecting critical cells. Pareto clusters would be the nodes and directed arcs would connect Pareto clusters admitting a V-path from at least one Pareto cell from one cluster to another. That graph representation would provide a fully discrete counterpart to the Reachability graph already proposed in the PL setting in [114].

8.3 Future perspectives

In this section, we provide our point of view for future directions of our work. As already done in the previous section, we refer to the issues numbering of the beginning of this chapter.

Developing of issue 1): on the one hand, the work of this thesis has given its contribution to the issue of MPH computation by providing a Morse reduction technique compatible with MPH. As already observed, this gives promising results but does not completely solve the issue, especially for computing the persistence module. Developing algorithms able to receive in input cell complexes rather than simple simplicial complexes will help in fully exploiting Morse reduction preprocessings. For instance, we think that a good starting point would be the visualization tool RIVET [197]. Enhancing the runtime storage cost of MPH computation would give the opportunity of evaluating MPH computational perspectives. On the other hand, in this thesis we have seen how the computational issues in MPH are strictly connected to theoretical issues of reducing and interpreting the information captured by MPH invariants. Some MPH invariants were not taken into account in this thesis for evaluation, such as QR-barcodes [141] for the lack

of a public available tool. Finally, we think that theoretical insights in the interplay between the optimality notion we proposed in this thesis and multigraded Betti numbers can help in developing a preprocessing technique able to exploit the filtration size reduction, not simply for the persistence module computationa, like we did in Section 6.2, but also for slice-compatible invariants we were not able to propose in this thesis. In the case of PH, feasibility of computations over larger data sets has opened the way to statistics of PH invariants in the direction of topological a preprocessing for machine learning techniques. In the case of MPH, this way is not yet feasible due to computational constraints and it is premature to propose this issue. Future perspectives in statistics contexts are determined by feasibility issues of MPH computational techniques and theoretical answers to the information captured by MPH invariants.

Developing of issue b): as already suggested in several parts of this thesis, we suggest an interplay between MPH and visualization of multivariate data which deserves further insights. The optimality notion we proposed in Chapter 5 and Theorem 7.1.6 reported from the work of Cerri and Frosini [46] goes in that direction. We believe that further insights in the topic can have positive impact both on defining topology-consistent noise removal techniques valid for multivariate data and on clarifying the relations among several Pareto-related notions available in the smooth and in the discrete case, as anticipated in Section 7.4.2.

Index

Bibliography

[1] Max K. Agoston. *Computer Graphics and Geometric Modeling: Mathematics*. Springer Verlag London Ltd., 2005 (Cited on page 38).

[2] Madjid Allili, Tomasz Kaczynski, and Claudia Landi. "Reducing complexes in multidimensional persistent homology theory". In: *Journal of Symbolic Computation* 78 (2017), pp. 61–75. DOI: 10.1016/j.jsc.2015.11.020 (Cited on page 51).

[3] Madjid Allili, Tomasz Kaczynski, Claudia Landi, and Filippo Masoni. "A New Matching Algorithm for Multidimensional Persistence". In: *ArXiv repository* (2015). arXiv: 1511.05427 (Cited on pages 69, 71, 83, 87, 89, 90, 92, 110).

[4] Madjid Allili, Tomasz Kaczynski, Claudia Landi, and Filippo Masoni. "Algorithmic Construction of Acyclic Partial Matchings for Multidimensional Persistence". In: *DGCI 2017: Discrete Geometry for Computer Imagery*. Ed. by Kropatsch W., Artner N., and Janusch I. Vol. 10502. Lecture Notes in Computer Science. Springer, Cham, 2017, pp. 375–387. DOI: 10.1007/978-3-319-66272-5_30 (Cited on pages xvi, 24, 51, 55, 68, 69, 75, 87, 89, 113).

[5] Chandrajit Bajaj, Valerio Pascucci, and Dan Schikore. "Visualization of Scalar Topology for Structural Enhancement". In: *Proceedings of the Conference on Visualization '98*. VIS '98. IEEE Computer Society Press, 1998, pp. 51–58. DOI: 10.1109/VISUAL.1998.745284 (Cited on page 139).

[6] Thomas Banchoff. "Critical points and curvature for embedded polyhedra". In: *Journal of Differential Geometry* 1.C (1967), pp. 245–256. DOI: 10.4310/jdg/1214428092 (Cited on pages 138, 139).

[7] Thomas Banchoff. "Critical points and curvature for embedded polyhedral surfaces". In: *The American Mathematical Monthly* 77.5 (1970), pp. 475–485. DOI: 10.2307/2317380 (Cited on pages 138, 139).

[8] Ulrich Bauer. "Persistence in discrete Morse Theory". In: *PhD Thesis, Niedersächsische Staats-und Universitätsbibliothek Göttingen.* (2011) (Cited on page 14).

[9] Ulrich Bauer. *Ripser*. 2016. URL: https://github.com/Ripser/ripser (Cited on pages 41, 42).

[10] Ulrich Bauer, Michael Kerber, and Jan Reininghaus. *Persistent Homology Algorithm Toolbox (PHAT)*. 2013. URL: https://github.com/blazs/phat (Cited on page 42).

[11] Ulrich Bauer, Michael Kerber, and Jan Reininghaus. “Clear and Compress: Computing Persistent Homology in Chunks”. In: *Topological Methods in Data Analysis and Visualization III*. Ed. by P. T. Bremer, I. Hotz, V. Pascucci, and R. Peikert. Mathematics and Visualization. Springer, Cham, 2014, pp. 103–117. DOI: 10.1007/978-3-319-04099-8_7 (Cited on pages 39, 40, 42, 43, 123, 127).

[12] 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)*. 2014, pp. 31–38. DOI: 10.1137/1.9781611973198.4 (Cited on page 40).

[13] Ulrich Bauer, Michael Kerber, Jan Reininghaus, and Hubert Wagner. “PHAT Persistent Homology Algorithms Toolbox”. In: *Journal of Symbolic Computation*. Ed. by Hoon Hong and Chee Yap. Vol. 78. Lecture Notes in Computer Science. Software available at https://github.com/blazs/phat. Springer Berlin Heidelberg, 2017, pp. 76–90. DOI: 10.1016/j.jsc.2016.03.008 (Cited on page 41).

[14] Mark de Berg and Constantinos Tsirogiannis. “Exact and approximate computations of watersheds on triangulated terrains”. In: *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. GIS ‘11. 2011, p. 74. DOI: 10.1145/2093973.2093985 (Cited on page 139).

[15] Silvia Biasotti, Aandrea Cerri, Daniela Giorgi, and Michela Spagnuolo. “PHOG: Photometric and geometric functions for textured shape retrieval”. In: *Computer Graphics Forum* 32.5 (2013), pp. 13–22. DOI: 10.1111/cgf.12168 (Cited on pages 49, 53).

[16] Silvia Biasotti, Andrea Cerri, Patrizio Frosini, and Daniela Giorgi. “A new algorithm for computing the 2-dimensional matching distance between size functions”. In: *Pattern Recognition Letters* 32.14 (2011), pp. 1735–1746. DOI: 10.1016/j.patrec.2011.07.014 (Cited on page 49).

[17] Silvia Biasotti, Andrea Cerri, Patrizio Frosini, Daniela Giorgi, and Claudia Landi. “Multidimensional size functions for shape comparison”. In: *Journal of Mathematical Imaging and Vision* 32.2 (2008), pp. 161–179. DOI: 10.1007/s10851-008-0096-z (Cited on pages 45, 47–49, 113, 121).

[18] Silvia Biasotti, Bianca Falcidieno, and Michela Spagnuolo. “Extended Reeb Graphs for Shape Analysis and Model Compression”. In: *Discrete Geometry for Computer Imagery, Lecture Notes in Computer Science*. Vol. 1953/2000. Springer, Berlin, Heidelberg, 2000, pp. 185–197. DOI: 10.1007/3-540-44438-6_16 (Cited on page 138).

[19] Silvia Biasotti, Daniela Giorgi, Michela Spagnuolo, and Bianca Falcidieno. “Reeb graphs for shape analysis and applications”. In: *Theoretical Computer Science* 392.1-3 (2008), pp. 5–22. DOI: 10.1016/j.tcs.2007.10.018 (Cited on page 138).

[20] Anders Björner. *Oriented matroids*. Cambridge University Press, 1999 (Cited on pages 64, 70).

[21] Jean Daniel Boissonnat, Tamal K. Dey, and Clément Maria. "The Compressed Annotation Matrix: en Efficient Data Structure for Computing Persistent Cohomology". In: *Algorithms - ESA 2013*. Ed. by Hans L. Bodlaender and Giuseppe F. Italiano. Vol. 8125. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 695–706. DOI: 10.1007/978-3-642-40450-4_59 (Cited on pages 36, 39, 40).

[22] Jean Daniel Boissonnat and Clément Maria. "The simplex tree: An efficient data structure for general simplicial complexes". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Lecture Notes in Computer Science 7501.June (2012), pp. 731–742. DOI: 10.1007/978-3-642-33090-2_63 (Cited on pages 36, 40, 42).

[23] Jean Daniel Boissonnat and Clément Maria. "Computing Persistent Homology with Various Coefficient Fields in a Single Pass". In: *Algorithms - ESA 2014*. Ed. by A.S. Schulz and D. Wagner. Vol. 8737. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2014, p. 16. DOI: 10.1007/978-3-662-44777-2_16 (Cited on page 42).

[24] Peer-Timo Bremer, Herbert Edelsbrunner, Bernd Hamann, and Valerio Pascucci. "A topological hierarchy for functions on triangulated surfaces". In: *IEEE Transactions on Visualization and Computer Graphics* 10.4 (2004), pp. 385–396. DOI: 10.1109/TVCG.2004.3 (Cited on page 139).

[25] Peter Bubenik. "Statistical topological data analysis using persistence landscapes". In: *The Journal of Machine Learning Research* 16.1 (2015), pp. 77–102 (Cited on page 54).

[26] Oleksiy Busaryev, Sergio Cabello, Chao Chen, Tamal K. Dey, and Yusu Wang. "Annotating simplices with a homology basis and its applications". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Ed. by Fedor V. Fomin and Petteri Kaski. Vol. 7357. Lecture Notes in Computer Science. Springer, 2012, pp. 189–200. DOI: 10.1007/978-3-642-31155-0_17 (Cited on pages 39, 40).

[27] Francesca Cagliari, Barbara Di Fabio, and Massimo Ferri. "One-dimensional reduction of multidimensional persistent homology". In: *Proceedings of the American Mathematical Society* 138.08 (2010), pp. 3003–3003. DOI: 10.1090/S0002-9939-10-10312-8 (Cited on pages 45, 47, 48, 53).

[28] David Canino. *The Mangrove TDS Library: a C++ Tool for the Fast Prototyping of Topological Data Structures*. 2012. URL: http://mangrovetds.sourceforge.net (Cited on page 36).

[29] David Canino and Leila De Floriani. "Representing Simplicial Complexes with Mangrove". In: *Proceedings of the 22nd International Meshing Roundtable*. Springer, 2013, pp. 465–483. DOI: 10.1007/978-3-319-02335-9_26 (Cited on page 36).

[30] David Canino, Leila De Floriani, and Kenneth Weiss. "IA*: An adjacency-based representation for non-manifold simplicial shapes in arbitrary dimensions". In: *Computers & Graphics* 35.3 (2011), pp. 747–753. DOI: 10.1016/j.cag.2011.03.009 (Cited on pages 36, 94).

[31] Gunnar Carlsson. "Topology and data". In: *Bulletin of the American Mathematical Society* 46.2 (2009), pp. 255–308. DOI: 10.1090/S0273-0979-09-01249-X (Cited on page 37).

[32] Gunnar Carlsson and Vin de Silva. "Zigzag Persistence". In: *Foundations of Computational Mathematics* 10.4 (2010), pp. 367–405. DOI: 10.1007/s10208-010-9066-0 (Cited on page 39).

[33] Gunnar Carlsson, Gurjeet Singh, and Afra Zomorodian. "Computing multidimensional persistence". In: ed. by Yingfei Dong, Ding-Zhu Du, and Oscar Ibarra. Vol. 5878. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, pp. 730–739. DOI: 10.1007/978-3-642-10631-6_74 (Cited on pages 22, 47, 48, 50).

[34] Gunnar Carlsson and Afra Zomorodian. "Computing Persistent Homology". In: *Discrete & Computational Geometry* 33.2 (2005), pp. 249–274. DOI: 10.1007/s00454-004-1124-4 (Cited on pages 17, 38).

[35] Gunnar Carlsson and Afra Zomorodian. "The Theory of Multidimensional Persistence". In: *Computational Geometry*. Vol. 42. 1. 2009, pp. 71–93. DOI: 10.1007/s00454-009-9176-0 (Cited on pages 29, 30, 32, 43, 44, 46).

[36] Hamish Carr and David Duke. "Joint contour nets". In: *IEEE Transactions on Visualization and Computer Graphics*. Vol. 20. 8. IEEE Computer Society, 2014, pp. 1100–1113. DOI: 10.1109/TVCG.2013.269 (Cited on page 141).

[37] Hamish Carr, Zhao Geng, Julien Tierny, Amit Chattopadhyay, and Aaron Knoll. "Fiber surfaces: generalizing isosurfaces to bivariate data". In: *Computer Graphics Forum* 34.3 (2015), pp. 241–250. DOI: 10.1111/cgf.12636 (Cited on page 141).

[38] Hamish Carr, Jack Snoeyink, and Ulrike Axen. "Computing contour trees in all dimensions". In: *Computational Geometry* 24.2 (2003), pp. 75–94. DOI: 10.1016/S0925-7721(02)00093-7 (Cited on page 137).

[39] Hamish Carr, Jack Snoeyink, and Michiel van de Panne. "Simplifying flexible isosurfaces using local geometric measures". In: *IEEE Visualization 2004*. IEEE Computer Society, 2004, pp. 497–504. DOI: 10.1109/VISUAL.2004.96 (Cited on page 138).

[40] E. Catmull and J. Clark. "Recursively generated B-spline surfaces on arbitrary topological meshes". In: *Computer-Aided Design* 10.6 (1978), pp. 350–355. DOI: https://doi.org/10.1016/0010-4485(78)90110-0 (Cited on page 114).

[41] Nicholas J. Cavanna, Mahmoodreza Jahanseir, and Donald R. Sheehy. "A Geometric Perspective on Sparse Filtrations *". In: *ArXiv repository* (June 2015), pp. 1–18. arXiv: 1506.03797v1 (Cited on page 37).

[42] Niccolò Cavazza, Marc Ethier, Patrizio Frosini, Tomasz Kaczynski, and Claudia Landi. "Comparison of persistent homologies for vector functions: From continuous to discrete and back". In: *Computers and Mathematics with Applications* 66.4 (2013), pp. 560–573. DOI: 10.1016/j.camwa.2013.06.004 (Cited on pages 37, 153).

[43] Frédéric Cazals, Frédéric Chazal, and Thomas Lewiner. "Molecular shape analysis based upon the morse-smale complex and the connolly function". In: *Proceedings of the nineteenth conference on Computational geometry*. SCG '03. ACM New York, 2003, pp. 351–360. DOI: 10.1145/777842.777845 (Cited on pages 139, 140).

[44] Andrea Cerri, Barbara Di Fabio, Massimo Ferri, Patrizio Frosini, and Claudia Landi. "Betti numbers in multidimensional persistent homology are stable functions". In: *Mathematical Methods in the Applied Sciences* 36 (2013), pp. 1543–1557. DOI: 10.1002/mma.2704 (Cited on pages 22, 53).

[45] Andrea Cerri, Barbara Di Fabio, Grzegorz Jablonski, and Filippo Medri. "Comparing shapes through multi-scale approximations of the matching distance". In: *Computer Vision and Image Understanding* 121 (2014), pp. 43–56. DOI: 10.1016/j.cviu.2013.11.004 (Cited on page 128).

[46] Andrea Cerri and Patrizio Frosini. "Discontinuities in Multidimensional Size Functions". In: *ArXiv repository* (2009), pp. 1–23. arXiv: 0811.1868 (Cited on pages 136, 160).

[47] Andrea Cerri and Patrizio Frosini. *A New Approximation Algorithm for the Matching Distance in Multidimensional Persistence*. 2010. URL: http://amsacta.cib.unibo.it/2971/ (Cited on page 49).

[48] Andrea Cerri, Patrizio Frosini, and Claudia Landi. "A global reduction method for multidimensional size graphs". In: *Electronic Notes in Discrete Mathematics* 26 (2006), pp. 21–28. DOI: 10.1016/j.endm.2006.08.004 (Cited on page 51).

[49] Andrea Cerri and Claudia Landi. "The persistence space in multidimensional persistent homology". In: *International Conference on Discrete Geometry for Computer Imagery*. Ed. by Gonzalez-Diaz, Rocio and Jimenez, Maria-Jose and Medrano, Belen. Vol. 7749. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 180–191. DOI: 10.1007/978-3-642-37067-0_16 (Cited on pages 25–29, 44, 45, 53).

[50] Amit Chattopadhyay, Hamish Carr, David Duke, and Zhao Geng. "Extracting Jacobi Structures in Reeb Spaces". In: *Eurographics Conference on Visualization - Short Papers* (2014). DOI: 10.2312/eurovisshort.20141156 (Cited on pages 141, 142).

[51] Amit Chattopadhyay, Hamish Carr, David Duke, Zhao Geng, and Osamu Saeki. "Multivariate topology simplification". In: *Computational Geometry: Theory and Applications* 58 (2016), pp. 1–24. DOI: 10.1016/j.comgeo.2016.05.006 (Cited on pages 141, 142).

[52] Frédéric Chazal, David Cohen-Steiner, Marc Glisse, Leonidas J Guibas, and Steve Y Oudot. "Proximity of Persistence Modules and their Diagrams". In: *Proceedings of the twenty-fifth annual symposium on Computational geometry*. SCG '09 November. ACM, 2009, pp. 237–246. DOI: 10.1145/1542362.1542407 (Cited on pages 22, 53).

[53] Chao Chen and Michael Kerber. "Persistent homology computation with a twist". In: *27th European Workshop on Computational Geometry ...* Vol. 45. 3. 2011, pp. 28–31. DOI: 10.1.1.224.6560 (Cited on pages 39, 42, 123).

[54] Chao Chen and Michael Kerber. "An output-sensitive algorithm for persistent homology". In: *Computational Geometry: Theory and Applications* 46.4 (2013), pp. 435–447. DOI: 10.1016/j.comgeo.2012.02.010 (Cited on page 39).

[55] David Cohen-Steiner, Herbert Edelsbrunner, and John Harer. "Stability of persistence diagrams". In: *Discrete and Computational Geometry* 37.1 (2007), pp. 103–120. DOI: 10.1007/s00454-006-1276-5 (Cited on pages 52, 53).

[56] René Corbet and Michael Kerber. "The representation theorem of persistent homology revisited and generalized". In: *ArXiv repository* (2017), pp. 1–28. arXiv: 1511.05427 (Cited on page 30).

[57] Michele D'Amico. "Δ reduction of size graphs as a new algorithm for computing size functions of shapes". In: *Proceedings of the International Conference on Computer Vision, Pattern Recognition and Image Processing. Feb. 27 - Mar. 3*. 2000, pp. 107–110 (Cited on page 49).

[58] Michele D'Amico, Patrizio Frosini, and Claudia Landi. "Natural pseudo-distance and optimal matching between reduced size functions". In: *Acta Applicandae Mathematicae* 109.2 (2010), pp. 527–554. DOI: 10.1007/s10440-008-9332-1 (Cited on pages 52, 53).

[59] Emanuele Danovaro, Leila De Floriani, Paola Magillo, Mohammed Mostefa Mesmoudi, and Enrico Puppo. "Morphology-driven simplification and multiresolution modeling of terrains". In: *Proceedings of the 11th ACM International Symposium on Advances in Geographic Information Systems*. Ed. by E. Hoel and P. Rigaux. ACM New York, 2003, pp. 63–70. DOI: 10.1145/956676.956685 (Cited on pages 138, 139).

[60] Emanuele Danovaro, Leila De Floriani, and Mohammed Mostefa Mesmoudi. "Topological analysis and characterization of discrete scalar fields". In: *Geometry, Morphology, and Computational Imaging*. Ed. by T. Asano, R. Klette, and C. Ronse. Vol. 2616. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2003, pp. 386–402. DOI: 10.1007/3-540-36586-9_25 (Cited on pages 138, 139).

[61] Leila De Floriani, Ulderico Fugacci, Federico Iuricich, and Paola Magillo. "Morse complexes for shape segmentation and homological analysis: discrete models and algorithms". In: *Computer Graphics Forum* 34.2 (2015), pp. 761–785. DOI: 10.1111/cgf.12596 (Cited on page 139).

[62] Leila De Floriani, David Greenfieldboyce, and Annie Hui. “A data structure for non-manifold simplicial d-complexes”. In: *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*. ACM. 2004, pp. 83–92. DOI: 10.1145/1057432.1057444 (Cited on page 36).

[63] Leila De Floriani, Annie Hui, Daniele Panozzo, and David Canino. “A Dimension-Independent Data Structure for Simplicial Complexes”. In: *Proceedings of the 19th International Meshing Roundtable*. Ed. by Suzanne Shontz. Springer Berlin Heidelberg, 2010, pp. 403–420. DOI: 10.1007/978-3-642-15414-0_24 (Cited on page 36).

[64] Vin De Silva and Gunnar Carlsson. “Topological estimation using witness complexes”. In: *Proceedings of the First Eurographics conference on Point-Based Graphics* (2004), pp. 157–166. DOI: 10.2312/spbg/spbg04/157-166 (Cited on page 37).

[65] Cecil Jose A. Delfinado and Herbert Edelsbrunner. “An incremental algorithm for Betti numbers of simplicial complexes on the 3-sphere”. In: *Computer Aided Geometric Design* 12.7 (1995), pp. 771–784. DOI: 10.1016/0167-8396(95)00016-Y (Cited on page 38).

[66] Tamal K. Dey, Fengtao Fan, and Yusu Wang. “Computing Topological Persistence for Simplicial Maps”. In: *Annual Symposium on Computational Geometry - SOCG’14*. ACM. 2014, pp. 345–354. DOI: 10.1145/2582112.2582165 (Cited on pages 39, 40, 42).

[67] Tamal K. Dey, Fengtao Fan, and Yusu Wang. *SimPers: Software for Topological Persistence under Simplicial Maps*. 2014. URL: %7Bhttp://web.cse.ohio-state.edu/%7B~%7Ddey.8/SimPers/Simpers.html%7D (Cited on pages 41, 42).

[68] Tamal K. Dey, Joachim Giesen, and Samrat Goswami. “Shape segmentation and matching with flow discretization”. In: *Algorithms and Data Structures. WADS 2003. Lecture Notes in Computer Science*. Ed. by Frank Dehne, Jörg-Rüdiger Sack, and Michiel Smid. Springer Berlin Heidelberg, 2003, pp. 25–36. DOI: 10.1007/978-3-540-45078-8_3 (Cited on pages 138, 139).

[69] Paweł Dłotko. *Persistence Landscape Toolbox*. 2017. URL: https://www.math.upenn.edu/~dlotko/persistenceLandscape.html (Cited on page 54).

[70] Paweł Dłotko, Tomasz Kaczynski, Marian Mrozek, and Thomas Wanner. “Coreduction Homology Algorithm for Regular CW-Complexes”. In: *Discrete and Computational Geometry* 46 (2011), pp. 361–388. DOI: 10.1007/s00454-010-9303-y (Cited on pages 40, 41).

[71] Paweł Dłotko and Hubert Wagner. “Simplification of complexes for persistent homology computations”. In: *Homology, Homotopy and Applications* 16.1 (2014), pp. 49–63. DOI: 10.4310/HHA.2014.v16.n1.a3 (Cited on pages 40, 41).

[72] Harish Doraiswamy and Vijay Natarajan. “Output-sensitive construction of reeb graphs”. In: *IEEE Transactions on Visualization and Computer Graphics* 18.1 (2012), pp. 146–159. DOI: 10.1109/TVCG.2011.37 (Cited on page 138).

[73] David Duke, Hamish Carr, Aaron Knoll, Nicolas Schunck, and Andrzej Nam Hai Ah andStaszczak. “Visualizing nuclear scission through a multifield extension of topological analysis”. In: *IEEE Transactions on Visualization and Computer Graphics* 18.12 (2012), pp. 2033–2040. DOI: 10.1109/TVCG.2012.287 (Cited on page 141).

[74] Herbert Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer Verlag, Berlin, 1987 (Cited on page 36).

[75] Herbert Edelsbrunner and John Harer. “Persistent homology – a survey”. In: *Contemporary Mathematics*. Vol. 453. Providence, RI: American Mathematical Society, 2008, pp. 257–282. DOI: 10.1090/conm/453/08802 (Cited on pages 25, 38–40, 42, 123).

[76] Herbert Edelsbrunner and John L. Harer. “Jacobi sets”. In: *Foundations of Computational Mathematics: Minneapolis, 2002*. Vol. 312. London Mathematical Society Lecture Note Series. Cambridge University Press, 2004, pp. 37–57. DOI: 10.1017/CBO9781139106962.003 (Cited on pages 141, 142, 144).

[77] Herbert Edelsbrunner, John L. Harer, Vijay Natarajan, and Valerio Pascucci. “Morse-smale complexes for piecewise linear 3-manifolds”. In: *Proceedings of the Nineteenth Annual Symposium on Computational Geometry*. SCG ’03. ACM New York, 2003, pp. 361–370. DOI: 10.1145/777792.777846 (Cited on page 139).

[78] Herbert Edelsbrunner, John L. Harer, and Amit K. Patel. “Reeb Spaces of Piecewise Linear Mappings”. In: *Proceedings of the Twenty-fourth Annual Symposium on Computational Geometry*. SCG ’08. ACM New York, 2008, pp. 242–250. DOI: 10.1145/1377676.1377720 (Cited on pages 134, 140).

[79] Herbert Edelsbrunner, John L. Harer, and Afra Zomorodian. “Hierarchical Morse-Smale complexes for piecewise linear 2-manifolds”. In: *Discrete and Computational Geometry*. Vol. 30. 1. 2003, pp. 87–107. DOI: 10.1007/s00454-003-2926-5 (Cited on pages 138, 139).

[80] Herbert Edelsbrunner, David Letscher, and Afra Zomorodian. “Topological persistence and simplification”. In: *Discrete and Computational Geometry* 28.4 (2002), pp. 511–533. DOI: 10.1007/s00454-002-2885-2 (Cited on pages 17, 26, 38, 42, 123, 144).

[81] Herbert Edelsbrunner and Ernst P Mücke. “Three-Dimensional Alpha Shapes”. In: *ACM Transactions on Graphics* 13.1 (1994), pp. 43–72. DOI: 10.1145/174462.156635 (Cited on page 37).

[82] Herbert Edelsbrunner and Ernst Peter Mücke. “Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms”. In: *ACM Transactions on Graphics* 9.1 (1990), pp. 66–104. DOI: 10.1145/77635.77639 (Cited on pages 64, 69, 78, 150).

[83] David Eisenbud. *The Geometry of Syzygies: A second course in Commutative Algebra and Algebraic Geometry*. Springer, New York, NY, 2005, p. 248. DOI: 10.1007/b137572 (Cited on pages 31, 46, 50).

[84] Brittany T. Fasy, Jisu Kim, Fabrizio Lecci, Clément Maria, and Vincent Rouvreau. *TDA: Statistical Tools for Topological Data Analysis*. 2017. URL: https://cran.r-project.org/web/packages/TDA/index.html (Cited on page 52).

[85] Federico Iuricich. *MDG: a C++ library for computing discrete gradients on multivariate data*. 2017. URL: https://github.com/IuricichF/NM_FormanMultifiltration (Cited on pages 114, 119, 123, 129).

[86] Riccardo Fellegara, Federico Luricich, Leila De Floriani, and Kenneth Weiss. "Efficient computation and simplification of discrete morse decompositions on triangulated terrains". In: *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems - SIGSPATIAL '14*. 2014, pp. 223–232. DOI: 10.1145/2666310.2666412 (Cited on pages 40, 41, 94, 115).

[87] Robin Forman. "Morse Theory for cell complexes". In: *Advances in Mathematics* 134.1 (1998), pp. 90–145. DOI: 10.1006/aima.1997.1650 (Cited on pages 14–16, 40, 132, 139).

[88] Robin Forman. "A user's guide to discrete Morse Theory". In: *Seminaire Lotharingin de Combinatoire* 48 (2002), p. 35 (Cited on pages 14, 15).

[89] Patrizio Frosini. "A distance for similarity classes of submanifolds of a Euclidean space". In: *Bulletin of the Australian Mathematical Society* 42.3 (1990), pp. 407–415. DOI: 10.1017/S0004972700028574 (Cited on pages 36, 48).

[90] Patrizio Frosini and Michele Mulazzani. "Size homotopy groups for computation of natural size distances". In: *Bulletin of the Belgian Mathematical ...* 6.November 1998 (1999), pp. 455–464 (Cited on page 37).

[91] Raphael Fuchs and Helwig Hauser. "Visualization of multi-variate scientific data". In: *Computer Graphics Forum* 28.6 (2009), pp. 1670–1690. DOI: 10.1111/j.1467-8659.2009.01429.x (Cited on page 137).

[92] Ulderico Fugacci, Federico Iuricich, and Leila De Floriani. "Efficient computation of simplicial homology through acyclic matching". In: *Proceedings - 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2014*. 2015, pp. 587–593. DOI: 10.1109/SYNASC.2014.84 (Cited on pages 40, 41, 118, 128).

[93] Ulderico Fugacci, Federico Iuricich, and Leila De Floriani. "Computing discrete Morse complexes from simplicial complexes". In: *Graphical Models* (2018). Manuscript submitted for publication (Cited on pages 77, 93, 94, 96).

[94] Oliver Gäfvert. "Algorithms for Multidimensional Persistence". In: *Master thesis, KTH Royal Institute of Technology* (2016) (Cited on pages 47, 48, 50).

[95] Robert Ghrist. "Barcodes: The persistent topology of data". In: *Bulletin of the American Mathematical Society* 45.1 (2008), pp. 61–75. DOI: 10.1090/S0273-0979-07-01191-3 (Cited on page 37).

[96] Mark Giesbrecht. “Probabilistic computation of the Smith normal form of a sparse integer matrix”. In: *Algorithmic Number Theory. ANTS 1996*. Ed. by H. Cohen. Vol. 1122. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 1996, pp. 173–186. DOI: 10.1007/3-540-61581-4_53 (Cited on page 38).

[97] Joachim Giesen and Matthias John. “The flow complex: a data structure for geometric modeling”. In: *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA ’03. Society for Industrial and Applied Mathematics, 2003, pp. 285–294 (Cited on pages 138, 139).

[98] Frederick M Goodman. *Algebra: abstract and concrete*. Prentice Hall, 1998 (Cited on page 9).

[99] David Günther, Jan Reininghaus, Hubert Wagner, and Ingrid Hotz. “Efficient computation of 3D Morse–Smale complexes and persistent homology using discrete Morse Theory”. In: *The Visual Computer* 28.10 (2012), pp. 959–969. DOI: 10.1007/s00371-012-0726-8 (Cited on pages 40, 41, 139, 140).

[100] Attila G. Gyulassy, Peer-Timo Bremer, Ray Grout, Hemanth Kolla, Jacqueline Chen, and Valerio Pascucci. “Stability of dissipation elements: a case study in combustion”. In: *Computer Graphics Forum* 33.3 (2014), pp. 51–60. DOI: 10.1111/cgf.12361 (Cited on page 139).

[101] Attila G. Gyulassy, Peer-Timo Bremer, Bernd Hamann, and Valerio Pascucci. “Practical considerations in Morse-Smale complex computation”. In: *Topological Methods in Data Analysis and Visualization: Theory, Algorithms, and Applications*. Ed. by Valerio Pascucci, Xavier Tricoche, Hans Hagen, and Julien Tierny. Mathematics and Visualization. Springer Berlin Heidelberg, 2011, pp. 67–78. DOI: 10.1007/978-3-642-15014-2_6 (Cited on pages 139, 140).

[102] Attila G. Gyulassy, Peer-Timo Bremer, and Valerio Pascucci. “Computing morse-smale complexes with accurate geometry”. In: *IEEE Transactions on Visualization and Computer Graphics* 18.12 (2012), pp. 2014–2022. DOI: 10.1109/TVCG.2012.209 (Cited on pages 139, 140).

[103] Attila G. Gyulassy, Mark A. Duchaineau, Vijay Natarajan, Valerio Pascucci, Eduardo M. Bringa, Andrew Higginbotham, and Bernd Hamann. “Topologically clean distance fields”. In: *IEEE Transactions on Visualization and Computer Graphics* 13.6 (2007), pp. 1432–1439. DOI: 10.1109/TVCG.2007.70603 (Cited on page 139).

[104] Attila G. Gyulassy, Aaron Knoll, Kah Chun Lau, Bei Wang, Peer-Timo Bremer, Michael E. Papka, Larry A. Curtiss, and Valerio Pascucci. “Interstitial and Interlayer Ion Diffusion Geometry Extraction in Graphitic Nanosphere Battery Materials”. In: *IEEE Transactions on Visualization and Computer Graphics* 22.1 (2016), pp. 916–925. DOI: 10.1109/TVCG.2015.2467432 (Cited on page 139).

[105] Attila G. Gyulassy, Vijay Natarajan, Valerio Pascucci, and Bernd Hamann. "Efficient computation of Morse-Smale complexes for three-dimensional scalar functions". In: *IEEE Transactions on Visualization and Computer Graphics* 13.6 (2007), pp. 1440–1447. DOI: 10.1109/TVCG.2007.70552 (Cited on pages 138, 139).

[106] Sariel Har-Peled and Manor Mendel. "Fast Construction of Nets in Low-Dimensional Metrics and Their Applications". In: *SIAM Journal on Computing* 35.5 (2006), pp. 1148–1184. DOI: 10.1137/S0097539704446281 (Cited on page 37).

[107] Heather A. Harrington, Nina Otter, Hal Schenck, and Ulrike Tillmann. "Stratifying multiparameter persistent homology". In: *ArXiv repository* Aug (2017), pp. 1–38. arXiv: 1708.07390 (Cited on pages 44, 46).

[108] Allen Hatcher. *Algebraic Topology*. Cambridge University Press, 2001 (Cited on pages 9, 11–13, 37).

[109] Christian Heine, Heike Leitte, Mario Hlawitschka, Federico Iuricich, Leila De Floriani, Gerik Scheuermann, Hans Hagen, and Christoph Garth. "A survey of topology-based methods in visualization". In: *Computer Graphics Forum* 35.3 (2016), pp. 643–667. DOI: 10.1111/cgf.12933 (Cited on page 137).

[110] Masaki Hilaga, Yoshihisa Shinagawa, Taku Kohmura, and Tosiyasu L. Kunii. "Topology matching for fully automatic similarity estimation of 3D shapes". In: *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM New York, 2001, pp. 203–212. DOI: 10.1145/383259.383282 (Cited on page 138).

[111] Lars Huettenberger, Nils Feige, Achim Ebert, and Christoph Garth. "Application of Pareto sets in quality control of series production in car manufacturing". In: *2015 IEEE Pacific Visualization Symposium (PacificVis)*. IEEE Pacific, 2015, pp. 135–139. DOI: 10.1109/PACIFICVIS.2015.7156369 (Cited on page 144).

[112] Lars Huettenberger and Christoph Garth. "A comparison of pareto sets and jacobi sets". In: *Mathematics and Visualization*. Ed. by J. Bennett, F. Vivodtzev, and V. Pascucci. Vol. 38. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 125–141. DOI: 10.1007/978-3-662-44900-4__8 (Cited on page 142).

[113] Lars Huettenberger, Christian Heine, Hamish Carr, Gerik Scheuermann, and Christoph Garth. "Towards multifield scalar topology based on pareto optimality". In: *Computer Graphics Forum* 32.3 PART3 (2013), pp. 341–350. DOI: 10.1111/cgf.12121 (Cited on pages 142–144, 152, 153).

[114] Lars Huettenberger, Christian Heine, and Chritopher Garth. "Decomposition and simplification of multivariate data using Pareto sets". In: *IEEE Transactions on Visualization and Computer Graphics* 20.12 (2014), pp. 2684–2693. DOI: 10.1109/TVCG.2014.2346447 (Cited on pages 142–144, 149, 151, 159).

[115] Federico Iuricich, Sara Scaramuccia, Claudia Landi, and Leila De Floriani. “A discrete Morse-based approach to multivariate data analysis”. In: *SIGGRAPH ASIA 2016 Symposium on Visualization*. SA ’16. ACM New York, 2016. DOI: 10.1145/3002151.3002166 (Cited on pages xvii, 77, 144, 146).

[116] Chris Johnson. “Top scientific visualization research problems”. In: *IEEE Computer Graphics and Applications* 24.4 (2004), pp. 13–17. DOI: 10.1109/MCG.2004.20 (Cited on page 137).

[117] Michael Joswig and Marc E Pfetsch. “Computing Optimal Discrete Morse Functions”. In: *Electronic Notes in Discrete Mathematics* 17.Supplement C (2004), pp. 191–195. DOI: 10.1016/j.endm.2004.03.038 (Cited on page 15).

[118] Mateusz Juda and Marian Mrozek. “CAPD::RedHom v2 - Homology software based on reduction algorithms”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Ed. by Hoon Hong and Chee Yap. Vol. 8592. Lecture Notes in Computer Science. Software available at http://capd.sourceforge.net/capdRedHom/. Springer Berlin Heidelberg, 2014, pp. 160–166. DOI: 10.1007/978-3-662-44199-2_27 (Cited on page 42).

[119] Tomasz Kaczynski, Konstantin Michael Mischaikow, and Marian Mrozek. *Computational homology*. Ed. by Morris W. Hirsch and Joel Robbin. Springer-Verlag, New York, 2004, p. 480 (Cited on page 4).

[120] Ravindran Kannan and Achim Bachem. “Polynomial Algorithms for Computing the Smith and Hermite Normal Forms of an Integer Matrix”. In: *SIAM Journal on Computing* 8.4 (1979), pp. 499–507. DOI: 10.1137/0208040 (Cited on page 38).

[121] Johannes Kehrer and Helwig Hauser. “Visualization and visual analysis of multifaceted scientific data: A survey”. In: *IEEE Transactions on Visualization and Computer Graphics* 19.3 (2013), pp. 495–513. DOI: 10.1109/TVCG.2012.110 (Cited on page 137).

[122] Michael Kerber, Dmitriy Morozov, and Arnur Nigmetov. *Hera*. 2016. URL: https://bitbucket.org/grey_narn/hera (Cited on page 52).

[123] Henry King, Kevin Knudson, and Neža Mramor. “Generating Discrete Morse Functions from Point Data”. In: *Experimental Mathematics* 14.4 (2005), pp. 435–444. DOI: 10.1080/10586458.2005.10128941 (Cited on pages 40, 51, 139, 140).

[124] Pavol Klacansky, Julien Tierny, Hamish Carr, and Zhao Geng. “Fast and Exact Fiber Surfaces for Tetrahedral Meshes”. In: *IEEE Transactions on Visualization and Computer Graphics* (2016), pp. 1–1. DOI: 10.1109/TVCG.2016.2570215 (Cited on pages 141, 142).

[125] Joe Kniss, Gordon Kindlmann, and Charles Hansen. “Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets”. In: *Visualization, 2001. VIS ’01. Proceedings*. IEEE Computer Society Washington, 2001, pp. 255–262. DOI: 10.1109/VISUAL.2001.964519 (Cited on page 141).

[126] Kevin P. Knudson. “A refinement of multi-dimensional persistence”. In: *ArXiv repository* (2007). DOI: 10.4310/HHA.2008.v10.n1.a11. arXiv: 0706.2608 (Cited on page 44).

[127] Kevin P. Knudson. “A refinement of multi-dimensional persistence”. In: *Homology, Homotopy and Applications* 10.1 (2008), pp. 259–281. DOI: 10.4310/HHA.2008.v10.n1.a11 (Cited on pages 31, 46, 50).

[128] Martin Kreuzer and Lorenzo Robbiano. *Computational Commutative Algebra 1*. 2000, pp. 1–327. DOI: 10.1007/978-3-540-70628-1 (Cited on pages 31, 46).

[129] Marc van Kreveld, René van Oostrum, Chandrajit Bajaj, Valerio Pascucci, and Dan Schikore. “Contour trees and small seed sets for isosurface traversal”. In: *Proceedings of the thirteenth annual symposium on Computational geometry*. SCG ’97. ACM New York, 1997, pp. 212–220. DOI: 10.1145/262839.269238 (Cited on pages 137, 138).

[130] Dan Laney, Peer-Timo Bremer, Ajith Mascarenhas, Paul C. Miller, and Valerio Pascucci. “Understanding the structure of the turbulent mixing layer in hydrodynamic instabilities”. In: *IEEE Transactions on Visualization and Computer Graphics* 12.5 (2006), pp. 1053–1060. DOI: 10.1109/TVCG.2006.186 (Cited on page 139).

[131] Solomon Lefschetz. *Algebraic Topology*. Vol. 27. Colloquium Publications. American Mathematical Society, 1942 (Cited on page 10).

[132] Michael Phillip Lesnick. “Multidimensional interleavings and applications to topological inference”. PhD thesis. Stanford University, 2012. arXiv: 1206.1365 (Cited on pages 22, 44, 50, 53).

[133] Michael Lesnick and Matthew Wright. “Interactive Visualization of 2-D Persistence Modules”. In: *ArXiv repository* (2015), pp. 1–75. arXiv: 1512.00180 (Cited on pages 29, 45, 47, 49).

[134] Thomas Lewiner. “Critical sets in discrete Morse theories: Relating Forman and piecewise-linear approaches”. In: *Computer Aided Geometric Design* 30.6 (2013). DOI: 10.1016/j.cagd.2012.03.012 (Cited on page 152).

[135] David Lipsky, Primoz Skraba, and Mikael Vejdemo-Johansson. “A spectral sequence for parallelized persistence”. In: *ArXiv repository* (Dec. 2011). arXiv: 1112.1245 (Cited on pages 39, 40).

[136] Alberto Lovison. “Singular Continuation: Generating Piece-wise Linear Approximations to Pareto Sets via Global Analysis”. In: *SIAM Journal on Optimization* 21.2 (2011), pp. 463–490. DOI: 10.1137/100784746 (Cited on pages 134, 135, 142, 144).

[137] Paola Magillo, Emanuele Danovaro, Leila De Floriani, Laura Papaleo, and Maria Vitali. “A discrete approach to compute terrain morphology”. In: *Communications in Computer and Information Science* 21 (2008), pp. 13–26. DOI: 10.1007/978-3-540-89682-1_2 (Cited on pages 138, 139).

[138] Clément Maria, Jean Daniel Boissonnat, Marc Glisse, and Mariette Yvinec. "The Gudhi library: Simplicial complexes and persistent homology". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Ed. by Hoon Hong and Chee Yap. Vol. 8592. Lecture Notes in Computer Science. Software available at https://project.inria.fr/gudhi/software/. Springer Berlin Heidelberg, 2014, pp. 167–174. DOI: 10.1007/978-3-662-44199-2_28 (Cited on pages 41, 42).

[139] Yukio Matsumoto. *An Introduction to Morse Theory*. Vol. 208. Translations of Mathematical Monographs. American Mathematical Society, 2002 (Cited on page 132).

[140] Fernando Meyer. "Topographic distance and watershed lines". In: *Signal Processing* 38.1 (1994), pp. 113–125. DOI: 10.1016/0165-1684(94)90060-4 (Cited on pages 138, 139).

[141] Ezra Miller. "Data structures for real multiparameter persistence modules". In: *ArXiv repository* (2017). arXiv: 1709.08155 (Cited on pages 22, 31, 44, 46, 159).

[142] John W. Milnor. *Morse Theory*. Princeton university press, New Jersey, 1963 (Cited on pages 14, 132).

[143] Nikola Milosavljević, Dmitriy Morozov, and Primož Škraba. "Zigzag Persistent Homology in Matrix Multiplication Time". In: *Proc. 27th Ann. Symp. Comput. Geom.* 2011, pp. 216–225. DOI: 10.1145/1998196.1998229 (Cited on page 39).

[144] Konstantin Mischaikow and Vidit Nanda. "Morse Theory for Filtrations and Efficient Computation of Persistent Homology". In: *Discrete & Computational Geometry* 50.2 (2013), pp. 330–353. DOI: 10.1007/s00454-013-9529-6 (Cited on pages 20, 40, 41).

[145] Dmitriy Morozov. *Dionysus library for computing persistent homology*. 2012. URL: http://www.mrzv.org/software/dionysus/ (Cited on pages 41, 42, 52).

[146] Marian Mrozek and Bogdan Batko. "Coreduction homology algorithm". In: *Discrete and Computational Geometry* 41.1 (2009), pp. 96–118. DOI: 10.1007/s00454-008-9073-y (Cited on pages 10, 12, 40, 41).

[147] Marian Mrozek, Paweł Pilarczyk, and Natalia Zelazna. "Homology algorithm based on acyclic subspace". In: *Computers and Mathematics with Applications* 55.11 (2008), pp. 2395–2412. DOI: 10.1016/j.camwa.2007.08.044 (Cited on pages 40, 41).

[148] Marian Mrozek and Thomas Wanner. "Coreduction homology algorithm for inclusions and persistent homology". In: *Computers & Mathematics with Applications* 60.10 (2010), pp. 2812–2833. DOI: 10.1016/j.camwa.2010.09.036 (Cited on pages 40, 41).

[149] James R. Munkres. *Elements of Algebraic Topology*. Perseus Books, 1984, p. 454 (Cited on pages 2, 38).

[150] Suthambhara Nagaraj and Vijay Natarajan. "Relation-aware isosurface extraction in multifield data". In: *IEEE Transactions on Visualization and Computer Graphics* 17.2 (2011), pp. 182–191. DOI: 10.1109/TVCG.2010.64 (Cited on pages 141, 142).

[151] Suthambhara Nagaraj and Vijay Natarajan. “Simplification of Jacobi sets”. In: *Topological Data Analysis and Visualization: Theory, Algorithms and Applications*. Springer Berlin Heidelberg, 2011, pp. 91–102. DOI: 10.1007/978-3-642-15014-2_8 (Cited on pages 141, 142, 144).

[152] Suthambhara Nagaraj, Vijay Natarajan, and Ravi S. Nanjundiah. “A gradient-based comparison measure for visual analysis of multifield data”. In: *Computer Graphics Forum* 30.3 (2011), pp. 1101–1110. DOI: 10.1111/j.1467-8659.2011.01959.x (Cited on pages 141, 142).

[153] Vidit Nanda. *Perseus: the persistent homology software*. 2012. URL: http://www.sas.upenn.edu/%7B~%7Dvnanda/perseus/index.html (Cited on page 42).

[154] Oliver Gäfvert. *TopCat: a Java library for computing invariants on multidimensional persistence modules*. 2016 (Cited on pages 47, 48, 50, 117).

[155] Nina Otter, Mason A Porter, Ulrike Tillmann, Peter Grindrod, and Heather A Harrington. “A roadmap for the computation of persistent homology”. In: *EPJ Data Science* 6.1 (2017), p. 17. DOI: 10.1140/epjds/s13688-017-0109-5 (Cited on page 43).

[156] Valerio Pascucci. “Topology diagrams of scalar fields in scientific visualisation”. In: *Topological Data Structures for Surfaces: An Introduction to Geographical Information Science*. Ed. by S. Rana. John Wiley & Sons Ltd, 2004. Chap. 8, pp. 121–129. DOI: 10.1002/0470020288.ch8 (Cited on page 139).

[157] Valerio Pascucci, Giorgio Scorzelli, Peer-Timo Bremer, and Ajith Mascarenhas. “Robust on-line computation of Reeb graphs”. In: *ACM Transactions on Graphics*. Vol. 26. 3. ACM New York, 2007, p. 58. DOI: 10.1145/1276377.1276449 (Cited on page 138).

[158] Giuseppe Patané, Michela Spagnuolo, and Bianca Falcidieno. “A minimal contouring approach to the computation of the reeb graph”. In: *IEEE Transactions on Visualization and Computer Graphics* 15.4 (2009), pp. 583–595. DOI: 10.1109/TVCG.2009.22 (Cited on page 138).

[159] Georges Reeb. “Sur les points singulièrs d’une forme de Pfaff complétement intégrable ou d’une fonction numérique”. In: *Comptes Rendu de l’Académie des Sciences* 222 (1946), pp. 847–849 (Cited on pages 132, 137).

[160] Jan Reininghaus, Ulrich Bauer, and Michael Kerber. *DIPHA*. 2014. URL: https://code.google.com/p/dipha/ (Cited on pages 41, 42).

[161] Jan Reininghaus, Stefan Huber, Ulrich Bauer, and Roland Kwitt. “A Stable Multi-Scale Kernel for Topological Machine Learning”. In: *ArXiv repository* (Dec. 2015), pp. 1–8. arXiv: 1412.6821 (Cited on pages 53, 54).

[162] Vanessa Robins, Peter John Wood, and Adrian P. Sheppard. “Theory and algorithms for constructing discrete morse complexes from grayscale digital images”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.8 (2011), pp. 1646–1658. DOI: 10.1109/TPAMI.2011.95 (Cited on pages xvi, 40, 41, 43, 51, 55, 56, 58, 60, 63–65, 67, 87, 89, 99–101, 108, 139, 140).

[163] Jos B. T. M. Roerdink and Arnold Meijster. “The watershed transform: definitions, algorithms and parallelization strategies”. In: *Fundamenta Informaticae* 41.1,2 (2000), pp. 187–228 (Cited on pages 138, 139).

[164] Joseph J. Rotman. *An introduction to homological algebra*. 2nd. Springer Science & Business Media, 2008, p. 709. DOI: 10.1007/978-0-387-68324-9e-ISBN: (Cited on page 106).

[165] Bernard Schneider and Jo Wood. “Construction of metric surface networks from raster-based DEMs”. In: *Topological Data Structures for Surfaces*. John Wiley & Sons, Ltd, 2004. Chap. 4, pp. 53–70. DOI: 10.1002/0470020288.ch4 (Cited on page 139).

[166] Bernhard Schneider. “Extraction of hierarchical surface networks from bilinear surface patches”. In: *Geographical Analysis* 37.2 (2005), pp. 244–263. DOI: 10.1111/j.1538-4632.2005.00638.x (Cited on page 139).

[167] Dominic Schneider, Christian Heine, Hamish Carr, and Gerik Scheuermann. “Interactive comparison of multifield scalar data based on largest contours”. In: *Computer Aided Geometric Design* 30.6 (2013), pp. 521–528. DOI: 10.1016/j.cagd.2012.03.023 (Cited on page 140).

[168] Dominic Schneider, Alexander Wiebel, Hamish Carr, Mario Hlawitschka, and Gerik Scheuermann. “Interactive comparison of scalar fields based on largest contours with applications to flow visualization”. In: *IEEE Transactions on Visualization and Computer Graphics* 14.6 (2008), pp. 1475–1482. DOI: 10.1109/TVCG.2008.143 (Cited on page 140).

[169] Martina Scolamiero, Wojciech Chachólski, Anders Lundman, Ryan Ramanujam, and Sebastian Öberg. “Multidimensional Persistence and Noise”. In: *Foundations of Computational Mathematics* (2016), pp. 1–40. DOI: 10.1007/s10208-016-9323-y (Cited on pages 44, 47).

[170] Donald R. Sheehy. “An Output-Sensitive Algorithm for Computing Weighted α -Complexes”. In: *Proceedings of the Canadian Conference on Computational Geometry*. 2015 (Cited on page 37).

[171] Philip Shilane, Patrick Min, Michael Kazhdan, and Thomas Funkhouser. “The Princeton Shape Benchmark”. In: *Shape Modeling Applications, 2004. Proceedings*. 2004, pp. 167–178. DOI: 10.1109/SMI.2004.1314504 (Cited on page 121).

[172] Yoshihisa Shinagawa, Tosiyasu L. Kunii, and Yannick L. Kergosien. “Surface coding based on Morse Theory”. In: *IEEE Computer Graphics and Applications* 11.5 (1991), pp. 66–78. DOI: 10.1109/38.90568 (Cited on pages 132, 138).

[173] Nithin Shivashankar, Senthilnathan Maadasamy, and Vijay Natarajan. “Parallel computation of 2D Morse-Smale complexes”. In: *IEEE Transactions on Visualization and Computer Graphics* 18.10 (2012), pp. 1757–1770. DOI: 10.1109/TVCG.2011.284 (Cited on pages 139, 140).

[174] Nithin Shivashankar and Vijay Natarajan. “Parallel computation of 3D morse-smale complexes”. In: *Computer Graphics Forum* 31.3 PART1 (2012), pp. 965–974. DOI: 10.1111/j.1467-8659.2012.03089.x (Cited on pages 40, 41).

[175] Nithin Shivashankar and Vijay Natarajan. “Parallel computation of 3D morse-smale complexes”. In: *Computer Graphics Forum* 31.3 PART1 (2012), pp. 965–974. DOI: 10.1111/j.1467-8659.2012.03089.x (Cited on pages 139, 140).

[176] Nithin Shivashankar, M. Senthilnathan, and Vijay Natarajan. “Parallel computation of 2D Morse-Smale complexes”. In: *IEEE Transactions on Visualization and Computer Graphics* 18.10 (2012), pp. 1757–1770. DOI: 10.1109/TVCG.2011.284 (Cited on pages 40, 41).

[177] Vin de Silva, Dmitriy Morozov, and Mikael Vejdemo-Johansson. “Dualities in persistent (co)homology”. In: *Inverse Problems* 124003.12 (2011), p. 16. DOI: 10.1088/0266-5611/27/12/124003 (Cited on pages 39, 42, 123).

[178] Gurjeet Singh, Facundo Mémoli, and Gunnar Carlsson. “Topological methods for the analysis of high dimensional data sets and 3D object recognition”. In: *Eurographics Symposium on Point-Based Graphics*. Ed. by Mario Botsch and Renato Pajarola. 2007, pp. 91–100. DOI: 10.2312/SPBG/SPBG07/091-100 (Cited on page 141).

[179] Stephen Smale. “Global analysis and economics - Pareto optimum and a generalization of Morse Theory”. In: *Synthese* 31.2 (1975), pp. 345–358. DOI: 10.1007/BF00485983 (Cited on pages 131, 134, 135, 143).

[180] Peter F. Stadler and Christoph Flamm. “Barrier trees on poset-valued landscapes”. In: *Genetic Programming and Evolvable Machines* 4.1 (2003), pp. 7–20. DOI: 10.1023/A:1021821009420 (Cited on pages 142, 143, 149, 150).

[181] Arne Storjohann. “Near Optimal Algorithms for Computing Smith Normal Forms of Integers Matrices”. In: *Proceedings of the 1996 international symposium on Symbolic and algebraic computation*. ISSAC ’96. ACM, 1996, pp. 267–274. DOI: 10.1145/236869.237084 (Cited on page 38).

[182] Shigeo Takahashi, Tetsuya Ikeda, Yoshihisa Shinagawa, Tosiyasu L. Kunii, and Minoru Ueda. “Algorithms for extracting correct critical points and constructing topological graphs from discrete geographical elevation data”. In: *Computer Graphics Forum*. Vol. 14. 3. 1995, pp. 181–192. DOI: 10.1111/j.1467-8659.1995.cgf143_0181.x (Cited on page 139).

[183] Andrew Tausz. *phom: Persistent Homology in R, Version 1.0. 1*. 2011. URL: http://cran.r-project.org (Cited on page 42).

[184] Andrew Tausz, Mikael Vejdemo-Johansson, and Henry Adams. "JavaPlex: A research software package for persistent (co)homology". In: *Proceedings of ICMS 2014*. Ed. by Han Hong and Chee Yap. Lecture Notes in Computer Science 8592. Software available at http://appliedtopology.github.io/javaplex/. 2014, pp. 129–136 (Cited on pages 41, 42).

[185] Dean P. Thomas, Rita Borgo, Hamish Carr, and Simon Hands. "Joint contour net analysis of lattice QCD data". In: *ArXiv repository* (Mar. 2017). arXiv: 1703.02488 (Cited on page 141).

[186] Dilip Mathew Thomas and Vijay Natarajan. "Multiscale symmetry detection in scalar fields by clustering contours". In: *IEEE Transactions on Visualization and Computer Graphics* 20.12 (Dec. 2014), pp. 2427–2436. DOI: 10.1109/TVCG.2014.2346332 (Cited on page 138).

[187] Julien Tierny and Hamish Carr. "Jacobi fiber surfaces for bivariate Reeb space computation". In: *IEEE Transactions on Visualization and Computer Graphics* 23.1 (2017), pp. 960–969. DOI: 10.1109/TVCG.2016.2599017 (Cited on page 142).

[188] Julien Tierny, Guillaume Favelier, Joshua A. Levine, Charles Gueunet, and Michael Michaux. "The Topology ToolKit". In: *IEEE Transactions on Visualization and Computer Graphics* 24.1 (2018). Software available at https://topology-tool-kit.github.io, pp. 832–842. DOI: 10.1109/TVCG.2017.2743938 (Cited on page 144).

[189] Julien Tierny, Attila G. Gyulassy, Eddie Simon, and Valerio Pascucci. "Loop surgery for volumetric meshes: Reeb graphs reduced to contour trees". In: *IEEE Transactions on Visualization and Computer Graphics* 15.6 (2009), pp. 1177–1184. DOI: 10.1109/TVCG.2009.163 (Cited on page 138).

[190] Mikael Vejdemo-Johansson. "GAP persistence – a computational topology package for GAP". In: *Book of Abstracts Minisymposium on Publicly Available Geometric Topological Software* 43 (2012), p. 43 (Cited on pages 41, 42).

[191] Luc Vincent and Pierre Soille. "Watersheds in digital spaces: an efficient algorithm based on immersion simulations". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 13.6 (1991), pp. 583–598. DOI: 10.1109/34.87344 (Cited on pages 138, 139).

[192] Gunther H. Weber, Scott E. Dillard, Hamish Carr, Valerio Pascucci, and Bernd Hamann. "Topology-controlled volume rendering". In: *IEEE Transactions on Visualization and Computer Graphics* 13.2 (2007), pp. 330–341. DOI: 10.1109/TVCG.2007.47 (Cited on page 138).

[193] Kenneth Weiss, Federico Iuricich, Riccardo Fellegara, and Leila De Floriani. "A primal/dual representation for discrete morse complexes on tetrahedral meshes". In: *Computer Graphics Forum* 32.3 PART3 (2013), pp. 361–370. DOI: 10.1111/cgf.12123 (Cited on pages 40, 41, 94, 115, 139, 140).

[194] Adalbert Wilhelm. “Linked views for visual exploration”. In: *Handbook of Data Visualization*. Springer Berlin Heidelberg, 2008, pp. 199–215. DOI: 10.1007/978-3-540-33037-0 (Cited on page 140).

[195] Pak Chung Wong and R. Daniel Bergeron. “30 years of multidimensional multivariate visualization”. In: *Scientific Visualization, Overviews, Methodologies, and Techniques, Dagstuhl, Germany, May 1994 - Overviews and Surveys*. IEEE Computer Society Press, 1997, pp. 3–33 (Cited on page 137).

[196] Zoë Wood, Hugues Hoppe, Mathieu Desbrun, and Peter Schröder. “Removing excess topology from isosurfaces”. In: *ACM Transactions on Graphics* 23.2 (2004), pp. 190–208. DOI: 10.1145/990002.990007 (Cited on page 138).

[197] Matthew Wright and Michael Lesnick. *RIVET: Rank Invariant Visualization and Exploration Tool*. 2016. URL: http://rivet.online (Cited on pages 49, 50, 53, 158, 159).

[198] Afra Zomorodian. “Fast construction of the Vietoris-Rips complex”. In: *Computers and Graphics (Pergamon)* 34.3 (2010), pp. 263–271. DOI: 10.1016/j.cag.2010.03.007 (Cited on page 37).