Discrete Morse Sandwich: Fast Computation of Persistence Diagrams for Scalar Data – An Algorithm and A Benchmark

Pierre Guillou, Jules Vidal, Julien Tierny

▶ **To cite this version:**

Pierre Guillou, Jules Vidal, Julien Tierny. Discrete Morse Sandwich: Fast Computation of Persistence Diagrams for Scalar Data – An Algorithm and A Benchmark. IEEE Transactions on Visualization and Computer Graphics, In press, pp.1-18. 10.1109/TVCG.2023.3238008 . hal-03736312

HAL Id: hal-03736312
https://hal.science/hal-03736312

Submitted on 22 Jul 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

arXiv:2206.13932v1 [cs.LG] 27 Jun 2022

Discrete Morse Sandwich: Fast Computation of Persistence Diagrams for Scalar Data – An Algorithm and A Benchmark

Pierre Guillou, Jules Vidal, and Julien Tierny

Abstract—This paper introduces an efficient algorithm for persistence diagram computation, given an input piecewise linear scalar field f defined on a d-dimensional simplicial complex $\mathcal{K}$, with $d \leq 3$. Our method extends the seminal *"PairCells"* algorithm [102] by introducing three main accelerations. *(i)* First, we express this algorithm within the setting of discrete Morse theory [32], which considerably reduces the number of input simplices to consider. *(ii)* Second, we introduce a stratification approach to the problem, that we call *sandwiching*. Specifically, minima-saddle persistence pairs ($\mathcal{D}_0(f)$) and saddle-maximum persistence pairs ($\mathcal{D}_{d-1}(f)$) are efficiently computed by respectively processing with a Union-Find the unstable sets of 1-saddles and the stable sets of $(d-1)$-saddles. Additionally, we provide a detailed description of the (optional) handling of the boundary components of $\mathcal{K}$ when processing $(d-1)$-saddles. This fast processing of the dimensions 0 and $(d-1)$ further reduces, and drastically, the number of critical simplices to consider for the computation of $\mathcal{D}_1(f)$, the intermediate layer of the *sandwich*. *(iii)* Third, we document several performance improvements via shared-memory parallelism. We provide an open-source implementation of our algorithm for reproducibility purposes. We also contribute a reproducible benchmark package, which exploits three-dimensional data from a public repository and compares our algorithm to a variety of publicly available implementations. Extensive experiments indicate that our algorithm improves by two orders of magnitude the time performance of the seminal *"PairCells"* algorithm it extends. Moreover, it also improves memory footprint and time performance over a selection of 14 competing approaches, with a substantial gain over the fastest available approaches, while producing a strictly identical output. We illustrate the utility of our contributions with an application to the fast and robust extraction of persistent 1-dimensional generators on surfaces, volume data and high-dimensional point clouds.

Index Terms—Topological data analysis, scalar data, persistence diagrams, discrete Morse theory.

1 Introduction

Scalar data is central to many fields of science and engineering. It can be the result of an *(i)* acquisition process (examples include CT-scans produced in medical imaging or one-dimensional time-series produced by punctual sensors) or it can be the result of a *(ii)* numerical computation (examples include simulations in computational fluid dynamics, material sciences, etc). In both cases, the data is typically provided as a low-dimensional scalar field (1D, 2D, or 3D) defined on the vertices of either *(i)* a regular grid (e.g. pixel or voxel images) or *(ii)* a mesh (e.g. polyhedral surfaces and volumes, AMR grids, etc.). An established strategy to generically process either cases of data provenance is to subdivide each cell of the input domain into simplices [48], [53], hence converting the input data into a generic representation that facilitates subsequent processing, namely a piecewise linear scalar field defined over a simplicial complex (i.e. poly-lines in 1D, triangulated surfaces in 2D and tetrahedral meshes in 3D). However, such scalar fields are provided in the applications with an ever-increasing size and geometrical complexity, which significantly challenges their interpretation by human users. This motivates the design of advanced data analysis tools, to support the interactive exploration and analysis of the features of interest present in large datasets. This is precisely the purpose of Topological Data Analysis (TDA) [28], which provides a toolbox of techniques for the generic, robust, and efficient extraction of structural features in data.

Topological methods have been investigated by the visualization community for more than twenty years [50], with applications to a variety of domains, including combustion [14], [41], [60] fluid dynamics [19], [56] material sciences [45], [61], chemistry [6], [35], [74], or astrophysics [82], [87] to name a few. Several topological data representations studied in TDA (such as the persistence diagram [28], the contour tree [15], [37], [89], the Reeb graph [7], [38], [75], [76], [77] or the Morse-Smale complex [13], [22], [42], [44], [46], [81]) have been specialized and used successfully in visualization, in particular for the explicit extraction and visual representation of structural patterns hidden in the data. An important aspect of TDA is its ability to provide multi-scale hierarchies of the above topological data representations, which consequently enables multi-scale visualization, exploration and analysis. In that setting, *Topological Persistence* [28] is an established importance measure which enables to distinguish the most salient topological structures present in the data from those corresponding to noise. In typical analysis pipelines (as shown in Fig. 1), this importance measure drives the simplification of the above topological representations, resulting in interactive, multi-scale data explorations. In practice, topological persistence can be obtained by computing *Persistence Diagrams* [28]. Several algorithms have been proposed for their computation (see Sec. 1.1) and

• *P. Guillou, J. Vidal and J. Tierny are with the CNRS and Sorbonne Université. E-mail: {firstname.lastname}@sorbonne-universite.fr*

Manuscript received May 21, 2021; revised May 11, 2021.

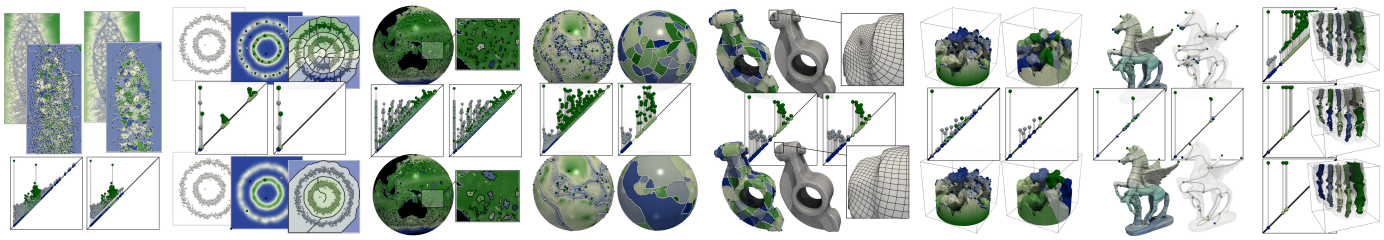

Fig. 1. Panorama of topological analysis pipelines in visualization and graphics applications. From left to right: cell segmentation in microscopy data with the Morse-Smale complex (acquired 2D image), persistence-driven clustering with the Morse-Smale complex (2D point cloud), vortex extraction with the merge tree in climate data (acquired sea surface height, triangulated surface), tectonic plate extraction with the Morse-Smale complex (simulated data, triangulated surface), quad-meshing with the Morse-Smale complex (eigenfunction of the Laplace Beltrami operator, triangulated surface), viscous finger extraction with the Morse-Smale complex (simulated data, tetrahedral mesh), skeleton extraction with the Reeb graph (harmonic function, triangulated surface), bone segmentation in a CT scan with the merge tree (acquired 3D image). In each case, topological persistence plays a central role to distinguish features from noise, enabling multi-scale analysis (left: original diagram, right: simplified diagram).

many software packages are publicly available. However, most of them generically target data defined in arbitrary dimension (with a specific focus towards high dimensional point clouds) and provide only limited specialization for low-dimensional scalar data.

In this work, we introduce a novel algorithm for the fast computation of persistence diagrams for scalar data defined on 1, 2 or 3-dimensional domains. In contrast to previous work, our approach specifically takes advantage of the low dimensionality of typical scalar data and exploits a stratification strategy, that we call *sandwiching* (see Sec. 3 for an overview). Our algorithm has several advantages over existing approaches. Our extensive experiments (Sec. 9) demonstrate a substantial gain over existing algorithms, both in memory footprint and time performance, while delivering a strictly identical output. Moreover, it is output sensitive and most of its internal procedures can be accelerated with shared-memory parallelism (Sec. 7). For reproducibility purposes, we provide a C++ implementation of our approach. We also contribute a benchmark package, which exploits three-dimensional data from a public repository [57] and compares our approach to a variety of publicly available implementations. We believe such a benchmark has the potential to become a reference experiment for future work on the topic. Finally, we present an application (Sec. 8) to the fast and robust extraction of generators for surfaces, volume data or high-dimensional point clouds.

1.1 Related work

This section describes the literature related to our work. First, we provide a quick overview of the usage of persistent homology in data visualization. Second, we briefly review the related computational methods.

Persistent Homology in Visualization Persistent Homology has originally been introduced independently by several research groups [2], [29], [33], [78]. In many applications involving data analysis, topological persistence quickly established itself as an appealing importance measure that helps distinguish salient topological structures in the data.

In data visualization, except a few approaches dealing with graph layouts [88] and dimensionality reduction [27], [73], Persistent Homology has been mostly used in previous work in *scientific visualization*, typically dealing with the interactive visual analysis of scalar data (coming from acquisitions or simulations). In that context, topological persistence is typically used as a measure of importance driving the simplification of the input data itself [62], [93], or the multi-scale hierarchical representation of topological abstractions [50], such as contour trees [15], [37], [89], Reeb graphs [7], [38], [75], [76], [77] or Morse-Smale complexes [13], [22], [42], [44], [46], [81]. For instance, in the *"Topology ToolKit"* (TTK) [8], [92] (an open-source library for topological data analysis and visualization), data is typically pre-simplified interactively, by removing low persistence features [62], [93], yielding a multi-scale hierarchy for the subsequent topological data representations (Fig. 1). Similar analysis pipelines have been documented in a number of applications, including combustion [14], [41], [60] fluid dynamics [19], [56] material sciences [45], [61], chemistry [6], [35], [74], or astrophysics [82], [87]. Topological persistence has also been used as an importance measure in several other scalar data analysis tasks, such as data segmentation [9], [16], isosurface extraction [96], data compression [86] or transfer function design for volume rendering [101]. The persistence diagram (Sec. 2.4) is a popular topological data representation, which concisely and robustly captures the number and salience of the features of interest present in the data. As such, it is an effective visual descriptor of the population of features in data, for ensemble summarization [31], [58], [97] or feature tracking [63], [84], [85].

Algorithms for Computing Persistent Homology In general, the standard approach to the computation of persistent homology involves the reduction of the boundary matrix [28] (which describes the facet/co-facet relations between the simplices of the input domain). This approach is now the core procedure of many software packages. This includes for instance *PHAT* [5] and *Dipha* [4] (which feature additional accelerations [18], [23], along with specific data structures for cubical cell complexes [100]), *Gudhi* [65] (which also features specific accelerations [10], [12], [24], [26] and data structures [11], in particular for cubical cell complexes [100]) and others [68], [90]. Certain packages have a special focus towards the persistent homology of Rips filtrations of high-dimensional point clouds, such as *Ripser* [3] (adapted to cubical complexes [55]) or *Eirene* [51], [52]. They have been integrated in several data analysis libraries [80], [91]. Some methods support parallel computations [4], [5], [69], [72].

All these methods are included in our benchmark (Sec. 9.3).

For low dimensional data, such as typical scalar fields, specific computational strategies can be considered. Specifically, in the case of surfaces, Edelsbrunner and Harer [28] observe (section VII.2, *"Efficient Implementations"*) that the persistence diagrams of the persistent homology groups of dimensions 0 and 1 can be computed very efficiently, by respectively tracking with a Union-Find data structure [21] the connected components of the sub-level sets of f and $-f$. A similar duality argument has been recently discussed for general cell complexes in higher dimensions [34]. This strategy has been the default computation method in TTK [8], [92] since its initial release, where the connected components of f and $-f$ are efficiently tracked thanks to a parallel merge tree algorithm [36], [37]. Recently, Vidal et al. presented *progressive* [98] and *approximate* [99] variants of this strategy, based on a multiresolution representation of the input. Our work exploits a similar strategy for the persistence diagrams of dimension 0 and $(d-1)$ (where d is the dimension of the input data) and further accelerates this process by restricting the sub-level set connectivity tracking to the unstable (and stable) sets of 1 and $(d-1)$ saddles.

Similarly to our work, previous approaches have investigated Morse theory [66], [70], specifically its discrete version [32], to accelerate the computation of persistence diagrams of scalar data. Robins et al. [25], [79] described an algorithm for computing a discrete gradient field whose critical simplices exactly coincide with the topological changes of the lower star filtration of the data. Thus, all the topological events occurring in the filtration of the scalar data can be equivalently encoded with a filtration of its discrete Morse complex. As the Morse complex is usually smaller in practice than the input data, this pre-process data reduction procedure accelerates subsequent, traditional algorithms for persistent homology [102]. Extensions of this idea have been investigated [39], [54], [67], [71], in particular for the support of high dimensional data. In contrast, our work specifically takes advantage of the low dimensionality of the data to expedite the computation, with a stratification approach that we call *sandwiching* (see Sec. 3 for an overview) as well as a novel specialization (Sec. 4) of the seminal algorithm *"PairCells"* [102] to the discrete Morse theory setting [32].

1.2 Contributions

This paper makes the following new contributions:

1) *A fast algorithm for the computation of persistence diagrams for 1D, 2D or 3D scalar data*: Our algorithm is based on a stratification strategy, called *sandwiching*, which leverages the low dimensionality of the data:
   - The persistence diagrams for the dimensions 0 and $(d-1)$ are efficiently computed by restricting a Union-Find [21] processing to the unstable (and stable) sets of 1 and $(d-1)$ saddles (Sec. 5);
   - For the 3D case, we introduce a specialization of the seminal algorithm *"PairCells"* [102] to the discrete Morse theory setting [32] for the processing of the remaining, unprocessed 1 and $(d-1)$ saddles (Sec. 4).

   Since *(i)* it is based on simple and inexpensive operations for the dimensions 0 and $(d-1)$ and that, for the dimension 1, *(ii)* it focuses the computation on a limited set of critical simplices (the remaining saddle-saddle pairs, the intermediate layer of the *sandwich*), our algorithm provides substantial gains with regard to reference algorithms. Moreover, it is output sensitive and several of its internal routines can be efficiently accelerated with shared-memory parallelism.
2) *An open-source implementation*: For reproduction purposes, we provide a C++ implementation of our approach, which is officially integrated in the source tree of TTK [8], [92] (Github commit: e14377b).
3) *A reproducible benchmark:* We provide a Python benchmark package (https://github.com/pierre-guillou/pdiags_bench), which uses three-dimensional data from a public repository [57] and compares the running times, memory footprints and output diagrams of a variety of publicly available implementations for persistence diagram computation. This reproducible benchmark may be used as a reference experiment for future developments on the topic.

2 Preliminaries

This section presents the theoretical background of our work. It contains definitions adapted from the Topology ToolKit [8], [92]. We refer the reader to textbooks [28], [102] for comprehensive introductions to computational topology.

2.1 Input data

The input data is provided as a piecewise linear (PL) scalar field $f : \mathcal{K} \to \mathbb{R}$ defined on d-dimensional simplicial complex $\mathcal{K}$, with $d \leq 3$. As discussed in the introduction, this input representation generically and homogeneously supports all types of typical scalar data, in 1D, 2D or 3D, coming from either acquisitions or numerical simulations. When the data is given on arbitrary cell complexes, cells are subdivided into simplices. In particular, regular grids are triangulated according to the Freudenthal triangulation [48], [53] (yielding a 6-vertex neighborhood in 2D and a 14-vertex neighborhood in 3D). Note that this triangulation is performed implicitly (i.e. no memory overhead), by emulating the simplicial structure upon traversal queries [92].

The input scalar field f is typically provided on the vertices of $\mathcal{K}$ and interpolated on the simplices of higher dimension. f is also assumed to be injective on the vertices of $\mathcal{K}$, which is easily achieved in practice with a symbolic perturbation inspired from Simulation of Simplicity [30].

2.2 Lexicographic filtration

Given the input function f, a global order between the simplices of $\mathcal{K}$ can be introduced by considering the so-called *lexicographic* comparison, as detailed below.

Given a d-simplex $\sigma \in \mathcal{K}$, let us consider the sequence $\{f(v_0(\sigma)), f(v_1(\sigma)), \ldots, f(v_d(\sigma))\}$ of its vertex data values, sorted in decreasing order, where $f(v_i(\sigma))$ denotes the i^{th} largest value among its vertices, i.e. $f(v_0(\sigma)) > f(v_1(\sigma)) > \ldots > f(v_d(\sigma))$.

Then, an order can be established between any two simplices σ_i and σ_j by comparing the above sorted sequences. In particular, σ_i will be considered *smaller* than σ_j if $f(v_0(\sigma_i)) < f(v_0(\sigma_j))$. On the contrary, if $f(v_0(\sigma_i)) >$

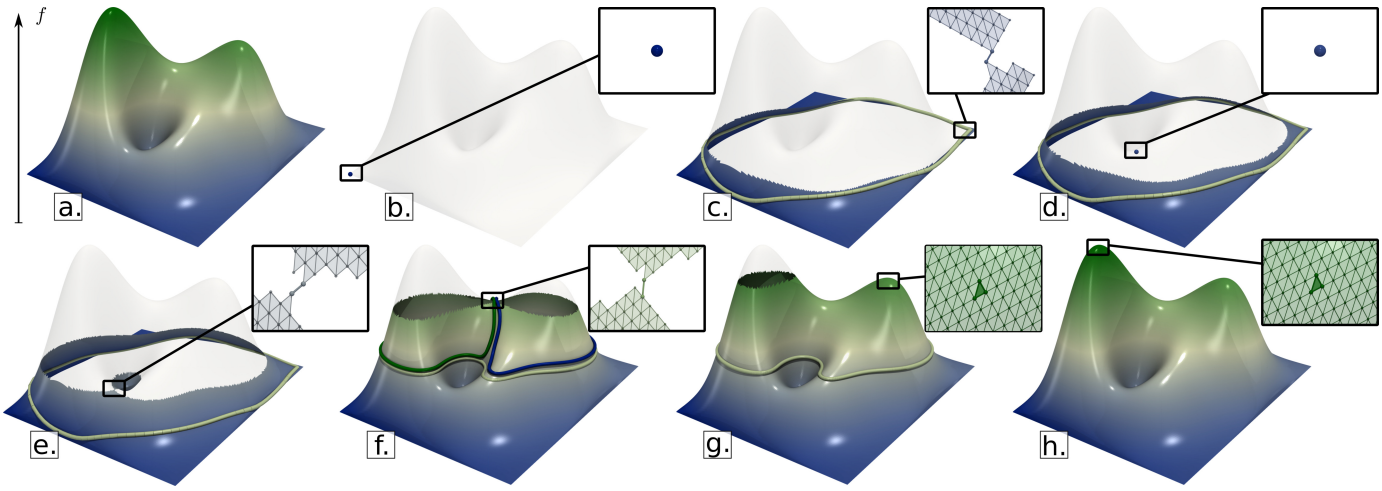

Fig. 2. Lexicographic filtration of a toy example (elevation f on a terrain $\mathcal{K}$, *(a)*). At step *(b)*, the introduction of the first vertex in the filtration $\mathcal{K}_b$ creates one connected component and $\beta_0(\mathcal{K}_b) = 1$. This component later loops back to itself *(c)*, creating a non trivial 1-cycle c_c (light green). At this point, we have: $\mathcal{H}_1(\mathcal{K}_c) = \{0, c_c\}$ and $\beta_1(\mathcal{K}_c) = rank(\mathcal{H}_1(\mathcal{K}_c)) = log_2(|\mathcal{H}_1(\mathcal{K}_c)|) = 1$. At step *(d)*, a new connected component is created and $\beta_0(\mathcal{K}_d) = 2$. At step *(e)*, two connected components merge into one and $\beta_0(\mathcal{K}_e) = 1$. At step *(f)*, the connected component of $\mathcal{K}_f$ loops back to itself, yielding three, independent, non trivial 1-cycles c_c (light green), c_f (dark green) and $c_{f'}$ (dark blue). At this point, we have: $\mathcal{H}_1(\mathcal{K}_f) = \{0, c_c, c_f, c_{f'}\}$ and $\beta_1(\mathcal{K}_f) = rank(\mathcal{H}_1(\mathcal{K}_f)) = log_2(|\mathcal{H}_1(\mathcal{K}_f)|) = 2$. At step *(g)*, the introduction of a triangle fills the *"hole"* left by the homology class $c_{f'}$ (dark blue, step *(f)*), which becomes trivial and disappears. Moreover, the class c_f (dark green, step *(f)*) becomes homologous to an *older* class, c_c (light green), and thus disappears and we have $\beta_1(\mathcal{K}_g) = 1$. Finally, at step *(h)*, the introduction of the last triangle fills the *"hole"* left by the homology class c_c (light green, step *(g)*) and we eventually have $\beta_0(\mathcal{K}_h) = 1$ and $\beta_1(\mathcal{K}_h) = 0$. The persistent diagram (Fig. 5) keeps track of all these events and records the birth, death and overall lifespan of the topological features responsible for changes in Betti numbers.

$f(v_0(\sigma_j))$, σ_i will be considered *greater* than σ_j. Otherwise, if $f(v_0(\sigma_i)) = f(v_0(\sigma_j))$, a tiebreak needs to be performed and the order will be decided by iteratively considering, similarly, the following vertices in the sequence (i.e. $v_k(\sigma_i)$ and $v_k(\sigma_j)$, with $k \in \{1, \dots, d\}$) until the conditions $f(v_k(\sigma_i)) < f(v_k(\sigma_j))$ (i.e. σ_i is smaller than σ_j) or $f(v_k(\sigma_i)) > f(v_k(\sigma_j))$ (i.e. σ_i is greater than σ_j) are satisfied. In the case where the dimensions d_i and d_j of σ_i and σ_j are such that $d_i < d_j$ and that $f(v_k(\sigma_i)) = f(v_k(\sigma_j)), \forall k \in \{0, \dots, d_i\}$ (i.e. σ_i is a *face* of σ_j), then σ_i is considered *smaller* than σ_j. Since f is injective on the vertices of $\mathcal{K}$ (Sec. 2.1), this lexicographic comparison guarantees a strict total order on the set of simplices of $\mathcal{K}$, such that all the faces of a simplex σ are by construction smaller than σ.

Let $\mathcal{K}_i$ be the union of the first i simplices of $\mathcal{K}$, given the above comparison. Then, the global lexicographic order induces a nested sequence of simplicial complexes $\emptyset = \mathcal{K}_0 \subset \mathcal{K}_1 \subset \dots \subset \mathcal{K}_n = \mathcal{K}$ (where n is the number of simplices of $\mathcal{K}$), which we call the *lexicographic filtration* of $\mathcal{K}$. Intuitively, it can be seen as a time-varying process, where the simplices of $\mathcal{K}$ are added one by one, given the lexicographic comparison of the vertex data values.

A central idea in Topological Data Analysis consists in encoding the evolution of the topological structures of $\mathcal{K}_i$ (for typical scalar data: its connected components, handles and voids, see Sec. 2.3) *along* the filtration, as i increases from 0 to n. In particular, as shown in Fig. 2, connected components progressively merge, handles get closed and voids get filled. This evolution is captured by the persistence diagram introduced later (Sec. 2.4).

We now discuss an alternate filtration, often considered in previous work [28], [79] and we describe its relation (used in Sec. 4) to the lexicographic filtration considered here. Let $St(v)$ be the star of a vertex v, i.e. the set of all its co-faces σ: $St(v) = \{\sigma \in \mathcal{K} \mid v < \sigma\}$. Let $St^-(v)$ be the *lower* star of v. It is the subset of the simplices of the star of v, for which v is the vertex with highest f value: $St^-(v) = \{\sigma \in St(v) \mid \forall u \in \sigma, f(u) \leq f(v)\}$. Since f is assumed to be injective on the vertices of $\mathcal{K}$, it follows that each simplex $\sigma \in \mathcal{K}$ belongs to a unique lower star. Let $\mathcal{K}'_j$ be the union of the first j lower stars, i.e. the union of the lower stars of the j-th lowest vertices of f. Then, the nested sequence of simplicial complexes $\emptyset = \mathcal{K}'_0 \subset \mathcal{K}'_1 \subset \dots \subset \mathcal{K}'_{n_v} = \mathcal{K}$ (where n_v is the number of vertices in $\mathcal{K}$) is called the *lower star filtration* of f [28]. $\mathcal{K}'_j$ is homotopy equivalent to the sub-level sets of $f(v_j)$ [28] and the topological changes occurring in $\mathcal{K}'_j$ during the lower star filtration thus precisely occur at the PL critical points [1] of f. Given the above definition, it follows that each sub-complex $\mathcal{K}'_j$ of the lower star filtration is equal to the sub-complex $\mathcal{K}_{i-1}$ of the lexicographic filtration, where σ_i is the *vertex* immediately after v_j in the global vertex order. In other words, the lower star filtration introduces simplices by *chunks* of lower stars (Fig. 9), while the lexicographic filtration introduces them one by one, yet in a compatible order. Then, it follows that each PL critical point includes in its lower star a simplex whose introduction via the lexicographic filtration changes the topology of $\mathcal{K}_i$.

2.3 Homology groups

The topology of a simplicial complex can be described with its homology groups, briefly summarized here from [28].

We call a p-chain c a formal sum (with modulo 2 coefficients) of p-simplices σ_i of $\mathcal{K}$: $c = \sum \alpha_i^c \sigma_i$ with $\alpha_i^c \in \{0, 1\}$.

Two p-chains $c = \sum \alpha_i^c \sigma_i$ and $c' = \sum \alpha_i^{c'} \sigma_i$ can be summed together componentwise to form a new p-chain

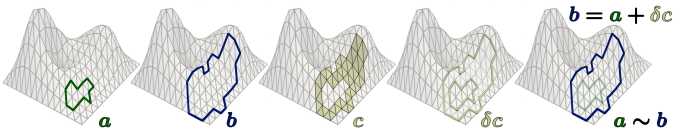

Fig. 3. Two 1-cycles a and b are *homologous* (noted $a \sim b$) if there exists a 2-chain c, such that $b = a + \partial c$. Here, $\partial c = a + b$. By adding a on both sides (modulo 2 coefficients), we indeed have: $b = a + \partial c$, thus $a \sim b$.

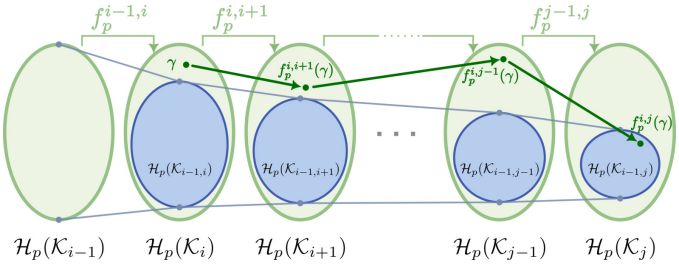

Fig. 4. Tracking homology classes along the filtration with homomorphisms (dark green arrows, illustration adapted from [28]). The class γ is born in $\mathcal{K}_i$: it is not the image through the homomorphism $f_p^{i-1,i}$ of any pre-existing class (i.e. it does not belong to $\mathcal{H}_p(\mathcal{K}_{i-1,i})$, blue set). γ dies in $\mathcal{K}_j$ as it merges with a pre-existing class, i.e. the image through the homomorphism $f_p^{i-1,j}$ of a class already existing at step $i-1$ and still persistent at step j (blue set). In Fig. 2, the dark green cycle c_f has a similar trajectory: it is born at step *(f)* and at step *(g)*, it *merges* with the pre-existing cycle c_c (i.e. c_f becomes homologous to c_c).

$c'' = \sum \alpha_i^{c''} \sigma_i$, where $\alpha_i^{c''} = \alpha_i^c + \alpha_i^{c'}$ and $\alpha_i^{c''} \in \{0, 1\}$. Intuitively, a p-chain can be interpreted as a selection of p-simplices, modeled with a bit mask, where a p-simplex is present in the selection (i.e. with its coefficient valued at 1) only if it has been added an odd number of times. Then, the set of all possible p-chains of $\mathcal{K}$ (along with their modulo 2 addition) forms the *group of chains*, noted $\mathcal{C}_p(\mathcal{K})$.

The boundary of a p-simplex σ_i, noted $\partial\sigma_i$, is given by the sum of its faces of dimension $(p-1)$. Then, the *boundary* of a p-chain c, noted ∂c, is the sum of the boundaries of the simplices of c: $\partial c = \sum \alpha_i^c \partial\sigma_i$. Note that ∂c is itself a $(p-1)$ chain, i.e. $\partial : \mathcal{C}_p(\mathcal{K}) \rightarrow \mathcal{C}_{p-1}(\mathcal{K})$, and that the boundary operator commutes with addition, i.e. $\partial(c + c') = \partial c + \partial c'$.

A p-cycle c is a p-chain such that $\partial c = 0$ and the group of all possible p-cycles is noted $\mathcal{Z}_p(\mathcal{K})$. A p-boundary is a p-chain $c \in \mathcal{C}_p(\mathcal{K})$ which is the boundary of a $(p+1)$-chain $c' \in \mathcal{C}_{p+1}(\mathcal{K})$: $c = \partial c'$. The group of p-boundaries is noted $\mathcal{B}_p(\mathcal{K})$. The fundamental lemma of homology states that $\partial\partial c = 0$ for every p-chain c, for any p [28]. This implies that p-boundaries are necessarily p-cycles ($\mathcal{B}_p \subseteq \mathcal{Z}_p$), but not the other way around: all p-cycles are not necessarily p-boundaries. Such cycles are specifically captured with the notion of *homology group*, which is the quotient group given by: $\mathcal{H}_p(\mathcal{K}) = \mathcal{Z}_p(\mathcal{K})/\mathcal{B}_p(\mathcal{K})$. Specifically, two p-cycles a and b of $\mathcal{Z}_p$ are called *homologous* (noted $a \sim b$), if $b = a + \partial c$ where ∂c is a p-boundary ($\partial c \in \mathcal{B}_p$). Intuitively, this means that two cycles a and b are homologous if one can be transformed into the other, by the addition of the boundary of a $(p+1)$ chain c (Fig. 3), as further discussed in Sec. 2.5. The set of all cycles which are homologous defines a *homology class* (from which anyone can be chosen as a representant). The *order* of $\mathcal{H}_p(\mathcal{K})$ is given by its cardinality, i.e. the number of homology classes. Given the modulo-2 addition between representants, the *rank* of $\mathcal{H}_p(\mathcal{K})$ is given by the maximum number of linearly independent classes (called *generators*) and it is called the p-th Betti number of $\mathcal{K}$, noted $\beta_p(\mathcal{K})$. Intuitively, the p-th Betti number gives the number of *p-dimensional holes* in $\mathcal{K}$, which cannot be filled with a $(p+1)$ chain of $\mathcal{K}$. In practice, given a 3-dimensional simplicial complex $\mathcal{K}$ embedded in $\mathbb{R}^3$, $\beta_0(\mathcal{K})$ corresponds to its number of connected components, $\beta_1(\mathcal{K})$ is its number of handles and $\beta_2(\mathcal{K})$ is its number of voids.

2.4 Persistence diagrams

Persistent diagrams are concise topological data representations which track the evolution of the homology groups during a filtration. In the remainder, we focus on the lexicographic filtration introduced in Sec. 2.2. Since $\mathcal{K}_i \subseteq \mathcal{K}_j$ for any $0 \leq i \leq j \leq n$, it follows that there exists a homomorphism [28] between the homology groups $\mathcal{H}_p(\mathcal{K}_i)$ and $\mathcal{H}_p(\mathcal{K}_j)$, noted $f_p^{i,j} : \mathcal{H}_p(\mathcal{K}_i) \rightarrow \mathcal{H}_p(\mathcal{K}_j)$. This homomorphism $f_p^{i,j}$ keeps track of the relations between the homology classes along the filtration, from $\mathcal{K}_i$ to $\mathcal{K}_j$ (Fig. 4).

Formally, for any $0 \leq i \leq j \leq n$, the p-th persistent homology group, noted $\mathcal{H}_p(\mathcal{K}_{i,j})$, is the image of the homomorphism $f_p^{i,j}$, noted $\mathcal{H}_p(\mathcal{K}_{i,j}) = f_p^{i,j}(\mathcal{H}_p(\mathcal{K}_i))$.

Specifically, we say that a homology class γ is *born at* $\mathcal{K}_i$ if $\gamma \in \mathcal{H}_p(\mathcal{K}_i)$ and $\gamma \notin \mathcal{H}_p(\mathcal{K}_{i-1,i})$ (see Fig. 4): γ is present in $\mathcal{H}_p(\mathcal{K}_i)$ but it is not included in the image by $f_p^{i-1,i}$ of the homology groups of the previous complex in the filtration, $\mathcal{K}_{i-1}$. In other words, γ is present in $\mathcal{H}_p(\mathcal{K}_i)$ but it is not associated to any pre-existing class of $\mathcal{H}_p(\mathcal{K}_{i-1})$ by $f_p^{i-1,i}$.

Symmetrically, we say that a homology class γ born at $\mathcal{K}_i$ *dies at* $\mathcal{K}_j$ if *(i)* $f_p^{i,j-1}(\gamma) \notin \mathcal{H}_p(\mathcal{K}_{i-1,j-1})$ and *(ii)* $f_p^{i,j}(\gamma) \in \mathcal{H}_p(\mathcal{K}_{i-1,j})$, see Fig. 4. In other words, *(i)* the class γ did not exist prior to i and *(ii)* it merged (through $f_p^{j-1,j}$) with another, pre-existing class γ', itself created before i. This destruction of a class upon its merge with another *older* class is often called the *Elder rule* [28]. Note that the birth of a p-dimensional homology class γ occurs on a p-simplex σ_i of $\mathcal{K}_i$, while its death occurs on a $(p+1)$-simplex σ_j of $\mathcal{K}_j$. The pair (σ_i, σ_j) is called a *persistence pair*.

The persistence of a homology class γ which was born in $\mathcal{K}_i$ and which died in $\mathcal{K}_j$ is given by the difference in the corresponding scalar values $\mathcal{P}(\gamma) = f(v_j) - f(v_i)$, where $f(v_j)$ and $f(v_i)$ are respectively the maximum vertex data values of the simplices σ_j and σ_i. Note that a homology class γ which was born in $\mathcal{K}_i$ and whose image by $f_p^{i,n}$ is still included in $\mathcal{H}_p(\mathcal{K}_{i,n})$ is said to have *infinite persistence* (i.e. it is still present in the final complex $\mathcal{K}_n = \mathcal{K}$).

The persistence diagram of dimension p, noted $\mathcal{D}_p(f)$, is a concise encoding of the p-dimensional persistent homology groups. In particular, it embeds each persistent generator γ in the 2D birth/death plane at position $\big(f(v_i), f(v_j)\big)$ and its persistence can be therefore directly read from its height to the diagonal. This has the practical implication that generators with large persistence (typically corresponding to salient features in the data) are located far away from the diagonal, whereas generators with small persistence (typically corresponding to noise) are located in the vicinity of the diagonal, as illustrated in Fig. 5.

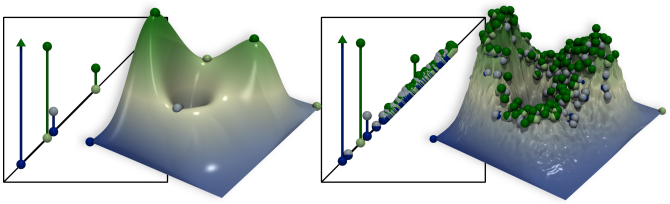

Fig. 5. Persistence diagrams for the lexicographic filtration of a clean (left) and noisy (right) version the terrain toy example (blue bars: $\mathcal{D}_0(f)$, green bars: $\mathcal{D}_1(f)$). Classes with infinite persistence are shown with an upward arrow. Critical simplices are reported with spheres (dark blue: minima, dark green: maxima, other: saddles). The Betti numbers of any step i of the filtration can be directly read from the diagram, by counting the intersections between a horizontal line at height i with the bars of the diagram. The *persistence* of each topological feature is given by the height of each bar. In practice, noise in data tends to create short bars (right) which can be easily distinguished from the main signal (long bars), in this case, two prominent hills and one salient pit.

To measure a distance between two diagrams $\mathcal{D}(f_i)$ and $\mathcal{D}(f_j)$ (as done in Sec. 9.3.2), a typical pre-processing step consists in augmenting each diagram with the diagonal projection of the off-diagonal points of the other diagram:

$$\mathcal{D}'(f_i) = \mathcal{D}(f_i) \cup \{\Delta(p_j) \mid p_j \in \mathcal{D}(f_j)\}$$
$$\mathcal{D}'(f_j) = \mathcal{D}(f_j) \cup \{\Delta(p_i) \mid p_i \in \mathcal{D}(f_i)\},$$

where $\Delta(p_i) = (\frac{x_i+y_i}{2}, \frac{x_i+y_i}{2})$ stands for the diagonal projection of the off-diagonal point $p_i = (x_i, y_i) \in \mathcal{D}(f_i)$. Intuitively, this augmentation phase inserts dummy features in the diagram (with zero persistence, along the diagonal), hence preserving the topological information of the diagrams. This augmentation guarantees that the two diagrams now have the same number of points ($|\mathcal{D}'(f_i)| = |\mathcal{D}'(f_j)|$), which facilitates the evaluation of their distance.

Given two points $p_i = (x_i, y_i) \in \mathcal{D}'(f_i)$ and $p_j = (x_j, y_j) \in \mathcal{D}'(f_j)$, the ground distance d_q ($q > 0$) in the 2D birth/death space is given by:

$$d_q(p_i, p_j) = (|x_j - x_i|^q + |y_j - y_i|^q)^{1/q} = \|p_i - p_j\|_q.$$

By convention, $d_q(p_i, p_j)$ is set to zero between diagonal points ($x_i = y_i$ and $x_j = y_j$). Then, the L^q-Wasserstein distance [95], noted W_q, is given by:

$$W_q(\mathcal{D}'(f_i), \mathcal{D}'(f_j)) = \min_{\phi \in \Phi} \Big(\sum_{p_i \in \mathcal{D}'(f_i)} d_q(p_i, \phi(p_i))^q \Big)^{1/q},$$

where Φ is the set of all possible assignments ϕ mapping a point $p_i \in \mathcal{D}'(f_i)$ to a point $p_j \in \mathcal{D}'(f_j)$ (possibly its diagonal projection, indicating its destruction).

2.5 The algorithm *"PairCells"*

Zomorodian [102] describes an iterative algorithm called *"PairCells"* for the computation of persistence diagrams. We sketch its main steps here as our approach builds on top of it. This algorithm (Alg. 1) observes, for each step i of the input filtration, the effect of the insertion of a d_i-simplex σ_i on the set of (d_i-1)-cycles homologous to its boundary $\partial\sigma_i$. In particular, if $\partial\sigma_i$ was not already trivial (i.e. homologous to an empty cycle) in $\mathcal{K}_{i-1}$, then the insertion of σ_i in $\mathcal{K}_i$ will now make $\partial\sigma_i$ become trivial ($\partial\sigma_i \sim 0$). By transitivity,

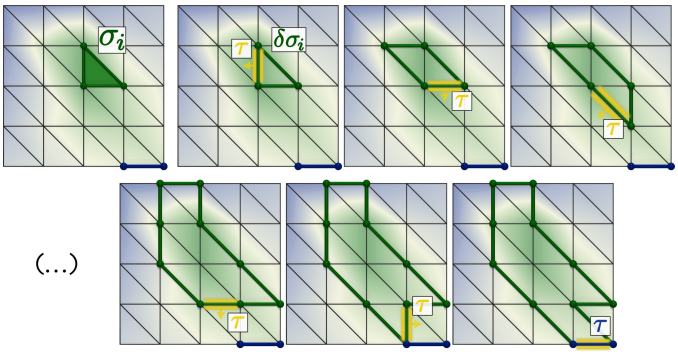

Fig. 6. *Homologous expansion* given a (d_i+1)-simplex σ_i (left to right, top to bottom). The boundary $\partial\sigma_i$ (green curve) is iteratively expanded by: *(i)* selecting its highest (d_i)-simplex τ (yellow, line 7, Alg. 1) and *(ii)* adding the boundary of the expanded (d_i+1)-chain paired with τ (line 13, Alg. 1), here the triangle adjacent to τ. The expansion stops (line 10, Alg. 1) when an unpaired (d_i)-simplex (blue edge) is included in the expanded boundary (bottom right). Then, a persistence pair (τ, σ_i) is created if the expanded boundary is not empty (line 21, Alg. 1).

Algorithm 1 Reference *"PairCells"* [102]

Input: Lexicographic filtration of $\mathcal{K}$ by f.
Output: Persistence diagrams $\mathcal{D}_0(f)$, $\mathcal{D}_1(f)$ and $\mathcal{D}_2(f)$.

1: **for** $j \in [1, n]$ **do**
2: // Process the (d_i+1)-simplex σ_j
3: $Pair(\sigma_j) \leftarrow \emptyset$
4: $Chain(\sigma_j) \leftarrow \sigma_j$
5: // Homologous expansion of $\partial\sigma_j$
6: **while** $\partial\big(Chain(\sigma_j)\big) \neq 0$ **do**
7: $\tau \leftarrow \max\Big(\partial\big(Chain(\sigma_j)\big)\Big)$
8: **if** $Pair(\tau) == \emptyset$ **then**
9: // τ created a (d_i)-cycle
10: **break**
11: **else**
12: // Expand chain (with homologous boundary)
13: $Chain(\sigma_j) \leftarrow Chain(\sigma_j) + Chain\big(Pair(\tau)\big)$
14: **end if**
15: **end while**
16: **if** $\partial\big(Chain(\sigma_j)\big) \neq 0$ **then**
17: // A non-trivial cycle homologous to $\partial\sigma_j$ exists (l. 10)
18: $\tau \leftarrow \max\Big(\partial\big(Chain(\sigma_j)\big)\Big)$
19: $Pair(\sigma_j) \leftarrow \tau$
20: $Pair(\tau) \leftarrow \sigma_j$
21: $\mathcal{D}_{d_i}(f) \leftarrow \mathcal{D}_{d_i}(f) \cup (\tau, \sigma_j)$
22: **end if**
23: **end for**

all the cycles c homologous to $\partial\sigma_i$ (its homology class) now become trivial as well, hence completing a persistence pair in $\mathcal{D}_{d_i-1}(f)$, that is, *filling* a d_{i-1}-dimensional hole of $\mathcal{K}_{i-1}$.

Thus, for each step i of the filtration, the algorithm reconstructs (d_i-1)-cycles in $\mathcal{K}_i$ which are homologous to $\partial\sigma_i$. This is achieved by a process that we call *homologous expansion* (Fig. 6), which iteratively expands a chain $Chain(\sigma_i)$, whose boundary $\partial\big(Chain(\sigma_i)\big)$ is homologous to $\partial\sigma_i$ by construction (Alg. 1, lines 6 to 15). This expansion is achieved by considering (d_i-1)-simplices in decreasing filtration order, i.e. by selecting at each iteration the highest simplex (Alg. 1, line 7), and by stopping at the first unpaired (d_i-1)-simplex τ (Alg. 1, line 10), responsible for the creation of the latest (i.e. youngest) homologous (d_i-1)-cycle

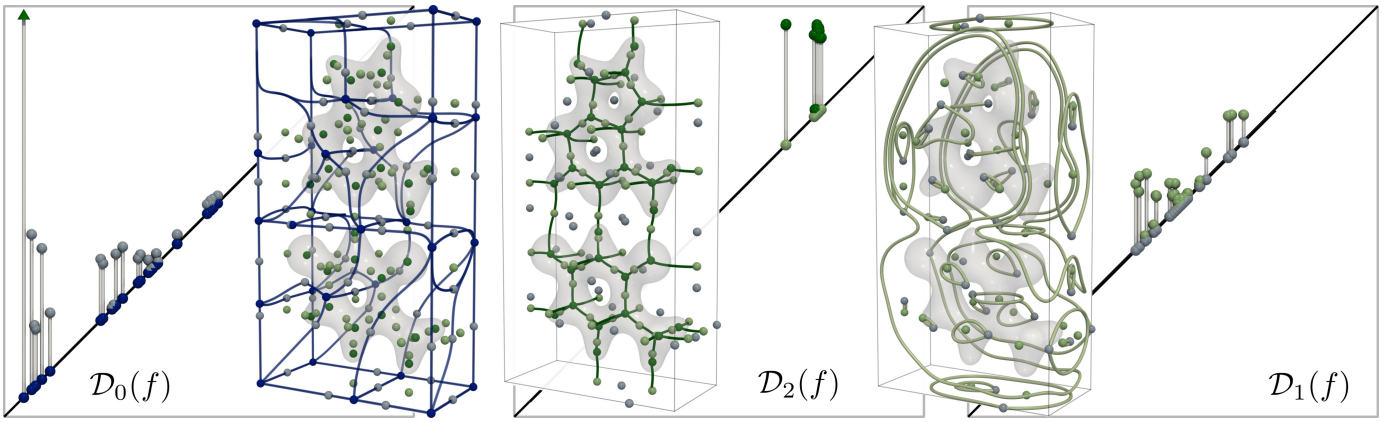

Fig. 7. Overview of our approach on a quantum chemistry dataset (grey surface: electron density level set). The diagram $\mathcal{D}_0(f)$ is first efficiently computed (left) by processing the unstable sets (blue curves) of the 1-saddles (light blue spheres) of f (Sec. 5.1). Simultaneously (center), $\mathcal{D}_{d-1}(f)$ is computed symmetrically, by processing the stable sets (green curves) of the $(d-1)$-saddles (light green spheres) of f (Sec. 5.2). Finally (right), only the remaining, unpaired 2-saddles (light green spheres) are considered by our new algorithm *"PairCriticalSimplices"* (Sec. 4) to efficiently compute $\mathcal{D}_1(f)$ by homologous expansions of their associated 1-cycles (white curves). This stratification strategy drastically reduces the number of unpaired critical simplices at each step, leaving only a small portion for the last and most computationally expensive part of our approach (right).

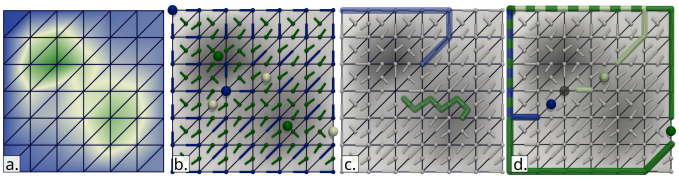

Fig. 8. Given an input PL scalar field f *(a)*, a *discrete gradient field* $\mathcal{G}$ pairs i-simplices with $(i+1)$-simplices *(b)* (blue: vertex-edge arrows, green: edge-triangle arrows). The remaining unpaired simplices are *critical* (large spheres, blue: minima, white: saddles, green: maxima). Two *discrete integral lines* (or *"v-paths"*) are shown in *(c)* (blue: vertex-edge integral line, green: edge-triangle integral line). The *discrete unstable set* of each saddle, *(d)*, is the collection of all discrete integral lines starting from it (each discrete unstable set is colored according to its saddle).

in $\mathcal{K}_{i-1}$, hence effectively enforcing the *Elder rule* (Sec. 2.4). Then the persistence pair (τ, σ_i) is created in $\mathcal{D}_{d_i-1}(f)$. If $\partial\sigma_i$ was trivial initially in $\mathcal{K}_{i-1}$ when starting the expansion, $Chain(\sigma_i)$ is extended until its boundary becomes empty and no pair will be created in $\mathcal{D}_{d_i-1}(f)$.

2.6 Discrete Morse Theory (DMT)

We now conclude this section of preliminaries with notions of discrete Morse theory [32], or DMT for short (which we restrict here to simplicial complexes), as it is instrumental in our approach to accelerate the algorithm *"PairCells"*.

We call a discrete vector a pair formed by a simplex $\sigma_i \in \mathcal{K}$ (of dimension i) and one of its co-facets σ_{i+1} (i.e. one of its co-faces of dimension $i+1$), noted $\{\sigma_i < \sigma_{i+1}\}$. σ_{i+1} is usually referred to as the *head* of the vector, while σ_i is its tail. Examples of discrete vectors include a pair between a vertex and one of its incident edges, or a pair between an edge and a triangle containing it (see Fig. 8). A *discrete vector field* on $\mathcal{K}$ is then defined as a collection $\mathcal{V}$ of pairs $\{\sigma_i < \sigma_{i+1}\}$ such that each simplex of $\mathcal{K}$ is involved in at most one pair. A simplex σ_i which is involved in no discrete vector of $\mathcal{V}$ is called a *critical simplex*.

A discrete integral line, or *v-path*, is a sequence of discrete vectors $\{\{\sigma_i^0 < \sigma_{i+1}^0\}, \ldots, \{\{\sigma_i^k < \sigma_{i+1}^k\}\}$ such that *(i)* $\sigma_i^j \neq \sigma_i^{j+1}$ (i.e. the tails of two consecutive vectors are distinct) and *(ii)* $\sigma_i^{j+1} < \sigma_{i+1}^j$ (the tail of a vector in the sequence is a face of the head of the previous vector in the sequence) for any $0 < j < k$. We say that a discrete integral line *terminates* at a critical simplex σ_i if σ_i is a face of the head of its last vector $\{\sigma_i^k < \sigma_{i+1}^k\}$ (i.e. $\sigma_i < \sigma_{i+1}^k$). Symmetrically, we say that a discrete integral line *starts* at a critical simplex σ_{i+1} if σ_{i+1} a co-face of the tail of its first vector σ_i^0 (i.e. $\sigma_i^0 < \sigma_{i+1}$). By analogy with the smooth setting, this notion of discrete integral lines therefore starts and terminates at critical points. The collection of all the discrete integral lines terminating in a given critical simplex σ_i is called the *discrete stable set* of σ_i and it is noted $\mathcal{K}(\sigma_i)$. Symmetrically, the collection of all the discrete integral lines starting at a given critical simplex σ_i is called the *discrete unstable set* of σ_i (Fig. 8) and it is noted $\mathcal{K}'(\sigma_i)$.

A discrete vector field that is such that all of its possible discrete integral lines (i.e. all of its v-paths) are acyclic is called a *discrete gradient field* [32], noted $\mathcal{G}$. Then, the critical simplices of $\mathcal{G}$ are discrete analogs to the critical points from the smooth setting [66], [70]. Their dimension i corresponds to the smooth notion of index (number of negative eigenvalues of the Hessian): local minima occur on vertices, i-saddles on i-simplices and maxima on d-simplices.

As detailed in Sec. 2.1, for typical scalar data, the input is generically provided as a PL scalar field f. Given this input, several algorithms have been proposed for the computation of a discrete gradient field $\mathcal{G}$ [40], [42], [43], [44], [79], [81], [92]. In particular, Robins et al. introduced an algorithm based on homotopic expansions [79], which guarantees that each resulting critical d_i-simplex σ_i belongs to the lower star of a PL critical point of index d_i.

3 Overview

Figure 7 provides an overview of our approach. We assume that $\mathcal{K}$ is connected (otherwise, each connected component

is processed independently by our algorithm). While we focus on simplicial complexes in our work (Sec. 1), our algorithm can be applied in principle to arbitrary cell complexes.

First, the discrete gradient of the input data is computed along with its critical simplices via homotopic expansion [79]. The rest of our approach consists in grouping the resulting critical simplices into persistence pairs. We describe for that a generic extension (Sec. 4) of the algorithm *"PairCells"* [102], which is expressed in the DMT setting for improved performances, and that we call *"PairCriticalSimplices"*. While each diagram $\mathcal{D}_0(f)$, $\mathcal{D}_1(f)$ and $\mathcal{D}_2(f)$ could be computed with this algorithm, we describe instead a stratification strategy, called *sandwiching* (described below), which further, drastically improves performances.

Second (Fig. 7, left), the diagram $\mathcal{D}_0(f)$ is obtained by processing the unstable sets of the 1-saddles of f (Sec. 5.1).

Third (Fig. 7, center) the diagram $\mathcal{D}_{d-1}(f)$ is obtained by processing the stable sets of the $(d-1)$-saddles of f (Sec. 5.2).

Next (Fig. 7, right) for 3D data only, the diagram $\mathcal{D}_1(f)$ is computed by restricting our novel algorithm *"PairCriticalSimplices"* (Sec. 4) to the remaining set of unpaired 2-saddles.

Last, the remaining critical simplices are necessarily involved in classes of infinite persistence, capturing the (infinitely persistent) homology groups of $\mathcal{K}$ (Sec. 6).

4 Pairing Critical Simplices

This section presents our adaptation of the seminal algorithm *"PairCells"* (Sec. 2.5) to the DMT setting (Sec. 2.6), resulting in substantial performance gains.

4.1 Observations

This section describes three main observations regarding the algorithm *"PairCells"* (Alg. 1), which are at the basis of our adaptation, described in Sec. 4.2.

***(a)* Dimension separability** First, one can observe that the different persistence diagrams $\mathcal{D}_0(f)$, $\mathcal{D}_1(f)$ and $\mathcal{D}_2(f)$ can be computed in a separated manner, one after the other. Indeed, a given d_i-simplex σ_i can only be involved in *(i)* the destruction of a (d_i-1)-cycle (if $\partial\sigma_i$ was not trivial), or *(ii)* the creation of a d_i-cycle (if $\partial\sigma_i$ was trivial). For instance, the addition of a 1-simplex in the lexicographic filtration connects two vertices belonging either *(i)* to distinct connected components (in which case a persistence pair is added to $\mathcal{D}_0(f)$, line 21, Alg. 1) or *(ii)* to the same connected component (in which case a new 1-cycle is created line 10, Alg. 1, to be later added to $\mathcal{D}_1(f)$). Thus, if the persistence diagram $\mathcal{D}_{i-1}(f)$ is available, the diagram $\mathcal{D}_i(f)$ can be efficiently computed by restricting Alg. 1 to the $i+1$ simplices of $\mathcal{K}$ (still processed in lexicographic order). Then, each i-simplex which has not been paired yet in $\mathcal{D}_{i-1}(f)$ will be guaranteed to be the creator of a i-cycle, and thus involved in a persistence pair of $\mathcal{D}_i(f)$. This dimension separability is at the basis of our *sandwiching* stratification strategy.

***(b)* Boundary caching** The original algorithm *"PairCells"* proceeds to the homologous expansion of d_i-cycles by iteratively growing (d_i+1)-chains (line 13, Alg. 1), and by explicitly extracting their boundary when needed (for instance, line 6, Alg. 1). However, each of these extractions requires a pass which is linear with the size of the chain. This can be improved by *caching* the boundary of the chain created at each simplex, and by manipulating boundaries directly in the expansion process, instead of the corresponding chains (as described in Sec. 4.2). This adaptation of the expansion process would still require a linear pass (to perform the modulo-2 addition of boundary simplices), but this time on a much smaller set (boundaries are in practice much smaller than their corresponding chains).

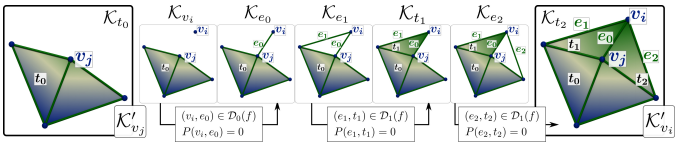

Fig. 9. The *lexicographic* filtration introduces simplices one by one (left to right) and can capture many zero-persistence pairs ($(v_i, e_0), (e_1, t_1)$ and (e_2, t_2)), which are not captured by the *lower-star* filtration, which introduces simplices by chunks of lower stars (steps $\mathcal{K}'_{v_j}$ and $\mathcal{K}'_{v_i}$).

***(c)* Zero persistence skip** By definition of a filtration (Sec. 2.2), a given simplex cannot be inserted in the filtration before its facets. A practical implication of this observation is that many, zero-persistence pairs are created by the algorithm *"PairCells"*. For instance, the insertion of the last edge e_1 of a given triangle t_1 often creates a new 1-cycle (step $\mathcal{K}_{e_1}$, Fig. 9) which is immediately filled by the subsequent insertion of t_1 (step $\mathcal{K}_{t_1}$, Fig. 9), creating a persistence pair (e_1, t_1) in $\mathcal{D}_1(f)$. However, since the persistence of a pair is given by the difference between the maximum vertex data values of e_1 and t_1 (Sec. 2.4), we have in such cases $\mathcal{P}(e_1, t_1) = 0$, as e_1 and t_1 have to share the highest vertex of t_1, given the lexicographic order. Thus, a significant time (evaluated in Sec. 9.2) is spent in practice by the algorithm *"PairCells"* to construct persistence pairs with zero persistence, which consequently do not contribute any information to the output diagram. We address this issue with DMT. In particular, the discrete gradient $\mathcal{G}$ computed via homotopic expansion [79] guarantees that each critical simplex belongs to the lower star of a PL critical point of f, which exactly coincide themselves to changes in the homology groups of the lower star filtration (Sec. 2.6). Then, all the remaining regular simplices (involved in a discrete vector of $\mathcal{G}$) induce homology changes which are *not* captured by the lower star filtration (steps marked with a black frame in Fig. 9, $\mathcal{K}'_{v_j}$ and $\mathcal{K}'_{v_i}$), and which, equivalently, have zero persistence. Then, it follows that all the zero-persistence pairs of the lexicographic filtration can be efficiently skipped in a pre-process, by discarding from the computation all the simplices which are not critical, as only the critical simplices will induce non-zero persistence homology changes (i.e., captured by the lower star filtration, Sec. 2.6).

4.2 Algorithm

Our adaptation of the algorithm *"PairCells"* to the DMT setting, called *"PairCriticalSimplices"*, directly results from the above observations.

***(a)* Zero-persistence skip** First, Alg. 2 is used in a pre-process to skip the zero-persistence pairs of the lexicographic filtration. This algorithm first computes the discrete gradient field $\mathcal{G}$ given the input PL scalar field $f : \mathcal{K} \to \mathbb{R}$

Algorithm 2 Our pre-processing algorithm *"Zero persistence skip"*

Input: Lexicographic filtration of $\mathcal{K}$ by f.
Output: Ordered sets C_{d_i} or critical d_i-simplices.

1: // Discrete gradient (homotopic expansion [79])
2: $\mathcal{G} \leftarrow DiscreteGradient(\mathcal{K}, f)$
3: **for** $i \in [1, n]$ **do**
4: // Process a d_i-simplex σ_i
5: **if** $\exists\ \{\sigma_i < \sigma_j\}$ or $\exists\ \{\sigma_j < \sigma_i\}$ **then**
6: // σ_i is involved in a discrete vector with σ_j
7: // Skip zero-persistence pair
8: $Pair(\sigma_i) \leftarrow \sigma_j$
9: $Pair(\sigma_j) \leftarrow \sigma_i$
10: **else**
11: // σ_i is a critical simplex
12: $Pair(\sigma_i) \leftarrow \emptyset$
13: $C_{d_i} \leftarrow C_{d_i} \cup \sigma_i$
14: **end if**
15: **end for**
16: **for** $d_i \in [0, d]$ **do**
17: $Sort(C_{d_i})$ // by lexicographic order
18: **end for**

Algorithm 3 Our algorithm *"PairCriticalSimplices"*.

Input: Ordered set C_{d_i+1} of critical (d_i+1)-simplices
Output: Persistence diagrams $\mathcal{D}_{d_i}(f)$.

1: **for** $j \in C_{d_i+1}$ **do**
2: // Process the (d_i+1)-simplex σ_j
3: $Boundary(\sigma_j) \leftarrow \partial\sigma_j$
4: // Homologous expansion of $\partial\sigma_j$
5: **while** $Boundary(\sigma_j) \neq 0$ **do**
6: $\tau \leftarrow \max\big(Boundary(\sigma_j)\big)$
7: **if** $Pair(\tau) == \emptyset$ **then**
8: // τ is unpaired and thus created a d_i-cycle.
9: **break**
10: **else**
11: // Expand boundary
12: $Boundary(\sigma_j) \leftarrow$
13: $Boundary(\sigma_j) + Boundary\big(Pair(\tau)\big)$
14: **end if**
15: **end while**
16: **if** $Boundary(\sigma_j) \neq 0$ **then**
17: // A non-trivial cycle homologous to $\partial\sigma_j$ exists (l. 9)
18: $\tau \leftarrow \max\big(Boundary(\sigma_j)\big)$
19: $Pair(\sigma_j) \leftarrow \tau$
20: $Pair(\tau) \leftarrow \sigma_j$
21: $\mathcal{D}_{d_i}(f) \leftarrow \mathcal{D}_{d_i}(f) \cup (\tau, \sigma_j)$
22: **end if**
23: **end for**

with homotopic expansion [79] (line 2). Then, each simplex σ_i involved in a discrete vector of $\mathcal{G}$ is marked as belonging to a zero-persistence pair. By convention, we pair together the head and the tail of a given vector (line 9). Otherwise, critical d_i-simplices are marked as unpaired (line 12) and are added to the set C_{d_i} of critical d_i-simplices (line 13). Once all discrete vectors have been processed, each set C_{d_i} is sorted by increasing lexicographic order (line 17).

***(b)* Pair Critical Simplices** We now present our algorithm *"PairCriticalSimplices"* (Alg. 3). For a given simplex dimension $d_i + 1$, this algorithm takes as an input the ordered set C_{d_i+1} of critical $(d_i + 1)$-simplices and produces the diagram $\mathcal{D}_{d_i}(f)$. This assumes that the diagram $\mathcal{D}_{d_i-1}(f)$ has already been computed (see the *Dimension separability* property, Sec. 4.1) and that consequently, the critical d_i-simplices involved in $\mathcal{D}_{d_i-1}(f)$ have already been paired. Since all the regular simplices inserted in between two critical simplices by the lexicographic filtration are guaranteed to belong to zero-persistence pairs (*Zero persistence skip* property, Sec. 4.1), Alg. 3 simply processes the critical simplices of C_{d_i+1} in increasing lexicographic order. For each critical simplex σ_j, the standard, downwards homologous expansion of the classical algorithm *"PairCells"* is employed (line 5). However, as discussed in Sec. 4.1 (*Boundary caching* property), our algorithm directly manipulates boundaries instead of the corresponding chains. Then, when a boundary homologous to $\partial\sigma_j$ is expanded (line 12), a modulo-2 addition is employed by manipulating a bit mask (indicating if a simplex σ_i is already preset in $Boundary(\sigma_j)$). The rest of the algorithm is identical to the original algorithm *"PairCells"*: if the expanded boundary for the simplex σ_j is not-empty (line 16), this means that a critical simplex τ, creating a d_i-cycle, has been found during the downward homologous expansion (line 9). Then, a persistence pair (τ, σ_j) is created between σ_j and the highest d_i-simplex τ of its expanded boundary $Boundary(\sigma_j)$ (line 21).

5 Extremum-Saddle Persistence Pairs

Our algorithm *"PairCriticalSimplices"* (Sec. 4.2) could be used as-is to compute the diagrams $\mathcal{D}_0(f)$, $\mathcal{D}_1(f)$ and $\mathcal{D}_2(f)$ one after the other, already resulting in substantial performance gains over the seminal algorithm *"PairCells"* (see Sec. 9.2). In this section, we further exploit the *Dimension separability* property (Sec. 4) to further speedup the process.

5.1 Minimum-Saddle Persistence Pairs

This section introduces a faster alternative to the algorithm *"PairCriticalSimplices"*, for the specific case of $\mathcal{D}_0(f)$.

***(a)* Unstable set restriction** This algorithm is based on the key observation that, for the specific case of $\mathcal{D}_0(f)$, given a critical 1-simplex σ_1^0, the homologous expansion described in Alg. 3 exactly coincides with the discrete unstable set (Sec. 2.6) of σ_1^0 (see Fig. 10). In particular, at the first iteration of the algorithm, τ will be selected as one of the two vertices of σ_1^0, noted σ_0^0. If σ_0^0 is not a minimum itself, it has to be paired (given the discrete gradient $\mathcal{G}$, Alg. 2) with another edge σ_1^1, being one of its co-facets, with $\sigma_1^1 \neq \sigma_1^0$. Since in simplicial complexes, edges are guaranteed to connect distinct vertices, we then have the property that the only other facet of σ_1^1 is another vertex $\sigma_0^1 \neq \sigma_0^0$. Thus, so far, the first iteration of the homologous expansion visited a sequence of edges and vertices $\{\sigma_1^0, \sigma_0^0, \sigma_1^1, \sigma_0^1\}$, such that for each item σ_i^j in this sequence we have: *(i)* $\sigma_i^j \neq \sigma_i^{j+1}$ and *(ii)* $\sigma_i^{j+1} < \sigma_{i+1}^j$, which exactly coincides with the definition of a discrete integral line (Sec. 2.6). Then, along the iterations of Alg. 3, two integral lines, started at each vertex of σ_1^0, will be iteratively constructed, by selecting at each iteration the highest extremity of the two integral lines (line 6). This process terminates when one of the two integral lines reaches a minimum σ_0' (i.e., an unpaired vertex, line 9). At this point, we have $Boundary(\sigma_1^0) = \{\sigma_0' + \sigma_0''\}$, where σ_0'' is the extremity of the other integral line (Fig. 10, left). Then, if $\sigma_0' \neq \sigma_0''$ (i.e. σ_1^0 did not create a 1-cycle), we have $Boundary(\sigma_1^0) \neq \emptyset$ (line 16) and a persistence pair (σ_0', σ_1^0) is created (line 21). Then, at this stage, the boundary

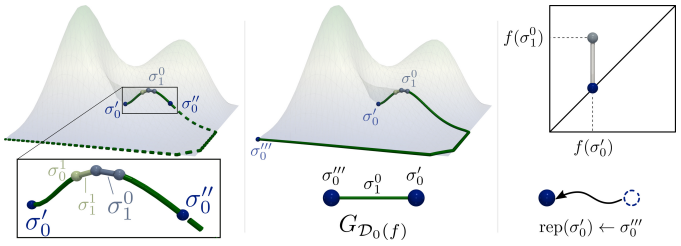

Fig. 10. Overview of our algorithm for the computation of $\mathcal{D}_0(f)$. *Left:* the homologous expansion of Alg. 3 applied to a critical edge σ_1^0 turns out to be restricted to the discrete unstable sets of σ_1^0, until it reaches a first unpaired vertex (σ_0'). There, the expansion pauses and a persistence pair (σ_0', σ_1^0) is created. At this point, the expanded boundary $Boundary(\sigma_1^0)$ is equal to $\{\sigma_0' + \sigma_0''\}$. *Center:* It is possible that the expanded boundary of another critical edge later hits the now paired vertex σ_0', in which case it will return the expanded boundary of its paired simplex (σ_1^0), that is $Boundary(\sigma_1^0)$. This has the effect of *resuming* the homologous expansion from σ_0'', still on the discrete unstable set of σ_1^0, down to σ_0'''. Our overall algorithm expedites this homologous expansion by collapsing all regular simplices along the unstable sets of critical edges, resulting in a graph $G_{\mathcal{D}_0(f)}$ (bottom), whose arcs are the critical edges and the nodes the vertices at the end of their unstable sets. *Right:* The graph $G_{\mathcal{D}_0(f)}$ is processed with a Union-Find data structure by adding its arcs in increasing (original) lexicographic order and a new persistence pair is created in $\mathcal{D}_0(f)$ after the insertion of each arc.

expansion completed the first integral line from σ_1^0 down to σ_0' and paused the second integral line at σ_0'' and we have, by construction, $Boundary(\sigma_1^0) \sim \partial\sigma_1^0$. Next, it is possible that later in the algorithm, the expanded boundary $Boundary(\sigma_1')$ of another critical 1-simplex σ_1' hits the minimum σ_0'. In such a case, the expanded boundary of its paired simplex (i.e. $Boundary(\sigma_1^0)$) will then be added (modulo-2) to $Boundary(\sigma_1')$ (line 13) and the second integral line started in σ_1^0 (paused at σ_0'') will eventually be resumed from σ_0'' until it hits another minimum (Fig. 10, center).

Overall, for $\mathcal{D}_0(f)$, Alg. 3 will exactly visit the edges and vertices of $\mathcal{K}$ which are located on the unstable sets of the critical 1-simplices. It follows that the homologous expansion of a critical 1-simplex σ_1 can be accelerated by directly considering it unstable set, whose boundary ($\{\sigma_0' + \sigma_0'''\}$, c.f. above) is homologous by construction to $\partial\sigma_1$.

Note that this observation no longer holds in higher dimensions. For instance, when constructing $\mathcal{D}_1(f)$ on a 3-dimensional simplicial complex, in contrast to the case of $\mathcal{D}_0(f)$ described above, the unstable set of a critical 2-simplex σ_2 may become non-manifold (as described by Gyulassy and Pascucci [47] in the study of Morse-Smale complexes, yielding multiple integral lines between a given pair of critical simplices). In such a case, the boundary of the unstable set of σ_2 is no longer exactly homologous to $\partial\sigma_2$ (due to the non-manifold elements of the surface) and the acceleration described above is no longer applicable.

***(b)* Unstable set compression** The computation of $\mathcal{D}_0(f)$ can be further accelerated by *compressing* all unstable sets. Given the unstable sets of the critical 1-simplices, we collapse all their regular edges (which are involved in zero-persistence pairs, Sec. 4.1). This collapse eventually results in a graph $G_{\mathcal{D}_0(f)}$ (Fig. 10, center), whose nodes and arcs respectively correspond to the vertices and edges of $\mathcal{K}$ which are left unpaired by $\mathcal{G}$ and whose adjacency relations are determined by the input unstable sets. At this point, $G_{\mathcal{D}_0(f)}$ can be directly given as an input to Alg. 3 to compute $\mathcal{D}_0(f)$.

***(c)* Connectivity tracking** Given $G_{\mathcal{D}_0(f)}$, we further accelerate the process and simplify Alg. 3 by exploiting the specific dimensionality of $\mathcal{D}_0(f)$. In particular, in the case of $\mathcal{D}_0(f)$, Alg. 3 visits the arcs of $G_{\mathcal{D}_0(f)}$ in increasing (original) lexicographic order. For a given arc σ_1, two cases can occur.

First *(i)*, the highest node σ_0 of σ_1 has not been visited yet by any expansion (line 7) and a persistence pair (σ_0, σ_1) is created in $\mathcal{D}_0(f)$ (line 21). Then the arc σ_1 can be collapsed (similarly to regular edge compression, above paragraph (b)) to indicate that it can no longer be paired by the algorithm. This collapse can be modeled by a *union* operation, indicating that the other node σ_0' of σ_1 becomes the *representant* of σ_0 (which can no longer be paired).

Second *(ii)*, the highest vertex σ_0 of σ_1 has already been visited by a prior expansion, in which case we need to efficiently *find* its other boundary vertex σ_0' (line 13) to resume the expansion there.

Overall, $\mathcal{D}_0(f)$ can be computed from $G_{\mathcal{D}_0(f)}$ by collapsing its arcs as they are visited and recording these collapses with a *union* operation, such that boundary nodes can later be retrieved with a *find* operation. This can be efficiently implemented with a Union-Find data structure [21], since for $\mathcal{D}_0(f)$, each node needs to record only one representant (the representant of the other node of its paired arc).

***(d)* Summary** Overall (Fig. 10), our algorithm computes $\mathcal{D}_0(f)$ by first constructing the unstable sets of each critical 1-simplex. Next, each regular edge in these unstable sets is collapsed to create the graph $G_{\mathcal{D}_0(f)}$. Finally, $G_{\mathcal{D}_0(f)}$ is processed with a Union-Find data structure [21] to compute $\mathcal{D}_0(f)$. Initially a Union-Find node $UF(\sigma_0)$ is created for each node σ_0 of $G_{\mathcal{D}_0(f)}$ and the arcs of $G_{\mathcal{D}_0(f)}$ are processed in increasing (original) lexicographic order. Given an arc σ_1, its two expanded boundary nodes σ_0 and σ_0' are efficiently retrieved by applying the *find* operation on the two nodes of σ_1. Then, if σ_0 is strictly higher than σ_0', the persistence pair (σ_0, σ_1) is created in $\mathcal{D}_0(f)$ and a *union* operation is performed between the nodes $UF(\sigma_0)$ and $UF(\sigma_0')$, and the unpaired node $UF(\sigma_0')$ is used as a representant. Overall, our algorithm for computing $\mathcal{D}_0(f)$ can be interpreted as an adaptation to the DMT setting of earlier work on monotone paths for merge tree construction [17], [20], [64], [83], [98].

5.2 Saddle-Maximum Persistence Pairs

In this section, we detail our strategy for the computation of $\mathcal{D}_{d-1}(f)$. In particular, we exploit within the DMT setting the duality argument discussed by Edelsbrunner and Harer [28] in the case of surfaces, recently discussed for general cell complexes in higher dimensions [34]. Specifically, this duality argument (illustrated in Fig. 11) states that, at a given step i of the filtration, the $(d-1)$-dimensional voids of $\mathcal{K}_i$, under certain conditions, exactly coincide with the connected components of the *complement* $\mathcal{K}_i^*$ of $\mathcal{K}_i$.

Formally, let $\mathcal{K}^*$ be the dual cell complex of $\mathcal{K}$. Specifically, each $(d-i)$-simplex σ_i of $\mathcal{K}$ is represented by an i-dimensional cell σ_i^* in $\mathcal{K}^*$. Moreover, given two simplices $\sigma_i < \sigma_j$ in $\mathcal{K}$, we have $\sigma_j^* < \sigma_i^*$ in $\mathcal{K}^*$ (i.e. face-coface relations are reversed). Then, it follows that each diagram $\mathcal{D}_{d-k-1}(f)$ of the lexicographic filtration of $\mathcal{K}$ is equal to the opposite of the diagram $\mathcal{D}_k(-f)$ (i.e. the diagram of the

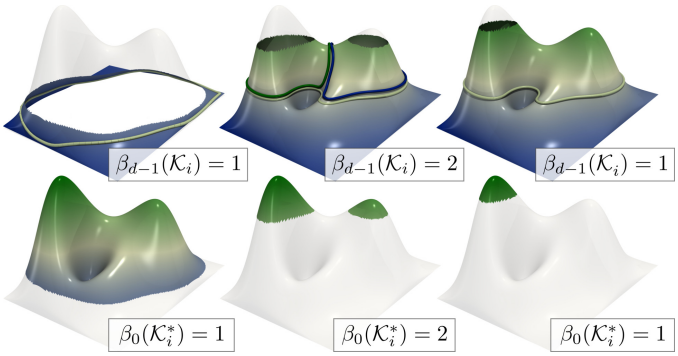

Fig. 11. The Betti number $\beta_{d-1}(\mathcal{K}_i)$ of a given step of the filtration $\mathcal{K}_i$ (top) equals the Betti number $\beta_0(\mathcal{K}_i^*)$ of the complement $\mathcal{K}_i^*$ of $\mathcal{K}_i$ (bottom), assuming that the boundary $\partial\mathcal{K}$ of $\mathcal{K}$ is fully included in $\mathcal{K}_i$.

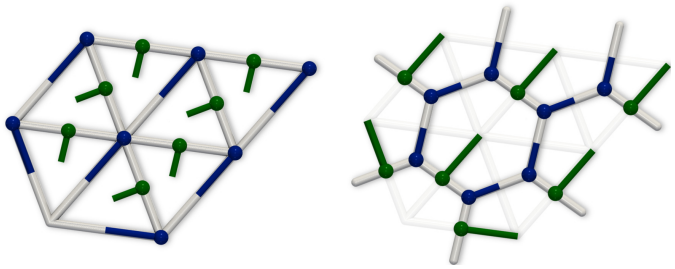

Fig. 12. Given a discrete gradient field $\mathcal{G}$ defined on $\mathcal{K}$ (left), its *dual* discrete gradient field $\mathcal{G}^*$ (right) is obtained by considering the dual cell complex $\mathcal{K}^*$ and reverting each arrow of $\mathcal{G}$: each vertex-edge arrow (blue, left) becomes an edge-face arrow (green, right) while each edge-triangle arrow (green, left) becomes a vertex-edge arrow (blue, right), along which unstable sets can be easily defined and computed.

backward lexicographic filtration, $-f$) of $\mathcal{K}^*$ (see Garin et al. [34], Theorem 2.1). This implies in particular that $\mathcal{D}_{d-1}(f)$ can be computed very efficiently by applying the algorithm for $\mathcal{D}_0(f)$ described in Sec. 5.1 to the *backward* filtration (i.e.$-f$, reverse order) of the dual $\mathcal{K}^*$ of $\mathcal{K}$. This observation nicely translates to the DMT setting, as described next.

A dual discrete gradient vector field $\mathcal{G}^*$ (Fig. 12) can be easily defined on the dual $\mathcal{K}^*$ of $\mathcal{K}$ by reverting each discrete vector of $\mathcal{G}$. In particular, each discrete vector $\{\sigma_{d-1}, \sigma_d\}$ between a $(d-1)$-simplex σ_{d-1} of $\mathcal{K}$ and one its cofacets can be reverted into $\{\sigma_d^*, \sigma_{d-1}^*\}$, where σ_d^* and σ_{d-1}^* are the simplices dual to σ_d and σ_{d-1} in $\mathcal{K}^*$. Then $\{\sigma_d^*, \sigma_{d-1}^*\}$ is a discrete vector between a 0-simplex (σ_d^*) and a 1-simplex (σ_{d-1}^*). Once this is established, the algorithm described in Sec. 5.1 can be applied as-is on $\mathcal{G}^*$. Additionally, one can observe that the critical 1-simplices of $\mathcal{G}^*$ will be, by construction, critical $(d-1)$-simplices of $\mathcal{G}$ and that their *unstable* sets in $\mathcal{G}^*$ will exactly coincide to *stable* sets in $\mathcal{G}$.

Thus, our algorithm for computing $\mathcal{D}_0(f)$ (Sec. 5.1) can be easily adapted to compute $\mathcal{D}_{d-1}(f)$ as follows. The stable sets of each critical $(d-1)$-simplex are first constructed. Next, each discrete vector in these stable sets is collapsed, to create a graph $G_{\mathcal{D}_{d-1}(f)}$, where each node represents a critical d-simplex and each arc a critical $(d-1)$-simplex. Finally, $G_{\mathcal{D}_{d-1}(f)}$ is processed with a Union-Find data structure (Sec. 5.1), but in decreasing lexicographic order, and a persistence pair (σ_{d-1}, σ_d) is created in $\mathcal{D}_{d-1}(f)$ for each connected component of $G_{\mathcal{D}_{d-1}(f)}$ created in σ_d and merged into another by the addition of the arc representing σ_{d-1}.

Domains with boundary When $\mathcal{K}$ is not closed, a slight variation of the above algorithm is considered. For domains with boundary, in specific configurations, the connected components of the backward lexicographic filtration of $\mathcal{K}^*$ may no longer exactly coincide with the voids of the forward lexicographic filtration of $\mathcal{K}$. In particular, when a connected component of the backward filtration of $\mathcal{K}^*$ first hits the outer boundary component of $\mathcal{K}$ (by construction, on a critical $(d-1)$-simplex), it no longer describes a void *inside* the object, as it merges with the rest of the outside space (thus deleting the corresponding cavity). To take this into account, we assign a *virtual* discrete maximum with infinite function value to the outer boundary component of $\mathcal{K}$ (representing the outside space) and apply the rest of the above algorithm as-is. Then, when a connected component of backward filtration hits the outer boundary, it is considered, given the above adjustment, to die there as it merged with an (infinitely) older component (the outside space).

Note that this specific adjustment comes with no additional computational overhead as the rest of our algorithm is used as-is (only one, extra virtual maximum is considered by the algorithm). In our implementation, this adjustment is optional as its practical relevance can be questionable for real-life data, as illustrated in Appendix 1.

6 Critical Simplices of Infinite Persistence

To summarize, our overall approach first computes $\mathcal{D}_0(f)$ (Sec. 5.1) and $\mathcal{D}_{d-1}(f)$ (Sec. 5.2). Finally, if $d = 3$, $\mathcal{D}_1(f)$ is computed with our novel algorithm *"PairCriticalSimplices"* (Sec. 4). During this process, certain critical simplices may remain unpaired after the above algorithms have finished. These correspond to homology classes with infinite persistence, which exactly characterize the homology of $\mathcal{K}$. Specifically, each remaining unpaired i-simplex σ_i yields a persistence class with infinite persistence in $\mathcal{D}_i(f)$, which we embed, by convention at location $(f(\sigma_i), f^*)$, where f^* denotes the maximum f value. Such points in the diagrams are marked with a specific flag (Fig. 5), as they describe more the domain $\mathcal{K}$ itself than the data f defined on it.

7 Computational Aspects

This section details the computational aspects of our algorithm, including time complexity and parallelism.

7.1 Time complexity

The first stage of our approach consists in establishing the lexicographic filtration with a global sort in $\mathcal{O}(nlog(n))$ steps (where n is the total number of simplices in $\mathcal{K}$).

The second stage computes a discrete gradient by homotopic expansion [79]. This operation takes $\mathcal{O}(n_v)$ where n_v is the number of vertices in $\mathcal{K}$.

The third stage consists in computing $\mathcal{D}_0(f)$ (Sec. 5.1). The first step of this algorithm computes the unstable sets of each 1-saddle to construct the graph $G_{\mathcal{D}_0(f)}$, which is done in $\mathcal{O}(n_e)$ steps in practice, where n_e is the number of edges

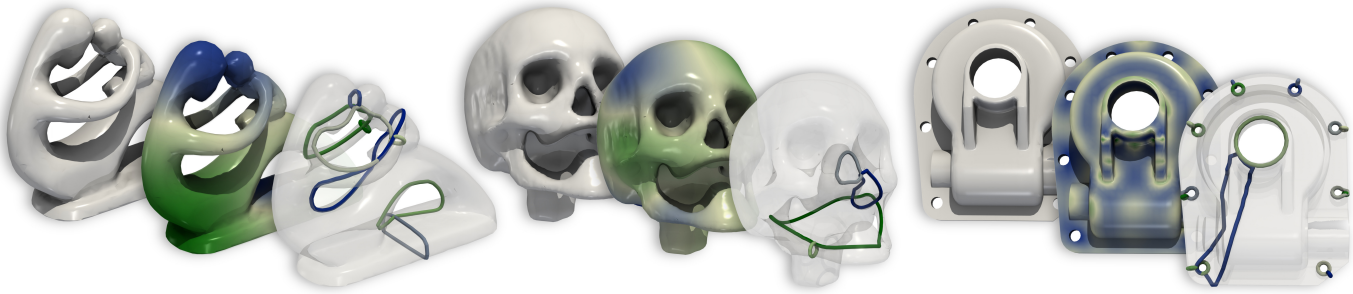

Fig. 13. Extracting surface generators with eigenfunctions of the Laplace-Beltrami operator (center). The infinitely persistent 1-cycles of this very smooth scalar field (curves, right) smoothly capture each topological handle, facilitating further surface post-processing (e.g. parametrization).

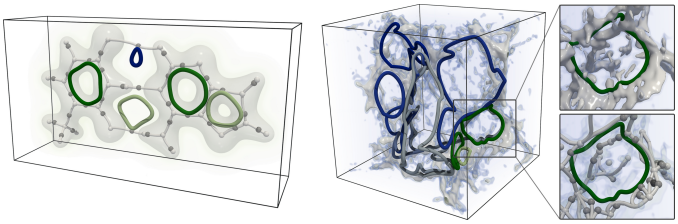

Fig. 14. Persistent 1-cycles (colored by persistence) in volume data capture prominent loops in the sub-level sets. In a quantum chemistry example (electron density, left), the 3 most persistent 1-cycles exactly coincide with the 3 carbon rings of the molecular system, while the 2 least persistent cycles coincide with weaker, non-covalent interactions. In an astrophysics example (dark matter density, right), persistent 1-cycles capture prominent loops in the *cosmic web* [87], indicating a very dense, circular pattern of galaxies, arranged around a prominent void (inset zoom). In both cases, 1-dimensional separatrices of the Morse-Smale complex are shown in the background for visual context.

in $\mathcal{K}$. Processing $G_{\mathcal{D}_0(f)}$ with a Union-Find data-structure to finally construct $\mathcal{D}_0(f)$ takes $\mathcal{O}(n_e\alpha(n_e))$ steps, where $\alpha()$ is the extremely slowly-growing inverse of the Ackermann function. Computing $\mathcal{D}_{d-1}(f)$ requires the same steps.

The fourth stage of our approach, only for $d = 3$, applies our algorithm *"PairCriticalSimplices"* (Alg. 3). Similarly to the seminal algorithm *"PairCells"* [102], our algorithm requires $\mathcal{O}(n^3)$ steps in the worst case. For each critical simplex (in the worst case, n steps, line 1), an homologous expansion is performed (in the worst case, in n steps, lines 5 to 15), which itself requires at each step a possibly linear pass to expand the boundary of the current critical simplex with modulo-2 additions (line 12). However, as documented in Sec. 9.2, our algorithm *"PairCriticalSimplices"* performs in practice significantly faster than the algorithm *"PairCells"* since: *(i)* it only considers the critical simplices (and not all the simplices of $\mathcal{K}$), *(ii)* the critical simplices already present in $\mathcal{D}_0(f)$ and $\mathcal{D}_{d-1}(f)$ are discarded from the computation (which provides further accelerations), *(iii)* it maintains the expanded boundary of the considered critical simplex and not its expanded chain (which is significantly bigger).

Overall, our approach has the advantage of being output-sensitive. In particular, the size (number of nodes) of the graphs $G_{\mathcal{D}_0(f)}$ and $G_{\mathcal{D}_{d-1}(f)}$ corresponds to the number of minima and maxima of f and consequently to the size of $\mathcal{D}_0(f)$ and $\mathcal{D}_{d-1}(f)$. The time complexity of our algorithm *"PairCriticalSimplices"* is parameterized by the number of remaining saddle-saddle pairs, which corresponds to the size of $\mathcal{D}_1(f)$. Then our approach will provide superior performances when considering smooth data sets, as typically found in various simulation domains.

7.2 Shared memory parallelism

Our approach can benefit from further accelerations thanks to shared-memory parallelism. The first stage (establishing the lexicographic filtration) can be done with parallel sorting (see the GNU parallel sort for an implementation example). The second stage (discrete gradient computation [79]) is trivially parallelizable on the vertices of $\mathcal{K}$. Regarding the third stage, computing $\mathcal{D}_0(f)$, the computation of the unstable sets is parallelized on a per 1-saddle basis and the processing of $G_{\mathcal{D}_0(f)}$ with the Union-Find data-structure is then done sequentially. In practice $\mathcal{D}_0(f)$ and $\mathcal{D}_{d-1}(f)$ are computed in parallel thanks to a task pool mechanism. Regarding the fourth stage (Alg. 3), the first iterations of the homologous expansions can be done independently for each critical 2-simplex, as long as no unpaired 1-simplex is visited (line 7). Thus, these first iterations are run in parallel, on a per critical 2-simplex basis. Once all homologous expansions have reached their first unpaired 1-simplex, the rest of Alg. 3 is run sequentially.

8 Application to Generator Extraction

This section presents an application of our contributions to the fast extraction of persistent 1-dimensional generators. While the topological persistence computed by our algorithm is a central simplification criterion in data visualization (Fig. 1), the information maintained by our algorithm can additionally be exploited directly for visualization purposes. Specifically, Iurichich [54] suggested to extract, for a given persistence pair (σ_i, σ_j), a representative d_i-cycle homologous to $\partial\sigma_j$, specifically, the earliest homologous d_i-cycle, created at σ_i. For that, Iurichich introduced a specific post-processing algorithm [54], requiring the persistence diagram to be computed in a pre-processing step.

In contrast, in our work, this information is precisely maintained throughout the entire computation, for all 1-dimensional persistence pairs, and is then readily available when our persistence diagram computation algorithm has finished, resulting in further accelerations. Specifically,

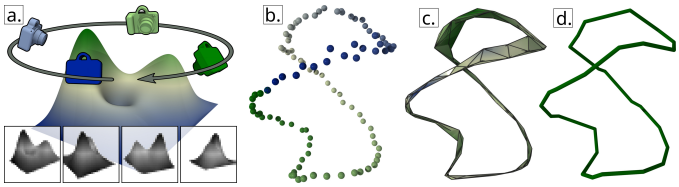

Fig. 15. Detecting circular patterns in high dimensional data. *(a)* 100 greyscale pictures of resolution 32×32 (bottom) of a synthetic terrain (center) are taken from 100 viewpoints (colored cameras) arranged along a circle (camera color: arc-length parameterization along the circle). *(b)* This set of images can be interpreted as 100 points in $\mathbb{R}^{32\times32}$, which can be projected down to 3D via Multi-Dimensional Scaling (MDS) [59] (color: camera arc-length parameterization). *(c)* The 2-dimensional Rips complex can be computed from the point cloud in the high-dimensional space ($\mathbb{R}^{32\times32}$) to infer the structure of the space sampled by the point cloud, by adding a triangle in the complex if its *diameter* (the maximum pairwise distance between its vertices) is smaller than a threshold ϵ. The Rips complex is shown in 3D (via MDS projection), although it is computed in $\mathbb{R}^{32\times32}$. *(d)* The infinitely persistent 1-cycle of the *diameter* function (for each vertex, average of the diameter of its adjacent triangles) robustly captures the circular pattern synthetically injected in the data (*(a)*), hence confirming the ability of persistent 1-cycles to recover circular patterns in high-dimensional data.

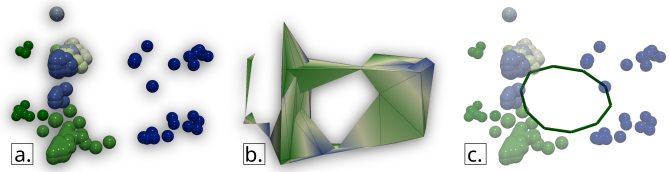

Fig. 16. Seven measurements of electric power consumption (including active power, voltage, intensity) for a single household over two years (daily sampling) [49]. This dataset can be interpreted as 700 points (one per day) in $\mathbb{R}^7$ (projected in 3D in *(a)* via MDS [59]). *(a)* The k-means clustering ($k = 8$) applied to this point cloud identifies dense clusters (point colors), corresponding to distinct consumption modes. *(b)* The 2D Rips complex (computed in $\mathbb{R}^7$, but projected in 3D via MDS [59]) exhibits a prominent topological handle. *(c)* The *diameter* function (see Fig. 15) yields a highly persistent 1-cycle (green curve) between four clusters (light blue, white, dark blue, light green). This indicates that these four clusters are organized along a circular pattern in $\mathbb{R}^7$, implying in practice that continuous displacements from the white consumption state to the light green one, are likely to imply a transition along the cycle, either through the light blue or dark blue consumption states.

for each critical 2-simplex σ_j, the homologous expansion of Alg. 3 (lines 5 to 15) iteratively reconstructs with $Boundary(\sigma_j)$ a sequence of 1-cycles homologous to $\partial\sigma_j$ and any of these can be chosen as a representative generator. Specifically, we store the earliest cycle, precisely obtained at the end of the homologous expansion (line 16), when the first unpaired simplex τ is visited. Note that the problem of extracting persistent 0 and $(d-1)$-dimensional generators is significantly simpler and has already been addressed via merge tree based segmentations [9], [16], [36].

Figs. 13, 14, 15 and 16 illustrate the ability of persistent 1-cycles to robustly capture circular patterns on surfaces, volume data and high-dimensional point clouds respectively.

9 Results

This section presents experimental results obtained with a C++/OpenMP implementation of our approach, integrated in TTK (commit: e14377b). Our experiments were mostly run on a commodity desktop computer with two Xeon CPUs (3.0 GHz, 2x4 cores, 64 GB of RAM), while specific scalability experiments were run on a large shared-memory system with 128 Xeon CPUs (2.6 GHz, 128x8 cores, 16TB of RAM).

9.1 Experimental data

We consider a list of 34 scalar datasets available on a public repository [57], provided as three-dimensional regular grids of various resolutions and data types. These datasets come from diverse application fields (bio-imaging, material sciences, combustion, quantum chemistry, fluid dynamics) and have been either acquired (e.g. CT scans) or simulated. We additionally consider two extreme cases: *(i)* an elevation function (yielding the smallest possible output, a single bar in $\mathcal{D}_0(f)$, with infinite persistence), *(ii)* a random function (yielding the largest outputs in practice). Since our approach is output sensitive, we re-sampled all datasets to a common resolution (192^3), to better observe runtime variations solely based on the output size. This common resolution has been chosen such that most of the implementations considered in our benchmark Sec. 9.3 could run on our experimental setup. Some of the available datasets [57] were too large to fit in the memory of our desktop computer and could not be downsampled to the common resolution. These have not been considered in the benchmark as we believe our desktop computer to be representative of the machines used by potential benchmark users.

We generated 2D datasets by taking a slice of each original 3D regular grid along the Z-coordinate (at mid-value). These 2D datasets were re-sampled to $4,096^2$. Finally, we generated 1D datasets by considering a line of each 2D dataset (Y-coordinate, mid-value). These 1D datasets were re-sampled to a common resolution of $1,048,576$ vertices.

Each of these 1D, 2D and 3D datasets were then triangulated into a simplicial complex by breaking up each cell into two triangles in 2D, and five tetrahedra in 3D. As discussed in Sec. 1, our approach focuses on this generic input representation based on simplicial complexes and we will therefore consider these representations for our experimentations. This results overall in 108 input datasets.

As described in Sec. 9.3, some of the public implementations considered in our benchmark are specialized (or include specialized backends) for regular grids. However, they do not all interpret the input data in a consistent manner. For instance, some implementations (such as Gudhi) consider the input scalars to be defined on a per voxel basis, while others (such as Dipha or CubicalRipser) consider them as defined on a per vertex basis, which results in cell complexes of significantly different sizes (in particular, penalizing Gudhi). Moreover, some implementations (such as Oineus, PairCells, PersistenceCycles, TTK-FTM, DMS) implicitly triangulate the input regular grid data [48], [53], which also changes the size of the input complex. First, since these internal data representations differ, the generated outputs will, consequently, not be exactly identical. Second, since these differences in internal representation result in cell complexes of significantly different sizes, they also induce a strong bias in runtime comparison. For these

TABLE 1
Public implementations considered in our benchmark (Sec. 9.3).

Implementation	Ref.	Version	Category	Language	Simplicial Support	Grid Support	Parallelism	Distance
PairCells	[102]	Github 362e69c	Explicit Homologous Expansion	C++	Native	Implicit	No	**0.0**
CubicalRisper	[55]	Github a063dac	Boundary Matrix Reduction	C++	No	Native	No	NA
Dionysus2	[68]	Pypi v2.0.8	Boundary Matrix Reduction	C++	Native	No	No	**0.0**
DIPHA	[4]	Github 0b87476	Boundary Matrix Reduction	C++	Native	Native	Controllable	**0.0**
Eirene.jl	[52]	Julia 1.3.6	Boundary Matrix Reduction	Julia	Native	No	No	9.0×10^3
Gudhi	[65]	Github 845b02ff	Boundary Matrix Reduction	C++	Native	Native	Observed	15.3×10^3
Javaplex	[90]	Github v4.3.4	Boundary Matrix Reduction	Java	Native	No	Observed	**0.0**
Oineus (Python API)	[72]	Github f2dd92e	Boundary Matrix Reduction	C++	No	Implicit	Controllable	NA
PHAT (Spectral Seq.)	[5]	Bitbucket 264f0a7	Boundary Matrix Reduction	C++	Native	No	Controllable	466.6×10^3
Ripser.py	[3], [94]	Pypi v0.6.0	Boundary Matrix Reduction	C++	Native	No	No	NA
Diamorse	[25]	Github 3416d7a	Discrete Morse Theory	C++	No	Native	No	NA
Perseus	[71]	Author WebPage	Discrete Morse Theory	C++	Native	Native	No	NA
PersistenceCycles	[54]	Github b68ae3e	Discrete Morse Theory	C++	Native	Implicit	Controllable	97.5×10^3
TTK-FTM	[37]	v0.9.9	Merge-Tree (2D)	C++	Native	Implicit	Controllable	122.5×10^6

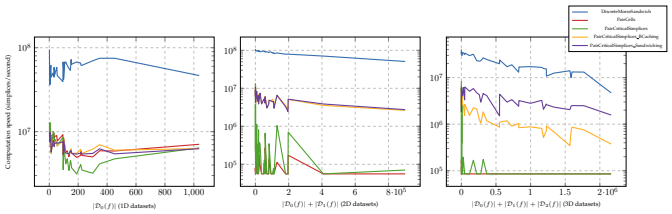

Fig. 17. Computation speeds (number of simplices per second, log scale), as a function of the output size, for the distinct accelerations of our algorithm, in comparison to the seminal algorithm *"PairCells"* (red, Alg. 1). Each non-red curve corresponds to a specific variant of our algorithm: *"PairCriticalSimplices"* (green, Alg. 3), with boundary caching (yellow, Sec. 4.2), with *"Sandwiching"* (purple, Sec. 5.1), in parallel (blue, 8 cores, Sec. 7.2). On average, our parallel algorithm computes in 0.05 (1D), 1.52 (2D) and **8.34** (3D) seconds, and achieves a parallel efficiency of 20.63% (1D), 45.44% (2D), and **59.38%** (3D) for an overall speedup over *"PairCells"* of $\times 9$ in 1D, $\times 922$ in 2D, and $\times$**264** in 3D.

reasons, we decided to focus our analysis on the methods which natively support simplicial complexes, for which a direct and unbiased comparison can be performed. For completeness, we provide performance numbers for regular grids in Appendix 2, but we stress that the inconsistencies in the internal representations and in the generated outputs prevent a direct and unbiased comparison.

9.2 Performance analysis

This section evaluates the time performance of our overall approach, named *"DiscreteMorseSandwich"* (*DMS* for short), and details the gains provided by each steps of our algorithm, in comparison to the original algorithm *"PairCells"*.

Fig. 17 provides time performance curves for the 1D, 2D and 3D versions of our 36 input datasets, where computation speeds (in simplices per second, log scale) are reported as a function of the output size, and where the seminal algorithm *"PairCells"* is compared to four variants of our approach, to evaluate the performance gain of each acceleration introduced in our algorithm.

This figure confirms the output-sensitive behavior of our overall approach (DMS, blue curves), as computation speeds decrease for increased output sizes. As expected by our time complexity analysis (Sec. 7.1), the lowest speeds occur for 3D datasets (2.34×10^7 simplices/sec on average) since there, the computation of $\mathcal{D}_1(f)$ (the intermediate layer of the sandwich) has a less favorable time complexity than for $\mathcal{D}_0(f)$ and $\mathcal{D}_2(f)$. This is confirmed by the increased speed in 2D (9.17×10^7 simplices/sec on average), while in 1D speed slightly decrease again as the unstable sets of 1-saddles now cover the entire domain (whereas they constitute only a small subset in 2D). In 1D (left), our overall approach (blue) is only about an order of magnitude faster than the seminal algorithm *"PairCells"*. In 2D (center), the simplest variant *"PairCriticalSimplices"* (green) starts to provide a significant acceleration (about one order of magnitude speedup) with regard to *"PairCells"* (red), while boundary caching provides another order of magnitude speedup. The effect of the sandwiching strategy becomes clearly visible in 3D (right). There, the *"PairCells"* and *"PairCriticalSimplices"* both timeout after 30 minutes of computation. In 3D, the benefit of the sandwiching approach is substantial (about a $\times 4$ speedup), as illustrated by the gap between the yellow and purple curves. For all dimensions, the parallelization of our overall approach (blue, 8 cores) provides about another order of magnitude speedup over the other variants. Overall, in 3D, our approach provides an average speedup of two orders of magnitude over *"PairCells"* ($\times 264$, when considering the runs where *"PairCells"* did not timeout after 30 minutes), with computations in less than 10 seconds, with about a 60% parallel efficiency, which can be considered as an efficient parallelization.

9.3 Performance benchmark

This section describes our benchmark for evaluating and comparing various public implementations for persistent diagram computation. Our Python benchmark package (https://github.com/pierre-guillou/pdiags_bench) *(i)* downloads and prepares the benchmark data (Sec. 9.1), *(ii)* downloads, builds and executes each implementation (Sec. 9.3.1) and *(iii)* aggregates the output information to produce the results provided in this section.

9.3.1 Implementations

Our benchmark includes the implementation of our algorithm *"DiscreteMorseSandwich"* (DMS) as well as 14 other implementations, whose specifications are reported in Tab. 1. A few clarifications are needed regarding certain implementations. In particular, *TTK-FTM* only computes $\mathcal{D}_0(f)$ and $\mathcal{D}_{d-1}(f)$. Ripser and its scikit-tda version both reported integer overflows for relatively large inputs (issue communicated to the authors). A number of implementations (among the category *"Boundary Matrix Reduction"*) require

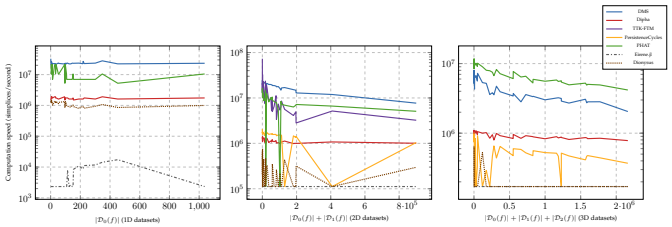

Fig. 18. Benchmark of sequential computation speeds (simplices/second, log scale) as a function of the output size.

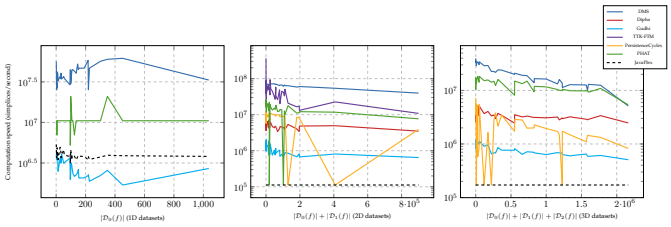

Fig. 19. Benchmark of parallel computation speeds (simplices/second, log scale, 8 cores) as a function of the output size.

an explicit boundary matrix as an input, which we compute in a pre-processing stage with TTK.

9.3.2 Output comparison

To check the correctness of our implementation, we compute the L_2-Wasserstein distance (Sec. 2.4) between our output and the one computed by each implementation included in the benchmark, for each dataset, for each output dimension. To enable the direct comparison of this distance across datasets, we consider as input scalar field f the vertex order, after sorting all input data values (i.e. $f(v) \in [0, n_v - 1]$). The average distance for all datasets, for all output dimensions, is reported in the column *"Distance"* of Tab. 1. These numbers show that our implementation generates outputs which are strictly identical to most other implementations (PairCells, Dionysus2, DIPHA, Javaplex, etc.). Variations from 0 seem to indicate slight inaccuracies for the corresponding implementation. For instance, for handling boundary effects, TTK-FTM only implements the second strategy described in Appendix 1 (i.e. it ignores the virtual maximum on the boundary), which impacts distance evaluations for $\mathcal{D}_{d-1}(f)$ (see Appendix 1). Note that this distance is only reported for the (non timed out) implementations natively supporting simplicial complexes, as outputs differ significantly in the case of regular grids (depending on the implementation's data interpretation, see Sec. 9.1).

9.3.3 Performance metrics

We evaluate performance along two major aspects: computation time and memory requirement.

Regarding computation time, we consider the timings reported by each implementation, from which we remove the input/output times (for reading the input from disk and writing the output to disk). We also do *not* include the pre-processing time dedicated to boundary matrix computation, for the implementations which require this input form. Thus, our timings only include the core computation phase and we report in the following computation speeds, expressed in number of simplices per second. To enable an acceptable overall runtime (for the entire benchmark), we decided to interrupt all computations after a pre-defined timeout threshold of 15 minutes for all experiments.

Memory usage is evaluated with Python's standard library resources and we report the maximum resident set size of each implementation (run in a dedicated subprocess).

9.3.4 Benchmark results

Fig. 18 first reports, for each input dimension, the computation speed in sequential mode for all the (non timed out) implementations supporting simplicial complexes natively, for which no parallelism was observed or for which the number of threads could be explicitly set to one. There, as can be expected, the only non-C++ implementation (*Eirene.jl*) provides the lowest speeds. Overall, the other C++ based implementations report computation speeds between 10^5 and 10^8 simplices per second. *PHAT* and our method *DMS* report the fastest sequential runtimes for all dimensions, with *DMS* improving over *PHAT* in 1D and 2D by 48% and 51% respectively, while *PHAT* provides the best average sequential times in 3D, with a gain of 41% over *DMS*. In Fig. 18, *PersistenceCycles* [54] is the method which is the most related to our approach conceptually. However, similar to other previous approaches based on DMT [39], [54], [67], [71], it computes the *full* Morse complex (prior to running a standard boundary matrix reduction on it). In contrast, our approach computes only the necessary *subparts* of the Morse complex: the (1D) unstable sets of 1-saddles for $\mathcal{D}_0(f)$, the (1D) stable sets of 2-saddles for $\mathcal{D}_2(f)$ – which are much smaller than their (2D) unstable sets, and the (2D) unstable sets for only a *small subset* of the 2-saddles, specifically, only those involved in $\mathcal{D}_1(f)$. This careful selection already provides a significant performance gain. Next, the construction of $\mathcal{D}_0(f)$ and $\mathcal{D}_2(f)$ uses a Union-Find data-structure, which is much more efficient than boundary matrix reduction. Overall, this results in runtime gains of 92% and 88% of *DMS* over *PersistentCycles* in 2D and 3D respectively.

Next, Fig. 19 reports, for each input dimension, the computation speed in parallel mode (using 8 cores) for all the (non timed out) implementations supporting simplicial complexes natively, for which parallelsim was observed (typically using all available cores, in certain phases of the algorithm, for instance sorting) or for which the number of threads could be controlled explicitly. In this figure, note that only the pre-processing sorting step of Gudhi benefits from parallelism, the core of its algorithm being sequential. Similarly to the sequential case, the only non-C++ implementation (*JavaPlex*) provides the lowest speeds, as can be expected. The other C++ based implementations all report computation speed increases when parallelism is activated. In comparison to *PersistenceCycles* specifically (the other method of Fig. 19 based on DMT), our method *DMS* improves runtimes by 87% in 2D and 90% in 3D. Overall, *DMS* reports the fastest parallel runtimes for all dimensions, improving runtimes by 73%, 51% and 35% over the fastest competing technique, in 1D, 2D and 3D respectively. These fastest runtimes can be explained by an improved parallel efficiency over competing techniques.

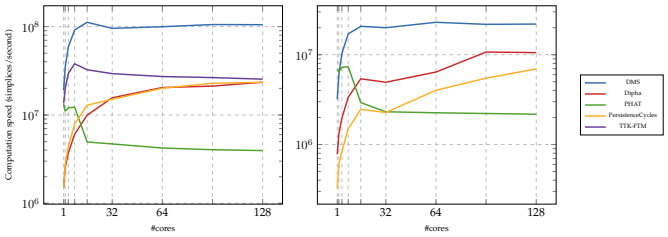

Fig. 20. Benchmark of parallel scalability (average speed, as a function of the number of used cores) in 2D (left) and 3D (right).

In particular, the discrete gradient computation is trivially parallel, while several other steps of our approach are also parallelized efficiently (Sec. 7.2).

We further investigate parallel scalability in Fig. 20, which reports for the most efficient parallel implementations, in 2D (left) and 3D (right), their average computation speeds (average over all datasets) when increasing the number of used cores. This figure indicates that no implementation scales significantly when increasing the number of cores and that most implementations reach a high plateau of speed (essentially due to the remaining sequential parts of the algorithms) beyond 32 cores in 2D and 96 cores in 3D. Moreover, *PHAT*, which presented encouraging parallel performances on a desktop computer, seems to suffer drastically from NUMA effects on our large system, resulting in an absence of parallel acceleration. Overall, *DMS* reports the fastest performances on this system, improving runtimes with 128 cores by 76% and 52% over the fastest competing technique, in 2D and 3D respectively.

Together, Figures 19 and 20 also indicate that *DMS* is more versatile than other approaches, as it outperforms the most adapted implementation for each system (*PHAT* for the desktop computer, and *DIPHA* for the large system).

Finally, Tab. 2 reports the memory footprint for all the implementations supporting simplicial complexes natively, and for which the computation completed successfully. There, one can observe that the methods supporting parallelism have a very similar (if not identical) memory footprint when parallelism is activated. Overall, the methods taking a boundary matrix as an input tend to have the largest memory footprints. In contrast, *DMS* uses *TTK*'s internal triangulation data-structure [92] for modeling the input simplicial complex, which can be interpreted as a sparse representation of the boundary matrix, resulting in substantial improvements over the most competitive techniques, by 25%, 5% and 15% in 1D, 2D and 3D respectively.

9.4 Limitations

While most of its steps are parallelized (Sec. 7.2), the final stage of our approach (homologous expansion, Alg. 3) is mostly sequential, which impairs parallel scalability (Fig. 20). We partially addressed this issue by parallelizing the first iterations of the homologous expansion (Sec. 7.2), which resulted in improved performance. Note that we have tried to also parallelize the subsequent iterations of homologous expansion (one thread per expansion), but the necessary synchronizations, upon the processing of an unpaired simplex τ, resulted in strong performance degradation.

TABLE 2
Maximum memory footprint over a single run for each implementation, in mega-bytes (average over all datasets, bold: smallest footprint).

Implementation	Seq. 1D	Seq. 2D	Seq. 3D	Para. (8c) 1D	Para. (8c) 2D	Para. (8c) 3D
Dionysus2	417.9	19,626.2	31,418.0	NA	NA	NA
DIPHA	271.1	11,835.0	19,672.1	NA	12,380.6	20,597.5
Eirene.jl	43,885.9	NA	NA	NA	NA	NA
Gudhi	NA	NA	NA	246.1	10,644.8	16,770.3
JavaPlex	NA	NA	NA	1,676.7	NA	NA
PHAT	251.0	10,399.1	15,326.8	250.0	10,396.9	15,472.1
PersistenceCycles	NA	13,153.7	32,505.8	NA	13,154.4	32,878.9
TTK-FTM	NA	5,692.9	NA	NA	5,701.8	NA
DiscreteMorseSandwich	**188.3**	**5,388.9**	**12,818.7**	**188.2**	**5,390.7**	**13,142.5**

Similarly to the orignal algorithm *"PairCells"* [102], our variant *"PairCriticalSimplices"* can work in principle in arbitrary dimension. However, Robin's homotopic expansion [79] provides strong guarantees – regarding the correspondence between critical simplices and PL critical points – for input datasets in up to three dimensions. Beyond, such a correspondence is no longer guaranteed and our *zero-persistence skip* procedure (Alg. 2) may no longer be valid.

10 Conclusion

This paper introduced an efficient algorithm for the computation of persistence diagrams for scalar data. Specifically, we introduced a stratification strategy, which *(i)* computes the *easiest* diagrams first ($\mathcal{D}_0(f)$ and $\mathcal{D}_{d-1}(f)$) with an efficient Union-Find based processing applied to a carefully selected subset of the stable and unstable sets of saddles and which then *(ii)* efficiently computes the remaining diagram ($\mathcal{D}_1(f)$) by revisiting the seminal algorithm *"PairCells"* [102] in the context of discrete Morse theory. Extensive experiments on 36 public datasets validated the performance improvements of our approach over the *"PairCells"* algorithms, with two orders of magnitude speedups in 3D. A comprehensive benchmark including 14 public implementations for persistent homology computation indicated that our approach provides the lowest memory footprints, as well as the fastest parallel performances. Additionally, our experiments illustrated the versatility of our approach, as it outperforms (in 1D, 2D and 3D) the competing methods the most adapted to each tested system (*PHAT* for the desktop computer and *DIPHA* for the large system), providing users with performance confidence irrespective of their system.

In the future, we will investigate alternative strategies for discrete gradient computation, in order to extend our *zero-persistence skip* procedure to arbitrary dimensions. Moreover, we will explore strategies for distributed computation, to further improve parallel scalability on large systems.

Acknowledgments

This work is partially supported by the European Commission grant ERC-2019-COG *"TORI"* (ref. 863464, https://erc-tori.github.io/). The authors thank Joshua Levine for his thoughtful proofreading.

References

[1] T. F. Banchoff. Critical points and curvature for embedded polyhedral surfaces. *The American Mathematical Monthly*, 1970.
[2] S. Barannikov. Framed Morse complexes and its invariants. *Adv. Soviet Math.*, 1994.

[3] U. Bauer. Ripser: efficient computation of Vietoris-Rips persistence barcodes. https://github.com/Ripser/ripser/, 2019.

[4] U. Bauer, M. Kerber, and J. Reininghaus. Distributed computation of persistent homology. In *Alg. Eng. & Exp.*, 2014.

[5] U. Bauer, M. Kerber, J. Reininghaus, and H. Wagner. Phat - persistent homology algorithms toolbox. *J. Symb. Comput.*, 2017.

[6] H. Bhatia, A. G. Gyulassy, V. Lordi, J. E. Pask, V. Pascucci, and P.-T. Bremer. TopoMS: Comprehensive Topological Exploration for Molecular and Condensed-Matter Systems. *J. C. C.*, 2018.

[7] S. Biasotti, D. Giorgio, M. Spagnuolo, and B. Falcidieno. Reeb graphs for shape analysis and applications. *TCS*, 2008.

[8] T. Bin Masood, J. Budin, M. Falk, G. Favelier, C. Garth, C. Gueunet, P. Guillou, L. Hofmann, P. Hristov, A. Kamakshidasan, C. Kappe, P. Klacansky, P. Laurin, J. Levine, J. Lukasczyk, D. Sakurai, M. Soler, P. Steneteg, J. Tierny, W. Usher, J. Vidal, and M. Wozniak. An Overview of TTK. In *TopoInVis*, 2019.

[9] A. Bock, H. Doraiswamy, A. Summers, and C. T. Silva. TopoAngler: Interactive Topology-Based Extraction of Fishes. *IEEE TVCG*, 2018.

[10] J. Boissonnat, T. K. Dey, and C. Maria. The compressed annotation matrix: An efficient data structure for computing persistent cohomology. *Algorithmica*, 2015.

[11] J. Boissonnat and C. Maria. The simplex tree: An efficient data structure for general simplicial complexes. *Algorithmica*, 2014.

[12] J. Boissonnat and S. Pritam. Edge collapse and persistence of flag complexes. In *SoCG*, 2020.

[13] P. Bremer, H. Edelsbrunner, B. Hamann, and V. Pascucci. A topological hierarchy for functions on triangulated surfaces. *IEEE TVCG*, 2004.

[14] P. Bremer, G. Weber, J. Tierny, V. Pascucci, M. Day, and J. Bell. Interactive exploration and analysis of large scale simulations using topology-based data segmentation. *IEEE TVCG*, 2011.

[15] H. Carr, J. Snoeyink, and U. Axen. Computing contour trees in all dimensions. In *Symp. on Dis. Alg.*, 2000.

[16] H. A. Carr, J. Snoeyink, and M. van de Panne. Simplifying Flexible Isosurfaces Using Local Geometric Measures. In *IEEE VIS*, 2004.

[17] H. A. Carr, G. H. Weber, C. M. Sewell, and J. P. Ahrens. Parallel peak pruning for scalable SMP contour tree computation. In *IEEE LDAV*, 2016.

[18] C. Chen and M. Kerber. Persistent homology computation with a twist. In *European Workshop on Computational Geometry*, 2011.

[19] F. Chen, H. Obermaier, H. Hagen, B. Hamann, J. Tierny, and V. Pascucci. Topology analysis of time-dependent multi-fluid data using the reeb graph. *Computer Aided Geometric Design*, 2013.

[20] Y. Chiang, T. Lenz, X. Lu, and G. Rote. Simple and optimal output-sensitive construction of contour trees using monotone paths. *Comput. Geom.*, 2005.

[21] T. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2009.

[22] L. De Floriani, U. Fugacci, F. Iuricich, and P. Magillo. Morse complexes for shape segmentation and homological analysis: discrete models and algorithms. *CGF*, 2015.

[23] V. de Silva, D. Morozov, and M. Vejdemo-Johansson. Dualities in persistent (co)homology. *Inverse Problems*, 2011.

[24] V. de Silva, D. Morozov, and M. Vejdemo-Johansson. Persistent cohomology and circular coordinates. *D. C. G.*, 2011.

[25] O. Delgado-Friedrichs, V. Robins, A. Sheppard, and P. Wood. Digital image analysis using discrete morse theory and persistent homology. https://github.com/AppliedMathematicsANU/diamorse/, 2020. Accessed: 2021-05-18.

[26] T. K. Dey, F. Fan, and Y. Wang. Computing topological persistence for simplicial maps. In *SoCG*, 2014.

[27] H. Doraiswamy, J. Tierny, P. J. S. Silva, L. G. Nonato, and C. T. Silva. Topomap: A 0-dimensional homology preserving projection of high-dimensional data. *IEEE TVCG*, 2020.

[28] H. Edelsbrunner and J. Harer. *Computational Topology: An Introduction*. American Mathematical Society, 2009.

[29] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. *DCG*, 2002.

[30] H. Edelsbrunner and E. P. Mucke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM ToG*, 1990.

[31] G. Favelier, N. Faraj, B. Summa, and J. Tierny. Persistence Atlas for Critical Point Variability in Ensembles. *IEEE TVCG*, 2018.

[32] R. Forman. A User's Guide to Discrete Morse Theory. *AM*, 1998.

[33] P. Frosini and C. Landi. Size theory as a topological tool for computer vision. *Pattern Recognition and Image Analysis*, 1999.

[34] A. Garin, T. Heiss, K. Maggs, B. Bleile, and V. Robins. Duality in persistent homology of images. In *SoCG*, 2020.

[35] D. Guenther, R. Alvarez-Boto, J. Contreras-Garcia, J.-P. Piquemal, and J. Tierny. Characterizing molecular interactions in chemical systems. *IEEE TVCG*, 2014.

[36] C. Gueunet, P. Fortin, J. Jomier, and J. Tierny. Contour forests: Fast multi-threaded augmented contour trees. In *IEEE LDAV*, 2016.

[37] C. Gueunet, P. Fortin, J. Jomier, and J. Tierny. Task-Based Augmented Contour Trees with Fibonacci Heaps. *IEEE TPDS*, 2019.

[38] C. Gueunet, P. Fortin, J. Jomier, and J. Tierny. Task-based Augmented Reeb Graphs with Dynamic ST-Trees. In *EGPGV*, 2019.

[39] D. Günther, J. Reininghaus, H. Wagner, and I. Hotz. Efficient computation of 3d morse-smale complexes and persistent homology using discrete morse theory. *Vis. Comput.*, 2012.

[40] A. Gyulassy. *Combinatorial construction of Morse-Smale complexes for data analysis and visualization*. PhD thesis, UC Davis, 2008.

[41] A. Gyulassy, P. Bremer, R. Grout, H. Kolla, J. Chen, and V. Pascucci. Stability of dissipation elements: A case study in combustion. *CGF*, 2014.

[42] A. Gyulassy, P. Bremer, and V. Pascucci. Shared-Memory Parallel Computation of Morse-Smale Complexes with Improved Accuracy. *IEEE TVCG*, 2018.

[43] A. Gyulassy, P. T. Bremer, B. Hamann, and V. Pascucci. A practical approach to Morse-Smale complex computation: Scalability and generality. *IEEE TVCG*, 2008.

[44] A. Gyulassy, D. Guenther, J. A. Levine, J. Tierny, and V. Pascucci. Conforming Morse-Smale complexes. *IEEE TVCG*, 2014.

[45] A. Gyulassy, V. Natarajan, M. Duchaineau, V. Pascucci, E. Bringa, A. Higginbotham, and B. Hamann. Topologically Clean Distance Fields. *IEEE TVCG*, 2007.

[46] A. Gyulassy, V. Natarajan, V. Pascucci, P. Bremer, and B. Hamann. A topological approach to simplification of three-dimensional scalar functions. *IEEE TVCG*, 2006.

[47] A. Gyulassy and V. Pascucci. Computing Simply-Connected Cells in Three-Dimensional Morse-Smale Complexes. In *Proc. of TopoInVis*, 2011.

[48] H. Freudenthal. Simplizialzerlegungen von beschrankter Flachheit. *Annals of Mathematics*, 43:580–582, 1942.

[49] G. Hebrail and A. Berard. Individual household electric power consumption – UCI Machine Learning Repository, 2012. https://archive-beta.ics.uci.edu/ml/datasets/individual+household+electric+power+consumption.

[50] C. Heine, H. Leitte, M. Hlawitschka, F. Iuricich, L. De Floriani, G. Scheuermann, H. Hagen, and C. Garth. A survey of topology-based methods in visualization. *CGF*, 2016.

[51] G. Henselman and R. Ghrist. Matroid Filtrations and Computational Persistent Homology. *ArXiv e-prints*, 2016.

[52] G. Henselman-Petrusek. Eirene.jl package for homological algebra. https://github.com/Eetion/Eirene.jl, 2018. Accessed: 2021-05-19.

[53] H.W. Kuhn. Some combinatorial lemmas in topology. *IBM Journal of Research and Development*, 45:518–524, 1960.

[54] F. Iuricich. Persistence cycles for visual exploration of persistent homology. *IEEE TVCG*, 2021. https://github.com/IuricichF/PersistenceCycles.

[55] S. Kaji, T. Sudo, and K. Ahara. Cubical ripser: Software for computing persistent homology of image and volume data. https://github.com/CubicalRipser/, 2020.

[56] J. Kasten, J. Reininghaus, I. Hotz, and H. Hege. Two-dimensional time-dependent vortex regions based on the acceleration magnitude. *IEEE TVCG*, 2011.

[57] P. Klacansky. Open Scientific Visualization Data Sets. https://klacansky.com/open-scivis-datasets/, 2020.

[58] M. Kontak, J. Vidal, and J. Tierny. Statistical parameter selection for clustering persistence diagrams. In *2019 IEEE/ACM UrgentHPC@SC*, 2019.

[59] J. Kruskal and M. Wish. Multidimensional scaling. *Sage University Publications*, 1978.

[60] D. E. Laney, P. Bremer, A. Mascarenhas, P. Miller, and V. Pascucci. Understanding the structure of the turbulent mixing layer in hydrodynamic instabilities. *IEEE TVCG*, 2006.

[61] J. Lukasczyk, G. Aldrich, M. Steptoe, G. Favelier, C. Gueunet, J. Tierny, R. Maciejewski, B. Hamann, and H. Leitte. Viscous

fingering: A topological visual analytic approach. In *PMVMSP*, 2017.
[62] J. Lukasczyk, C. Garth, R. Maciejewski, and J. Tierny. Localized topological simplification of scalar data. *IEEE TVCG*, 2020.
[63] J. Lukasczyk, C. Garth, G. H. Weber, T. Biedert, R. Maciejewski, and H. Leitte. Dynamic nested tracking graphs. *IEEE TVCG*, 2019.
[64] S. Maadasamy, H. Doraiswamy, and V. Natarajan. A hybrid parallel algorithm for computing and tracking level set topology. In *Proc. of HiPC*, 2012.
[65] C. Maria, J. Boissonnat, M. Glisse, and M. Yvinec. The gudhi library: Simplicial complexes and persistent homology. In *Mathematical Software*, 2014. https://github.com/GUDHI/.
[66] J. Milnor. *Morse Theory*. Princeton University Press, 1963.
[67] K. Mischaikow and V. Nanda. Morse theory for filtrations and efficient computation of persistent homology. *D. C. G.*, 2012.
[68] D. Morozov. Dionysus2. http://www.mrzv.org/software/dionysus2, 2017. Accessed: 2021-05-19.
[69] D. Morozov and A. Nigmetov. Brief announcement: Towards lockfree persistent homology. In *Symposium on Parallelism in Algorithms and Architectures*, 2020.
[70] M. Morse. The calculus of variations in the large. In *AMS*, 1934.
[71] V. Nanda. Perseus, the persistent homology software. http://people.maths.ox.ac.uk/nanda/perseus/, 2021. Accessed: 2021-05-18.
[72] A. Nigmetov and D. Morozov. Oineus. https://github.com/grey-narn/oineus, 2021. Accessed: 2021-05-19.
[73] P. Oesterling, C. Heine, H. Jänicke, G. Scheuermann, and G. Heyer. Visualization of high-dimensional point clouds using their density distribution's topology. *IEEE TVCG*, 2011.
[74] M. Olejniczak, A. S. P. Gomes, and J. Tierny. A Topological Data Analysis Perspective on Non-Covalent Interactions in Relativistic Calculations. *International Journal of Quantum Chemistry*, 2019.
[75] S. Parsa. A deterministic *o(m log m)* time algorithm for the reeb graph. In *SoCG*, 2012.
[76] V. Pascucci, G. Scorzelli, P. T. Bremer, and A. Mascarenhas. Robust on-line computation of Reeb graphs: simplicity and speed. *ACM ToG*, 2007.
[77] G. Reeb. Sur les points singuliers d'une forme de Pfaff complètement intégrable ou d'une fonction numérique. *Comptes Rendus des séances de l'Académie des sciences*, 222(847-849):76, 1946.
[78] V. Robins. Toward computing homology from finite approximations. *Topology Proceedings*, 1999.
[79] V. Robins, P. J. Wood, and A. P. Sheppard. Theory and Algorithms for Constructing Discrete Morse Complexes from Grayscale Digital Images. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2011.
[80] N. Saul and C. Tralie. Scikit-tda: Topological data analysis for python, 2019. https://doi.org/10.5281/zenodo.2533369. doi: 10.5281/zenodo.2533369
[81] N. Shivashankar and V. Natarajan. Parallel Computation of 3D Morse-Smale Complexes. *CGF*, 2012.
[82] N. Shivashankar, P. Pranav, V. Natarajan, R. van de Weygaert, E. P. Bos, and S. Rieder. Felix: A topology based framework for visual exploration of cosmic filaments. *IEEE TVCG*, 2016.
[83] D. Smirnov and D. Morozov. Triplet Merge Trees. In *TopoInVis*, 2017.
[84] M. Soler, M. Petitfrere, G. Darche, M. Plainchault, B. Conche, and J. Tierny. Ranking Viscous Finger Simulations to an Acquired Ground Truth with Topology-Aware Matchings. In *IEEE LDAV*, 2019.
[85] M. Soler, M. Plainchault, B. Conche, and J. Tierny. Lifted Wasserstein matcher for fast and robust topology tracking. In *IEEE LDAV*, 2018.
[86] M. Soler, M. Plainchault, B. Conche, and J. Tierny. Topologically controlled lossy compression. In *IEEE PV*, 2018.
[87] T. Sousbie. The persistent cosmic web and its filamentary structure: Theory and implementations. *R. A. S.*, 2011.
[88] A. Suh, M. Hajij, B. Wang, C. Scheidegger, and P. A. Rosen. Persistent homology guided force-directed graph layouts. *IEEE TVCG*, 2019.
[89] S. Tarasov and M. Vyali. Construction of contour trees in 3d in o(n log n) steps. In *SoCG*, 1998.
[90] A. Tausz, M. Vejdemo-Johansson, and H. Adams. JavaPlex: A research software package for persistent (co)homology. In *ICMS*, 2014. http://appliedtopology.github.io/javaplex/.
[91] G. Tauzin, U. Lupo, L. Tunstall, J. B. Pérez, M. Caorsi, A. M. Medina-Mardones, A. Dassatti, and K. Hess. giotto-tda: : A topological data analysis toolkit for machine learning and data exploration. *J. Mach. Learn. Res.*, 2021.
[92] J. Tierny, G. Favelier, J. A. Levine, C. Gueunet, and M. Michaux. The Topology ToolKit. *IEEE TVCG*, 2017. https://topology-tool-kit.github.io/.
[93] J. Tierny and V. Pascucci. Generalized topological simplification of scalar fields on surfaces. *IEEE TVCG*, 2012.
[94] C. Tralie, N. Saul, and R. Bar-On. Ripser.py: A lean persistent homology library for python. *The Journal of Open Source Software*, 3(29):925, Sep 2018. doi: 10.21105/joss.00925
[95] K. Turner, Y. Mileyko, S. Mukherjee, and J. Harer. Fréchet Means for Distributions of Persistence Diagrams. *DCG*, 2014.
[96] M. van Kreveld, R. van Oostrum, C. Bajaj, V. Pasucci, and D. Schikore. Contour trees and small seed sets for isosurface traversal. In *SoCG*, 1997.
[97] J. Vidal, J. Budin, and J. Tierny. Progressive Wasserstein Barycenters of Persistence Diagrams. *IEEE TVCG*, 2019.
[98] J. Vidal, P. Guillou, and J. Tierny. A Progressive Approach to Scalar Field Topology. *IEEE TVCG*, 2021.
[99] J. Vidal and J. Tierny. Fast approximation of persistence diagrams with guarantees. In *IEEE LDAV*, 2021.
[100] H. Wagner, C. Chen, and E. Vucini. Efficient computation of persistent homology for cubical data. In *Proc. of TopoInVis*, 2011.
[101] G. H. Weber, S. E. Dillard, H. A. Carr, V. Pascucci, and B. Hamann. Topology-controlled volume rendering. *IEEE TVCG*, 2007.
[102] A. J. Zomorodian. Topology for computing. In M. J. Atallah and M. Blanton, eds., *Algorithms and Theory of Computation Handbook (Second Edition)*, chap. 3, pp. 82–112. CRC Press, 2010.

Pierre Guillou is a research engineer at Sorbonne Université. After graduating from MINES ParisTech, a top French engineering school in 2013, he received his Ph.D., also from MINES ParisTech, in 2016. His Ph.D. work revolved around parallel image processing algorithms for embedded accelerators. Since 2019, he has been an active contributor to TTK and the author of many modules created for the VESTEC and TORI projects.

Jules Vidal is a post-doctoral researcher, currently at Sorbonne Université, from where he received the Ph.D. degree in 2021. He received the engineering degree in 2018 from ENSTA Paris. He is an active contributor to the Topology ToolKit (TTK), an open source library for topological data analysis. His notable contributions to TTK include the efficient and progressive approximation of distances, barycenters and clusterings of persistence diagrams.

Julien Tierny received the Ph.D. degree in Computer Science from the University of Lille in 2008 and the Habilitation degree (HDR) from Sorbonne University in 2016. He is currently a CNRS research director, affiliated with Sorbonne University. Prior to his CNRS tenure, he held a Fulbright fellowship (U.S. Department of State) and was a post-doctoral researcher at the Scientific Computing and Imaging Institute at the University of Utah. His research expertise lies in topological methods for data analysis and visualization. He is the founder and lead developer of the Topology ToolKit (TTK), an open source library for topological data analysis.

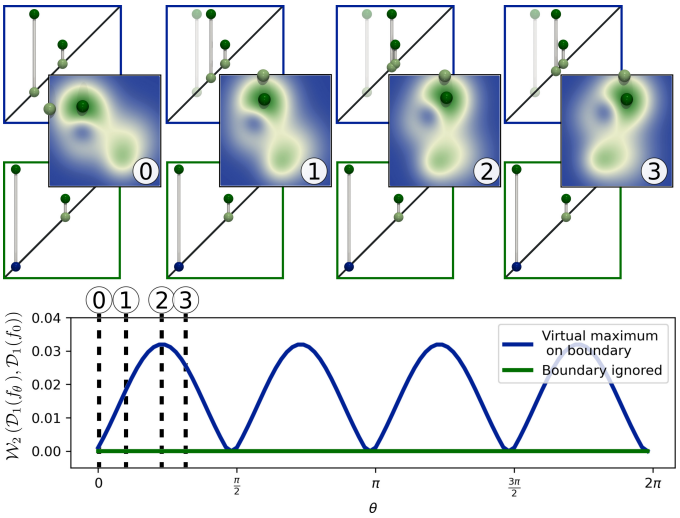

Fig. 21. Instability in $\mathcal{D}_{d-1}(f_\theta)$ induced by boundary perturbations.

APPENDIX

1 BOUNDARY PERTURBATIONS

As discussed in the section 5.2 of the main manuscript, the insertion of a virtual maximum on the outer boundary component of the data is optional, as it may result in an unstable assessment of the importance of the global maximum. Fig. 21 shows the toy example terrain from Fig. 2 (main manuscript) which is rotated in the plane and cropped back to a square, for a varying angle θ (left to right), hence simulating a typical boundary data-cutting artifact observed in real-life data. In the exact diagram $\mathcal{D}_{d-1}(f_\theta)$ (top, computed by the introduction of a virtual maximum on the outer boundary $\partial\mathcal{K}$ of $\mathcal{K}$, Sec. 5.2 of the main manuscript), the global maximum (dark green sphere in the data) is paired with a boundary saddle (light green sphere in the data), whose function value is dictated by the shape of the boundary (of the cut). As θ increases, the corresponding bar in $\mathcal{D}_{d-1}(f_\theta)$ oscillates horizontally (transparent: initial position for $\theta = 0$). Thus, the L_2-Wasserstein distance (blue curve, bottom, Sec. 2.4 of the main manuscript) to the original diagram $\mathcal{D}_{d-1}(f_0)$ also oscillates with θ. In contrast, by optionally disabling the introduction of a virtual maximum on $\partial\mathcal{K}$ (center), the global maximum always gets paired (by convention, Sec. 6 of the main manuscript) to the global minimum, inducing a zero L_2-Wasserstein distance throughout. From our experience, this latter strategy (center) provides in practice a more stable assessment of the importance of the global maximum.

2 REGULAR GRIDS

As discussed in the main manuscript, some of the public implementations considered in our benchmark are specialized (or include specialized backends) for regular grids. However, they do not all interpret the input data in a consistent manner. For instance, some implementations (such as Gudhi) consider the input scalars to be defined on a per voxel basis, while others (such as Dipha or CubicalRipser) consider them as defined on a per vertex basis, which results in cell complexes of significantly different sizes (in particular, penalizing Gudhi). Moreover, some implementations (such as Oineus, PairCells, PersistenceCycles, TTK-FTM, DMS) implicitly triangulate the input regular grid data [48], [53], which also changes the size of the input complex. First, since these internal data representations differ, the generated outputs will, consequently, not be exactly identical. Second, since these differences in internal representation result in cell complexes of significantly different sizes, they also induce a strong bias in runtime comparison. For these reasons, we decided to focus our analysis on the methods which natively support simplicial complexes, for which a direct and unbiased comparison can be performed. For completeness, we provide performance numbers for regular grids in Fig. 22, but we would like to stress that the inconsistencies in the internal representations and in the generated outputs prevent a direct and unbiased comparison.

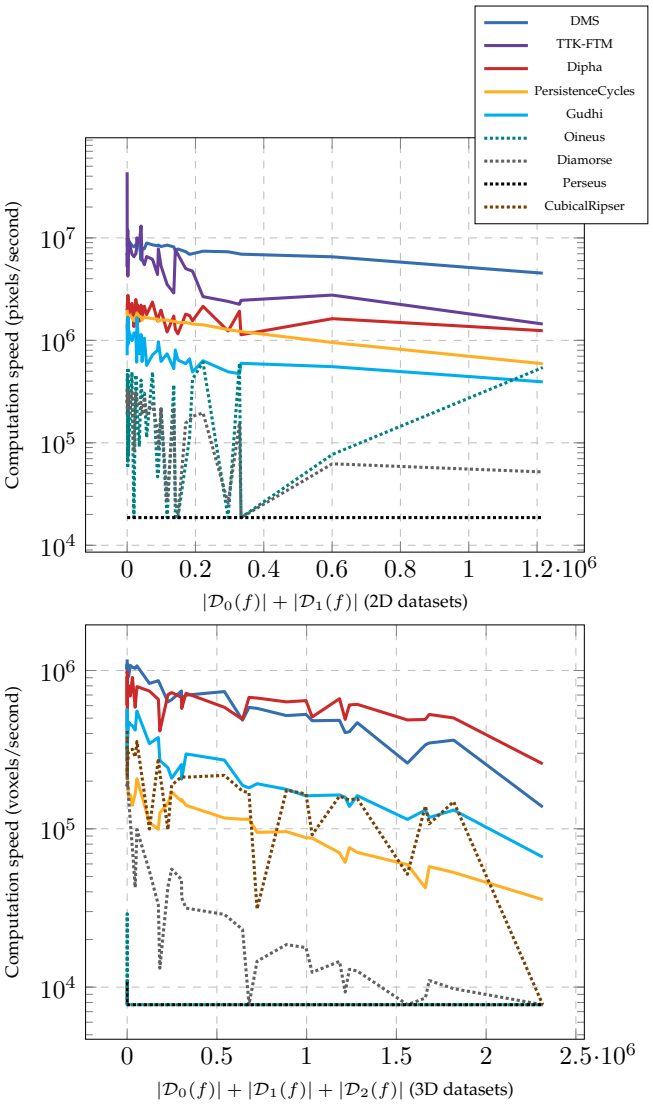

Fig. 22. Computation speeds (top: 2D data, in pixel/s – bottom: 3D data, in voxel/s, 8 cores) as a function of the output size.