A Progressive Approach to Scalar Field Topology

Jules Vidal, Pierre Guillou, Julien Tierny

A Progressive Approach to Scalar Field Topology

Jules Vidal, Pierre Guillou, and Julien Tierny

Abstract—This paper introduces progressive algorithms for the topological analysis of scalar data. Our approach is based on a hierarchical representation of the input data and the fast identification of *topologically invariant vertices*, which are vertices that have no impact on the topological description of the data and for which we show that no computation is required as they are introduced in the hierarchy. This enables the definition of efficient coarse-to-fine topological algorithms, which leverage fast update mechanisms for ordinary vertices and avoid computation for the topologically invariant ones. We demonstrate our approach with two examples of topological algorithms (critical point extraction and persistence diagram computation), which generate interpretable outputs upon interruption requests and which progressively refine them otherwise. Experiments on real-life datasets illustrate that our progressive strategy, in addition to the continuous visual feedback it provides, even improves run time performance with regard to non-progressive algorithms and we describe further accelerations with shared-memory parallelism. We illustrate the utility of our approach in batch-mode and interactive setups, where it respectively enables the control of the execution time of complete topological pipelines as well as previews of the topological features found in a dataset, with progressive updates delivered within interactive times.

Index Terms—Topological data analysis, scalar data, progressive visualization.

1 Introduction

The ever-increasing size and complexity of the datasets produced in engineering and sciences constitute a major challenge for their interpretation by human users. To address these issues, advanced data analysis tools are designed to efficiently capture the main features of interest in large datasets, in order to support interactive visualization and analysis tasks. The tools developed in Topological Data Analysis (TDA) precisely serve this purpose. They form a family of techniques which focus on the generic, robust, and efficient extraction of structural features in data [1]. Over the last years, many data analysis and visualization methods have been built around these concepts [2], with applications to a large spectrum of domains, including astrophysics [3], [4], biological imaging [5], [6], [7], chemistry [8], [9], [10], fluid dynamics [11], material sciences [12], [13], [14], or turbulent combustion [15], [16], [17]. In the case of scalar data, TDA algorithms rely on established topological data abstractions, such as contour trees [18], [19], [20], [21], [22], [23], Reeb graphs [24], [25], [26] or Morse-Smale complexes [27], [28], [29], [30]. In particular, the Persistence diagram [31] is a concise data representation, which visually summarizes the population of features of interest in a dataset, as a function of a measure of importance called *topological persistence*. Its conciseness, combined with its stability, made it increasingly popular in machine learning [32], [33], [34] and in interactive data analysis, where it quickly provides visual hints regarding the number and importance of the features in the data.

Although the core algorithms in TDA have practicable asymptotic complexities (usually from linear to quadratic time), the construction of the above topological abstractions can still require significant computation times for real-life datasets. Thus, when they are integrated into larger interactive systems, TDA algorithms can become a serious time bottleneck. This is a concern in data exploration scenarios, where users may need to wait between seconds and minutes to get a feedback when they adjust the parameters of the topological analysis.

• *J. Vidal, P. Guillou, J. Tierny are with Sorbonne Université and CNRS. E-mail: {jules.vidal, pierre.guillou, julien.tierny}@sorbonne-universite.fr*

Manuscript accepted Feb 17, 2021

In his seminal paper on response times of interactive systems, Miller [35] studied the impact of response time on the ability of users to maintain focus on a given task. For response times below a second (*continuity preserving latency*), the system appears fully responsive to user adjustments, allowing truly interactive sessions. For response times below a few seconds (*flow preserving latency*), users still manage to maintain their focus but the pauses in the exploration, due to the computation, challenge user interpretation skills. Above ten seconds (*attention preserving latency* threshold), users tend to disengage from the task to pursue other activities in parallel, which is highly detrimental to the interpretation process. Unfortunately, for real-life data, existing TDA algorithms correspond to the latter category of response times.

To address the discontinuities in user experience implied by excessive computation times, the notion of *progressive* data analysis has been explored by several authors in information visualization [36], [37], [38], [39]. In this context, a progressive algorithm is a technique capable of providing an interpretable output upon interruption requests, and of refining it otherwise, progressively converging towards the final solution. In an interactive setup, progressive algorithms can improve user experience in two ways. First, users can let such algorithms refine progressively their outputs, while receiving continuously visual feedbacks, and stop them when the outputs are deemed satisfactory. Second, users can also define a priori an upper limit on the computation time, after which the computation is interrupted. This enables to design interactive systems with guaranteed response time. This is particularly useful for algorithms whose actual execution times are difficult to anticipate, such as in TDA, where many popular algorithms, although with known time complexity, may have an output-sensitive computation time in practice. Progressive algorithms can also be beneficial to non-interactive setups, such as batch-mode computations on high performance computers, where the allocation of computing resources often needs to be finely controlled. In this context, progressivity enables the assignment of precise computation time budget to data analysis programs. This is relevant for time-critical applications such as numerical simulation in support of urgent decision making in catastrophe management

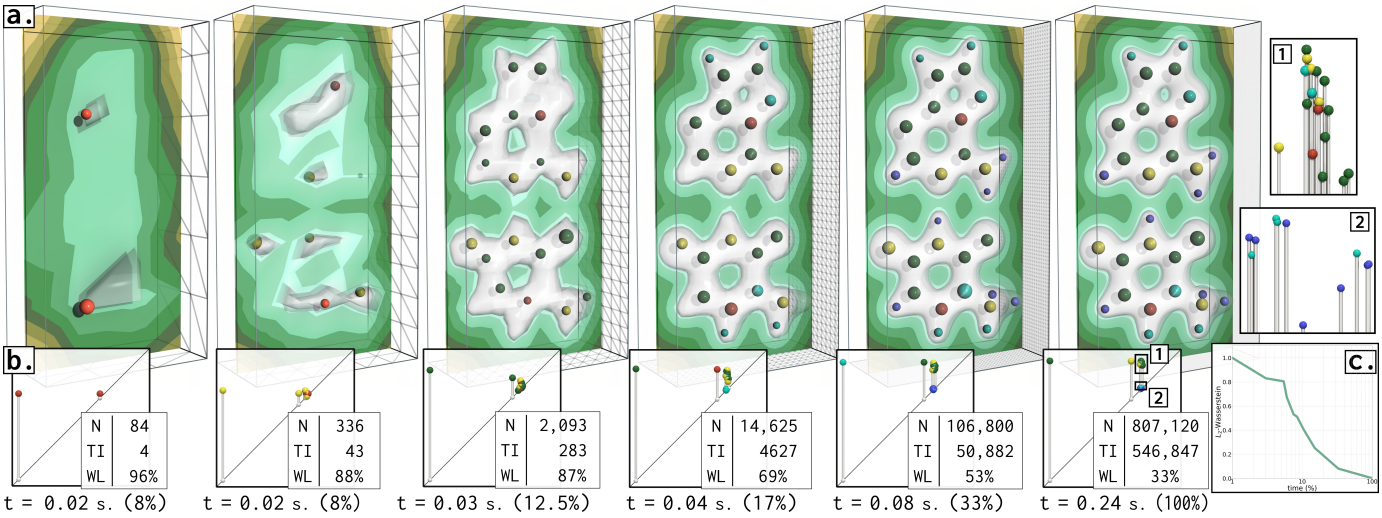

Fig. 1. Progressive persistence diagrams (saddle-maximum pairs) of the electron density of the adenine-thymine (AT) molecular system (an isosurface shows the two molecules), for a few steps of the progressive computation. Our coarse-to-fine approach efficiently refines the persistence diagram by progressing down a hierarchical representation of the input (from left to right). Maxima (denoting the atoms) are shown in the domain (a) with spheres, scaled by topological persistence and colored by lifetime in the data hierarchy (from red to dark blue). In this example, the persistence diagram (b) progressively captures the main features of the data. As of 8% of the computation time (leftmost), two persistent maxima are captured, denoting the presence of two molecules. As the computation progresses, atoms are progressively captured, heavier atoms first: oxygens, then nitrogens and carbons, and finally hydrogens are respectively all captured as of 12.5%, 17% and 33% of the computation time. At this point the diagram is complete and its accuracy is then improved until the final, exact result (rightmost). This qualitative progression is confirmed quantitatively by the empirical convergence of the L_2-Wassserstein distance to the final output (c), which is monotonically decreasing: more computation time indeed yields more accuracy. Our algorithms leverage efficient update mechanisms and *topologically invariant vertices* (*TI*), which can be quickly identified and for which we show that no computation is required, thus drastically reducing the workload (*WL*) of the algorithm with time. Overall, our progressive approach efficiently computes the persistence diagram of the data, while continuously providing relevant visual feedback.

(wildfires, floods, outbreaks, etc). This is also relevant to more general applications, towards the control of the power consumption of high performance computers, which becomes an increasingly important societal issue.

In this work, we introduce, to our knowledge, the first progressive algorithms for the topological analysis of scalar data. Our overall approach is based on the key idea that *critical points* (which correspond to topological events in data, Sec. 2.2) correspond to singular events, which usually have a reasonably low probability of appearance. In particular, most of the points of a dataset are regular points in practice, and do not imply topological events. For those, fast update mechanisms can be derived, allowing for efficient progressive algorithms. Our overall approach is based on a hierarchical representation of the data (Sec. 3), where *topologically invariant vertices* (vertices of the input which do not produce topological events upon insertion, and which, therefore, have no impact on the topological representation of the data, Sec. 3.3) can be identified and processed very efficiently. This enables the definition of coarse-to-fine topological algorithms, which provide an exact output for each level of the data hierarchy, and which quickly update their outputs between consecutive hierarchy levels. This strategy is motivated by two key practical observations: *(i)* the main features of interest of a dataset often appear early in the data hierarchy in practice and *(ii)* critical points represent a small portion of the input vertices, allowing for fast update mechanisms leveraging the inherent regularity of the data (when present). These two practical observations are quantitatively evaluated in our experiments (Sec. 6). We demonstrate our progressive strategy with two examples of topological analysis algorithms: critical point extraction (Sec. 4) and persistence diagram computation (for extremum-saddle pairs, Sec. 5). In both cases, the inherent regularity of the data is leveraged to derive efficient progressive algorithms, which continuously provide visual feedback, and which compute the final result even faster in practice than existing non-progressive methods. We also present simple parallelizations of our algorithms, to further improve performance. Experiments on real-life datasets validate the relevance of our progressive representations, both at a qualitative and quantitative level. We illustrate the practical utility of our approach, both for batch-mode and interactive setups, where it respectively enables the control of the run time of a TDA pipeline, as well as progressive previews of the topological features found in a dataset, continuously updated within interactive times.

1.1 Related Work

Many approaches based on topological methods have been documented over the last two decades. We refer the reader to the survey by Heine et al. [2] for a comprehensive overview. In the following, we focus on algorithms for constructing topological data abstractions, which are the most related to our work.

While Morse theory has originally been developed in the smooth setting [40], many of its concepts can be translated to discretized data, in particular in the form of piecewise-linear (PL) scalar fields defined on PL manifolds. Banchoff [41] introduced a formalism for the combinatorial characterization of the *critical points* (Sec. 2.2) of an input PL scalar field. These points correspond to locations where the sub-level sets of the function change their topology. They correspond to notable events in the data. In practice, extrema are often associated with features of interest. In presence of noise however, many critical points can be reported by this characterization, which motivates the introduction of an importance

measure on critical points, to distinguish noise artifacts from salient features.

Topological persistence [1], [31] has been established as a reference measure to assess the importance of critical points. It can be directly read from the *Persistence diagram* (Sec. 2.3) which plots topological features of the data according to their *birth* and *death*, both of which exactly coincide with critical points. Thus, the critical points of the input data are arranged in the diagram in pairs. The Persistence diagram can be computed generically by matrix reduction operations [1], [31]. The pairs of critical points in the diagram which involve extrema, often associated to features of interest in applications, can be computed more efficiently, with a Union-Find data structure [1], [42], or equivalently, they can be read directly from the merge tree (presented further down). For the special case of point cloud data, the topology of the underlying manifold sampled by the point cloud can be inferred [43] by considering the persistence diagram of the Vietoris-Rips filtration, for which tailored algorithms have been developped [44].

Although persistence diagrams are stable [45] (i.e. a small perturbation to the input data will only yield a small perturbation of its persistence diagram), their discriminative power may be insufficient for some applications. This motivates the design of more discriminative topological abstractions, such as merge and contour trees, which track the connected components of sub-level sets and level sets. Intuitively, these trees indicate how level sets connect and disconnect when passing critical points. The first algorithms for computing these tree structures focused on the 2D [18], [19] and 3D [20] cases. In their seminal paper, Carr et al. [21] introduced an efficient algorithm, with optimal time complexity, for computing the contour tree in all dimensions. Recently, several algorithms have been documented to compute this structure in parallel [22], [23], [46], [47], [48], [49], [50], [51]. If the input domain is not simply connected (intuitively, if it contains handles), the Reeb graph [24] needs to be considered instead of the contour tree to correctly track connected components of level sets, which involves more sophisticated methods (as the Reeb graph may now contain loops). The first Reeb graph computation algorithms were based on a slicing strategy [52], [53], [54], later improved by solely slicing along critical contours [55], [56]. Several techniques focused on practical performance [26], [57], while algorithms with optimal time complexity have been introduced, first in 2D [58], later in arbitrary dimension [59], and then parallelized [60]. Recently, efficient algorithms have been investigated for the computation of the generalization of the Reeb graph to multivariate functions, called the Reeb space [61], [62], [63].

The Morse-Smale complex is another typical topological abstraction for scalar data [27]. It decomposes the input domain into cells which have identical gradient integration extremities. Intuitively, it segments the data into regions, bounded by gradient flow separatrices, where the gradient shows a homogeneous behaviour. While the initial algorithms for its computation were developed in the PL setting [64], [65], modern alternatives [28], [29] are based on Discrete Morse theory [66] and parallel algorithms have been documented [30], [67].

To our knowledge, no algorithm has been described so far for the *progressive* computation of the above structures. In this work, we introduce the first progressive algorithms for the computation of topological abstractions, namely critical points (Sec. 4) and persistence diagrams (for extremum-saddle pairs, Sec. 5). Our approach is based on a hierarchical representation of the data. Multiresolution hierarchies have been considered before, for the Reeb graph [68], the contour tree [69] and the Morse-Smale complex [70], [71], [72], but the hierarchical aspect dealt with the *output* data structure, while the input was processed without multiresolution, with existing algorithms [21], [53], [64]. In contrast, in our work, the *input* data is represented as a multiresolution hierarchy and the output is efficiently, progressively updated in a coarse-to-fine manner, by iterating through the hierarchy levels.

Our progressive scheme relies on a hierarchical representation of the input data. In the visualization community, many types of hierarchies have been defined to encode and extract visual representations from volumetric data at different levels of details [73], [74], [75], [76], [77], [78]. For example, Gerstner and Pajarola [78] introduce a method for the robust extraction of isosurfaces in multiresolution volume representations. For this, their algorithm extracts the critical points of the input scalar field, for each level of their hierarchical scheme. However, they use for this the standard, non-progressive, procedure [41]. In contrast, our approach extracts the critical points for all of our hierarchy levels *progressively*, i.e. without recomputing from scratch critical points at each new hierarchy level, but instead by efficiently and minimally updating the information already computed at the previous levels. Generally, in our work, we focus on a specific scheme based on the so-called *red* subdivision [79], [80], [81], [82], [83] applied to regular grids [84], [85], in particular to investigate progressive and efficient *coarse-to-fine* computations, in contrast to the traditional fine-to-coarse hierachical approaches found in the visualization literature.

The approaches which are the most related to our work are probably the streaming algorithms for computing the Reeb graph [26] and the merge tree [16]. These algorithms are capable of computing their output in a streaming fashion: the simplices of the input domain can be processed in arbitrary order and these algorithms maintain and iteratively complete their output data structure. However, while they can be interrupted, these algorithms are not, strictly speaking, *progressive*: upon interruption, they do not provide interpretable but *partial* results, which are very far in practice from the final result. For instance, the streaming Reeb graph [26] can typically count at a given time a very large number of loops (which will be iteratively filled as the algorithm progresses). In contrast, our coarse-to-fine algorithms provide interpretable results upon interruption, which are visually similar to the exact, final outputs and which empirically quickly converge towards them.

1.2 Contributions

This paper makes the following new contributions:

1) *A progressive data representation (Sec. 3)* We present an approach for the progressive topological analysis of scalar data, to generate interpretable outputs upon interruption requests. Our approach relies on a hierarchical representation of the input data (derived from established triangulation subdivision schemes [79], [80], [81], [82], [83], [84], [85]) and the fast identification of the new notion of *topologically invariant vertices*, for which we show that no computation is required as they are introduced in the hierarchy.
2) *A progressive algorithm for critical point extraction (Sec. 4)* We introduce a progressive algorithm for critical point extraction. As it progresses down the data hierarchy, our algorithm leverages efficient update mechanisms for ordinary vertices and avoids computation for the topologically invariant ones. This enables a progressive output refinement, which results in even faster overall computations than non-progressive methods. We also

introduce a fast heuristic to evaluate the lifetime of critical points in the data hierarchy.

3) *A progressive algorithm for persistence diagram computation (Sec. 5)* We introduce a progressive algorithm for the computation of persistence diagrams of extremum-saddle pairs, built on top of the above contributions. In practice, our algorithm tends to capture the main features of the data first, and then progressively improves its accuracy. This is confirmed quantitatively by the empirical convergence of the Wasserstein distance to the final output, which is monotonically decreasing (more computation time indeed yields more accuracy). Our approach enables a continuous visual feedback, while being in practice even faster overall than non-progressive methods.
4) *A reference implementation* We provide a reference C++ implementation of our algorithms (publicly available at: https://github.com/julesvidal/progressive-scalar-topology) that can be used to replicate our results, and for future benchmarks.

2 Preliminaries

This section briefly presents the technical background of our work. We refer the reader to the textbook by Edelsbrunner and Harer [1] for a detailed introduction to computational topology.

2.1 Input Data

The input is modeled as a piecewise linear (PL) scalar field $f : \mathcal{M} \rightarrow \mathbb{R}$ defined on a PL d-manifold $\mathcal{M}$, with d equals 2 or 3 in our applications. The scalar values are given at the vertices of $\mathcal{M}$ and are linearly interpolated on the other simplices (with barycentric coordinates). f is assumed to be injective on the vertices of $\mathcal{M}$ (i.e. each vertex has a distinct f value). This is enforced in practice with a symbolic perturbation inspired by Simulation of Simplicity [86]. Specific requirements on the structure of the triangulation $\mathcal{M}$ are discussed in Secs. 3.1 and 6.5.

2.2 Critical Points

Topological features of f can be tracked with the notion of *sub-level* set, noted $f^{-1}_{-\infty}(w) = \{p \in \mathcal{M} \mid f(p) < w\}$. It is simply the subset of the data below a certain threshold w. In particular, the topology of these sub-level sets (in 3D their connected components, cycles and voids) can only change at specific locations, named the *critical points* of f [40]. In the PL setting, Banchoff [41] introduced a local characterization of critical points, defined as follows.

A *face* τ of a simplex $\sigma \in \mathcal{M}$ is a simplex of $\mathcal{M}$ that is defined by a non-empty, strict subset of the vertices of σ. We call σ a *co-face* of τ and we note $\tau < \sigma$. The *star* of a vertex $v \in \mathcal{M}$, noted $St(v)$, is the set of its co-faces: $St(v) = \{\sigma \in \mathcal{M} \mid v < \sigma\}$. This can be viewed as a small, combinatorial, neighborhood around v. The *link* of v, noted $Lk(v)$, is the set of the faces τ of the simplices σ of $St(v)$ with empty intersection with v: $Lk(v) = \{\tau \in \mathcal{M} \mid \tau < \sigma,\ \sigma \in St(v),\ \tau \cap v = \emptyset\}$. This can be viewed as the *boundary* of a small, combinatorial, neighborhood around v. The *lower link* of v, noted $Lk^-(v)$, is given by the set of simplices of $Lk(v)$ which only contain vertices *lower* than v: $Lk^-(v) = \{\sigma \in Lk(v) \mid \forall v' \in \sigma,\ f(v') < f(v)\}$. The upper link is defined symmetrically: $Lk^+(v) = \{\sigma \in Lk(v) \mid \forall v' \in \sigma,\ f(v') > f(v)\}$. A vertex v is *regular* if both $Lk^-(v)$ and $Lk^+(v)$ are simply connected. For such vertices, the sub-level sets enter the neighborhood of v, $St(v)$, through the lower part of the neighborhood boundary, $Lk^-(v)$, and exit through its upper part, $Lk^+(v)$, *without* changing their topology. Otherwise, v is a *critical point*. These can be classified with regard to their *index* $\mathcal{I}(v)$, which intuitively corresponds to the number of independent directions of decreasing f values around v. It is equal to 0 for local minima ($Lk^-(v) = \emptyset$), to d for local maxima ($Lk^+(v) = \emptyset$) and otherwise to i for i-saddles ($0 < i < d$).

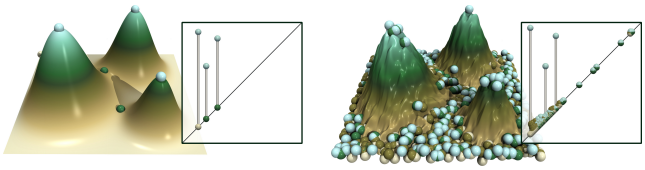

Fig. 2. Persistence diagrams of a clean (left) and noisy (right) scalar field (light brown spheres: minima, cyan: maxima, others: saddles). The main three hills are clearly apparent in the diagrams (high persistence pairs), whereas small pairs near the diagonal indicate noisy features.

Adjacency relations between critical points can be captured with the notion of *integral line*. Given a vertex v, its *forward* integral line, noted $\mathcal{L}^+(v)$, is a path along the edges of $\mathcal{M}$, initiated in v, such that each edge of $\mathcal{L}^+(v)$ connects a vertex v' to its highest neighbor v''. Then, forward integral lines are guaranteed to terminate in local maxima of f. When encountering a saddle s, we say that an integral line *forks*: it yields one new integral line per connected component of $Lk^+(s)$. Note that several integral lines can *merge* (and possibly fork later). A *backward* integral line, noted $\mathcal{L}^-(v)$ is defined symmetrically (i.e. integrating downwards towards minima). Critical points play a central role in TDA as they often correspond to features of interest in various applications: centers of vortices in fluid dynamics [11], atoms in chemistry [8], [9], [10] or clusters of galaxies in astrophysics [3], [4].

2.3 Persistence Diagrams

Several importance measures for critical points have been studied [5], including *topological persistence* [31], which is tightly coupled to the notion of Persistence diagram [1], which we briefly summarize here. In practical applications, features of interest are often characterized by the extrema of the field. Thus, in the following, we will first focus our description on local minima, and then discuss generalizations. The importance of a local minimum can be assessed with its *persistence*, which describes the lifetime of the topological feature (i.e. the connected component) it created in $f^{-1}_{-\infty}(w)$. As w increases, new connected components of $f^{-1}_{-\infty}(w)$ are created at the minima of f. The Elder rule [1] indicates that if two connected components, created at the minima m_0 and m_1 with $f(m_0) < f(m_1)$, meet at a given 1-saddle s, the *youngest* of the two components (the one created at m_1) *dies* in favor of the *oldest* one (created at m_0). In this case, a *persistence pair* (m_1, s) is created and its *topological persistence* p is given by $p(m_1, s) = f(s) - f(m_1)$. All local minima can be unambiguously paired following this strategy, while the global minimum is usually paired, by convention, with the global maximum. Symmetrically, persistence assesses the importance of a local maximum paired with a $(d-1)$-saddle, based on the lifetime of the topological feature it destroyed in $f^{-1}_{-\infty}(w)$.

Generally, as one continuously increases an isovalue w, topological structures in $f^{-1}_{-\infty}(w)$ (connected components, cycles, voids) are created and destroyed at critical points. Thus, each topological feature is characterized by a pair of critical points, which indicate its birth and death, and whose difference in function values indicates its lifespan in the data, its *persistence*. As described

above, the persistence of connected components of $f_{-\infty}^{-1}(w)$ is encoded with minimum/1-saddle pairs. In 3D, 2-saddle/maximum pairs characterize the life time of the voids of $f_{-\infty}^{-1}(w)$, while 1-saddle/2-saddle pairs characterize its independent cycles. As mentioned above, features of interest are often characterized by the extrema of the field. Thus, in the following, when considering persistence diagrams, we will focus on minimum/1-saddle pairs and $(d-1)$-saddle/maximum pairs.

Persistence pairs are usually visualized with the *Persistence diagram* $\mathcal{D}(f)$ [1], which embeds each pair (c,c'), with $f(c) < f(c')$, as a point in the 2D plane, at location $\big(f(c), f(c')\big)$. There, the pair persistence can be visualized as the height of the point to the diagonal. In other words, in the persistence diagram, each topological feature of $f_{-\infty}^{-1}(w)$ (connected component, cycle, void) can be readily visualized as a bar (Fig. 2), whose height to the diagonal denotes its importance in the data. Features with a high persistence stand out, away from the diagonal, while noisy features are typically located in its vicinity. The conciseness, stability [31] and expressiveness of this diagram made it a popular tool for data summarization tasks. As shown in Fig. 2, it provides visual hints about the number, ranges and salience of the features of interest.

3 Progressive Data Representation

This section details our hierarchical scheme for the progressive representation of the input data, which relies on a hierarchy of triangulations $\mathcal{H}$ derived from established subdivision schemes [79], [80], [81], [82], [83]. In particular, our goal is to define a hierarchical scheme that will enable efficient update mechanisms between hierarchy levels. This will avoid, at each new level of the hierarchy, the recomputation from scratch of the topological data representations presented in sections 4 and 5, and this will instead enable their progressive update. After a generic description of the employed triangulation hierarchy (Sec. 3.1), we present for completeness an efficient implementation [84], [85] for the special case of triangulations of regular grids (Sec. 3.2), on which we focus in this paper (Sec. 6.5 discusses generalizations). Next, we resume our generic description (Sec. 3.3) and show how to leverage the specific structure of the employed triangulation hierarchy to accelerate the topological analysis of the data. For this, we introduce the novel notion of *Topologically Invariant Vertices*, which is central to our work.

3.1 Edge-Nested Triangulation Hierarchy

Our progressive representation of the input data is based on a multiresolution hierarchy of the input PL-manifold $\mathcal{M}$, which relies on established subdivision schemes [79], [80], [81], [82], [83]. Intuitively, our goal is to define a multiresolution hierachy that will enable the efficient update of the topological information computed at the previous levels, in order to avoid full re-computations (Sec. 4). In order to construct such a hierarchical scheme, as formalized next, we impose that, as one progresses down the hierarchy, new vertices are only inserted along pre-existing edges (exactly one new vertex per edge, typically at their center), and that the additional new edges only connect new vertices (Fig. 3). This will have the beneficial effect of preserving, from one hierarchy level to the next, the *structure* of the local neighborhood around each pre-existing vertex (of its link, as discussed in Sec. 3.3), which will in turn effectively enable fast updates of the pre-existing local topological information (Sec. 4). We call such a hierarchy *edge-nested* and we formalize it in the following, to introduce the notations that will

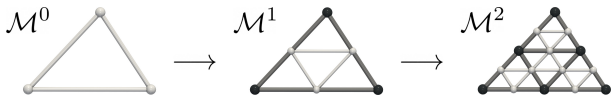

Fig. 3. Edge-nested triangulation hierarchy for a simple 2D example. Old vertices/edges are shown in black/gray. New vertices and edges are shown in white.

be used in the rest of the paper. Let $\mathcal{H} = \{\mathcal{M}^0, \mathcal{M}^1, \ldots, \mathcal{M}^h\}$ be a hierarchy of PL d-manifolds, which respects the following key conditions.

1) **Old Vertex Condition:** Each vertex of $\mathcal{M}^i$ (the triangulation at level i) also belongs to the vertex set, noted $\mathcal{M}_0^{i+1}$, of $\mathcal{M}^{i+1}$:
$$\mathcal{M}_0^i \subset \mathcal{M}_0^{i+1} \tag{1}$$
The vertices of $\mathcal{M}^{i+1}$ already present in $\mathcal{M}^i$ are called *old vertices* (black spheres in Fig. 3).
2) **New Vertex Condition:** Each vertex of $\mathcal{M}^{i+1}$ not present in $\mathcal{M}^i$ has to be located on an edge (v_0, v_1) of $\mathcal{M}^i$ (typically at its center), as summarized below, where $\mathcal{M}_1^i$ stands for the edge set of $\mathcal{M}^i$:
$$\forall v \in \mathcal{M}_0^{i+1}, v \notin \mathcal{M}_0^i : \quad \exists (v_0, v_1) \in \mathcal{M}_1^i,\ v \in (v_0, v_1) \tag{2}$$
The vertices of $\mathcal{M}^{i+1}$ not present in $\mathcal{M}^i$ are called *new vertices* (white spheres in Fig. 3).
3) **Old Edge Condition:** Each edge (v_0, v_1) of $\mathcal{M}^i$ has to be subdivided at level $i+1$ at exactly one new vertex v of $\mathcal{M}^{i+1}$:
$$\begin{aligned} \forall (v_0, v_1) \in \mathcal{M}_1^i : \quad & |\{v \in (v_0, v_1),\ v \notin \mathcal{M}_0^i,\ v \in \mathcal{M}_0^{i+1}\}| = 1 \\ & (v_0, v) \in \mathcal{M}_1^{i+1}, \quad (v, v_1) \in \mathcal{M}_1^{i+1} \\ & (v_0, v_1) \notin \mathcal{M}_1^{i+1} \end{aligned} \tag{3}$$
The edges of $\mathcal{M}^{i+1}$ obtained by subdivision of an edge of $\mathcal{M}^i$ are called *old edges*, they connect old vertices to new vertices (gray cylinders in Fig. 3).
4) **New Edge Condition:** Each edge of $\mathcal{M}^{i+1}$ which is not an old edge has to connect two new vertices, and it is called a *new edge* (white cylinders in Fig. 3).

Fig. 3 presents a simple example of 2D edge-nested triangulation hierarchy. Note that the Loop subdivision [81] is compatible with the above formalization, which is more generally termed as *red* subdivision in the scientific computing literature, and which has been extensively studied for domains of two [80], three [82], [83] and arbitrary dimensions [79]. An input PL manifold $\mathcal{M}$ admits an edge-nested triangulation hierarchy if there exists a hierarchy $\mathcal{H}$ for which $\mathcal{M}$ is the last element ($\mathcal{M} = \mathcal{M}^h$).

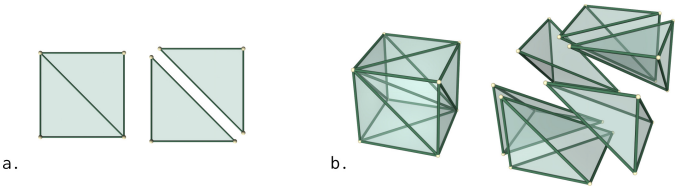

Fig. 4. Translation invariant local triangulation pattern for the cells of a 2D and 3D regular grid. In 2D, quadrilaterals are subdivided into two triangles (a), always along the same diagonal. In 3D, the generalization of this pattern subdivides each hexahedron into six tetrahedra (b).

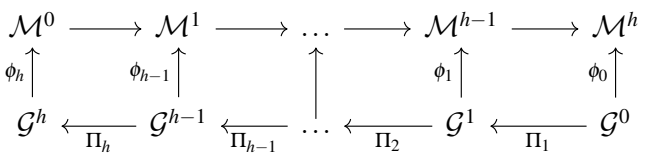

Fig. 5. Commutative diagram for the generation of an edge-nested triangulation hierarchy $\mathcal{H} = \{\mathcal{M}^0, \mathcal{M}^1, \dots, \mathcal{M}^h\}$ from a regular grid $\mathcal{G}^0$. The hierarchy can be obtained by a sequence of decimation operators Π_i, accompanied with triangulation operators ϕ_i.

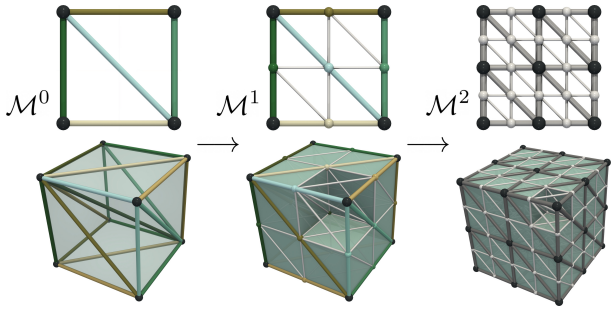

Fig. 6. Edge-nested triangulation hierarchy generated from a regular grid. Old vertices/edges are shown in black/gray in $\mathcal{M}^2$. There is a one-to-one mapping (colors from $\mathcal{M}^0$ to $\mathcal{M}^1$) between the edges of $\mathcal{M}^0$ and the new vertices of $\mathcal{G}^{h-1}$, inserted in each i-dimensional cell of $\mathcal{G}^h$ $(0 < i \leq d)$.

3.2 Edge-Nested Triangulations of Regular Grids

While the construction of an edge-nested triangulation hierarchy given an arbitrary input manifold $\mathcal{M}$ is an open question which we leave for future work (see Sec. 6.5), it can be shown that such a hierarchy exists for regular grids, and that it can be implemented very efficiently, as discussed by Bey [85]. We describe this implementation in the following for the sake of completeness, by detailing how to efficiently retrieve an arbitrarily coarse version of the fine triangulation $\mathcal{M}^h$ from an input regular grid $\mathcal{G}^0$.

Let $\mathcal{G}^0$ be a d-dimensional regular grid, with d equal to 2 or 3 in our applications, of dimensions L_x^0, L_y^0, L_z^0 $\big($*i.e.* of number of vertices $|\mathcal{G}_0^0| = (L_x^0+1) \times (L_y^0+1) \times (L_z^0+1)$, in 2D: $L_z^0 = 0\big)$. We will first assume that L_x^0, L_y^0 and L_z^0 are all powers of 2. Let ϕ_0 be the *triangulation operator*, which transforms $\mathcal{G}^0$ into a valid triangulation $\mathcal{M}^h$, i.e. $\mathcal{M}^h = \phi_0(\mathcal{G}^0)$, by preserving vertex sets, i.e. $\mathcal{M}_0^h = \mathcal{G}_0^0$, and by inserting exactly one edge for each i-dimensional cell of $\mathcal{G}^0$ $(1 < i \leq d)$, according to a unique pattern, which is *invariant by translation* along the cells of the grid, known as Kuhn's triangulation [84]. In 2D, each quadrilateral is subdivided into two triangles by inserting one edge always along the *same diagonal*. In 3D, each hexahedron is subdivided into six tetrahedra by always inserting the *same diagonal* edges (Fig. 4).

Let Π_1 be the *decimation operator*, which transforms the regular grid $\mathcal{G}^0$ into a regular grid $\mathcal{G}^1$, i.e. $\mathcal{G}^1 = \Pi_1(\mathcal{G}^0)$, by selecting one vertex every two vertices in each dimension. Let (i,j,k) be the grid coordinates of a vertex $v \in \mathcal{G}^0$. Then the grid $\mathcal{G}^1$ is obtained by only selecting the vertices with even grid coordinates (i,j,k) in $\mathcal{G}^0$. In 2D, each quadrilateral of $\mathcal{G}^1$ corresponds in the general case to four quadrilaterals of $\mathcal{G}^0$ and in 3D, each hexahedron of $\mathcal{G}^1$ corresponds to eight hexadra of $\mathcal{G}^0$. Note that the decimation operator Π_1 induces a reciprocal *subdivision* operator, which, given $\mathcal{G}^1$, yields $\mathcal{G}^0$ by inserting a new vertex in the center of each i-dimensional cell of $\mathcal{G}^1$ $(0 < i \leq d)$.

We now introduce by recurrence a sequence of decimation operators Π_i (Fig. 5), which decimate each grid $\mathcal{G}^{i-1}$ into a grid $\mathcal{G}^i$ by sub-sampling its vertices with even grid coordinates as described above. It follows that for a given level of decimation i, the dimensions of $\mathcal{G}^i$ are given by $L_x^i = L_x^0/2^i$, $L_y^i = L_y^0/2^i$, and $L_z^i = L_z^0/2^i$. Let us now consider the sequence of triangulation operators ϕ_i, which triangulate each grid $\mathcal{G}^i$ into a triangulation $\mathcal{M}^{h-i}$, i.e. $\mathcal{M}^{h-i} = \phi_i(\mathcal{G}^i)$, as illustrated by the commutative diagram of Fig. 5. Then, it can be verified (Fig. 6) that each condition of Sec. 3.1 is indeed satisfied by the sequence $\mathcal{H} = \{\mathcal{M}^0, \mathcal{M}^1, \dots, \mathcal{M}^h\}$ and that $\mathcal{H}$ is a valid edge-nested triangulation hierarchy. In particular, as described by Bey [85], any triangulation $\mathcal{M}^i$ can be equivalently obtained either: *(i)* by applying the red subdivision scheme [80], [82], [83] i times on $\mathcal{M}^0$ or *(ii)* by considering the Kuhn triangulation [84] of $\mathcal{G}^{h-i}$ (itself obtained by i regular subdivisions of $\mathcal{G}^h$). In other words, any triangulation $\mathcal{M}^i$ in the commutative diagram of Fig. 5 can be obtained by starting either *(i)* from $\mathcal{M}^0$ or *(ii)* from $\mathcal{G}^h$. In our work, we exploit this equivalence property, but in *reverse*: we use it to efficiently retrieve an arbitrarily coarse version of the fine triangulation $\mathcal{M}^h$ of the input grid $\mathcal{G}^0$.

In particular, the edge-nested triangulation hierarchy $\mathcal{H}$ can be implemented very efficiently, by encoding the entire hierarchy implicitly, and by only maintaining the grid $\mathcal{G}^0$ in memory. At a given hierarchy level i, adjacency relations in $\mathcal{M}^i$ between two vertices v_0 and v_1 can be inferred based on their grid coordinates at level i, (i_0, j_0, k_0) and (i_1, j_1, k_1), and given the triangulation pattern shown in Fig. 4. Then, the data values associated to the vertices v_0 and v_1 can be retrieved by mapping these vertices back to their original locations in $\mathcal{G}^0$, given by the grid coordinates $(i_0 \times 2^{h-i}, j_0 \times 2^{h-i}, k_0 \times 2^{h-i})$ and $(i_1 \times 2^{h-i}, j_1 \times 2^{h-i}, k_1 \times 2^{h-i})$. This approach is easily extended to support regular grids whose dimensions, L_x^0, L_y^0 or L_z^0 are not necessarily powers of 2. In particular, when considering the decimation operator Π_i, in case some of the dimensions L_x^{i-1}, L_y^{i-1} or L_z^{i-1} are not even, Π_i systematically adds the last vertex of $\mathcal{G}^{i-1}$ for each odd dimension. In our progressive algorithms (Sec. 4 and 5), these few extra vertices will require full recomputations. Below, we resume our generic description for arbitrary edge-nested triangulation hierarchies, not necessarily obtained from regular grids (Sec. 6.5 discusses generalizations).

3.3 Topologically Invariant Vertices

The input edge-nested triangulation hierarchy $\mathcal{H}$ yields a hierarchy of PL scalar fields $\{f^0, f^1, \dots, f^h\}$, such that each old vertex v maintains by construction its scalar value: $f^i(v) = f^j(v) = f(v)$, $\forall j \ / \ i \leq j \leq h$. In the following, we show how the specific structure of edge-nested triangulation hierarchies described in Sec. 3.1 can be leveraged to efficiently update topological information while progressing down the hierarchy. First we show that edge-nested triangulations preserve the topology of the link of vertices when progressing from one hierarchy level to the next. This enables the quick identification, discussed next, of vertices which do not change their criticality when progressing down the hierarchy. We call these vertices *topologically invariant old vertices*, as they will need no update during subsequent analyses (Sec. 4 and Sec. 5). Last, we show how to efficiently identify new vertices that are guaranteed by construction to be regular points of f^i, which we call *topologically invariant new vertices* and for which no computation will be required in subsequent analyses.

1) Link Topological Invariance: A first insight is that the link $Lk(v)$ of a vertex v is topologically invariant throughout the

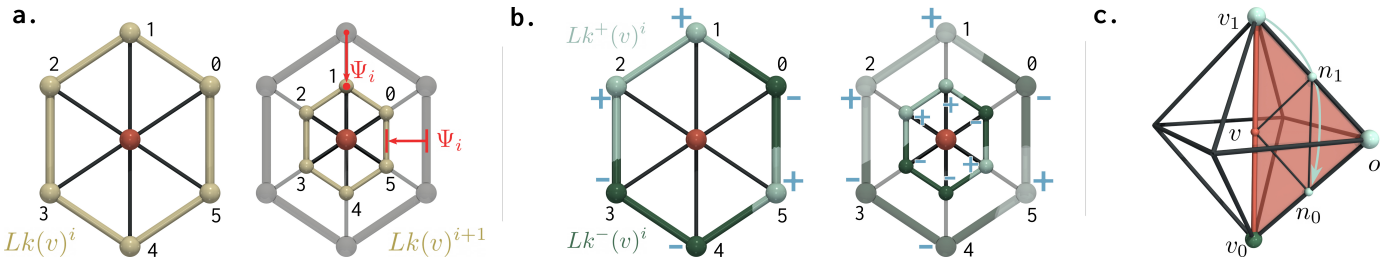

Fig. 7. Important properties of edge-nested triangulations, enabling fast updates of local topological information. (a) Left: From one hierarchy level (i) to the next ($i+1$), edge-nested triangulations preserve the local structure of the link $Lk(v)^i$ of an old vertex v (red sphere). In particular, there exists a one-to-one mapping Ψ_i between the vertices and the edges (red arrows) of $Lk(v)^i$ and $Lk(v)^{i+1}$. (b) Center: this link invariance enables the fast identification of old vertices which do not change their criticality: these are old vertices (red sphere) for which the *polarity* (blue signs) remains unchanged from one hierarchy level (i) to the next ($i+1$) and for which, therefore, connected components of lower and upper links (green and blue components, respectively) do not change (thus, requiring no update). Such vertices are called *topologically invariant old vertices*. (c) Right: A new vertex v which is *monotonic* (i.e. $f(v_0) < f(v) < f(v_1)$, with v_0 and v_1 being respectively the lowest and highest vertex of the edge (v_0, v_1) where v is inserted) is guaranteed to be regular if all its adjacent new neighbors (in the figure, n_0 and n_1) are also *monotonic* (see Sec. 3.3 for further discussion).

hierarchy. This property is important because it will enable the fast identification of vertices which do not change their criticality (next paragraph). Let $Lk(v)^i$ be the link of v at level i, then there exists a one-to-one mapping Ψ_i (Fig. 7(a)) between the simplices of $Lk(v)^i$ and $Lk(v)^{i+1}$ – such that $Lk(v)^{i+1} = \Psi_i\big(Lk(v)^i\big)$ – which preserves the simplicial structure of $Lk(v)^i$ (which preserves adjacencies). Indeed, *(i)* new vertices are only inserted on old edges (this maps the k^{th} neighbor of v at level i to its k^{th} new neighbor at level $i+1$, top red arrow in Fig. 7(a)) and *(ii)* new edges are only inserted between new vertices (this maps the k^{th} edge of $Lk(v)^i$ to the k^{th} new edge of $Lk(v)^{i+1}$, right red arrow in Fig. 7(a)). This mapping Ψ_i can be viewed as a combinatorially invariant *zoom* in the neighborhood of v as one progresses down the hierarchy.

2) Topologically Invariant Old Vertices: A second insight deals with the evolution of the data values on the link of an *old* vertex, as one progresses down the hierarchy and zooms with the above mapping Ψ_i. We define the *polarity* of $Lk(v)^i$, noted $\delta : Lk(v)^i \rightarrow \{-1, 1\}$ as the field which assigns to each neighbor n of v at level i the sign of its function difference with v: $\delta(n) = sgn\big(f(n) - f(v)\big)$. The polarity is positive in the upper link, negative in the lower link (Fig. 7(b), blue signs). Let (v_0, v_1) be an edge at level i, which gets subdivided at level $i+1$ along a new vertex v_n. Assuming that $f(v_0) < f(v_1)$, we say that v_n is *monotonic* if $f(v_n) \in \big(f(v_0), f(v_1)\big)$. Otherwise, v_n is *non-monotonic*. In that case, if v_n's polarity in $Lk(v_0)^{i+1}$ is the opposite of v_1's polarity in $Lk(v_0)^i$, we say that v_0 is *impacted* by its neighbor v_n. Now, if an old vertex v is not *impacted* by any of its non-monotonic neighbors, its link polarity is maintained (i.e. the blue signs in Fig. 7(b) remain unchanged when going from the hierarchy level i to $i+1$). This implies that v is therefore guaranteed to maintain its *criticality*: it maintains its critical index (i.e., $\mathcal{I}(v)^{i+1} = \mathcal{I}(v)^i$) or it remains regular. Indeed, each neighbor n which does not impact v maintains its classification as being upper or lower. Then, since there is a one-to-one mapping Ψ_i (see Fig. 7(a)) between $Lk(v)^i$ and $Lk(v)^{i+1}$ which preserves their simplicial structure, it follows that the complexes $Lk^-(v)^{i+1}$ and $Lk^+(v)^{i+1}$ are respectively identical to $Lk^-(v)^i$ and $Lk^+(v)^i$. Thus, the number of connected components of lower and upper links are maintained, preserving the criticality of v. Old vertices which are not impacted by their non-monotonic neighbors are called *topologically invariant old vertices*.

3) Topologically Invariant New Vertices: A third insight deals with the link of *new* vertices. Given a new monotonic vertex v (small red sphere in Fig. 7(c)) subdividing an edge (v_0, v_1) at level i (red cylinder in Fig. 7(c)), if its new neighbors are all monotonic as well, v is then called an *interpolating vertex* and it can be shown that v must be a regular vertex. First, since v is monotonic, it cannot be an extremum, since by definition it is connected to one lower (v_0) and one upper (v_1) old vertex (large green and blue spheres in Fig. 7(c)). Note that v_0 and v_1 are the only old vertices adjacent to v. Second, to show that v is regular, we argue that $Lk^+(v)^i$ is necessarily connected (and so is $Lk^-(v)^i$, symmetrically). Let (v_0, v_1, o) be a triangle at level $i-1$ (red triangle in Fig. 7(c)). At level i, the edges (v_0, o) and (v_1, o) are subdivided along the new vertices n_0 and n_1 and the *new* edges (v, n_0), (v, n_1), and (n_0, n_1) are inserted to connect the new vertices. Let us assume that $f(n_0) > f(v)$. n_0 is then an *upper* neighbor of v ($n_0 \in Lk^+(v)^i$). Since n_0 is monotonic, this means that the *outer* old vertex o (which is not in $Lk(v)^i$) must also be upper: $f(o) > f(n_0) > f(v)$. Since n_1 is monotonic as well, it follows that n_1 is upper too. Thus, there exists a path $\{v_1, n_1, n_0\} \in Lk(v)^i$ (blue arrow in Fig. 7(c)), which connects v_1 to n_0 and which is only composed of *upper* vertices. Thus n_0 and v_1 belong to the same connected component of $Lk^+(v)^i$. The same reasoning holds for all the new *upper* neighbors of v. It follows that $Lk^+(v)^i$ and $Lk^-(v)^i$ are both made of a single connected component, containing exactly one old vertex each, v_1 and v_0 respectively. Thus, v is regular. Note that this reasoning readily applies to 2D and 3D. Since interpolating vertices, such as v, imply no topological event in the sub-level sets, we call them *topologically invariant new vertices*.

The three key insights of edge-nested triangulations discussed above (summarized in Fig. 8) form the cornerstone of our progressive approach to topological analysis. As detailed next, checking if vertices are topologically invariant turns out to be less computationally expensive in practice than computing their criticality from scratch. Moreover, the set of topologically invariant vertices tends to represent the majority of the hierarchy (see Sec. 6). This allows for the design of efficient progressive algorithms, presented in the next sections.

4 Progressive Critical Points

Our progressive algorithm for critical point extraction starts at the first level of the hierarchy, $\mathcal{M}^0$, and progresses level by level down the hierarchy $\mathcal{H}$ until reaching its final level, $\mathcal{M}^h$, or until interrupted by a user. At each level i, our approach delivers the

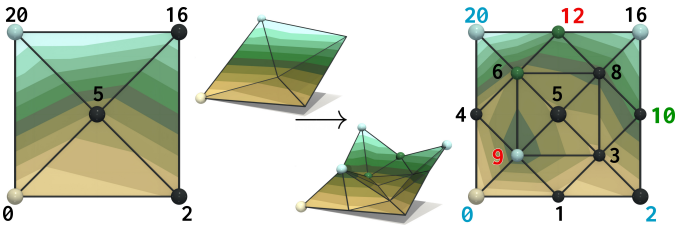

Fig. 8. *Topologically Invariant* (TI) vertices (numbers denote f values). When progressing down the hierarchy, two *non-monotonic* vertices appear (red labels). This yields new critical points (cyan: maxima, green: saddles, brown: minimum). *Old* TIs (blue labels), whose link polarity is unchanged, maintain their criticality. *New* TIs are regular (green label). For *topologically invariant* vertices (blue and green labels), no computation is required. As illustrated in Tab. 1, TI vertices represent the majority of the data in real-life datasets.

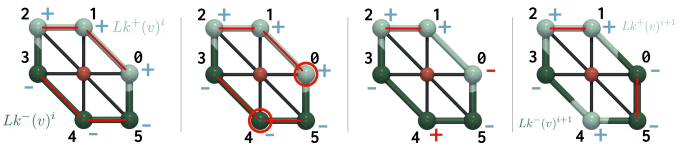

Fig. 9. Updating the criticality of a non-topologically invariant old vertex. From left to right: initial state, identification of *non-monotonic* vertices (red circles), update of the *link polarity* (red $+/-$ signs), and update of the connected components of $Lk^+(v)$ and $Lk^-(v)$. At each step, edges present in the dynamic tree [87] are highlighted in red. Only the edges impacted by polarity flips need to be updated in the dynamic tree: edges $(0,1)$, $(3,4)$ and $(4,5)$ are removed, and the edge $(0,5)$ is added.

entire list of critical points of the data for the current resolution ($f^i : \mathcal{M}^i \rightarrow \mathbb{R}$). For this, our strategy consists in avoiding recomputation as much as possible and instead efficiently and minimally update the information computed at the previous level $(i-1)$.

4.1 Initialization and Updates

This section focuses on the vertices of $\mathcal{H}$ which are not *topologically invariant*. The case of topologically invariant vertices is discussed in Sec. 4.2. In short, our approach computes the criticality of each vertex with the traditional method [41] at the first hierarchy level. However, for the following levels, instead of re-starting this computation from scratch, our algorithm maintains the criticality information computed at the previous levels and only minimally updates this information, where needed, by using dynamic trees [87], a specialized data structure for dynamic connectivity tracking.

At the first hierarchy level, $\mathcal{M}^0$ only contains new vertices for which the criticality needs to be initialized. As of the second level, old and new vertices start to co-exist in $\mathcal{M}^1$ and fast update mechanisms can be considered to efficiently update the criticality of the old vertices. For this, we leverage the topological invariance of the link of each old vertex throughout the hierarchy (Sec. 3.3). This allows to store relevant topological information on the link and to quickly update them when progressing down the hierarchy. In particular, we initialize for each *new* vertex v at level i the following information:

- *Link 1-skeleton:* We store the list of *local* edges (and their adjacencies) of $Lk(v)^i$, encoded with pairs of local indices for the neighbors of v. This remains invariant through $\mathcal{H}$ (Sec. 3.3).
- *Link polarity:* We store for each vertex of $Lk(v)^i$ its *polarity* (Sec. 3.3), i.e. its classification as being upper or lower than v. This is encoded with one bit per vertex of $Lk(v)^i$.
- *Link dynamic tree:* An efficient data structure [87] for maintaining connected components in dynamic graphs, discussed below.

For each new vertex v which is not topologically invariant, the following data structures are initialized: a list of pairs of local neighbor indices denoting the local edges of $Lk(v)^i$ (up to 24 pairs in a 3D grid), a list of bits denoting the polarity of each neighbor (up to 14 neighbors in a 3D grid), and the dynamic tree, detailed below. The criticality of v is computed with the traditional approach (Sec. 2.2), by enumerating the connected components of $Lk^+(v)^i$ and $Lk^-(v)^i$. This is usually achieved with breadth-first search traversals or with a Union-Find (UF) data structure [42]. However, in our setting, we would like to update these connected components as the algorithm progresses down the hierarchy. In particular, if a *local* edge e belongs to the upper link of v at level i, but not anymore at level $i+1$, the connected components of $Lk^+(v)^{i+1}$ need to be updated accordingly, preferably without recomputing them completely. For this, we use dynamic trees [87], which, like the UF data structure, maintain connected components in a graph upon edge insertion, but unlike the UF, also maintain them upon edge removal. In particular, all the vertices of $Lk(v)^i$ are initially inserted in the dynamic tree associated to v. Next, we insert each local edge of $Lk(v)^i$ in the dynamic tree, if both its ends have the same polarity. The criticality of v is then deduced by enumerating the connected components with positive and negative polarity, thanks to the dynamic tree.

For each old vertex v which is not topologically invariant (Fig. 9), its link polarity is quickly updated based on the non-monotonic new vertices of $Lk(v)^i$. Each local edge e of $Lk(v)^i$ which is *impacted* by a polarity flip of its vertices (Sec. 3.3) is removed from the dynamic tree associated to v if it was present in it (to account for the corresponding disconnection of lower/upper link component), and added to it otherwise, if both its ends have the same polarity (if they belong to the same lower/upper link component). Then, the criticality of v is quickly updated with the fast enumeration of the connected components of positive and negative polarity provided by the dynamic tree. Note that such an efficicent update of the criticality of v would not be feasible with a simple UF data structure, as the connected components of the link of v would need to be recomputed from scratch upon edge removal.

4.2 Computation Shortcuts

When moving from the hierarchy level i to $i+1$, topologically invariant old vertices are guaranteed to maintain their criticality (Sec. 3.3). For these, the dynamic trees (Sec. 4.1) do not need to be updated. Moreover, when moving from the hierarchy level i to $i+1$, each topologically invariant new vertex v is guaranteed to be regular. For these, the dynamic trees (Sec. 4.1) are not even initialized (they will only be used when v becomes no longer topologically invariant). Overall, our procedure to update vertex criticality can be summarized as follows:

1) Mononotic vertices: in this step, we loop over all new vertices to check whether or not they are monotonic.

2) Link polarity: in this step, we loop over all vertices to initialize/update their link polarity. For old vertices, updates are only needed for their non-mononotic neighbors. If an old vertex v is topologically invariant, no more computation is required for it at this hierarchy level.

3) Old vertices: each old vertex v which is not topologically invariant efficiently updates its criticality in f^i as described in Sec. 4.1.
4) New vertices: if a new vertex v is topologically invariant, it is classified as regular and no more computation is required for it at this hierarchy level. Otherwise, its criticality is updated (Sec. 4.1).

4.3 Parallelism

Critical point computation is an operation which is local to the link of each vertex. Thus, each of the four steps introduced above can be trivially parallelized over the vertices of $\mathcal{M}^i$ with shared-memory parallelism. This implies no synchronization, at the exception of the sequential transition between two consecutive steps.

4.4 Extremum Lifetime

As our algorithm progresses down $\mathcal{H}$, the population of critical points evolves. In practice, this means that some features of interest may be captured by the progressive algorithm earlier than others, denoting their importance in the data. To evaluate this, we consider for each extremum e the notion of *Lifetime*, defined as $l(e) = l_d(e) - l_a(e)$, where $l_a(e)$ and $l_d(e)$ stand for the levels where e appeared and disappeared respectively. The evaluation of this measure requires a correspondence between the extrema computed at the levels i and $i+1$, which is in general a challenging assignment optimization problem [16], [88], [89], [90], [91], [92]. For simplicity, we focus here on a simple yet time-efficient heuristic for estimating these correspondences, which can be enabled optionally.

Given a vertex v, identified as maximum at hierarchy level $i-1$, our heuristic consists of computing, for each neighbor n of v, an integral line $\mathcal{L}^+(n)^i$. Each of these lines terminates on local maxima of f^i, which we add to the set of *candidates* for v. At the end this step, we establish the correspondence between v and its highest candidate in terms of f^i values, noted m^*, and we say that v *maps* to m^* from $i-1$ to i. To focus the integration on a reasonable neighborhood, we restrict the number of edges on each integral line to a user parameter L_{max}, set to 10 in our experiments. If the set of *candidates* of v is empty, the maximum present in v at level $i-1$ is considered to disappear at level i $\big(l_d(v) = i\big)$. It is possible, given a maximum m at level i, that no maximum from the level $i-1$ maps to it. In this case, m is said to appear at the level i $\big(l_a(m) = i\big)$. Finally, if multiple maxima at level $i-1$ map to the same maximum at level i, they are all considered to disappear at the level i, at the exception of the *oldest* maximum (minimizing l_a), as suggested by the *Elder* rule in the case of persistence [1]. This optional procedure is run at each hierarchy level and enables the progressive estimation of the lifetime of the maxima. Note that the lifetime of minima is estimated with the symmetric procedure.

5 Progressive Persistence Diagrams

Our approach for progressive persistence diagrams leverages and combines the insights and algorithms introduced in the previous sections. It starts at the coarsest hierarchy level, $\mathcal{M}^0$, and then iterates progressively through the hierarchy levels, producing the exact persistence diagram $\mathcal{D}(f^i)$ for each level i, until $i = h$. We first introduce our approach in the non-progressive case (Sec. 5.1, Fig. 10), and then present our progressive strategy (Sec. 5.2). We focus on minimum-saddle persistence pairs, saddle-maximum pairs being treated symmetrically.

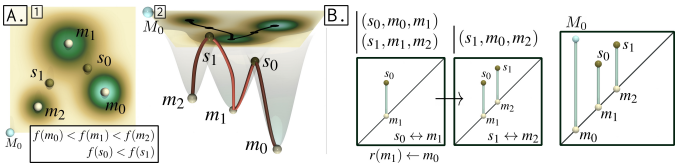

Fig. 10. Computing the minimum-saddle persistence diagram from critical points. Downwards monotonic paths are initiated at saddles to extract a list of critical point triplets (part A, left), which forms a reduced topological representation of the data. This reduced reprensentation is efficiently processed to produce the persistence diagram (part B, right).

5.1 Persistence Diagram from Critical Points

The diagram $\mathcal{D}(f)$ of the extremum-saddle pairs of an input field $f : \mathcal{M} \to \mathbb{R}$ is computed as follows. In short, critical points are used as seeds for the computation of monotonic paths, specifically linking saddles down to minima. This first step identifies merge events occurring at saddle points (*part A*). The merge events are processed in a second step (*part B*) to track the connected components of sub-level sets. Similarly to previous topological techniques based on monotonic paths [22], [47], [50], [93], our approach emulates the usage of a Union-Find data structure with path compression [42] (traditionally used for connectivity tracking) by propagating *representants* between merged components. However our strategy is specialized for the production of persistence diagrams, and only visits monotonic paths of minimal length (i.e. integral lines).
Part A:
From data to reduced topological information
1) Critical points. First, critical points are extracted (Sec. 2.2).
2) Saddle monotonic paths. The second step consists in initiating monotonic paths from each saddle s downwards, to identify at least one minimum for each connected component of sub-level set merging at s (Fig. 10). For this, we initiate backward integral lines (Sec. 2.2), for each connected component of lower link $Lk^-(s)$ of each saddle s. These integral lines are guaranteed to terminate in local minima of f. Once a backward integral line $\mathcal{L}^-(s)$ terminates in a local minimum m, we back-propagate the vertex identifier of m and store it for each vertex $v \in \mathcal{L}^-(s)$. Then, m is called a *representant* of v, which is noted $r(v) = \{m\}$. This strategy enables the early termination of an integral line $\mathcal{L}^-(s_1)$ when it merges with another one, $\mathcal{L}^-(s_0)$, computed previously. In that case, we back-propagate the representants reported by the merge vertex on $\mathcal{L}^-(s_1)$ back to s_1. At the end of this step, each saddle s is associated with the list of representants collected by its backward integral lines. These denote local minima which may have initially created the sub-level set components merging at s.

Part B:
From reduced topological information to persistence diagrams
3) Critical triplets. For each saddle s, we create a list of *critical triplets*, in the form (s, m_0, m_1), where m_0 and m_1 are representants of s and thus are local minima. These are obtained by considering pairs among the set of representants of s (computed previously). Note that in practice, for nearly all saddles, this list consists of only one triplet, which describes the fact that s separates two pits, m_0 and m_1. Note that in case of degenerate saddles, multiple triplets emerge. For a degenerate saddle associated with d representants $(m_0, \ldots, m_{d-1})$ in ascending values of f, we create the $d-1$ triplets (s, m_0, m_i) with $0 < i < d$.

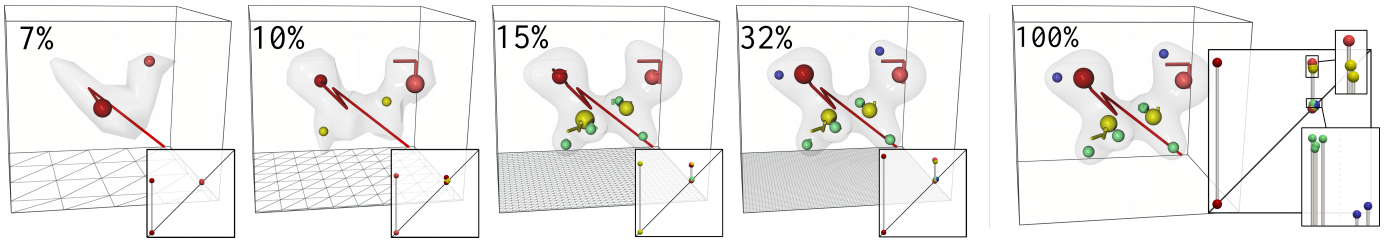

Fig. 11. Progressive persistence diagrams (saddle-maximum pairs, left to right) for the electron density of the ethane-diol molecule (transparent isosurface), at a few steps of the progressive computation. Maxima (denoting the atoms) are shown in the domain with spheres, scaled by persistence and colored by lifetime (red to blue), while their trajectory through the data hierarchy (Sec. 4.4) is shown with a curve (matching color). Our progressive approach captures the heaviest atoms first: two oxygens (at 7% of the computation time), then two carbons (10%) and finally the six hydrogens (32%).

4) Critical point pairing. This step iterates over the global list of critical triplets (computed previously) in increasing order of saddle values. The first triplet (s_0, m_0, m_1) represents the earliest merge event between connected components of sub-level sets of f. We introduce its *simplified version*, $\big(s_0, r(m_0), r(m_1)\big)$, which is initially equal to (s_0, m_0, m_1) (initially, a local minimum is itself its own representant). The highest of the two minima, for instance m_1, is then selected to create in $\mathcal{D}(f)$ the critical point pair (s_0, m_1). Indeed, since s_0 is the earliest merge event, m_1 is guaranteed to be the *youngest* minimum, according to the Elder rule [1], which created a component of sub-level set merging with another one at s_0. To model the *death* of m_1's component (its merge with the component containing m_0), we update its representant as follows: $r(m_1) \leftarrow r(m_0)$. Thus, all future merging events involving m_1 will re-direct to m_0, as the component born at m_1 died by merging with that containing m_0 (following the Elder rule [1]). This simplification process is iterated over the (sorted) global list of critical triplets. At each step, when constructing a simplified triplet $\big(s, r(m_0), r(m_1)\big)$, we recursively retrieve the representants of $r(m_0)$ and $r(m_1)$, until we reach minima only representing themselves. This guarantees that for each merge event of the sub-level set occurring at a saddle s, we can efficiently retrieve the deepest minimum for each of the components merging in s and therefore pair it adequately in $\mathcal{D}(f)$. Note that the recursive update of representants is equivalent to the so-called *path compression* of UF data structures [42]. Overall, iterating as described above over the list of triplets results in populating $\mathcal{D}(f)$ with pairs from bottom to top (by increasing death values).

5.2 Progressive Strategy

The above algorithm is divided in two parts (A and B, Sec. 5.1). In particular, only part A can leverage our progressive representation of the input data (Sec. 3), as part B processes reduced topological information which has been abstracted from it and which therefore become completely independent. Thus, we focus our progressive strategy on part A. This has a negligible impact on practical performance. In our experience, part B represents less than 5% of the computation on average. Critical points (Step 1) can be extracted progressively as described in Sec. 4. For Step 2, we investigated multiple shortcut mechanisms (similar to Sec. 4.2), to maintain the monotonic paths which remain valid from level i to $i+1$. However, from our experience, the overhead induced by this global maintenance is not compensated by the acceleration it induces at level $i+1$, as monotonic paths are usually highly localized and thus already inexpensive to compute (less than 10% of the non-progressive computation on average). Thus, our overall strategy for

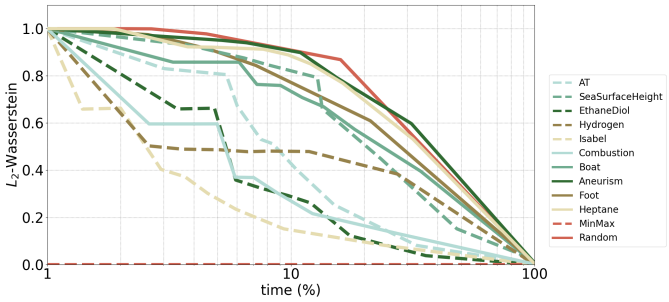

Fig. 12. Empirical convergence of the normalized L_2-Wasserstein distance. Each curve plots the distance between the currently estimated diagram, $\mathcal{D}(f^i)$, and the final, exact diagram, $\mathcal{D}(f)$, as a function of the percentage of computation time (logarithmic scale).

progressive persistence diagrams simply consists, at each level i of the triangulation hierarchy $\mathcal{H}$, in updating progressively the critical points (Sec. 4) and then triggering the fast, remaining steps of persistence diagram computation (2, 3, 4) as described in Sec. 5.1.

5.3 Parallelism

Our progressive algorithm for persistence diagram computation can be easily parallelized. The initial critical point computation (Step 1, Sec. 5.1) is parallelized as described in Sec. 4.3. Saddle integration (Step 2, Sec. 5.1) can be trivially parallelized over saddles. However, locks need to be used during representant back propagation (to guarantee consistency over concurrent accesses by distinct monotonic paths). Critical triplet generation (Step 3, Sec. 5.1) is also parallelized over saddles. In Step 4 (critical point pairing Sec. 5.1), triplets are sorted in parallel using the efficient GNU implementation [94]. The reminder of Step 4 is intrinsically sequential (as representants need to be updated in order of simplification), but in practice, this step represents less than 1% of the sequential execution, which does not impact parallel efficiency.

6 Results

This section presents experimental results obtained on a computer with two Xeon CPUs (3.0 GHz, 2x4 cores, 64GB of RAM), with a C++ implementation of our algorithms (publicly available at: https://github.com/julesvidal/progressive-scalar-topology), written as modules for the Topology ToolKit (TTK) [95]. The datasets are 3-dimensional (at the exception of *SeaSurfaceHeight*, which is 2-dimensional) and they have been downloaded from public repositories [96], [97].

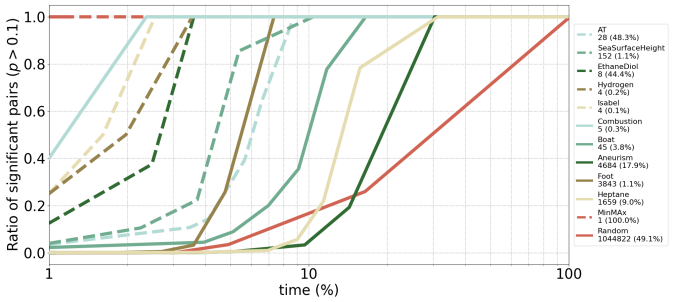

Fig. 13. Ratio of captured *significant pairs* in $\mathcal{D}(f^i)$ (cf. Sec. 6.1) as a function of computation time (logarithmic scale).

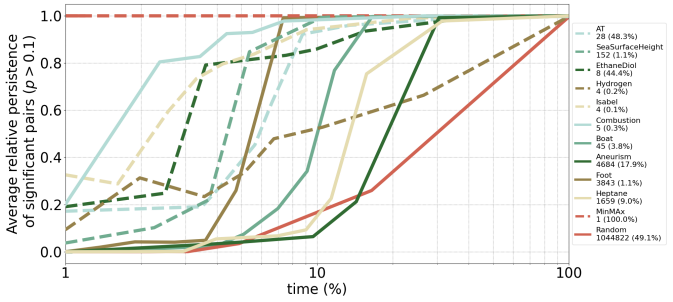

Fig. 14. Average persistence of the *significant pairs* captured in $\mathcal{D}(f^i)$ (cf. Sec. 6.1), relatively to the same average in $\mathcal{D}(f)$, as a function of computation time (logarithmic scale).

6.1 Progressive Data Representation

In this section, we study the practical relevance of our progressive data representation (Sec. 3). First, we evaluate its qualitative relevance. Our approach for persistence diagram computation (Sec. 5) progressively refines an estimation of the output $\mathcal{D}(f)$, by efficiently updating $\mathcal{D}(f^i)$ at each new hierarchy level i. To evaluate quantitatively the relevance of this estimation $\mathcal{D}(f^i)\big)$, we measure its similarity to the final, exact result $\mathcal{D}(f)$ with the *Wasserstein distance*, an established practical metric inspired by optimal transport [98], [99]. Intuitively, this distance aims at optimizing a matching between the features of two diagrams to compare and penalizes *mismatches* between these diagrams. Given two diagrams $\mathcal{D}(f)$ and $\mathcal{D}(g)$, a pointwise distance d_q, inspired from the L^p norm, can be introduced in the 2D birth/death space between two points $a=(x_a,y_a)\in\mathcal{D}(f)$ and $b=(x_b,y_b)\in\mathcal{D}(g)$, with $q>0$, as:

$$d_q(a,b)=(|x_b-x_a|^q+|y_b-y_a|^q)^{1/q}=\|a-b\|_q \tag{4}$$

By convention, $d_q(a,b)$ is set to zero if both a and b exactly lie on the diagonal ($x_a=y_a$ and $x_b=y_b$). The L_q-Wasserstein distance, noted W_q, between $\mathcal{D}(f)$ and $\mathcal{D}(g)$ can then be introduced as:

$$W_q\big(\mathcal{D}(f),\mathcal{D}(g)\big)=\min_{\phi\in\Phi}\left(\sum_{a\in\mathcal{D}(f)} d_q\big(a,\phi(a)\big)^q\right)^{1/q} \tag{5}$$

where Φ is the set of all possible assignments ϕ mapping each point $a\in\mathcal{D}(f)$ to a point $b\in\mathcal{D}(g)$ or to its projection onto the diagonal. W_q can be computed via assignment optimization, for which exact [100] and approximate [101], [102] implementations are publicly available [95].

For each level i, we measure the L_2-Wasserstein distance $W_2\big(\mathcal{D}(f),\mathcal{D}(f^i)\big)$. We normalize this distance by dividing it by $W_2\big(\mathcal{D}(f),\emptyset\big)$. Then, along the hierarchy $\mathcal{H}$, this normalized distance progresses from 1 to 0 for all datasets. Although this distance may increase in theory from one level to the next, Fig. 12 shows that it is monotonically decreasing for our datasets (see the appendix for similar convergence curves on an extended collection of datasets, including two examples containing a minor oscillation at the beginning of the computation). This shows that in practice, the accuracy of our progressive outputs indeed improves over time. This empirical convergence evaluation gives a global picture of the quality of our progressive data representation. To further evaluate its relevance, we report in Fig. 13 the ratio of captured *significant pairs* in the diagram $\mathcal{D}(f^i)$ as a function of the computation time. To evaluate this ratio, we select the *significant* pairs of $\mathcal{D}(f)$, i.e. with a relative persistence greater than 0.1. Let n_p be the number of such significant pairs (reported for each dataset in the legend of Fig. 13, right, along with its percentage over the total number of pairs in $\mathcal{D}(f)$, in parenthesis). Next, we select the n_p most persistent pairs in $\mathcal{D}(f^i)$ and divide the resulting number of selected pairs, noted $n_p^i\leq n_p$, by n_p. In short, this indicator helps appreciate the number of significant features captured by the hierarchy early in the computation. In particular, Fig. 13 shows that for most of the datasets, the number of captured significant pairs matches the final estimation as of 10% of the computation time. Fig. 14 reports the average persistence of the n_p^i significant pairs in $\mathcal{D}(f^i)$ as a function of the computation time, relatively to the average persistence of the n_p significant pairs in $\mathcal{D}(f)$. This indicator helps appreciate how well the significant pairs are captured in the data hierarchy. In particular, this figure shows a clear global trend across datasets: the persistence of the significant pairs tends to be underestimated early in the computation and this estimation improves over time. These quantitative observations (early capture of the significant pairs and underestimation of persistence at the beginning of the computation) can be visually observed in Fig. 11, which shows that the significant pairs are captured early in the data hierarchy (red and yellow pairs) but that their persistence is indeed underestimated: the corresponding points are initially close to the diagonal in the corresponding diagrams and then, they progressively move away from it.

Next, we evaluate the computational relevance of our progressive data representation, by reporting the number of Topologically Invariant (TI) vertices (Sec. 3.3), for which no computation is needed. Table 1 shows that for real-world datasets, TI vertices represent 72% of the data on average, which indicates that efficient update mechanisms can indeed be derived from our progressive data representation. This table also includes the memory overhead induced in progressive mode by the data structures employed by our topological analysis algorithms (Sec. 2.2 and Sec. 2.3). In particular, this overhead is estimated by measuring the memory footprint of all the data-structures which are present in our progressive algorithms (Sec. 4 and Sec. 5) but *not* present in the TTK implementation of the state-of-the-art methods. Thus, this column depicts the additional memory needed by our approach in comparison to the standard procedures available in TTK. In particular, this column shows a linear evolution of this memory overhead with the size of the data hierarchy. Note that our implementation is not optimized for memory usage and that important gains can be expected by re-engineering our data structures at a low level.

6.2 Time Performance

The time complexity of our progressive algorithm for critical point extraction is linear with the number of input vertices, which results

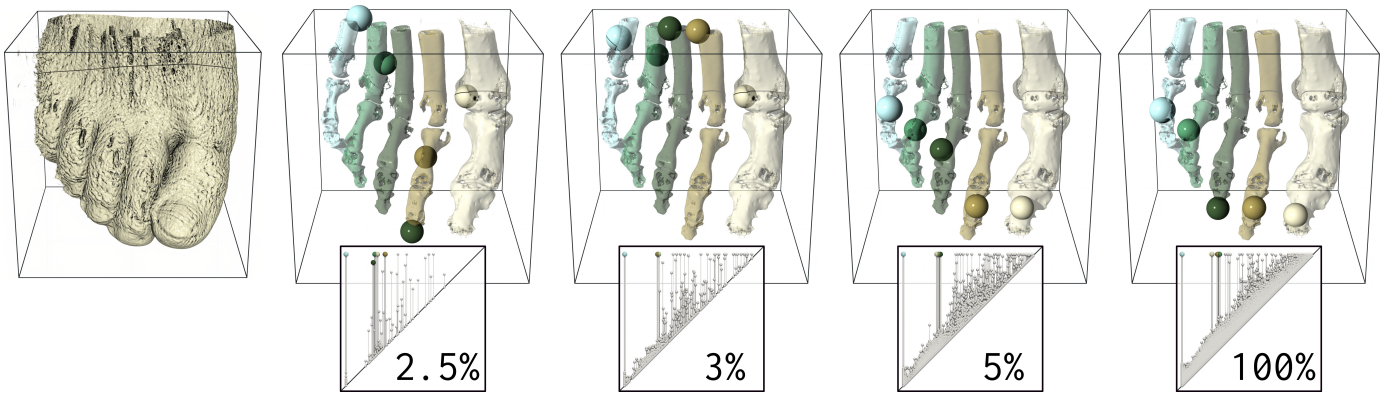

Fig. 15. Progressive persistence diagrams (saddle-maximum pairs, from left to right) of the CT scan of a foot (leftmost: isosurface), at a few steps of the computation. A merge tree based segmentation (colored regions, computed with TTK [95]) reveals the 5 most persistence structures in the data. Colored spheres show the 5 most persistent maxima reported by the current diagram estimation, illustrating a correct capture of the main structures early in the computation (as of 3% of computation).

Dataset	$\sum_{i=0}^{i=h}\|\mathcal{M}_0^i\|$	h	M. (Mb)	# old TIs	# new TIs	Total TIs	% **TIs**
AT	931,110	9	242	93,145	509,504	602,649	**64.7%**
SeaSurfaceHeight (2D)	1,384,626	11	240	241,264	608,460	849,724	**61.4%**
EthaneDiol	2,057,388	9	527	227,376	1,371,537	1,598,913	**77.7%**
Hydrogen	2,413,532	8	626	245,541	1,528,607	1,774,148	**73.5%**
Isabel	3,605,604	9	970	274,590	1,329,116	1,603,706	**44.5%**
Combustion	4,378,386	9	1,149	421,208	2,445,192	2,866,400	**65.5%**
Boat	4,821,326	9	1,221	575,646	3,690,624	4,266,270	**88.5%**
Random	18,117,518	9	5,389	169,603	54	169,657	**0.9%**
MinMax	18,994,899	9	4,742	2,394,619	16,474,429	18,869,048	**99.3%**
Aneurism	19,240,277	9	4,841	2,367,190	16,027,209	18,394,399	**95.6%**
Foot	19,240,277	9	5,109	1,648,823	10,746,624	12,395,447	**64.4%**
Heptane	31,580,914	10	8,117	3,453,180	22,512,477	25,965,657	**82.2%**

TABLE 1
Statistics of our progressive data hierarchy. From left to right: number of vertices, number of levels, memory overhead (over TTK), and number of topologically invariant (TI) vertices (Sec. 3.3) in the data hierarchy. For real-world datasets (Random and MinMax excluded), topologically invariant vertices represent 72% of the data on average.

	Critical Points				Persistence Diagram			
Dataset	TTK [41]	NP	Prog	Speedup	TTK [23]	NP	Prog	Speedup
AT	4.41	0.34	0.25	1.36	0.66	0.31	0.24	1.29
SeaSurfaceHeight (2D)	0.92	0.16	0.26	0.62	0.70	0.24	0.40	0.60
EthaneDiol	9.56	0.73	0.42	1.74	1.45	0.68	0.41	1.66
Hydrogen	11.55	0.92	0.59	1.56	1.90	0.89	0.63	1.41
Isabel	17.98	1.40	1.43	0.98	2.76	1.50	1.62	0.93
Combustion	21.87	1.74	1.33	1.31	4.36	1.82	1.50	1.21
Boat	22.47	1.74	0.73	2.38	3.38	1.81	0.82	2.21
Random	113.12	13.99	21.04	0.66	74.39	25.65	34.40	0.75
MinMax	82.23	6.94	1.56	4.45	14.68	7.00	1.64	4.27
Aneurism	84.03	7.39	2.19	3.37	12.85	8.03	3.43	2.34
Foot	100.58	9.26	8.13	1.14	18.20	12.18	12.06	1.01
Heptane	149.30	12.43	6.46	1.92	19.45	13.22	8.16	1.62

TABLE 2
Sequential computation times (in seconds) of our algorithms for critical point extraction (left) and persistence diagram computation (right). The columns *TTK* report the run times of the default implementations provided by the Topology ToolKit [95]. The columns *NP* and *Prog* respectively report the timings for the non-progressive (directly initialized at the final hierarchy level) and progressive versions of our algorithms.

	Critical Points				Persistence Diagram			
Dataset	TTK [41]	NP	Prog	Speedup	TTK [23]	NP	Prog	Speedup
AT	0.54	0.06	0.05	1.20	0.29	0.06	0.06	1.00
SeaSurfaceHeight (2D)	0.15	0.04	0.05	0.80	0.28	0.07	0.11	0.64
EthaneDiol	1.18	0.12	0.07	1.71	0.42	0.13	0.11	1.18
Hydrogen	1.41	0.14	0.12	1.17	0.97	0.18	0.19	0.95
Isabel	2.11	0.21	0.21	1.00	0.89	0.25	0.29	0.86
Combustion	2.54	0.25	0.21	1.19	0.82	0.29	0.30	0.97
Boat	2.71	0.26	0.15	1.73	0.81	0.30	0.24	1.25
Random	11.46	1.82	2.67	0.68	19.10	8.96	10.89	0.82
MinMax	9.87	1.25	0.43	2.91	3.24	1.39	0.69	2.01
Aneurism	10.19	1.14	0.50	2.28	4.08	1.89	1.48	1.28
Foot	11.30	1.93	1.75	1.10	5.39	3.37	3.29	1.02
Heptane	17.72	2.38	1.51	1.58	5.74	2.79	2.50	1.12

TABLE 3
Parallel computation times (in seconds, 8 cores) of our algorithms for critical point extraction (left) and persistence diagram computation (right). The columns *TTK* report the run times of the default implementations provided by the Topology ToolKit [95]. The columns *NP* and *Prog* respectively report the timings for the non-progressive (directly initialized at the final hierarchy level) and progressive versions of our algorithms.

in our hierarchical setup in $\mathcal{O}(\sum_{i=0}^{i=h}|\mathcal{M}_0^i|)$ steps. For persistence diagrams, in the worst possible configuration (degenerate saddles with systematic integral line forking), each saddle would generate monotonic paths which would hit every minimum. This would yield $n_s \times (n_m - 1)$ critical triplets, where n_s and n_m stand for the number of saddles and minima of f. This would yield $n_s \times (n_m - 1)$ merge events for the critical pairing step, each with an amortized complexity of $\mathcal{O}\big(\alpha(n_m)\big)$, where α is the inverse of the Ackermann function. However, such configurations are extremely rare in practice and most saddles only yield one triplet, resulting in an overall practical time complexity of $\mathcal{O}\big(\sum_{i=0}^{i=h}(|\mathcal{M}_1^i| + n_s^i \log n_s^i + n_s^i \alpha(n_m^i))\big)$ steps, also accounting for the sorting of triplets.

Table 2 reports computation times (sequential run) for the default algorithms (Sec. 2.2) [41], [23] available in TTK [95] and the non-progressive and progressive versions of our algorithms. Non-progressive methods (*TTK* and *NP* columns) compute from scratch only the last hierarchy level h directly. We only report the run times of TTK as an indicative baseline as the differences in triangulation implementations already induce alone important run time variations (TTK emulates implicitly triangulations for regular grids *at query time*, while our implementation stores the explicit list of link edges for each vertex, Sec. 4.1). Interestingly, the *Speedup* columns show that, in addition to their ability to provide continuous visual feedback, our progressive algorithms are also faster than their non-progressive versions (on average, 1.8 times faster for critical points, 1.6 for persistence diagrams). These speedups confirm that the overhead of processing an entire hierarchy ($\sum_{i=0}^{i=h}|\mathcal{M}_0^i|$ vertices in progressive mode, instead of $|\mathcal{M}_0^h|$ in non-progressive mode) and of detecting TI vertices is largely compensated by the gains these vertices provide. Note that the datasets with the most (resp. least) TI vertices (Tab. 1) are also those for which the largest (resp. smallest) speedups are obtained, confirming the importance of TI vertices in the computation.

Table 3 details the performance of the shared-memory parallelization of our progressive algorithms, using OpenMP [103], again in comparison to the default algorithms (Sec. 2.2) [41], [23] available in TTK [95] and to the non-progressive version of our algorithms. As mentioned in Sec. 4.3, critical point extraction can

be trivially parallelized over vertices, for each of the four steps of our algorithm, resulting in an average parallel efficiency of 66%. The persistence diagram computation results in a more modest efficiency (45%) as monotonic path computations are subject to locks, in addition to be possibly imbalanced.

6.3 Stress Cases

Our experiments include two synthetic datasets, whose purpose is to illustrate the most and the least favorable configurations for our approach, to better appreciate the dependence of our algorithms to their inputs. The *MinMax* dataset is an elevation field which only contains one global minimum and one global maximum. It exhibits therefore a lot of regularity. In contrast, the *Random* dataset assigns a random value to each vertex. Thus, no local coherency can be expected between consecutive levels in the data hierarchy (which is an important hypothesis in our framework).

Table 1 confirms the best/worst case aspect of these datasets, as they respectively maximize and minimize the ratio of TI vertices: *MinMax* has nearly only TI vertices (99.3%) while *Random* has nearly none (0.9%).

Table 2 confirms, as can be expected, that these two datasets also maximize and minimize the speedup induced by our progressive approach. In particular, our progressive algorithms report a speedup greater than 4 over their non-progressive versions for the *MinMax* dataset. This further confirms the observation made in Sec. 6.2 that processing an entire data hierarchy with the acceleration induced by TI vertices can indeed be faster than computing criticality from scratch at the final hierarchy level only (in particular, up to 4 times). In contrast, this table also shows that in the worst possible case (*Random*, nearly no TI vertices), the processing of the entire hierarchy can be up to 50% slower (for critical points, 30% for persistence diagrams) than computing in non-progressive mode at the final hierarchy level only. All the other datasets exhibit speedups included within these lower (*Random*) and upper (*MinMax*) bounds (on average 1.8 for critical points, 1.6 for persistence diagrams).

In terms of quality, the best/worst case aspect of *MinMax* and *Random* is also illustrated in Figs. 12, 13 and 14, where *MinMax* converges immediately, while *Random* describes the worst case (slow convergence, slow and inaccurate capture of the significant pairs). In these curves, the other datasets cover the span of possible behaviors between these two extreme cases.

6.4 Progressive Topological Visualization and Analysis

This section discusses the progressive visualizations and analyses enabled by our approach. Fig. 1 presents a typical example of progressive persistence diagram computation on the electron density of the adenine-thymine (AT) molecular system. In this figure, the estimated diagrams progressively capture the features in a meaningful way, as the heaviest atoms are captured first and the lighest ones last. In particular, in the diagrams, the introduced points progressively stand out from the diagonal towards their final locations. As of 33% of the computation, the diagram is complete and its accuracy is further improved over time. This illustrates the capacity of our approach to deliver relevant previews of the topological features of a dataset and to improve them progressively. Fig. 11 further illustrates our estimation of the lifetime of extrema and their trajectory in the data hierarchy. There, as one progresses down the hierarchy, the prominent maxima are progressively captured and they quickly stabilize in the vicinity of their final location. Fig. 15 illustrates progressive persistence diagrams for an acquired dataset. There, a merge tree based segmentation (computed with TTK [95]) is shown in the background. It represents the regions of the five most persistent leaf arcs of the merge tree. The five most persistent maxima reported by the current diagram estimation are reported with spheres. As of 3% of the computation, these maxima are correctly assigned to the final structures (one per toe), while their positional accuracy is further improved with time. Overall, the diagrams (bottom) capture the main features early in the computation, while smaller features and noise are progressively captured as the computation unfolds. Fig. 16 presents a gallery of progressive persistence diagrams for several datasets. The diagram estimations capture well the overall shape of the final, exact output (i.e. the number and salience of its main features) and are progressively refined over time. This gallery complements the quantitative analysis reported in Figs. 12, 13, 14 and confirms visually the interest of our progressive representations, which provide relevant previews of the topological features present in a dataset.

We used the TTK library [95] to integrate our implementation within the popular visualization system ParaView [104], as shown in the companion video (supplemental material). This video illustrates the progressive updates of our topological previews within interactive times, and further demonstrates their interest for interactive visualization environments.

Fig. 17 illustrates the interest of our progressive representations for the control of the run time of batch-mode analysis pipelines. We consider the Isabel ensemble [97], [106] (12 members illustrating 3 hurricane behaviors: formation, drift and landfall, Fig. 17, left to right). Our progressive algorithm is used to generate a persistence diagram for each member, and is interrupted at a predefined threshold of computation time (Fig. 17, top). Then, the TTK implementation of the algorithm by Vidal et al. [105] is used to cluster these diagrams (with a time constraint of one second). Overall, this results in a topological clustering pipeline whose time execution is fully controlled, from the feature extraction to their clustering. For reasonable computation thresholds (5% in Fig. 17), this pipeline returns the same, correct classification (one color per cluster) as the one returned with the exact diagrams (bottom). This demonstrates that the main trends of an ensemble in terms of features (the main clusters) can still be estimated reliably, while additionally controlling the execution time of the clustering pipeline.

6.5 Limitations and Discussion

Our progressive persistence diagrams tend in practice to capture the main features first. However, this cannot be guaranteed theoretically. For instance, sharp spikes in the data (e.g. high amplitude *and* high frequency noise) can yield persistent maxima only at the last levels of the hierarchy, as illustrated in Fig. 16 where the global maximum of the *Hydrogen* dataset (fourth row) belongs to a sharp spike in the center of the data (as also reported by the quantitative plots Figs. 12 and 14). This behavior prevents the definition of theoretical error bounds on our estimations. However, the empirical monotonic decrease of the Wasserstein distance (Fig. 12) indicates that our progressive representations actually provide reliable estimations, as confirmed by the indicators of Figs. 13, 14, where the real-world datasets cover the span of possible behaviors between the two stress cases (*MinMax*, *Random*). This can be explained by the fact that, in practice, persistent pairs often coincide with large features in the domain, which get captured early in the data hierarchy.

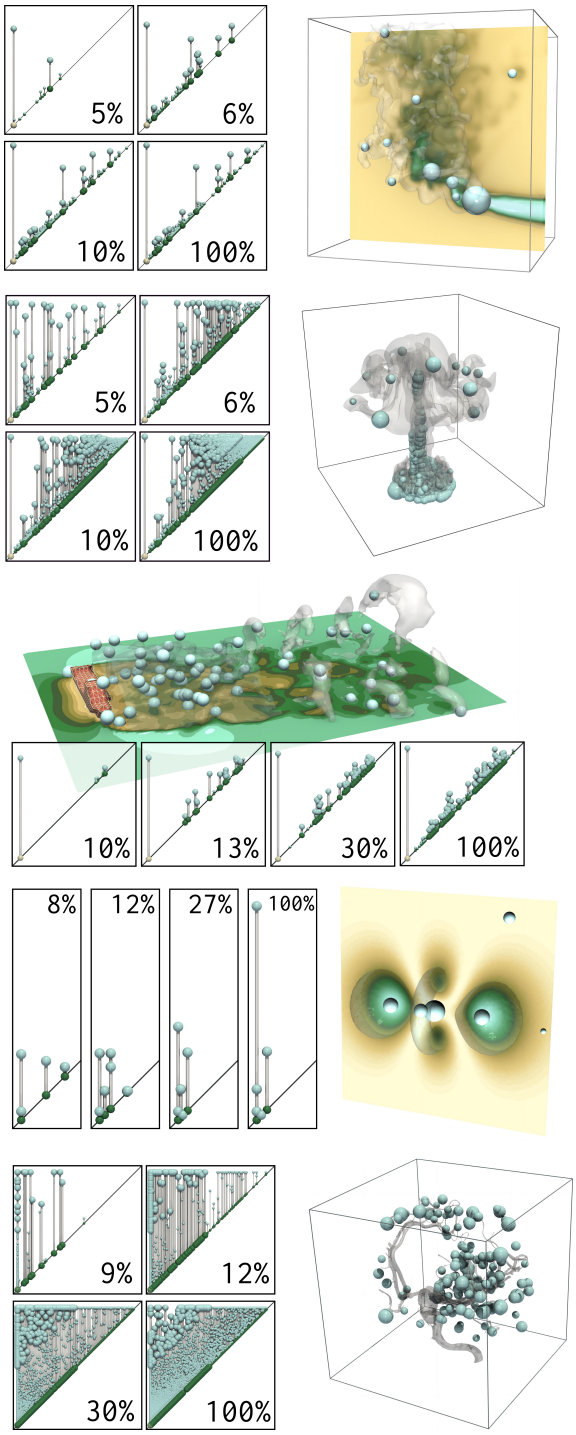

Fig. 16. Progressive persistence diagrams (saddle-maximum pairs) for several data sets (combustion, heptane, boat, hydrogen, aneurism), at a few steps of the computation. Persistent maxima are represented with spheres in the domain (scaled by persistence). The progressive diagrams capture well the overall shape (number and salience of features) of the final, exact output (100%) early in the computation and refine it over time.

Although we described our approach generically, we focused in this paper on an efficient implementation of edge-nested triangulations for regular grids (Sec. 3.2). The generalization of our approach to generic domains requires to investigate triangulation subdivision schemes. Several of them seem compliant with the notion of edge-nested triangulation (Sec. 3.1), such as the Loop subdivision

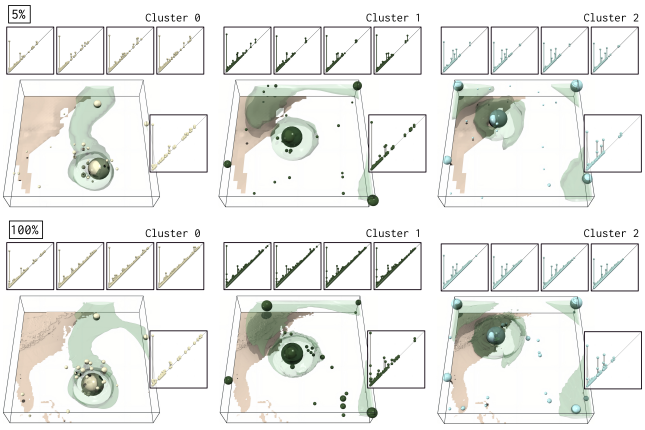

Fig. 17. Topological clustering of the Isabel ensemble dataset. Progressive persistence diagrams (top, interrupted at 5% of computation) are used as an input to the clustering approach by Vidal et al. [105] (constrained to 1 second of computation). This time-constrained ensemble clustering yields the same, correct classification (one color per cluster, from left to right) as the one returned with the exact diagrams (bottom).

[81] and the *red* triangulation refinement [79], [80], [82], [83]. However, efficiently transforming an arbitrary triangulation into a triangulation which admits an edge-nested hierarchy is an orthogonal question which we leave for future work. Similarly, the reliable tracking of extrema through the hierarchy (for lifetime estimation, Sec. 4.4) relates to another orthogonal problem, for which computationally expensive optimizations may need to be considered. Our algorithms complete a given hierarchical level before moving on to the next one. This results in increasing update times as the computation converges. In the future, finer update strategies will be considered, by considering adaptive, feature-centric, variable level-of-detail refinement methods. Finally, our algorithm for persistence diagrams does not support saddle-saddle pairs in 3D. However, from our experience, the interpretation of these structures is not obvious in the applications.

Our progressive scheme seems to be particularly efficient for algorithms which visit *all* the vertices of the domain (e.g. critical point extraction), but less beneficial for inexpensive operations which only visit small portions of the data (e.g. integral line computation, Sec. 5.2). This is a lesson learned from our experiments which could serve as guideline for future extensions to other topological analyis algorithms. Also, there is a trade off between the benefits of the progressive scheme and its cost in terms of memory usage. Future work is needed to improve the memory footprint of our approach by optimizing our data structures at a low level. For instance, for triangulations of regular grids and real-life tetrahedral meshes, the maximum number of neighbors around a vertex is typically small, which enables the encoding of local neighbor identifiers with very few bits, instead of full integers (as done in our current implementation). Other variables (such as the polarity, currently stored with a boolean for each neighbor) could also benefit from a more compact bit representation.

7 Conclusion

This paper introduced an approach for the progressive topological analysis of scalar data. Our work is based on a hierarchical representation of the input data and the fast identification of *topologically invariant vertices*, for which we showed that no computation was required as they were introduced in the hierarchy. This enables the

definition of efficient coarse-to-fine topological algorithms, capable of providing interpretable outputs upon interruption requests, and of progressively refining them otherwise until the final, exact output. We instantiated our approach with two examples of topological algorithms (critical point extraction and persistence diagram computation), which leverage efficient update mechanisms for ordinary vertices and avoid computation for the topologically invariant ones. For real-life datasets, our algorithms tend to first capture the most important features of the data and to progressively refine their estimations with time. This is confirmed quantitatively with the empirical convergence of the Wasserstein distance to the final, exact output, which is monotonically decreasing. More computation time indeed results in more accuracy. Our experiments also reveal that our progressive computations even turn out to be faster overall than non-progressive algorithms and that they can be further accelerated with shared-memory parallelism. We showed the interest of our approach for interactive data exploration, where our algorithms provide progressive previews, continuously refined over time, of the topological features found in a dataset. We also showed that in batch-mode, our approach enables to control the run time of a complete TDA pipeline (topological clustering of ensemble data).

We believe our work sets the foundations for several exciting research avenues for future work. First, we identified several improvement directions regarding the management of the input hierarchy, including the extensions to arbitrary triangulations or the addition of intermediate hierarchy levels. Second, since our progressive algorithms can be given a time-budget constraint and still be resumed afterwards if needed, we would like to investigate in the future how such preemptable data analysis algorithms can help for optimizing scheduling policies in high-performance environments, where data analysis is often run at the same time as data production and where the allocation of computation resources need to be finely optimized. Third, we believe our approach can be generalized to higher dimensions (with tailored sampling methods) as well as to other topological abstractions, in order to re-visit the entire TDA arsenal (merge trees, Reeb graphs, Morse-Smale complexes) in the light of progressivity. In that perspective, the generalization of topologically invariant vertices to Discrete Morse Theory [66] looks particularly promising.

Acknowledgments

We would like to thank the reviewers for their thoughtful remarks and suggestions. This work is partially supported by the European Commission grant H2020-FETHPC-2017 *"VESTEC"* (ref. 800904).

References

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

[2] 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," *Comp. Grap. For.*, 2016.

[3] T. Sousbie, "The persistent cosmic web and its filamentary structure: Theory and implementations," *Royal Astronomical Society*, 2011, http://www2.iap.fr/users/sousbie/web/html/indexd41d.html.

[4] 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 Transactions on Visualization and Computer Graphics*, 2016, http://vgl.serc.iisc.ernet.in/felix/index.html.

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

[6] A. Bock, H. Doraiswamy, A. Summers, and C. T. Silva, "TopoAngler: Interactive Topology-Based Extraction of Fishes," *IEEE Transactions on Visualization and Computer Graphics (Proc. of IEEE VIS)*, 2018.

[7] K. Anderson, J. Anderson, S. Palande, and B. Wang, "Topological data analysis of functional MRI connectivity in time and space domains," in *MICCAI Workshop on Connectomics in NeuroImaging*, 2018.

[8] 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. of Computational Chemistry*, 2018.

[9] D. Guenther, R. Alvarez-Boto, J. Contreras-Garcia, J.-P. Piquemal, and J. Tierny, "Characterizing molecular interactions in chemical systems," *IEEE Transactions on Visualization and Computer Graphics (Proc. of IEEE VIS)*, 2014.

[10] 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.

[11] J. Kasten, J. Reininghaus, I. Hotz, and H. Hege, "Two-dimensional time-dependent vortex regions based on the acceleration magnitude," *IEEE Transactions on Visualization and Computer Graphics*, 2011.

[12] A. Gyulassy, M. A. Duchaineau, V. Natarajan, V. Pascucci, E. Bringa, A. Higginbotham, and B. Hamann, "Topologically clean distance fields," *IEEE Transactions on Visualization and Computer Graphics (Proc. of IEEE VIS)*, 2007.

[13] A. Gyulassy, A. Knoll, K. Lau, B. Wang, P. Bremer, M. Papka, L. A. Curtiss, and V. Pascucci, "Interstitial and interlayer ion diffusion geometry extraction in graphitic nanosphere battery materials," *IEEE Transactions on Visualization and Computer Graphics (Proc. of IEEE VIS)*, 2015.

[14] 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 Symposium on Large Data Analysis and Visualization*, 2019.

[15] D. E. Laney, P. Bremer, A. Mascarenhas, P. Miller, and V. Pascucci, "Understanding the structure of the turbulent mixing layer in hydrodynamic instabilities," *IEEE Transactions on Visualization and Computer Graphics (Proc. of IEEE VIS)*, 2006.

[16] 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 Transactions on Visualization and Computer Graphics*, 2011.

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

[18] R. L. Boyell and H. Ruston, "Hybrid techniques for real-time radar simulation," in *Proc. of the IEEE Fall Joint Computer Conference*, 1963.

[19] M. De Berg and M. van Kreveld, "Trekking in the alps without freezing or getting tired," *Algorithmica*, 1997.

[20] S. Tarasov and M. Vyali, "Construction of contour trees in 3d in o(n log n) steps," in *S. o. C. G.*, 1998.

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

[22] D. Smirnov and D. Morozov, "Triplet Merge Trees," in *TopoInVis*, 2017.

[23] C. Gueunet, P. Fortin, J. Jomier, and J. Tierny, "Task-Based Augmented Contour Trees with Fibonacci Heaps," *IEEE Trans. Parallel Distrib. Syst.*, 2019.

[24] 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*, vol. 222, no. 847-849, p. 76, 1946.

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

[26] V. Pascucci, G. Scorzelli, P. T. Bremer, and A. Mascarenhas, "Robust on-line computation of Reeb graphs: simplicity and speed," *ACM Trans. on Graph.*, 2007.

[27] L. De Floriani, U. Fugacci, F. Iuricich, and P. Magillo, "Morse complexes for shape segmentation and homological analysis: discrete models and algorithms," *Comp. Grap. For.*, 2015.

[28] A. Gyulassy, P. T. Bremer, B. Hamann, and V. Pascucci, "A practical approach to Morse-Smale complex computation: Scalability and generality," *IEEE Transactions on Visualization and Computer Graphics (Proc. of IEEE VIS)*, 2008.

[29] 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.

[30] A. Gyulassy, P. Bremer, and V. Pascucci, "Shared-Memory Parallel Computation of Morse-Smale Complexes with Improved Accuracy," *IEEE Transactions on Visualization and Computer Graphics (Proc. of IEEE VIS)*, 2018.

[31] H. Edelsbrunner, D. Letscher, and A. Zomorodian, "Topological persistence and simplification," *Disc. Compu. Geom.*, 2002.

[32] M. Carrière, M. Cuturi, and S. Oudot, "Sliced Wasserstein Kernel for Persistence Diagrams," *ICML*, 2017.

[33] J. Reininghaus, S. Huber, U. Bauer, and R. Kwitt, "A stable multi-scale kernel for topological machine learning," in *IEEE CVPR*, 2015.

[34] B. Rieck, F. Sadlo, and H. Leitte, "Topological machine learning with persistence indicator functions," in *Proc. of TopoInVis*, 2017.

[35] R. B. Miller, "Response time in man-computer conversational transactions," in *Fall Joint Computer Conference*, 1968.

[36] M. Williams and T. Munzner, "Steerable, Progressive Multidimensional Scaling," in *Proc. of IEEE InfoVis*, 2004.

[37] J. Fekete and R. Primet, "Progressive Analytics: A Computation Paradigm for Exploratory Data Analysis," *arXiv*, 2016. [Online]. Available: http://arxiv.org/abs/1607.05162

[38] E. Zgraggen, A. Galakatos, A. Crotty, J. Fekete, and T. Kraska, "How progressive visualizations affect exploratory analysis," *IEEE Transactions on Visualization and Computer Graphics*, 2017.

[39] J. Jo, J. Seo, and J. Fekete, "PANENE: A Progressive Algorithm for Indexing and Querying Approximate k-Nearest Neighbors," *IEEE Transactions on Visualization and Computer Graphics*, 2020.

[40] J. Milnor, *Morse Theory*. Princeton University Press, 1963.

[41] T. F. Banchoff, "Critical points and curvature for embedded polyhedral surfaces," *The American Mathematical Monthly*, 1970.

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

[43] F. Chazal and S. Oudot, "Towards persistence-based reconstruction in euclidean spaces," in *S. o. C. G.*, 2008.

[44] U. Bauer, "Ripser: efficient computation of Vietoris-Rips persistence barcodes," 2019.

[45] D. Cohen-Steiner, H. Edelsbrunner, and J. Harer, "Stability of persistence diagrams," in *S. o. C. G.*, 2005.

[46] V. Pascucci and K. Cole-McLaughlin, "Parallel Computation of the Topology of Level Sets," *Algorithmica*, 2004.

[47] S. Maadasamy, H. Doraiswamy, and V. Natarajan, "A hybrid parallel algorithm for computing and tracking level set topology," in *Proc. of HiPC*, 2012.

[48] D. Morozov and G. H. Weber, "Distributed contour trees," in *Topological Methods in Data Analysis and Visualization III, Theory, Algorithms, and Applications*, 2014.

[49] A. Acharya and V. Natarajan, "A parallel and memory efficient algorithm for constructing the contour tree," in *Proc. of IEEE PacificVis*, 2015.

[50] H. A. Carr, G. H. Weber, C. M. Sewell, and J. P. Ahrens, "Parallel peak pruning for scalable SMP contour tree computation," in *IEEE Symposium on Large Data Analysis and Visualization*, 2016.

[51] C. Gueunet, P. Fortin, J. Jomier, and J. Tierny, "Task-based Augmented Merge Trees with Fibonacci Heaps,," in *IEEE LDAV*, 2017.

[52] Y. Shinagawa, T. L. Kunii, and Y. L. Kergosien, "Surface coding based on morse theory," *IEEE Computer Graphics and Applications*, 1991.

[53] S. Biasotti, B. Falcidieno, and M. Spagnuolo, "Extended Reeb Graphs for Surface Understanding and Description," in *Discrete Geometry for Computer Imagery*, 2000.

[54] Z. J. Wood, H. Hoppe, M. Desbrun, and P. Schröder, "Removing excess topology from isosurfaces," *ACM Trans. on Graph.*, 2004.

[55] G. Patanè, M. Spagnuolo, and B. Falcidieno, "Reeb graph computation based on a minimal contouring," in *Shape Modeling International*, 2008.

[56] H. Doraiswamy and V. Natarajan, "Output-Sensitive Construction of Reeb Graphs," *IEEE Transactions on Visualization and Computer Graphics*, 2012.

[57] J. Tierny, A. Gyulassy, E. Simon, and V. Pascucci, "Loop surgery for volumetric meshes: Reeb graphs reduced to contour trees," *IEEE Transactions on Visualization and Computer Graphics (Proc. of IEEE VIS)*, 2009.

[58] K. Cole-McLaughlin, H. Edelsbrunner, J. Harer, V. Natarajan, and V. Pascucci, "Loops in Reeb graphs of 2-manifolds," in *S. o. C. G.*, 2003.

[59] S. Parsa, "A deterministic *o(m log m)* time algorithm for the reeb graph," in *S. o. C. G.*, 2012.

[60] C. Gueunet, P. Fortin, J. Jomier, and J. Tierny, "Task-based Augmented Reeb Graphs with Dynamic ST-Trees," in *Eurographics Symposium on Parallel Graphics and Visualization*, 2019.

[61] H. Edelsbrunner, J. Harer, and A. K. Patel, "Reeb spaces of piecewise linear mappings," in *S. o. C. G.*, 2008.

[62] H. A. Carr and D. J. Duke, "Joint Contour Nets," *IEEE Transactions on Visualization and Computer Graphics*, 2014.

[63] J. Tierny and H. A. Carr, "Jacobi Fiber Surfaces for Bivariate Reeb Space Computation," *IEEE Transactions on Visualization and Computer Graphics (Proc. of IEEE VIS)*, 2016.

[64] H. Edelsbrunner, J. Harer, and A. Zomorodian, "Hierarchical morse complexes for piecewise linear 2-manifolds," in *S. o. C. G.*, 2001.

[65] H. Edelsbrunner, J. Harer, V. Natarajan, and V. Pascucci, "Morse-smale complexes for piecewise linear 3-manifolds," in *S. o. C. G.*, 2003.

[66] R. Forman, "A User's Guide to Discrete Morse Theory," *Advances in Mathematics*, 1998.

[67] N. Shivashankar and V. Natarajan, "Parallel Computation of 3D Morse-Smale Complexes," *Comp. Graph. For.*, 2012.

[68] M. Hilaga, Y. Shinagawa, T. Komura, and T. L. Kunii, "Topology matching for fully automatic similarity estimation of 3d shapes," in *Proc. of ACM SIGGRAPH*, 2001.

[69] V. Pascucci, K. Cole-McLaughlin, and G. Scorzelli, "Multi-resolution computation and presentation of contour trees," in *Proc. IASTED conference on visualization, imaging, and image processing*, 2004.

[70] P. Bremer, H. Edelsbrunner, B. Hamann, and V. Pascucci, "A Multi-Resolution Data Structure for 2-Dimensional Morse Functions," in *Proc. of IEEE VIS*, 2003.

[71] D. Guenther, J. Reininghaus, S. Prohaska, T. Weinkauf, and H.-C. Hege, "Efficient computation of a hierarchy of discrete 3d gradient vector fields," in *Proc. of TopoInVis*, 2012, pp. 15–29.

[72] F. Iuricich and L. D. Floriani, "Hierarchical forman triangulation: A multiscale model for scalar field analysis," *Comput. Graph.*, vol. 66, pp. 113–123, 2017.

[73] B. F. Gregorski, M. A. Duchaineau, P. Lindstrom, V. Pascucci, and K. I. Joy, "Interactive view-dependent rendering of large isosurfaces," in *Proc. of IEEE VIS*, 2002, pp. 475–482.

[74] K. Weiss and L. D. Floriani, "Supercubes: A high-level primitive for diamond hierarchies," *IEEE Transactions on Visualization and Computer Graphics (Proc. of IEEE VIS)*, vol. 15, no. 6, pp. 1603–1610, 2009.

[75] ——, "Diamond hierarchies of arbitrary dimension," *Comput. Graph. Forum*, vol. 28, no. 5, pp. 1289–1300, 2009.

[76] V. Pascucci and C. L. Bajaj, "Time critical isosurface refinement and smoothing," in *Proc. of the Volume Visualization and Graphics Symposium*, 2000, pp. 33–42.

[77] T. Lewiner, L. Velho, H. Lopes, and V. Mello, "Hierarchical isocontours extraction and compression," in *SIBGRAPHI*, 2004, pp. 234–241.

[78] T. Gerstner and R. Pajarola, "Topology preserving and controlled topology simplifying multiresolution isosurface extraction," in *Proc. of IEEE VIS*, 2000, pp. 259–266.

[79] H. Freudenthal, "Simplizialzerlegungen von beschrankter Flachheit," *Annals of Mathematics*, vol. 43, pp. 580–582, 1942.

[80] R.E. Bank, and A.H. Sherman, and A. Weiser, "Refinement algorithms and data structures for regular local mesh refinement," *Scientific Computing*, pp. 3–17, 1983.

[81] C. Loop, "Smooth Subdivision Surfaces Based on Triangles," Master's thesis, University of Utah, 1987.

[82] J. Bey, "Tetrahedral grid refinement," *Computing*, vol. 55, pp. 355–378, 1995.

[83] S. Zhang, "Successive subdivision of tetrahedra and multigrid methods on tetrahedral meshes," *Houston Journal of Mathematics*, vol. 21, pp. 541–556, 1995.

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

[85] J. Bey, "Simplicial grid refinement: on Freudenthal's algorithm and the optimal number of congruence classes," *Numer. Math.*, vol. 85, pp. 1–29, 1998.

[86] H. Edelsbrunner and E. P. Mucke, "Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms," *ACM Trans. on Graph.*, 1990.

[87] D. Sleator and R. Tarjan, "A data structure for dynamic trees," *Journal of Computer and System Sciences*, 1983.

[88] G. Ji and H.-W. Shen, "Feature Tracking using Earth Mover's Distance and Global Optimization," in *Proc. of IEEE PacificVis*, 2006.

[89] T. Klein and T. Ertl, "Scale-Space Tracking of Critical Points in 3D Vector Fields," in *Topology-based Methods in Visualization*, ser. Mathematics and Visualization. Springer, 2007.

[90] J. Reininghaus, J. Kasten, T. Weinkauf, and I. Hotz, "Efficient Computation of Combinatorial Feature Flow Fields," *IEEE Transactions on Visualization and Computer Graphics*, 2012.

[91] H. Saikia and T. Weinkauf, "Global Feature Tracking and Similarity Estimation in Time-Dependent Scalar Fields," *Comp. Graph. For.*, 2017.

[92] M. Soler, M. Plainchault, B. Conche, and J. Tierny, "Lifted Wasserstein matcher for fast and robust topology tracking," in *IEEE Symposium on Large Data Analysis and Visualization*, 2018.

[93] Y. Chiang, T. Lenz, X. Lu, and G. Rote, "Simple and optimal output-sensitive construction of contour trees using monotone paths," *Comput. Geom.*, 2005.
[94] J. Singler and B. Konsik, "The GNU libstdc++ Parallel Mode: Software Engineering Considerations," in *Proc. of International Workshop on Multicore Software Engineering*, 2008.
[95] J. Tierny, G. Favelier, J. A. Levine, C. Gueunet, and M. Michaux, "The Topology ToolKit," *IEEE Transactions on Visualization and Computer Graphics (Proc. of IEEE VIS)*, 2017, https://topology-tool-kit.github.io/.
[96] P. Klacansky, "Open Scientific Visualization Data Sets," https://klacansky.com/open-scivis-datasets/, 2020.
[97] TTK Contributors, "TTK Data," https://github.com/topology-tool-kit/ttk-data/tree/dev, 2020.
[98] L. Kantorovich, "On the translocation of masses," *AS URSS*, 1942.
[99] G. Monge, "Mémoire sur la théorie des déblais et des remblais," *Académie Royale des Sciences de Paris*, 1781.
[100] J. Munkres, "Algorithms for the assignment and transportation problems," *Journal of the Society for Industrial and Applied Mathematics*, 1957.
[101] D. P. Bertsekas, "A new algorithm for the assignment problem," *Mathematical Programming*, 1981.
[102] M. Kerber, D. Morozov, and A. Nigmetov, "Geometry helps to compare persistence diagrams," *ACM Journal of Experimental Algorithmics*, 2016.
[103] L. Dagum and R. Menon, "OpenMP: an industry standard API for shared-memory programming," *IEEE computational science and engineering*, vol. 5, no. 1, pp. 46–55, 1998.
[104] J. Ahrens, B. Geveci, and C. Law, "ParaView: An End-User Tool for Large-Data Visualization," *The Visualization Handbook*, pp. 717–731, 2005.
[105] J. Vidal, J. Budin, and J. Tierny, "Progressive wasserstein barycenters of persistence diagrams," *IEEE Transactions on Visualization and Computer Graphics (Proc. of IEEE VIS)*, 2019.
[106] I. SciVisContest, "Simulation of the isabel hurricane," http://sciviscontest-staging.ieeevis.org/2004/data.html, 2004.

Jules Vidal is a Ph.D. student at Sorbonne Université. 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.

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 project.

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 permanent scientist, affiliated with Sorbonne University (Paris, France). 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.