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, 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 instantiate our approach with two examples of topological algorithms (critical point extraction and persistence diagram computation), which generate exploitable 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 performances with regard to non-progressive algorithms and we describe further accelerations with shared-memory parallelism. We illustrate the utility of our approach in (i) batch-mode and (ii) interactive setups, where it respectively enables (i) the control of the execution time of complete topological pipelines as well as (ii) 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 admit 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.

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 exploitable 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 admit 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 can help optimize scheduling, as progressive tasks can be put on pause, and still provide an exploitable result which can be later refined upon computation resume. Progressivity also enables the assignment...
of a 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 (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, and that most of the points of a dataset do not imply topological events in general. 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. 2), where topologically invariant vertices (vertices of the input which do not produce topological events upon insertion, 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 instantiate 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 performances. 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 is 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. For instance, extrema are often associated in practice 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] established itself 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 the 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, which are often associated to features of interest in applications, can be computed more efficiently, with a Union-Find data structure [1, 22], 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 developed [44].

Although persistence diagrams are stable [45], their discriminative power may be insufficient for some applications. This motivates the design of more discriminative topological abstractions, such as the merge and contour trees, which respectively track the connected components of sub-level sets and level sets. 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, 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. 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 performances [26, 57], while algorithms with optimal time complexity have been introduced, first in 2D [53], 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 admit identical gradient integration extremities. 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 in 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]. 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 hierarchical 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 exploitable 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 exploitable 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 exploitable 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 even results in 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: [86]).
2 Preliminaries

This section briefly presents the technical background of our work. We refer the reader to the reference 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. \( f \) is assumed to be injective on the vertices of \( \mathcal{M} \). 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

The topological features of \( f \) can be tracked with the notion of sub-level set, noted \( f^{-1}(w) = \{ p \in \mathcal{M} \mid f(p) \leq w \} \). 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 \( f \) [49]. In the PL setting, Banchoff [49] introduced a local characterization of critical points, defined as follows.

The star of a vertex \( v \in \mathcal{M} \), noted \( \text{St}(v) \), is the set of its co-faces: \( \text{St}(v) = \{ \sigma \in \mathcal{M} \mid v \subset \sigma \} \). The link of \( v \), noted \( \text{Lk}(v) \), is the set of the faces \( \tau \) of the simplices \( \sigma \in \text{St}(v) \) with empty intersection with \( v: \text{Lk}(v) = \{ \tau \in \mathcal{M} \mid \tau \subset \sigma, \sigma \in \text{St}(v), \tau \cap v = \emptyset \} \). The lower link of \( v \), noted \( \text{Lk}^{-}(v) \), is given by the set of simplices of \( \text{Lk}(v) \) which only contain vertices lower than \( v: \text{Lk}^{-}(v) = \{ \sigma \in \text{Lk}(v) \mid \forall \nu \in \sigma, f(\nu) < f(v) \} \). The upper link is defined symmetrically: \( \text{Lk}^{+}(v) = \{ \sigma \in \text{Lk}(v) \mid \forall \nu \in \sigma, f(\nu) > f(v) \} \). A vertex \( v \) is regular if both \( \text{Lk}^{-}(v) \) and \( \text{Lk}^{+}(v) \) are simply connected.

For such vertices, the sub-level sets do not change their topology as they span \( \text{St}(v) \). Otherwise, \( v \) is a critical point. These can be classified with regard to their index \( \Sigma(v) \). It is equal to 0 for local minima (\( Lk^{-}(v) = \emptyset \)), to 1 for local maxima (\( Lk^{+}(v) = \emptyset \)) and otherwise to 1 for i-saddles (\( 0 < i < d \)).

Adjacency relations between critical points can be captured with the notion of integral line. Given a vertex \( v \), its forward integral line, noted \( Lk^{+}(v) \), is a path along the edges of \( \mathcal{M} \), initiated in \( v \), such that each edge of \( Lk^{+}(v) \) 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 \( Lk^{-}(v) \) 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 [9], [10] or clusters of galaxies in astrophysics [5], [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.

Persistence assesses the importance of a critical point, based on the lifetime of the topological feature it created in \( f^{-1}(w) \), as one continuously increase the isosurface \( w \). In particular, as \( w \) increases, new connected components of \( f^{-1}(w) \) 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 the local minima can be unambiguously paired following this strategy, while the global minimum is usually paired, by convention, with the global maximum. The symmetric reasoning can be applied in 3D to characterize, with 2-saddle/maximum pairs, the life time of the voids of \( f^{-1}(w) \), while the 1-saddle/2-saddle pairs characterize its independent cycles. In practical applications, 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 \( D(f) \), which embeds each pair \((c, c')\), with \( f(c) < f(c') \), as a point in the 2D plane, at location \((f(c), f(c'))\). There, the pair persistence can be visualized as the height of the point to the diagonal. 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.

To evaluate quantitatively the relevance of the progressive outputs continuously provided by our algorithms, we measure their distance to the final, exact result with the Wasserstein distance, an established practical metric inspired by optimal transport [57], [68]. Given two diagrams \( D(f) \) and \( D(g) \), a pointwise distance \( d_q \), inspired from the \( L^p \) norm, can be introduced in the 2D birth/death space between two points \( a = (x_a, y_a) \in D(f) \) and \( b = (x_b, y_b) \in D(g) \), with \( q > 0 \), as:

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

(1)

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 \( D(f) \) and \( D(g) \) can then be introduced as:

\[
W_q(D(f), D(g)) = \min_{\Phi \in \Phi} \left( \sum_{a \in D(f)} d_q(a, \Phi(a))^q \right)^{1/q}
\]

(2)

where \( \Phi \) is the set of all possible assignments \( \Phi \) mapping each point \( a \in D(f) \) to a point \( b \in D(g) \) or to its projection onto the diagonal. \( W_q \) can be computed via assignment optimization, for which exact [69] and approximate [90], [91] implementations are publicly available [92].
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 \( H \) derived from established subdivision schemes \([79],[80],[81],[82],[83]\). After a generic description of the employed triangulation hierarchy (Sec. 3.1), we detail 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 investigate how the specific structure of the employed triangulation hierarchy can be leveraged to accelerate the topological analysis of scalar data. For this, we introduce the novel notion of \textit{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 \( M \), which relies on established subdivision schemes \([79],[80],[81],[82],[83]\). We call such a hierarchy \textit{edge-nested} and formalize it in the following, to introduce the notations that will be used in the rest of the paper. Let \( H = \{ M^0,M^1,\ldots,M^h \} \) be a hierarchy of PL \( d \)-manifolds, which respects the following key conditions.

1) \textbf{Old Vertex Condition:} Each vertex of \( M^i \) (the triangulation at level \( i \)) also belongs to the vertex set, noted \( M^0 \), of \( M^{i+1} \):

\[ M_0^i \subset M_0^{i+1} \]

(3)

The vertices of \( M^{i+1} \) already present in \( M^i \) are called \textit{old vertices}.

2) \textbf{New Vertex Condition:} Each vertex of \( M^{i+1} \) not present in \( M^i \) has to be located on an edge \( (v_0,v_1) \) of \( M^i \) (typically in its center), as summarized below, where \( M_1^i \) stands for the edge set of \( M^i \):

\[ \forall v \in M_0^{i+1}, v \notin M_0^i : \exists (v_0,v_1) \in M_1^i, v \in (v_0,v_1) \]

(4)

The vertices of \( M^{i+1} \) not present in \( M^i \) are called \textit{new vertices}.

3) \textbf{Old Edge Condition:} Each edge \( (v_0,v_1) \) of \( M^i \) has to be subdivided at level \( i+1 \) along exactly one new vertex \( v \) of \( M^{i+1} \):

\[ \forall (v_0,v_1) \in M_1^i : |\{ v \in (v_0,v_1), v \notin M_0^i, v \in M_0^{i+1} \}| = 1 \]

(5)

\[ (v_0,v) \in M_1^{i+1}, (v,v_1) \in M_1^{i+1} \]

The edges of \( M^{i+1} \) obtained by subdivision of an edge of \( M^i \) are called \textit{old edges}.

4) \textbf{New Edge Condition:} Each edge of \( M^{i+1} \) which is not an old edge has to connect two new vertices, and it is called a \textit{new edge}.

Overall, this hierarchical scheme imposes that, as one progresses down the hierarchy, new vertices are inserted along edges (exactly one new vertex per edge, typically in their center), and that the additional edges only connect new vertices. The Loop subdivision \([81]\) is compatible with the above formalization, which is more generally termed as \textit{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 \( M \) admits an edge-nested triangulation hierarchy if there exists a hierarchy \( H \) for which \( M \) is the last element (\( M = M^0 \)).

3.2 Edge-Nested Triangulations of Regular Grids

While the construction of an edge-nested triangulation hierarchy given an arbitrary input manifold \( M \) 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 Bev \([85]\). We describe this implementation in the following for the sake of completeness.

Let \( G^0 \) be a \( d \)-dimensional regular grid, with \( d \) equals 2 or 3 in our applications, of dimensions \( L_0^0, L_0^1, L_0^2 \) (\( |G^0| = (L_0^0 + 1) \times (L_0^1 + 1) \times (L_0^2 + 1) \)), in 2D: \( L_0^2 = 0 \). We will first assume that \( L_0^0, L_0^1 \) and \( L_0^2 \) are all powers of 2. Let \( \phi_h \) be the \textit{triangulation operator}, which transforms \( G^0 \) into a valid triangulation \( M^h \), i.e. \( M^h = \phi_h(G^0) \), by preserving vertex sets, i.e. \( M_0^h = G_0^0 \), and by inserting exactly one edge for each \( i \)-dimensional cell of \( G^0 \) (\( 1 \leq i \leq d \)), according to a unique pattern, which is \textit{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 \textit{same diagonal}. In 3D, each hexahedron is subdivided into six tetrahedra by always inserting the \textit{same diagonal edges} (Fig. 3).

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

We now introduce by recurrence a sequence of decimation operators \( \Pi_i \) (Fig. 4), which decimate each grid \( G^{i-1} \) into a grid \( G^i \) 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 \( G^i \) are given by \( L_i^1 = L_0^1/2^i, L_i^2 = L_0^2/2^i \), and
1) Link Topological Invariance: We now consider the sequence of triangulation operators \( \phi_i \), which triangulate each grid \( G^h \) to a triangulation \( M^{h-1} \), i.e., \( M^{h-1} = \phi_i(G^h) \), as illustrated by the commutative diagram of Fig. 4.

Then, it can be verified (Fig. 5) that each condition of Sec. 3.1 is indeed satisfied by the sequence \( H = \{ M^h, M^1, \ldots, M^0 \} \) and that \( H \) is a valid edge-nested triangulation hierarchy. In particular, as described by Bey [85], any triangulation \( M^t \) can be equivalently obtained either: (i) by applying the red subdivision scheme [90], [92], [93] \( i \) times on \( M^0 \) or (ii) by considering the Kuhn triangulation [94] of \( G^{h-i} \) (itself obtained by \( i \) regular subdivisions of \( G^h \)). In other words, any triangulation \( M^t \) in the commutative diagram of Fig. 4 can be obtained by starting either (i) from \( M^0 \) or (ii) from \( G^h \). 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 \( M^0 \) of the input grid \( G^h \).

In particular, the edge-nested triangulation hierarchy \( H \) can be implemented very efficiently, by encoding the entire hierarchy implicitly, and by only maintaining the grid \( G^h \) in memory. At a given hierarchy level \( i \), adjacency relations in \( M^t \) 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. 3. 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 \( G^h \), given by the grid coordinates \((i_0 \times 2^{-i}, j_0 \times 2^{-i}, k_0 \times 2^{-i})\) and \((i_1 \times 2^{-i}, j_1 \times 2^{-i}, k_1 \times 2^{-i})\). This approach is easily extended to support regular grids whose dimensions, \( L^0 \), \( L^0 \) or \( L^0 \) are not necessarily powers of 2. In particular, when considering the decimation operator \( \Pi_i \) in case some of the dimensions \( L^i \times 1 \) or \( L^i \times 1 \) are not even, \( \Pi_i \) systematically adds the last vertex of \( G^{h-i} \) for each odd dimension. In our progressive algorithms (Sec. 4 and 5), these few extra vertices will require full re-computations. 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 \( H \) yields a hierarchy of PL scalar fields \( \{ f^0, f^1, \ldots, f^h \} \), such that each old vertex \( v \) maintains by construction its scalar value: \( f^j(v) = f^j(v) = f(v) \), \( \forall j / i \leq j \leq h \). In the following, we show how the specific structure of edge-nested triangulation hierarchies can be leveraged to efficiently update topological information while progressing down the hierarchy.

1) Link Topological Invariance: A first insight is that the link \( Lk(v) \) of a vertex \( v \) is topologically invariant throughout the hierarchy. Let \( Lk(v)^i \) be the link of \( v \) at level \( i \), then there exists a one-to-one mapping \( \Psi_i \) (right inset) between the simplices of \( Lk(v)^i \) and \( Lk(v)^{i+1} \) – such that \( Lk(v)^{i+1} = \Psi_i(Lk(v)^i) \) – 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 \( k^h \) neighbor of \( v \) at level \( i \) to its \( k^h \) new neighbor at level \( i+1 \)) and (ii) new edges are only inserted between new vertices (this maps the \( k^h \) edge of \( Lk(v)^i \) to the \( k^h \) new edge of \( Lk(v)^{i+1} \), inset). This mapping \( \Psi_i \) 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 \( \Psi_i \). We define the polarity of \( Lk(v)^i \), noted \( \delta(v) = Lk(v)^i \rightarrow \{ -1, 1 \} \) as the field which assigns to each neighbor \( n \) of \( v \) at level \( i \) the sign of its function difference with \( v \): \( \delta(n) = sgn(f(n) - f(v)) \). The polarity is positive in the upper link, negative in the lower link (blue signs, above inset). Let \((v_0, v_1)\) be an edge at level \( i \), which gets subdivided at level \( i+1 \) along a new vertex \( v_n \). We say that \( v_n \) is monotonic if \( f(v_0) < f(v_n) < f(v_1) \) or \( f(v_0) > f(v_n) > f(v_1) \). Otherwise, \( v_n \) is non-monotonic. In that case, if \( v_n \)'s polarity in \( Lk(v)^i \) is the opposite of \( v \)'s polarity in \( Lk(v)^i \), 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 (above inset) and \( v \) is guaranteed to maintain its criticality: it maintains its critical index (i.e., \( I(v)^i = I(v)^i \)) or it remains regular. Indeed, each neighbor \( n \) which does not impact \( v \) maintains it classification as being upper or lower. Then, since there is a one-to-one mapping \( \Psi_i \) (see the inset figure in the above paragraph) between \( Lk(v)^i \) and \( Lk(v)^{i+1} \) which preserves their simplicial structure, it follows that the complexes \( Lk^{-1}(v)^i \) and \( Lk^{-1}(v)^{i+1} \) are respectively identical to \( Lk^{-1}(v)^i \) and \( Lk^{-1}(v)^{i+1} \). 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 \( v \) subdividing an edge at level \( i \), 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. 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 \( Lk^{-1}(v)^i \) is necessarily connected (and so is \( Lk^{-1}(v)^i \), symmetrically). Let \((v_0, v_1, o)\) be a triangle at level \( i \) (above inset). 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 \((v_0, n_0)\), \((v_1, n_1)\), 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 \( v \) \((n_0 \in Lk^{-1}(v)^i) \). Since \( n_0 \) is monotonic, this means that the outsider old vertex \( o \) 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 \((v_1, n_1, n_0) \in Lk(v)^i \), 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 \( Lk^{-1}(v)^i \) and \( Lk^{-1}(v)^i \) 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.
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. 6) form the cornerstone of our progressive approach to topological analysis. As detailed next, checking if vertices are topologically invariant reveals to be less computationally expensive in practice than computing their criticality from scratch. Moreover, the set of topologically invariant vertices tend to represent in practice 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 initializes 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 user interruption.

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. 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 informations on the link and to quickly update them when progressing down the hierarchy. In particular, we initialize for each new vertex \( v \) at the level \( i \) the following information:

- **Link 1-skeleton:** We store the list of local edges (and their adjacencies) of \( \text{Lk}(v) \). This remains invariant through \( \mathcal{H} \) (Sec. 3.3).
- **Link polarity:** We store for each vertex of \( \text{Lk}(v) \) 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 \( \text{Lk}(v) \).
- **Link dynamic tree:** An efficient data structure \([92]\) for maintaining connected components in dynamic graphs, discussed below.

For each new vertex \( v \) which is not topologically invariant, these data structures are initialized and the criticality of \( v \) is computed with the traditional approach (Sec. 2.2), by enumerating the connected components of \( \text{Lk}^+(v) \) and \( \text{Lk}^-(v) \). 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 the level \( i \), but not anymore at the level \( i+1 \), the connected components of \( \text{Lk}^+(v) \) need to be updated accordingly, preferably without recomputing them completely. For this, we use dynamic trees \([93]\), 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 \( \text{Lk}(v) \) are initially inserted in the dynamic tree associated to \( v \). Next, we insert each local edge of \( \text{Lk}(v) \) in the dynamic tree, if both its extremities admit the same polarity. Then, the criticality of \( v \) is 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. 7), its link polarity is quickly updated based on the non-monotonic new vertices of \( \text{Lk}(v) \). Each local edge \( e \) of \( \text{Lk}(v) \) 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 extremities admit 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.

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) **Monotonic 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-monotonic 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 \( f' \) 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 \( M^l \) 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*, noted \( l(e) = l^+_e - l^-_e \), where \( l^+_e \) and \( l^-_e \) 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 [15, 24, 25, 96, 97, 98]. 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 in computing, for each neighbor \( n \) of \( v \), an integral line \( \mathcal{L}^+(n) \). Each of these lines terminates on local maxima of \( f \), which we add to the set of *candidates* for \( v \). At the end of this step, we establish the correspondence between \( v \) and its highest candidate in terms of \( f \) 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 \( i \) \((l^+_e(i) = i)\). It is possible, given a maximum \( m \) at level \( i \), that no maximum from the level \( i-1 \) maps to it. This case, \( m \) is said to appear at the level \( i \) \((l^-_e(m) = i)\). 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 \( l^+_e \)), 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 initializes at the coarsest hierarchy level, \( M^0 \), and then iterates progressively through the hierarchy levels, producing the exact persistence diagram \( D(f^i) \) for each level \( i \), until \( i = h \). We first introduce our approach in the non-progressive case [Fig. 8], and then present our progressive strategy [Sec. 5.2]. We focus on minimum-saddle persistence pairs, saddle-maximum pairs being treated symmetrically.

5.1 Persistence Diagram from Critical Points

The diagram \( D(f) \) of the extremum-saddle pairs of an input field \( f : M \rightarrow \mathbb{R} \) is computed as follows.

**Part A:**

*From data to reduced topological informations*

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. 8]. For this, we initiate backward integral lines [Sec. 2.2], for each connected component of lower link \( L^-(s) \) of each saddle \( s \). These integral lines are guaranteed to terminate in local minima of \( f \). In practice, once a backward integral line \( L^-(s) \) terminates in a local minimum \( m \), we back-propagate the vertex identifier of \( m \) and store it for each vertex \( v \in L^-(s) \). Then, \( m \) is called a *representant* of \( v \), which is noted \( r(v) = \{ m \} \). This strategy enables the early termination of an integral line \( L^-(s) \) when it merges with another one, \( L^-(s_0) \), computed previously. In that case, we back-propagate the representatives reported by the merge vertex on \( L^-(s_1) \) 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 \( m \), we create a list of critical triplets, in the form \((s, m_0, m_1)\). These are obtained by considering all the possible (order-independent) pairs, among the set of representants of \( s \) (computed previously). Note that in practice, for nearly all saddles, this list counts only one triplet, which describes the fact that \( s \) separates two pits, \( m_0 \) and \( m_1 \). Only in case of forking or degenerate saddles, multiple triplets can emerge at a given saddle.

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 \((s_0, m_0, m_1)\) models the earliest merge event between connected components of sub-level sets of \( f \). We introduce its *simplified version*, \((s_0, r(m_0), r(m_1))\), 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 \( D(f) \) 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 $(s, r(m_0), r(m_1))$, 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 $D(f)$. Note that our strategy bears global similarities with previous topological techniques based on monotonic paths [22, 47, 50, 99]. However, our strategy is specialized for the production of persistence diagrams. Overall, iterating as described above over the list of triplets results in populating $D(f)$ 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 informations which have 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 performances. In practice, 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 progressive persistence diagrams simply consists, at each level $i$ of the triangulation hierarchy $H$, 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 GNU’s efficient implementation [100]. 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](https://github.com/julesvidal/progressive-scalar-topology)), written as modules for the Topology ToolKit (TTK) [92]. The datasets have been downloaded from public repositories [101, 102].
These quantitative observations (early capture of the significant visually observed in Fig. 9, which shows that the significant pairs 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), which exhibits a linear evolution 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 in our hierarchical setup in $O(\sum_{i=1}^{n} |M_{hi}|)$ steps. For persistence
diagrams, in the worst possible configuration (degenerate saddles with systematic integral line forking), each saddle would generate monotone paths which would hit every minimum. This would yield \( n_s \times (n_m \times (n_m - 1))/2 \) critical triplets, where \( n_s \) and \( n_m \) stand for the number of saddles and minima of \( f \). However, such configurations are extremely rare in practice and most saddles only yield one triplet, resulting in an overall practical time complexity of \( O(\sum_{i=0}^{h}(|M_1^i| + n_p^i)) \).

Table 2 reports computation times (sequential run) for the default algorithms \( \text{TTK and NP columns} \) available in TTK \( \text{[92]} \) and the non-progressive and progressive versions of our algorithms. Non-progressive methods (\( \text{TTK and NP columns} \)) 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 \textit{query time}, while our implementation stores the explicit list of link edges for each vertex [Sec. 4.1]). Interestingly, the \textit{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 (\( \sum_{i=0}^{h} |M_0^i| \) vertices in progressive mode, instead of \( |M_0^h| \) 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 \( \text{[103]} \), again in comparison to the default algorithms \( \text{Sec. 2.2, [41, 23]} \) available in TTK \( \text{[92]} \) 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%.

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 \textit{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 \textit{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: \textit{MinMax} has nearly only TI vertices (99.3%) while \textit{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 \textit{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 (\textit{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 (\textit{Random}) and upper (\textit{MinMax}) bounds (on average 1.8 for critical points, 1.6 for persistence diagrams).

In terms of quality, the best/worst case aspect of \textit{MinMax} and \textit{Random} is also illustrated in Figs. 10, 11 and 12 where \textit{MinMax} converges immediately, while \textit{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. Figure 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 lightest 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.

Figure 9 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. Figure 13 illustrates progressive persistence diagrams for an acquired dataset. There, a merge tree based segmentation (computed with TTK [92]) 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. Figure 14 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. 10, 11, 12 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 [92] 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.

Figure 15 illustrates the interest of our progressive representations for the control of the run time of batch-mode analysis pipelines. We consider the Isabel ensemble [102], [106] (12 members illustrating 3 hurricane behaviors: formation, drift and landfall, [Fig. 15] 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. 15 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. 15), 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 in practice, 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. 14 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. 10 and 12). This behavior prevents the definition of theoretical error...
which computationally expensive optimizations may need to be applied to other topological analysis algorithms. Experiments which could serve as guidelines for future extensions include the improvement directions regarding the management of the input hierarchy, including the extensions to arbitrary triangulations or the addition of intermediate hierarchy levels. Second, we believe our approach can be generalized to higher dimensions (with tailored refinement schemes). Several of them seem compliant with the notion 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 [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. In practice, 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 guidelines for future extensions to other topological analysis algorithms.

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 exploitable 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, 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
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://vgii.erci.enset.in/felix/index.html.
[5] H. A. Carr, J. Shawe-Taylor, and M. van de Panne, “Smoothing Flexible Isosurfaces Using Local Geometric Measures,” in IEEE VIS, 2004.
[6] A. Bock, H. Doraaszamy, 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.
