Vis Comput (2012) 28:959–969
DOI 10.1007/s00371-012-0726-8

ORIGINAL ARTICLE

Efficient computation of 3D Morse–Smale complexes and persistent homology using discrete Morse theory

David Günther · Jan Reininghaus · Hubert Wagner · Ingrid Hotz

Published online: 26 May 2012

Abstract We propose an efficient algorithm that computes the Morse–Smale complex for 3D gray-scale images. This complex allows for an efficient computation of persistent homology since it is, in general, much smaller than the input data but still contains all necessary information. Our method improves a recently proposed algorithm to extract the Morse–Smale complex in terms of memory consumption and running time. It also allows for a parallel computation of the complex. The computational complexity of the Morse–Smale complex extraction solely depends on the topological complexity of the input data. The persistence is then computed using the Morse–Smale complex by applying an existing algorithm with a good practical running time. We demonstrate that our method allows for the computation of persistent homology for large data on commodity hardware.

Keywords Persistent homology · Morse–Smale complex · Discrete Morse theory · Large data

D. Günther (✉)
Computer Graphics, Max-Planck Institute for Informatics, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany
e-mail: dguenther@mpi-inf.mpg.de

J. Reininghaus · I. Hotz
Zuse Institute Berlin, Takustraße 7, 14165 Berlin, Germany

J. Reininghaus
e-mail: reininghaus@zib.de

I. Hotz
e-mail: hotz@zib.de

H. Wagner
Institute of Computer Science, Jagiellonian University, Lojasiewicza 6, 30-348 Krakow, Poland
e-mail: hubert.wagner@ii.uj.edu.pl

1 Introduction

It is clear that with the rapid increase of the amount of data produced, availability of efficient tools to analyze these data is of great importance. Computational topology [9], due to its ability to extract essential features of the analyzed data, is becoming a widely-used method. In particular, persistent homology introduced by Edelsbrunner et al. [10] has drawn much attention, since it robustly extracts the topological structure of the data.

While algorithms with good practical running times have been proposed [7], exact computation of persistence for large 3D image data remains a challenging problem due to huge memory requirements.

Forman's discrete Morse theory [11, 12], which is the theoretical foundation of our algorithm, allows us to reduce data in a way which preserves the topological structure. This representation of the data, called the Morse–Smale complex, is much more compact but still contains all the necessary topological information for persistent homology computation.

Inspired by the work of Robins et al. [25], we use discrete Morse theory to compute persistent homology. Günther et al. [14] showed that the Morse–Smale complex can be computed in $O(cn) \subseteq O(n^2)$, where n denotes the size of the input data and c the number of its critical points. In this paper, we propose an improved version of Algorithm 3 in [14]. The memory consumption of our improved algorithm depends solely on the topological complexity of the input data and it also allows for a parallel computation of the Morse–Smale complex.

We present results of an efficient implementation, which show that our algorithm is suitable for real-world applications. The method introduced in this paper allows for memory-efficient computation of persistent homology of

large 3D images. For example, we only need about 14 GB of memory for a data set of size $1120 \times 1131 \times 1552$, in contrast to the 500 GB that would be necessary using standard algorithms.

The remaining part of this paper is organized as follows. The related research is described in Sect. 2. In Sect. 3, the theoretical background of persistence and discrete Morse theory is introduced. In Sects. 4 and 5, we present our method and show computational results. Finally, we summarize the paper with a brief discussion in Sect. 6.

2 Related work

Persistence We will focus on previous work on *computing* persistence. For general applications of persistence, see [9]; for application in the context of image data, see [4, 23].

The standard, algebraic algorithm [9] for persistence has cubic running time in the size of the input (i.e., image). While a simplicial complex example was constructed by Morozov [22], showing that this pessimistic execution can actually occur, the behavior of this algorithm is only slightly superlinear in practical situations [7].

When focusing on 0-dimensional homology, union-find data structures can be used to compute persistence in time $O(n\alpha(n))$ [9], where α is the inverse of the Ackermann functions and n the size of the input.

Milosavljevic et al. [21] showed that persistent homology can be computed in matrix multiplication time $O(n^\omega)$ where the currently best estimation of ω is 2.376. Chen and Kerber [6] proposed a randomized algorithm to compute only pairs with persistence above a chosen threshold. Despite showing an improved theoretical complexity, it is unclear whether these methods are better than the standard persistence algorithm in practice.

A recent variation of the standard algebraic algorithm [9], called *killing*, introduced by Chen and Kerber [7] significantly reduces the amount of computations. This idea was also used by Wagner et al. [27] to compute persistence for n-dimensional images.

In general, purely algebraic methods suffer from high memory requirements. In our approach, we alleviate this effect by reducing the size of data.

Discrete Morse theory Morse theory [20] is a mathematical theory which relates the topology of the domain of a function with critical points of this function. For example, every continuous function defined on a sphere has at least one critical point. The set of critical points extracted should therefore satisfy the constraints described by Morse theory. Note that due to the global nature of topological consistency it is difficult to enforce these constraints in local numerical algorithms. Fortunately, Forman [11, 12] developed a discrete version of Morse theory, which allows for algorithms that provably result in a consistent set of critical points.

The first such algorithm was proposed by Lewiner et al. [18, 19] who also conjectured that persistence could be efficiently computed using discrete Morse theory. Recently, several other such algorithms were suggested [3, 13, 24]. Gyulassi et al. [15] introduced a fast streaming approach to extract the essential critical points of large data. The resulting complex is iteratively simplified to differentiate between spurious and important critical points. However, this approach is not suited for exact persistence computation since not all points in this complex can be paired.

Robins et al. [25] presented the first algorithm which is provably correct in 3D in a sense that the computed critical points correspond one-to-one to the topological changes in the sublevel sets of the image data. Günther et al. [14] built on the method by Robins et al. and proposed an optimal Morse–Smale complex extraction algorithm. In this paper, we further improve the algorithm by Günther et al. to enable also a memory-efficient parallel computation of the complex.

3 Theoretical background

Complexes The input of the persistent homology computation is a 3D gray-scale image: an array $\Omega = m \times k \times \ell$ and a function $f{:}\Omega \to \mathbb{R}$. To capture the topological information, we need to represent this as a *complex*, which is a decomposition of a space into *cells* of *different dimensions*. See Fig. 1a for an example. During the first part of computations, we use *cubical complexes* [17], whose cells consist of vertices, edges, squares, and full cubes. The *Morse–Smale* complex we extract later belongs to the class of *CW-complexes*, which is more general and its p-cells are only required to be homeomorphic to p-spheres [16].

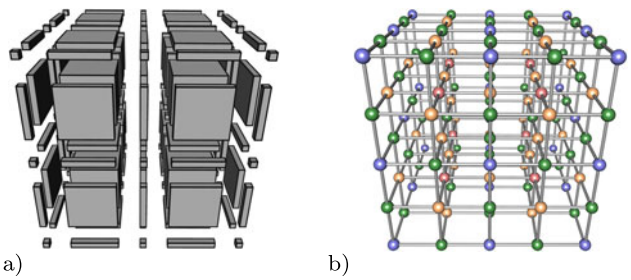


Fig. 1 Illustration of a cubical complex and its derived cell graph. Image (**a**) shows the cells of a small uniform grid in an exploding view. Image (**b**) shows the derived cell graph G_C. The nodes representing the 0-, 1-, 2-, and 3-cells are shown as *blue*, *green*, *yellow*, and *red* spheres, respectively

Boundary maps and matrices Cells of different dimensions are connected by boundary relations. For example, the boundary of an edge $E = (a, b)$ are the vertices a and b. If a $(p-1)$-cell α is in the boundary of a p-cell β, we say α is a proper face of β. Note that if a complex contains a cell c, it must also contain all the faces of c.

For any p-dimensional cell c, its *boundary*, denoted by $\partial_p c$, is the set of its $(p-1)$-dimensional faces. We now define this relation algebraically. Let a *p-chain* be a formal sum of p-cells with $\mathbb{Z}_2$ coefficients (other groups of coefficients can be used, but this one is the most suitable for our task). This enables us to extend the boundary operator linearly to p-chains. For any p-chain $c = \sum a_i c_i$, we have $\partial_p c = \sum a_i \partial_p c_i$. The p-chains, together with (modulo 2) addition, form a *group of p-chains*, denoted by C_p.

If we specify a unique index for each cell, a p-chain corresponds to a vector in $\mathbb{Z}_2^{n_p}$, where n_p is the number of p-dimensional cells in the complex. The p-dimensional boundary operator ∂_p can be written as an $n_p \times n_{p-1}$ binary matrix (also denoted ∂_p) whose columns are the boundaries of the p-cells.

The above is summarized by the *chain complex*, which can be viewed as an *algebraic representation* of a complex C [11]

$$C : C_3 \xrightarrow{\partial_3} C_2 \xrightarrow{\partial_2} C_1 \xrightarrow{\partial_1} C_0. \tag{1}$$

Filtration For a given complex K, a filtration is a nested sequence of complexes: $\emptyset = K_0 \subseteq K_1 \subseteq \cdots \subseteq K_n = K$ [9, 10]. In our case, it is induced by the input data $f{:}\Omega \to \mathbb{R}$ as follows. First, the values given by f on the 0-cells of K, are extended to all cells of K by a so-called *lower-star filtration*: each cell is assigned the maximum function value of the vertices it contains. The filtration of K with respect to f is then defined by the sublevel complexes $K^t = f^{-1}(-\infty, t]$. Imagine that we start with an empty complex and at each step of the filtration one or more cells are added.

Persistence First, we will give a basic intuition behind homology and persistent homology. For this paper, we can say that homology detects topological features: connected components, tunnels, and voids for a *fixed* thresholding (sublevel set) of a gray-scale image. *Persistent* homology, in turn, describes the *evolution* of topological features looking at consecutive thresholds.

More precisely, given a complex K and a *filtering function* $f{:}K \to \mathbb{R}$, *persistent homology* studies homological changes of the sublevel complexes, $K^t = f^{-1}(-\infty, t]$. The algorithm captures the birth and death times of homology classes of the sublevel complexes, as the threshold t grows from $-\infty$ to $+\infty$. By birth, we mean that a homology feature comes into being; by death, we mean it either becomes trivial or becomes identical to some other class born earlier.

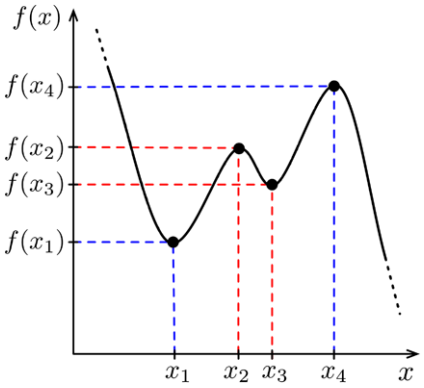


Fig. 2 Homological persistence of a 1D function $f(x)$. The persistence pairs consist of (x_1, x_4) and (x_3, x_2). The persistence of x_1 and x_4 is therefore given by $f(x_4) - f(x_1)$, while the persistence of x_2 and x_3 is given by $f(x_2) - f(x_3)$

See Fig. 2 for an example. The *persistence*, or lifetime of a class, is the difference between the death and birth times. Homology classes with larger persistence reveal information about the global structure of the space K, described by the function f.

The overall output of the computations is a list of *persistence pairs* of the form (birth, death). This information can be visualized in different ways. One well-accepted idea is the persistence diagram [8], which is a set of points in a two-dimensional plane, each corresponding to a persistent homology class. The coordinates of such a point are the birth and death time of the related class.

An important justification for the usage of persistence is the stability theorem. Cohen-Steiner et al. [8] proved that for any two filtering functions f and g, the difference of their persistence is always upper bounded by the L^∞ norm of their difference:

$$\|f - g\|_\infty := \max_{x \in K} \big|f(x) - g(x)\big|. \tag{2}$$

This enables robust estimation of how persistence is affected by perturbation of the input (e.g., noise). This guarantees that persistence can be used as a signature. Whenever two persistence outputs are different, we know that the functions are definitely different.

Discrete Morse theory In the following, we assume that a three-dimensional cubical complex C is given. We use the lower-star filtration defined above to extend the input function to all cells. The *cell graph* $G_C = (N, E)$ encodes the combinatorial information contained in C. The nodes N of the graph consist of the cells of the complex C and each node u^p is labeled with the dimension p of the cell it represents. The edges E of the graph encode the neighborhood relation of the cells in C. If the cell u^p is in the boundary of the cell w^{p+1}, then $e^p = \{u^p, w^{p+1}\} \in E$. We label each edge with the dimension of its higher dimensional node. An

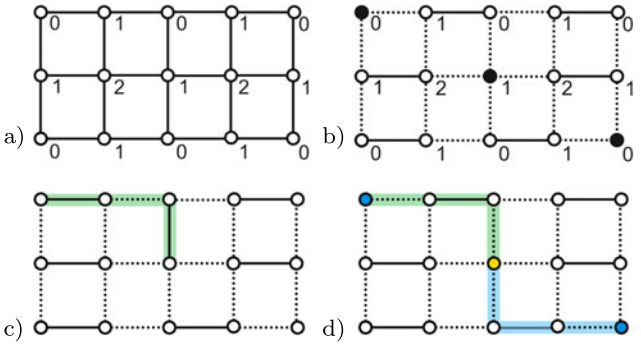


Fig. 3 Basic definitions of discrete Morse theory: (**a**) the cell graph G_C, the node labels indicate the dimension of the represented cells; (**b**) a combinatorial gradient field V defined on G_C, the edges contained in V are depicted by *solid lines*, the unmatched nodes—the critical nodes—are shown as *black spheres*; (**c**) a combinatorial streamline alternating between V and its complement; (**d**) two 1-separatrices of V (*blue* and *green*) emanating at a 1-saddle (*yellow*) and ending in a minimum (*blue*)

illustration of a cell graph is shown in Fig. 1b. Note that the node indices, their adjacency, and their geometric embedding in $\mathbb{R}^3$ are given implicitly by the regular grid structure of Ω.

A subset of pairwise nonadjacent edges is called a *matching* $M \subset E$. Using these definitions, a *combinatorial gradient field* V on a regular cell complex C can be defined as a certain acyclic matching of G_C [5]. The set of combinatorial gradient vector fields on C is given by the set of these matchings, i.e., the set of *Morse matchings* $\mathcal{M}^\phi$ of the cell graph G_C. An illustration of a 2D Morse matching is shown in Fig. 3b.

We now define the extremal structures of a combinatorial gradient vector field V in G_C. The unmatched nodes are called *critical nodes*. If u^p is a critical node, we say that it has *index* p. A *critical node* of index p is called minimum ($p = 0$), 1-saddle ($p = 1$), 2-saddle ($p = 2$), or maximum ($p = 3$). A combinatorial *p-streamline* is a path in the graph whose edges are of dimension p and alternate between $V \subset E$ and its complement $E \setminus V$. In a Morse matching, there are no closed p-streamlines. This defines the acyclic constraint for Morse matchings. A p-streamline connecting two critical nodes is called a *p-separatrix*. A *p-separation surface* is given by all combinatorial 2-streamlines that emanate from a critical point of index p. An illustration of extremal structures is shown in Fig. 3.

Using the above definitions, we can now define the chain complex associated to the *Morse–Smale complex* C_V with coefficients in $\mathbb{Z}_2$

$$C_V : C_3 \xrightarrow{\partial_3} C_2 \xrightarrow{\partial_2} C_1 \xrightarrow{\partial_1} C_0. \tag{3}$$

The Morse–Smale complex C_V is induced by a given combinatorial gradient field $V \subset E$ in a cell graph $G_C = (N, E)$. The chain groups C_ℓ are generated by the critical nodes of V with index ℓ. The boundary maps ∂_ℓ are defined by the combinatorial streamlines of V: if $u^\ell \in C_\ell$ is connected to $w^{\ell-1} \in C_{\ell-1}$ by an *odd number* of combinatorial streamlines, then $w^{\ell-1}$ is in the boundary of u^ℓ. Considering only pairs $(w^{\ell-1}, u^\ell)$ with an odd number of connections reduces the general formula by Forman [11] to a simplified version for $\mathbb{Z}_2$ coefficients.

Forman proves that the homology of C is always isomorphic to the homology of C_V [11]. If the critical nodes contained in V correspond one-to-one to the topological changes in the sublevel complexes, the persistent homology of C therefore coincides with the persistent homology of C_V [25].

Since, in practice, C_V is a lot smaller than C, we can use discrete Morse theory to devise a memory-efficient algorithm for persistent homology.

4 Method

In this section, we describe our overall algorithmic pipeline to compute persistent homology in a memory efficient manner. The input of the pipeline consists of a 3D gray-scale image. It is represented by a 3D array $\Omega = m \times k \times \ell$ and a function $f\colon \Omega \to \mathbb{R}$. To compute persistent homology, we initially represent Ω by the cubical complex C.

The pipeline consists of three steps. In Sect. 4.1, we describe the construction of the discrete gradient field V associated to C and f. In Sect. 4.2, we propose an improved algorithm to extract the Morse–Smale complex C_V defined by V. For completeness, we also describe the computation of the persistent homology of C_V in Sect. 4.3. We conclude this section with a brief analysis of the computational complexity, memory consumption, and some implementational details in Sects. 4.4 and 4.5.

4.1 Discrete gradient field

To compute the discrete gradient vector field V, we use the algorithm *ProcessLowerStar* [25]. The basic idea of this algorithm is to apply simple homotopic expansions in the lower star of each 0-node. The algorithm results in a combinatorial gradient field V whose critical nodes coincide with the changes of the topology of the sub-level complexes of C. For more algorithmic details and the proof, we refer the interested reader to [25].

4.2 Morse–Smale complex extraction

We now describe how we compute the chain complex associated to the Morse–Smale complex (3) induced by the combinatorial gradient field $V \subset E$.

Algorithm 1 ComputeBoundary(G_C, V, j, ℓ)

Input: $G_C = (N, E), V \subset E, j \in \{0, 1\}, \ell \in \{1, 2, 3\}$
Output: Binary matrix ∂_ℓ

```
1: S ← GetAllManifolds(G_C, V, ℓ, j)
2: I ← GetIntersection(G_C, V, S, ℓ, j)
3: for all c^p ∈ C_{ℓ−j} do
4:     C_c ← CountPaths(G_C, V, I, c^p, ℓ)
5:     for all w^k ∈ C_c do
6:         if k < p then
7:             ∂_ℓ(c^p, w^k) ← ∂_ℓ(c^p, w^k) + 1
8:         else
9:             ∂_ℓ(w^k, c^p) ← ∂_ℓ(w^k, c^p) + 1
```

While the chain groups C_ℓ can be easily extracted from N by collecting the nodes not covered by V, efficient computation of ∂_ℓ is challenging. The algorithm *ComputeBoundaryB* in [14] computes ∂_ℓ by counting the number of paths between pairs of critical nodes. However, manifolds emanating at different critical points can merge. This yields a partial multiple traversal of manifolds during a breadth-first search.

We now present our improved method to compute ∂_ℓ with a worst case complexity of $O(n^2)$. The main idea of Algorithm 1 is the following. We first collect all critical (unmatched) nodes in V. For each of these nodes, we then integrate the corresponding manifolds to collect the critical nodes in the respective (*co*-)boundaries but avoid multiple traversals of the manifolds. Since the connections between critical points are defined as intersection of their manifolds, we apply a backintegration for each of the (*co*-)boundary nodes restricted to the already integrated manifold. This results in a set of edges describing all connections between critical nodes.

The challenging task is now to check whether a pair of critical nodes is connected by an odd number of separatrices. If this is the case, these nodes are connected in the sense of $\mathbb{Z}_2$ and are inserted in the boundary matrix. To count the number of connections, we compute the multiplicity of paths from one critical node to another critical node but restricted to the intersection of the corresponding manifolds.

The input of Algorithm 1 consists of the cell graph $G_C = (N, E)$, a discrete gradient field $V \subset E$, a flag j, and the index ℓ of the resulting boundary map ∂_ℓ. If $j = 0$, the algorithm computes ∂_ℓ by finding the boundaries of the elements contained in C_ℓ. If $j = 1$, the algorithm computes ∂_ℓ by finding the *co*-boundaries of the elements contained in $C_{\ell-1}$. Note that both cases result in the same ∂_ℓ, the choice of j only affects the running time; see Sect. 4.5.

For notational simplicity in the following explanation, we only describe the algorithms in detail for $j = 0$—we consider the boundary of $c^\ell \in C_\ell$. However, all algorithms are designed to work also for $j = 1$.

Algorithm 2 GetAllManifolds(G_C, V, ℓ, j)

Input: $G_C = (N, E), V \subset E, \ell \in \{1, 2, 3\}$
Output: $S \subset E$

```
1: E_ℓ ← {e^k ∈ E : k = ℓ}
2: S ← ∅
3: for all c^p ∈ C_{ℓ−j} do
4:     E_ℓ ← E_ℓ \ S
5:     S ← S ∪ AlternatingRestrictedBFS(G_C, V, E_ℓ, c^p)
```

Algorithm 3 AlternatingRestrictedBFS(G_C, V, R, c^p)

Input: $G_C = (N, E), V \subset E, R \subset E, c^p \in N$
Output: $T \subset R \subset E$

```
1: T ← ∅
2: Q.push({c^p, false})
3: while Q ≠ ∅ do
4:     {u^p, flag} ← Q.pop()
5:     W ← AlternatingEdges(G_C, V, u^p, flag)
6:     W ← (W ∩ R) \ T
7:     for all {u^p, w^k} ∈ W do
8:         T ← T ∪ {u^p, w^k}
9:         Q.push({w^k, ¬flag})
```

Algorithm 4 GetIntersection(G_C, V, S, ℓ, j)

Input: $G_C = (N, E), V \subset E, S \subset E,$
$\ell \in \{1, 2, 3\}, j \in \{0, 1\}$
Output: $I \subset E$

```
1: C_S ← {u^{ℓ−1+j} ∈ N : ∃{u^{ℓ−1+j}, w^k} ∈ S} ∩ C_{ℓ−1+j}
2: I ← ∅
3: for all c^p ∈ C_S do
4:     S ← S \ I
5:     I ← I ∪ AlternatingRestrictedBFS(G_C, V, S, c^p)
```

We first compute the edges $S \subset E$ that are covered by the combinatorial ℓ-streamlines emanating from all elements of C_ℓ (Line 1) using Algorithms 2, 3, and 7. Note that each edge is traversed only once. All already visited edges are removed from the set of admissible edges (line 4, Algorithm 2 and line 6, Algorithm 3).

We then collect all critical nodes $C_S \subset C_{\ell-1}$ that are covered by S. These nodes are the possible boundary nodes. To compute the boundary of an individual c^ℓ, we need to count the number of combinatorial streamlines connecting c^ℓ with $c^{\ell-1} \in C_S$.

To do this efficiently, we first compute the set of edges $I \subset S$ of all combinatorial streamlines connecting the elements of C_ℓ with C_S (line 2) using Algorithms 4, 3, and 7. Each edge is again only traversed once.

All edges $I \subset E$ between the elements of C_ℓ and C_S are now computed. In the following we need to count the number of paths in I that connect $c^{\ell-1} \in C_S$ to $c^\ell \in C^\ell$.

Algorithm 5 CountPaths(G_C, V, I, c^p, ℓ)

Input: $G_C = (N, E), V \subset E, I \subset E, c^p \in N$
Output: $C_c \subset N$

```
1:  N_S ← GetManifoldNodes(G_C, V, I, c^p, ℓ)
2:  C_V ← {u^p ∈ N : ∄{u^p, w^k} ∈ V}
3:  P ← ∅
4:  L ← {c^p}
5:  Q.push({c^p, false})
6:  while Q ≠ ∅ do
7:      {u^p, flag} ← Q.pop()
8:      P ← P ∪ {u^p}
9:      W ← AlternatingEdges(G_C, V, u^p, flag)
10:     W ← W ∩ I
11:     for all {u^p, w^k} ∈ W do
12:         if w^k ∈ N_S then
13:             if u^p ∈ L then
14:                 L ← LΔ{w^k}
15:             Z ← AlternatingEdges(G_C, V, w^k, flag)
16:             Z ← Z ∩ I
17:             N_Z ← {z^q ∈ N : ∃{z^q, w^k} ∈ Z}
18:             if N_Z ⊂ P then
19:                 Q.push({w^k, ¬flag})
20: C_c ← L ∩ C_V \ c^p
```

Algorithm 6 GetManifoldNodes(G_C, V, I, c^p, ℓ)

Input: $G_C = (N, E), V \subset E, c^p \in N, \ell \in \{1, 2, 3\}$
Output: $N_S \subset N$

```
1: E_ℓ ← {e^k ∈ E : k = ℓ} ∩ I
2: S ← AlternatingRestrictedBFS(G_C, V, E_ℓ, c^p)
3: N_S ← ∅
4: for all {u^p, w^k} ∈ S do
5:     if u^p ∉ N_S then
6:         N_S ← N_S ∪ {u^p}
7:     if w^k ∉ N_S then
8:         N_S ← N_S ∪ {w^k}
```

This is done in line 4 using a simple graph algorithm shown in Algorithm 5. Since I represents all paths between critical nodes, we need to extract the paths for a given pair $(c^{\ell-1}, c^{\ell})$. The nodes covered by these paths then restrict the counting of the paths. This is obtained in line 1 using Algorithm 6. Since we are only interested in the Morse–Smale complex with coefficients in $\mathbb{Z}_2$, it suffices to count the number of paths modulo 2. This is obtained by taking the symmetric difference Δ in line 14 in Algorithm 5.

4.3 Persistence

To compute persistence, we use the standard algebraic algorithm [9] with a modification by Chen and Kerber [7]. This algorithm operates on a boundary matrix, ∂_ℓ, of the complex

Algorithm 7 AlternatingEdges$(G_C, V, u^p, flag)$

Input: $G_C = (N, E), V \subset E, u^p \in N, flag \in \{false, true\}$
Output: $W \subset E$

```
1: if flag = true then
2:     W ← {{u^p, w^k} ∈ E : {u^p, w^k} ∈ V}
3: else
4:     W ← {{u^p, w^k} ∈ E : {u^p, w^k} ∈ E \ V}
```

C, representing the input data. A *reduced matrix* is computed, from which the list of *persistent pairs*, as defined in Sect. 3, can be easily read. We refer the reader to [7] for more details.

In contrast to previous work [7, 27], we apply the matrix reduction algorithm to the Morse–Smale complex C_V instead of the initial complex C. Since C_V is much smaller than C in typical situations, storing the boundary matrices consumes significantly less memory (see Table 1).

4.4 Computational complexity

We now give a brief analysis of the computational complexity of our method. We denote the number of vertices of C by n and the number of critical nodes in a combinatorial gradient field by c. Note that the pseudocode shown in the algorithms in this section has been optimized for compactness and clarity instead of a best computational complexity. In the following analysis, we consider an optimal implementation of these algorithms. The realization of such an implementation from the pseudo code only poses some minor technical difficulties.

The complexity for the construction of the combinatorial gradient field using the algorithm proposed in [25] is $O(n)$—for each node of index 0 we only work on its lower star which has a constant size in the case of cubical complexes.

Analyzing the complexity of the Morse–Smale complex extraction described in Algorithm 1 is more intricate.

We start with the essential Algorithm 3. Due to line 6, the union in line 8 is disjoint, which implies that the complexity of Algorithm 3 is $O(|T|)$. Since Algorithm 2 only calls Algorithm 3, its complexity is $O(|S|)$. The complexity of Algorithm 4 is $O(|I|)$, since due to line 4, the union in line 5 is disjoint.

Finally, we need to consider the complexity of the loop in Algorithm 1. The loop in line 3 is executed $O(c)$ times. The complexity of its body is given by the complexity of Algorithm 5. In line 1 of Algorithm 5 Algorithm 6 is called, which is a direct application of Algorithm 3. Each node is uniquely inserted and its complexity is therefore $O(|N_S|) \subseteq O(|I|)$. The complexity of the body of the *while* loop in line 6 of Algorithm 5 is constant. It therefore suffices to count the number of times that line 19 is executed. The node w^k

Table 1 Running times and memory consumption for 3D images of different size and topological complexity. The second column shows the topological properties of the data sets. The third column shows the total memory consumption of our method compared to Wagner et al. [27] using 1 and 24 logical cores. The total running time for the construction of the initial gradient field, the boundary matrix and the matrix reduction using 1 and 24 logical cores are compared to the times of Wagner et al. [27] in the fourth column

Data	Properties			Memory (MB)			Time (sec)		
	Dimensions	$\sum \lvert C_\ell \rvert$	$\lvert I \rvert$	[27]	1×	24×	[27]	1×	24×
Silicium	98 × 34 × 34	1,109	13,882	30	1	1	5	1	<1
Fuel	64 × 64 × 64	667	4,982	82	1	1	1	3	<1
Neghip	64 × 64 × 64	5,709	47,025	82	2	3	2	3	<1
Hydrogen	128 × 128 × 128	24,257	168,626	538	17	19	37	21	2
Engine	256 × 256 × 128	1,035,127	7,331,167	2,127	296	236	82	141	16
X-Mas present	246 × 246 × 221	4,836,087	21,201,265	3,112	727	901	16,007	444	153
Aneurysm	256 × 256 × 256	75,485	1,308,765	4,250	135	146	211	170	14
Bonsai	256 × 256 × 256	344,277	5,886,696	4,250	225	273	175	219	22
Foot	256 × 256 × 256	1,658,617	12,162,264	4,250	405	505	171	270	30
Noise	256 × 256 × 256	11,761,873	42,389,191	*	1,517	1,866	*	619	101
Supine	512 × 512 × 426	27,440,949	142,886,726	26,133	4,701	5,876	1,496	2,369	339
Prone	512 × 512 × 463	28,976,885	152,326,748	28,406	5,009	6,262	2,180	2,525	354
X-Mas tree	512 × 499 × 512	50,043,123	215,181,923	*	7,392	9,162	*	3,374	562
Molecule	1,120 × 1131 × 1552	1,766,615	40,178,429	*	13,837	14,168	*	18,134	1,504

is only inserted into Q if it belongs to the local intersection (line 12) and all neighboring nodes $z^q \in N_Z$ have already been processed (lines 8 and 18). Since w^k can only be inserted by a neighboring node z^q, it can therefore be inserted only once. The complexity of Algorithm 5 is hence $O(|I|)$, since only nodes contained in I can enter the queue at all.

Note that there holds $O(|I|) \subseteq O(|S|) \subseteq O(n)$. The overall complexity of Algorithm 1 is hence $O(|S|+|I|+c|I|) \subseteq O(cn)$. Since there is a lower bound on the worst-case complexity for the Morse–Smale complex extraction problem in 3D of $O(n^2)$ [25], our proposed algorithm is optimal.

The choice of j does not affect the overall computational complexity—it only affects the practical running time of the algorithm; see Sect. 4.5.

The computational complexity of the matrix reduction algorithm [7] applied to the Morse complex with c critical cells is $O(c^3)$.

The overall complexity for our method is $O(cn + c^3)$.

4.5 Implementational details

Running time To compute the combinatorial gradient field, the cell graph is decomposed into lower stars of the 0-nodes. Since this is a disjoint decomposition, each lower star can be processed in parallel. Also, the boundaries of the critical points are independent of each other, which allows a parallel computation. We process Algorithms 1, 2, and 4 in parallel using OpenMP.

The flag j of Algorithm 1 influences solely its running time. In 3D, the combinatorial 1-streamlines can only merge, while the 3-streamlines can only split. As shown in [25], the computation of the co-boundaries of all 0-nodes ($j = 1, \ell = 1$) has thereby only a complexity of $O(n)$. The same applies to the boundaries of the 3-nodes ($j = 0, \ell = 3$). In contrast, the computation of ∂_2 has a worst case complexity of $O(n^2)$, regardless of j. The choice of j thereby does not affect the overall complexity of Algorithm 1. The practical running time, however, depends on j. For most inputs, the best choice is ($j = 0, \ell = 1$), ($j = 0, \ell = 2$), ($j = 1, \ell = 3$), since the computation of the (co) boundaries of the 2- and 1-nodes only amounts to a line integration, as in this setting, $|W| \leq 1$ in Algorithm 3, line 5.

Memory requirements We need only to compute the boundary matrices ∂_ℓ of the Morse–Smale complex C_V, which does not require much memory. On the other hand, explicit representation of the initial cubical complex C would require enormous amounts of memory. We therefore represent C only implicitly, using the regular structure induced by the grid [13, 27]. The adjacency information represented in the cell graph $G_C = (N, E)$ is always computed on-the-fly using index calculations. Since we enumerate the nodes N and the edges E without gaps, we can represent the combinatorial gradient field V simply by an array of bits of length $|E|$. The sets used in the algorithms depicted in this section can also be represented using such Boolean arrays. However, Boolean arrays of size $|E|$ would result in a huge memory overhead using a parallel computation (see Table 1). Since we only work on the intersection of manifolds, arrays of size $O(|I|)$ are sufficient. A look-up map

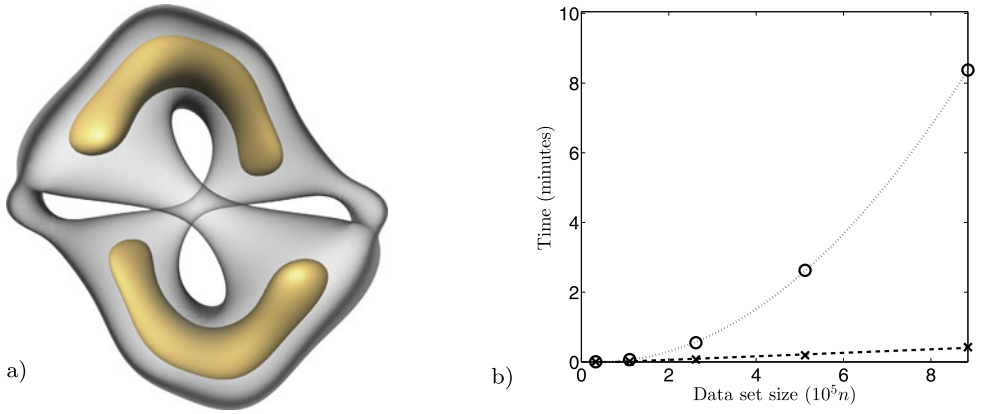


Fig. 4 Comparison of running times for an analytic function. (**a**) The *gray* and *yellow* surfaces depict two different isolevels of the analytic function g. (**b**) The *circle* and *cross* markers show the running times to construct the boundary matrix over different resolutions for the algorithms in [25] and Algorithm 1, respectively. The *semisolid line* depicts a least-square fitting of a linear function for the cross markers. The *dotted line* depicts a fitting of a quadratic function for the circle markers

translates global into local indices. This allows for efficient set operations.

If the data values on the 0-cells of the complex are defined by 32-bit single precision floats, then the total memory overhead factor of our method is about 3 in our current implementation.

5 Results

In the following, we present some examples to illustrate our method. All experiments were performed on a machine with two Intel Xeon E5645 CPUs, which provide 12 physical and 24 logical cores, and 24 GB RAM.

Table 1 shows the running time and memory consumption for different 3D data sets provided by [1, 2, 26]. We measured the total memory usage of our method with one and 24 cores. We also included the memory consumption of a persistence method [27] working on the boundary matrices of the initial cubical complex.

In the last column of Table 1, we compared the total running times of our method with the method proposed by Wagner et al. [27]. Note that a further comparison to other techniques is also given in [27].

The total memory consumption of our method is up to a factor 30 less than using a standard persistence approach. In practice, the Morse–Smale complex C_V is much smaller than the cubical complex C. This enables the persistence computation of large data. However, the memory consumption compared to [14] is in most of the examples slightly larger. This results from the usage of a translation map. Its size depends on the size of the intersection. The memory overhead for the parallel computation, however, is neglectable.

The overall running time using a single core is of similar order as the timings as in [27]. Using a parallel computation, we observe a speedup factor up to ten in our current implementation.

To investigate the behavior of Algorithm 1 for noisy data, we sampled pure uniform noise in the range of [0, 1] on a uniform 256^3 grid. While the cell complex consists of about 10^8 nodes, its gradient field contains only about 10^7 critical points. Even in the case of pure noise, the majority of nodes in G are noncritical nodes, which yields a reasonable memory consumption of a factor 2 less than a standard persistence approach. Due to the large number of critical points and the absence of large scale structures, the corresponding separatrices are relatively small and well distributed. Hence, they can be efficiently integrated. The timings as well as the memory consumption are also given in Table 1.

Figure 4 shows the running times of the Morse–Smale complex extraction step using the existing algorithm by Robins et al. [25] as well as of Algorithm 1. The data set is given by sampling an analytic function g on a uniform grid of increasing resolution and adding a small amount of uniform noise in the range of $[-0.5, 0.5]$ to the samples. The algorithm in [25] scales quadratically with the number of vertices n in the complex. In contrast, our method scales only linearly. The individual running times using a single core of Algorithm 1 applied to the analytic function g sampled on a uniform 96^3 grid are: (Sect. 4.1) 10 sec, (Sect. 4.2) 25 sec and (Sect. 4.3) 0.1 sec. In contrast, the algorithm in [25] needs (Sect. 4.2) 486 seconds.

The reason for this behavior stems from the structure of the data. The function g contains some large scale structures. Adding noise to it results in many critical points and the combinatorial 2-streamlines often merge and split. While this property dramatically increases the practical running

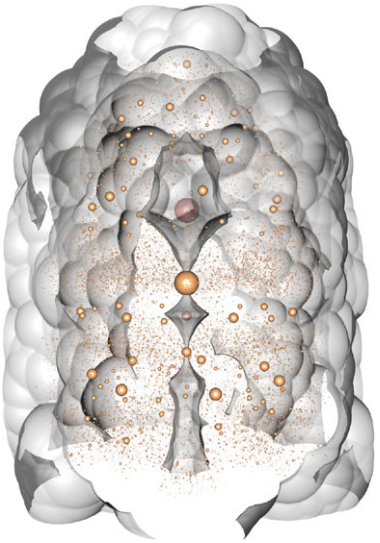


Fig. 5 Distance field of a molecule. An isosurface of a distance field, computed from a molecule, as *gray* transparent surface is shown. The maxima and the 2-saddles are shown as *red* and *yellow spheres*, respectively. Each sphere is scaled by its persistence

time of the algorithm by Robins et al. [25], our Algorithm 1 is not affected by this perturbation of the data.

We applied our method to a distance field, computed from a Chaperone protein, shown in Fig. 5. The objective is the extraction of the maxima and 2-saddles. While the maxima represent the points with the greatest distance to the atoms, the 2-saddles correspond to the narrow points of the field. These points define the minimal size of an atom to enter the molecule from the outside. The data set is of dimension $1120 \times 1131 \times 1552$ and contains 1,766,615 critical points. A standard persistence computation would theoretically require about 500 GB memory. Our approach, in contrast, only requires about 14 GB even using multiple cores, and can thereby be applied on commodity hardware. The total running time as well as the memory consumption for this example are shown in the last row of Table 1.

6 Conclusion and future work

We presented an improvement of the algorithm proposed in [14] to extract the Morse–Smale complex from a given combinatorial gradient field induced by a 3D gray scale image. This allows for a parallel computation of the Morse–Smale complex as well as a memory-efficient persistent homology computation. As shown in Sects. 4 and 5, our algorithm combines many useful properties:

1. The algorithm for the Morse–Smale complex extraction is optimal with a worst-case complexity of $O(cn) \subset O(n^2)$.
2. The overall complexity for the persistence computation is $O(cn + c^3)$.
3. The Morse–Smale complex computation requires significantly less memory and can be done in parallel.

There are some limitations of our approach:

1. Extending our techniques to more general inputs like simplicial complexes is possible, but would result in high memory-usage—we heavily exploit the compact representation of the initial, cubical complex.
2. Our current method is limited to three dimensions.

Despite these drawbacks, we believe that our method enables the application of persistent homology in new fields. Our current implementation can already be used to analyze very large, complex datasets.

A fundamental question, which is still an open problem in the homological persistence literature, is the relation of the topological complexity of a given input data and the persistence computation times. Since matrix reduction is a global operation, the structure of the underlying Morse–Smale complex is crucial. This structure also depends on the imaging process and the data format. For instance, the aneurysm and bonsai data are given as 8-bit integer while the prone and supine data are 16-bit integer CT scans. This may also contribute to the different timings shown in Table 1.

Acknowledgements This work was supported by the MPI of Biochemistry, the MPI for Informatics, the DFG Emmy-Noether research program, and Foundation for Polish Science Geometry and Topology in Physical Models program. We thank Daniel Baum for providing the molecule data set. We also thank Herbert Edelsbrunner and Chao Chen for many fruitful discussions.

Appendix

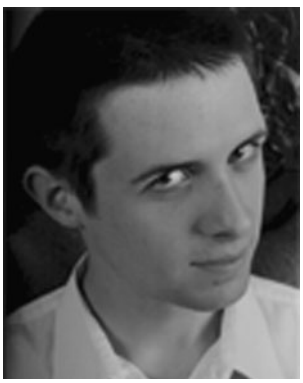
Let $\Omega = [-2, 2]^3$. The function $g{:}\Omega \to \mathbb{R}$ is given by

$$\begin{aligned} g(x, y, z) = {} & 1 \sin(1\,x) \sin(2\,y) \sin(3\,z) \\ & + 2 \sin(2\,x) \sin(1\,y) \sin(3\,z) \\ & + 3 \sin(3\,x) \sin(2\,y) \sin(1\,z) \\ & + 4 \sin(1\,x) \sin(3\,y) \sin(2\,z) \\ & + 5 \sin(2\,x) \sin(3\,y) \sin(1\,z) \\ & + 6 \sin(3\,x) \sin(1\,y) \sin(2\,z) \\ & + 1 \cos(3\,x) \cos(1\,y) \cos(2\,z) \\ & + 2 \cos(2\,x) \cos(1\,y) \cos(3\,z) \\ & + 3 \cos(1\,x) \cos(2\,y) \cos(3\,z) \\ & + 4 \cos(3\,x) \cos(2\,y) \cos(1\,z) \\ & + 5 \cos(2\,x) \cos(3\,y) \cos(1\,z) \\ & + 6 \cos(1\,x) \cos(3\,y) \cos(2\,z). \end{aligned}$$

References

1. The Institute of Computer Graphics and Algorithms. http://www.cg.tuwien.ac.at/research/vis/datasets/
2. Volvis: Voxel Data Repository (2010). http://www.volvis.org/
3. Bauer, U., Lange, C., Wardetzky, M.: Optimal topological simplification of discrete functions on surfaces. Discrete & Computational Geometry **47**(2), 1–31
4. Bendich, P., Edelsbrunner, H., Kerber, M.: Computing robustness and persistence for images. Proc. - IEEE Conf. Inf. Vis. **16**, 1251–1260 (2010)
5. Chari, M.K.: On discrete Morse functions and combinatorial decompositions. Discrete Math. **217**(1–3), 101–113 (2000)
6. Chen, C., Kerber, M.: An output-sensitive algorithm for persistent homology. In: Proceedings of the 27th Annual ACM Symposium on Computational Geometry (SoCG '11), pp. 207–216. ACM, New York (2011)
7. Chen, C., Kerber, M.: Persistent homology computation with a twist. In: 27th European Workshop on Comp. Geometry (EuroCG 2011) (2011)
8. Cohen-Steiner, D., Edelsbrunner, H., Harer, J.: Stability of persistence diagrams. Discrete Comput. Geom. **37**, 103–120 (2007)
9. Edelsbrunner, H., Harer, J.: Computational Topology. An Introduction. American Mathematical Society (2010)
10. Edelsbrunner, H., Letscher, D., Zomorodian, A.: Topological persistence and simplification. Discrete Comput. Geom. **28**(4), 511–533 (2002)
11. Forman, R.: Morse theory for cell complexes. Adv. Math. **134**, 90–145 (1998)
12. Forman, R.: A user's guide to discrete Morse theory. In: Seminaire Lotharingien de Combinatoire, vol. B48c, pp. 1–35 (2002)
13. Günther, D., Reininghaus, J., Prohaska, S., Weinkauf, T., Hege, H.C.: Efficient computation of a hierarchy of discrete 3d gradient vector fields. In: Peikert, R., Hauser, H., Carr, H., Fuchs, R. (eds.) Topological Methods in Data Analysis and Visualization. II. Mathematics and Visualization (TopoInVis 2011), Zürich, Switzerland, April 4–6, pp. 15–30. Springer, Berlin (2012)
14. Günther, D., Reininghaus, J., Wagner, H., Hotz, I.: Memory efficient computation of persistent homology for 3D image data using discrete Morse theory. In: Lewiner, T., Torres, R. (eds.) Proceedings of Conference on Graphics, Patterns and Images (SIBGRAPI), Maceió, Los Alamitos, vol. 24, pp. 25–32. IEEE Press, New York (2011)
15. Gyulassy, A., Bremer, P.T., Hamann, B., Pascucci, V.: A practical approach to Morse–Smale complex computation: scalability and generality. IEEE Trans. Vis. Comput. Graph. **14**, 1619–1626 (2008)
16. Hatcher, A.: Algebraic Topology. Cambridge University Press, Cambridge (2002)
17. Kaczynski, T., Mischaikow, K., Mrozek, M.: Computational Homology. Applied Math. Sciences, vol. 157. Springer, Berlin (2004)
18. Lewiner, T.: Geometric discrete Morse complexes. Ph.D. thesis, Dept. of Mathematics, PUC-Rio (2005)
19. Lewiner, T., Lopes, H., Tavares, G.: Optimal discrete Morse functions for 2-manifolds. Comput. Geom. **26**(3), 221–233 (2003)
20. Milnor, J.: Topology from the Differentiable Viewpoint. University of Virginia Press, Charlottesville (1965)
21. Milosavljević, N., Morozov, D., Skraba, P.: Zigzag persistent homology in matrix multiplication time. In: Proceedings of the 27th Annual ACM Symposium on Computational Geometry (SoCG '11), pp. 216–225. ACM, New York (2011)
22. Morozov, D.: Persistence algorithm takes cubic time in the worst case. In: BioGeometry News. Duke Computer Science, Durhmam (2005)
23. Mrozek, M., Wanner, T.: Coreduction homology algorithm for inclusions and persistent homology. Comput. Math. Appl. **60**, 2812–2833 (2010)
24. Reininghaus, J., Günther, D., Hotz, I., Prohaska, S., Hege, H.C., TADD: A computational framework for data analysis using discrete Morse theory. In: Mathematical Software (ICMS 2010), pp. 198–208. Springer, Berlin (2010)
25. Robins, V., Wood, P., Sheppard, A.: Theory and algorithms for constructing discrete Morse complexes from grayscale digital images. IEEE Trans. Pattern Anal. Mach. Intell. **33**(8), 1646–1658 (2011)
26. Röttger, S.: http://www9.informatik.uni-erlangen.de/External/vollib/
27. Wagner, H., Chen, C., Vucini, E.: Efficient computation of persistent homology for cubical data. In: Peikert, R., Hauser, H., Carr, H., Fuchs, R. (eds.) Topological Methods in Data Analysis and Visualization. II. Mathematics and Visualization (TopoInVis 2011), Zürich, Switzerland, April 4–6, pp. 91–108. Springer, Berlin (2012)

David Günther studied mathematics with a focus on nonlinear finite element calculus at the Humboldt University of Berlin, Germany, where he received his M.S. degree in 2007. He is a main author of OpenFFW, an open source finite element framework in Matlab. Currently, he is a junior researcher in the research group "Feature-Based Data Analysis for Computer Graphics and Visualization" at the Max-Planck Institute for Informatics in Saarbrücken, Germany, where he is working on his Ph.D. thesis. His research interests are topological data analysis, discrete Morse theory, and mathematical programming.



Jan Reininghaus received the M.S. degree in Mathematics from the Humboldt University of Berlin, Germany, for a thesis on the numerical treatment of Maxwell's equations. He is a main author of OpenFFW, an open source finite element framework written in Matlab. Currently he is with the Scientific Visualization Department of Zuse Institute Berlin (ZIB) where he is working on his Ph.D. thesis. His current research interests include Hodge theory, volume rendering, finite element exterior calculus, and discrete Morse theory.

Hubert Wagner received the M.S degree in Computer Science from the Jagiellonian University, Krakow, Poland. Currently he is a Ph.D. student at the same university, supervised by Professor Marian Mrozek. He is a coauthor of the CAPD/RedHom library for homology computations. His current research interests include computational topology, discrete Morse theory with applications to image, and text data analysis.

Ingrid Hotz Ingrid Hotz received the M.S. degree in theoretical Physics from the Ludwig Maximilian University in Munich Germany and the Ph.D. degree from the Computer Science Department at the University of Kaiserslautern, Germany. During 2003–2006, she worked as a postdoctoral researcher at the Institute for Data Analysis and Visualization (IDAV) at the University of California. Currently, she is the leader of a junior research group at the Zuse Institute in Berlin, Germany. Her research interests are in the area of data analysis and scientific visualization with focus on tensor and vector fields.