An output-sensitive algorithm for persistent homology

Chao Chen

Institute of Science and Technology (IST) Austria, Klosterneuburg, Austria

Michael Kerber

Institute of Science and Technology (IST) Austria, Klosterneuburg, Austria

Abstract

In this paper, we present the first output-sensitive algorithm to compute the persistence diagram of a filtered simplicial complex. For any $\Gamma > 0$, it returns only those homology classes with persistence at least Γ. Instead of the classical reduction via column operations, our algorithm performs rank computations on submatrices of the boundary matrix. For an arbitrary constant $\delta \in (0, 1)$, the running time is $O(C_{(1-\delta)\Gamma} R_d(n) \log n)$, where $C_{(1-\delta)\Gamma}$ is the number of homology classes with persistence at least $(1-\delta)\Gamma$, n is the total number of simplices in the complex, d its dimension, and $R_d(n)$ is the complexity of computing the rank of an $n \times n$ matrix with $O(dn)$ nonzero entries. Depending on the choice of the rank algorithm, this yields a deterministic $O(C_{(1-\delta)\Gamma} n^{2.376})$ algorithm, a $O(C_{(1-\delta)\Gamma} n^{2.28})$ Las-Vegas algorithm, or a $O(C_{(1-\delta)\Gamma} n^{2+\epsilon})$ Monte-Carlo algorithm for an arbitrary $\epsilon > 0$. The space complexity of the Monte-Carlo version is bounded by $O(dn) = O(n \log n)$.

Keywords: computational topology, persistent homology, randomized algorithms, rank computation

1. Introduction

Persistent homology is a general framework to measure the relevance of topological features of a space with respect to a function. Its ability to handle noisy data and to provide homological information in arbitrary dimensions

Email addresses: `chao.chen@ist.ac.at` (Chao Chen), `mkerber@ist.ac.at` (Michael Kerber) has turned it into a successful tool for the analysis of various types of data; see [1, 2, 3, 4, 5, 6] for applications in image analysis, sensor networks, and biological research. Parallel to applications, the theoretical basis of persistent homology has grown during the last decade, e.g. [7, 8, 9, 10, 11, 12, 13]. See also [14] for a recent textbook covering both theory and applications.

We are concentrating on the computation of persistent homology of a simplicial complex over $\mathbb{Z}_2$ in this work. The first algorithm by Edelsbrunner et al. [15] as well as other related algorithms [16, 10], are based on column-wise matrix reduction of the boundary matrix of the simplicial complex. Their running time is $O(n^3)$, where n is the number of simplices of the given complex, and Morozov [17] provides an example where the algorithm indeed has cubic complexity. A simple optimization of that algorithm which avoids columns operations on about half of the columns has been presented in [18]. Milosavljević et al. [19] present an algorithm to compute persistence in matrix multiplication time $O(n^\omega)$, where the best estimation of ω is currently 2.376 [20]. Better bounds are only known for the special case of 0-dimensional homology, for which a complexity of $O(n\alpha(n))$ can be achieved by union-find, where $\alpha(\cdot)$ is the inverse of the Ackermann function [14]. Numerous efficient solution, in theory and practice, have been presented for related problems such as computing the Reeb graph of a simplicial complex [21] and computing persistence on special structures like clique complexes [22], point clouds in higher dimensions [23], regular cubical grids [24], or dual complexes of cubical subdivisions in $\mathbb{R}^3$ [25].

In this paper, we present the first output-sensitive algorithm for computing homology classes above a predefined threshold: Given some filtration, namely, a d-dimensional simplicial complex filtered according to a filter function, and $\Gamma > 0$, our algorithm only returns those homology classes within the filtration that have a persistence of at least Γ. In various applications, classes with low persistence are considered as noise and one is only interested in the (typically few) classes with high persistence. Denoting by $C_{(1-\delta)\Gamma}$ the number of homology classes with persistence at least $(1-\delta)\Gamma$ for some $\delta > 0$, our algorithm has a running time of $O(C_{(1-\delta)\Gamma} R_d(n) \log n)$, where $R_d(n)$ is the complexity of computing the rank of an $n \times n$ binary matrix with $O(dn)$ nonzero entries. In this bound we assume δ to be a constant; a more detailed bound covering the dependence on δ is derived in the body of the paper.

Our algorithm is the first one that does not transform the boundary matrix into a reduced form (by row/column operations or matrix multiplications). Instead, it accesses the boundary matrix exclusively by computing the

Type	Ref.	Rank Complexity	Persistence Computation
Monte-Carlo	[26]	$O(n^2 \log^2 n \log\log n)$	$O(C_{(1-\delta)\Gamma} n^2 \log^3 n \log\log n)$
Las-Vegas	[27]	$\widetilde{O}(n^{3-1/(\omega-1)}) = O(n^{2.28})$	$O(C_{(1-\delta)\Gamma} n^{2.28})$
Deterministic	[28]	$O(n^\omega)= O(n^{2.376})$	$O(C_{(1-\delta)\Gamma} n^\omega \log n)$

Table 1: Time bounds of our algorithm using different types of rank computation algorithms. $\widetilde{O}$ means that logarithmic factors are omitted in the bound.

ranks of submatrices. We obtain different types of results depending on the choice of the rank algorithm in our method as shown in Table 1. The running time of the Las-Vegas algorithm is only in expectation, and the Monte-Carlo algorithm has a success probability of at least q for any predefined constant $q < 1$.

None of the presented algorithms improves the worst-case complexity for persistence computation in general because $C_{(1-\delta)\Gamma}$ can be as large as $n/2$. However, under the not too unrealistic assumption that the number of relevant persistent homology class is small (say, logarithmic in n or even constant), we obtain (randomized) algorithms with best known complexity. Furthermore, the Monte-Carlo version only needs $O(dn) = O(n \log n)$ memory, independent of the choice of Γ; the standard persistence algorithm needs $O(n^2)$ memory.

The randomized variants need to switch over to an extension field of $\mathbb{Z}_2$ with $O(n^2 \log n)$ elements for the computation, and the bounds refer to the number of arithmetic operations in this finite field. When considering bit complexities (i.e., number of bit operations performed in the algorithm), another factor of $O(\log n \log\log n \log\log\log n)$ appears in the bound. We emphasize that although the base field changes internally, the final result still yields persistence for $\mathbb{Z}_2$-homology.

The paper is organized as follows: We revisit the most relevant concepts from the theory of persistent homology in Section 2. We then give a method to compute certain areas of the persistence diagram by rank computations in Section 3. The main algorithm and its analysis is described afterwards; we split the discussion into the persistent inessential classes in Section 4 and the essential classes in Section 5. We discuss more details of how to achieve the running times from Table 1 in Section 6. In Section 7, we conclude with final remarks.

2. Preliminaries

In this section, we review persistent homology to the extent it is needed for this work. We assume that the reader is familiar with the basic notions of homology groups of topological spaces and of simplicial complexes. We refer to [14, 29] for introductory textbooks. We focus on homology over $\mathbb{Z}_2$ in this work.

Persistent Homology. Given a topological space $\mathbb{X}$ and a continuous *filter function* $f : \mathbb{X} \to \mathbb{R}$, we call $\mathbb{X}_t = f^{-1}(-\infty, t]$ the sublevel set of f for t. We denote $\mathsf{H}_p(\mathbb{X}_t)$ as the homology group of $\mathbb{X}_t$ in dimension p, and $\mathsf{H}(\mathbb{X}_t) = \bigoplus_{p\geq 0} \mathsf{H}_p(\mathbb{X}_t)$ be the direct sum of all homology groups. For $s \leq t$, there is a homomorphism $f^{s,t} : \mathsf{H}(\mathbb{X}_s) \to \mathsf{H}(\mathbb{X}_t)$ on homology groups that is induced by inclusion of the sublevel sets. *Persistent homology* tracks homological changes of the sublevel sets by capturing the *birth* and *death* times of homology classes of $\mathbb{X}_t$ as t grows from $-\infty$ to $+\infty$ in the following sense: A homology class α is born at s if α is in $\mathsf{H}(\mathbb{X}_s)$, but not in the image of $f^{s-\epsilon,s}$ for any $\epsilon > 0$. Such a class dies at some $t \geq s$, if t is the smallest value such that $f^{s,t}(\alpha) \in \mathrm{Im} f^{s-\epsilon,t}$ for some $\epsilon > 0$. In other words, the homology class α either becomes trivial or identical to another class that was born earlier. The *persistence*, or lifetime, of the class α is defined as $t - s$, the difference between its birth and death time. Intuitively, the classes with large persistence reveal information about the global structure of $\mathbb{X}$ filtered by the function f.

In computation, the input is usually a simplicial complex K (of size n) with a filter function $f : K \to \mathbb{R}$ that assigns each simplex a real value, with the constraint that the function value of a simplex is no smaller than those of its faces. We write all simplices in K in ascending order according to their filter function values, under the condition that a simplex has to be after his faces in the order. Denote $f_i = f(\sigma_i)$ for $i \geq 1$, $f_0 = -\infty$, and $K_i = f^{-1}(-\infty, f_i]$. We have a *filtration* of K, namely, a nested sequence of subcomplexes, $\emptyset = K_0 \subseteq K_1 \subseteq \ldots \subseteq K_n = K$. This filtration is not uniquely determined by the filter function if simplices have the same function value, but possible ties can be broken arbitrarily without changing the persistence diagram (as defined later). We can imagine that the simplices with function value f_j are inserted into K_{j-1} one by one. Then, the addition of a simplex σ_j to K_{j-1} either causes a birth or a death of exactly one homology class. In the former case, the simplex is called *creator*, in the latter case it is called *destroyer*. Homology classes in the filtration can thus be represented as *persistence pairs* (σ_i, σ_j) with $i < j$, meaning that the class is created when σ_i is added, and destroyed when σ_j is added. For convenience, we say σ_i *creates* and σ_j *destroys* the class. Also, we say a persistence pair is created (resp. destroyed) *in the index range* $[i_1, i_2]$ when the creator (resp. destroyer) has an index between i_1 and i_2.

An *essential class* is a class that is created in the filtration, but never destroyed later on. These classes exactly define the homology of K. We define the persistence of an essential class to be ∞.

Boundary matrix. Given a filtration and the corresponding ordered sequence of simplices, $\sigma_1, \cdots, \sigma_n$, the *(ordered) boundary matrix* ∂ is the $n \times n$ binary matrix defined by $\partial_{i,j} = 1$ if and only if σ_i is a face of σ_j of codimension 1. In other words, the i-th column of ∂ encodes the boundary of the simplex σ_i. Because a simplex enters a filtration after its boundary by definition, ∂ is upper-triangular. Also, each column has no more than $(d + 1)$ nonzero entries, where d is the dimension of the complex. Therefore, ∂ has $O(dn)$ nonzero entries, and any $k \times k$ submatrix of ∂ has $O(dk)$ nonzero entries. It is easy to see that $d \leq \log(n + 1) - 1$ because a d-simplex has $2^{d+1} - 1$ faces, and thus,

$$O(dn) = O(n \log n). \tag{1}$$

This sparsity of the boundary matrices will be exploited in Section 6. This sparsity was also exploited by Chen and Freedman [30, 31], in which the Wiedemann's algorithm was used for homology localization.

Reduction algorithm. For a non-zero column i, of a matrix, we call its *lowest entry* the largest row index of a nonzero entry, denoted as $\mathrm{low}(i)$. The standard persistence algorithm [14] performs left-to-right column reductions on the boundary matrix. For a column i, assuming that columns $1, \ldots, i-1$ are already reduced, we reduce i as follows. If there exists a reduced column j, $j < i$, such that $\mathrm{low}(j) = \mathrm{low}(i)$, subtract j from i. This is repeated until either the i-th column becomes zero, or $\mathrm{low}(i)$ is unique among the columns $1, \ldots, i$. Let $\mathcal{R}$ be the matrix that is obtained after applying this algorithm to all columns. The i-th column of $\mathcal{R}$ is 0 if and only if σ_i is a creator [14, §VII.1]. Moreover, let the i-th column of $\mathcal{R}$ be nonzero, and let $j = \mathrm{low}(i)$ denote its lowest entry. Then, (σ_j, σ_i) is a persistence pair. Unpaired simplices create essential homology classes. The given algorithm has cubic running time in the size of the complex.

The persistence diagram. Persistent homology can be described using a *persistence diagram*, $\mathrm{Dgm}(f)$, which is a multiset of points in $\mathbb{R} \times (\mathbb{R} \cup \{+\infty\})$; we call these points the *dots* of the diagram. In the simplicial complex case, the diagram contains a dot (f_i, f_j) for every persistent pair (σ_i, σ_j), and a dot (f_i, ∞) for every essential class that is created by σ_i. Of course, two persistence pairs (as well as two essential classes) might define the same dot; we call the *multiplicity* of a dot to be the number of persistence pairs that it represents. Often, the infinitely many points on the diagonal are also considered to belong to the diagram, but we will ignore them for the rest of the paper. The persistence of a homology class is the horizontal (or vertical) distance of the representing dot from the diagonal line. Therefore, homology classes with high persistence are further away from the diagonal. The persistence diagram is stable under perturbations of the filter function [9].

3. Computing Persistence Pairs

From now on, we assume that a simplicial complex K, its dimension d, and a filter function f are fixed, as well as a filtration of K with respect to f, determined by the simplex order $\sigma_1, \ldots, \sigma_n$. We let P denote the set of all persistence pairs of the form (σ_i, σ_j), that means, simplex σ_j destroys the homology class created by σ_i.

μ-queries. Instead of performing row or column operations on the ordered boundary matrix, ∂, we will access it exclusively by querying *interval multiplicities (μ-values)* of the filtered complex.

Definition 1. *For integers $i_1 \leq i_2 \leq j_1 \leq j_2$,*

$$\begin{aligned} P_{i_1,i_2}^{j_1,j_2} &:= \{(\sigma_k, \sigma_\ell) \in P \mid k \in [i_1, i_2] \wedge \ell \in [j_1, j_2]\} \\ \mu_{i_1,i_2}^{j_1,j_2} &:= \mathrm{Card}\, P_{i_1,i_2}^{j_1,j_2} \end{aligned}$$

In words, $P_{i_1,i_2}^{j_1,j_2}$ is the set of persistence pairs whose creators (resp. destroyers) fall into the index range $[i_1, i_2]$ (resp. $[j_1, j_2]$). In the persistence diagram, this corresponds to the set of dots within the box $[f_{i_1}, f_{i_2}] \times [f_{j_1}, f_{j_2}] \subseteq \mathbb{R}^2$, except for dots on the boundary whose creators (resp. destroyers) have out-of-range indices. In particular, for a dot on the boundary corresponding to multiple persistence pairs, only a fraction of them belong to $P_{i_1,i_2}^{j_1,j_2}$.

We show next that $\mu_{i_1,i_2}^{j_1,j_2}$ is determined by the ranks of 4 submatrices of ∂ with at most $j_2 - i_1 + 1$ columns (resp. rows). The result generalizes the "Pairing Uniqueness Lemma" from [32] which handles the special case of $i_1 = i_2$, $j_1 = j_2$.

For any matrix A, denote $A^{b_1,b_2}_{a_1,a_2}$ as the $(a_2-a_1+1)\times(b_2-b_1+1)$-submatrix of A consisting of rows $a_1, a_1+1, \cdots, a_2$ and columns $b_1, b_1+1, \cdots, b_2$.

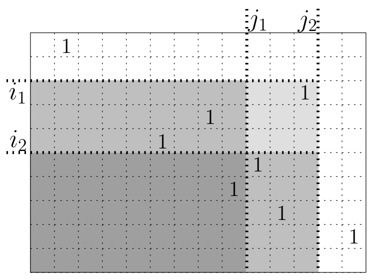


Figure 1: Part of the completely reduced matrix $\mathcal{S}$. $\mu^{j_1,j_2}_{i_1,i_2}$ counts the number of 1's in the light gray submatrix.

Theorem 2 (μ-query)**.**

$$\begin{aligned}\mu^{j_1,j_2}_{i_1,i_2} \;=\;& \operatorname{rank}(\partial^{i_1,j_2}_{i_1,j_2}) - \operatorname{rank}(\partial^{i_1,j_1-1}_{i_1,j_1-1}) - \\ & \operatorname{rank}(\partial^{i_2+1,j_2}_{i_2+1,j_2}) + \operatorname{rank}(\partial^{i_2+1,j_1-1}_{i_2+1,j_1-1}). \qquad (2)\end{aligned}$$

Proof. Consider the reduced matrix $\mathcal{R}$ which is constructed from ∂ by left-to-right column operations, as described in Section 2. We transform $\mathcal{R}$ into $\mathcal{S}$ by setting all entries in a nonzero column to zero except the lowest entry (Figure 1). It is straightforward to see by induction that $\mathcal{S}$ arises from $\mathcal{R}$ by a sequence of bottom-up row operations. Moreover, $\mathcal{S}$ has a 1 at position (i,j) if and only if (σ_i, σ_j) form a persistence pair. Therefore, $\mu^{j_1,j_2}_{i_1,i_2}$ is equal to the number of 1's in $\mathcal{S}^{j_1,j_2}_{i_1,i_2}$. By inclusion-exclusion,

$$\begin{aligned}\mu^{j_1,j_2}_{i_1,i_2} =& \left(\#\text{ 1's in } \mathcal{S}^{1,j_2}_{i_1,n}\right) - \left(\#\text{ 1's in } \mathcal{S}^{1,j_1-1}_{i_1,n}\right) - \\ & \left(\#\text{ 1's in } \mathcal{S}^{1,j_2}_{i_2+1,n}\right) + \left(\#\text{ 1's in } \mathcal{S}^{1,j_1-1}_{i_2+1,n}\right). \qquad (3)\end{aligned}$$

Since each row and column of $\mathcal{S}$ has at most one nonzero entry, the number of 1's in a submatrix of $\mathcal{S}$ is equal to its rank. Since we transform ∂ into $\mathcal{S}$ by a sequence of left-to-right column operations and bottom-up row operations, any lower-left submatrix of ∂ and its corresponding submatrix of $\mathcal{S}$ have the same rank. Therefore we can replace the first term in the right hand side of (3) with $\mathrm{rank}(\partial^{1,j_2}_{i_1,n})$, which is equal to $\mathrm{rank}(\partial^{i_1,j_2}_{i_1,j_2})$ because ∂ is upper-triangular. This also applies to the other three terms and thus leads to (2). $\square$

Corollary 3. *Let $R_d(k)$ denote the cost for computing the rank of a $k \times k$ square matrix with $O(dk)$ nonzero entries. Then, computing $\mu^{j_1,j_2}_{i_1,i_2}$ has a complexity of $O(R_d(j_2 - i_1 + 1))$.*

Computing $P^{j_1,j_2}_{i_1,i_2}$. Next, we compute $P^{j_1,j_2}_{i_1,i_2}$ with a sequence of μ-queries. Our algorithm proceeds in two steps. First, we compute the index of each creating simplex of the set (the first element of a pair). Second, we compute the index of the corresponding destroying simplex for every creating simplex.

To compute all creating simplices in $P^{j_1,j_2}_{i_1,i_2}$, we consider a binary tree whose nodes are subintervals of $[i_1, i_2]$. The root is the whole interval $[i_1, i_2]$ and the children of an interval $[a, b]$ are $[a, m]$ and $[m+1, b]$, where $m = \lfloor \frac{a+b}{2} \rfloor$. The leaves are the singleton intervals. Obviously, each $[i, i]$ with $i_1 \leq i \leq i_2$ appears as a leaf in the tree. We call such a tree the *bisection tree* of $[i_1, i_2]$ (Figure 2).

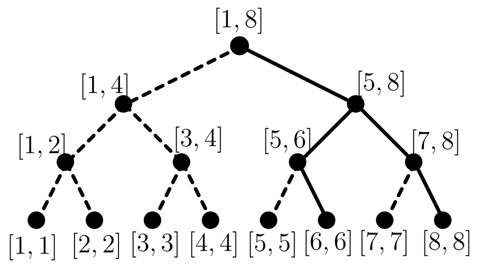


Figure 2: An illustration of the bisection tree of an interval $[i_1, i_2] = [1, 8]$, which contains two creators, σ_6 and σ_8. Only the nodes on the two solid paths (and their siblings) are explored by the algorithm.

For any node $[a, b]$ of the bisection tree, we define its *μ-value* to be $\mu^{j_1,j_2}_{a,b}$. The μ-value of every internal node equals the sum of μ-values of its two children, and a leaf has a μ-value of either 0 or 1. By construction, σ_i is a creating simplex in $P^{j_1,j_2}_{i_1,i_2}$ if and only if the μ-value of the leaf $[i, i]$ in the bisection tree is 1. Therefore, computing the creating simplices is simply computing the leaves of the bisection tree with μ-value 1.

To find these leaves, we compute the μ-value of the root and explore the tree recursively as follows. Let I be a node with known μ-value, denoted as

μ. If $\mu = 0$, we do nothing. Otherwise, if I is a leaf, we add I to the output list. If I has two children I_1 and I_2, we compute μ_1, the value of I_1. Then, the value of I_2 is given by $\mu - \mu_1$, and we recurse on I_1 and I_2. See Figure 2 for an illustration of which part of the tree is explored by the algorithm.

Lemma 4. *Computing the creating simplices of $P_{i_1,i_2}^{j_1,j_2}$ has a complexity of*

$$O((1 + \mu_{i_1,i_2}^{j_1,j_2} \log(i_2 - i_1 + 1))R_d(j_2 - i_1 + 1)).$$

Proof. We perform one μ-query initially for the root, and then one μ-computation per non-leaf node whose value is nonzero. The tree has nonzero μ-value only at nodes which lie on a path from the root to a nonzero leaf. Each path is at most $\log(i_2 - i_1 + 1)$ long. There are at most $\mu_{i_1,i_2}^{j_1,j_2}$ such paths, and thus at most $(1 + \mu_{i_1,i_2}^{j_1,j_2} \log(i_2 - i_1 + 1))$ μ-queries are executed. The claim follows by Corollary 3. $\square$

In the second step, we compute the destroyer simplex for each creator computed in step 1. Let us fix a creating simplex σ_i. By definition $\mu_{i,i}^{j_1,j_2} = 1$, and the index of its destroying counterpart is the unique integer $j \in [j_1, j_2]$ with $\mu_{i,i}^{j,j} = 1$. Clearly, we can find j by a binary search on $[j_1, j_2]$ which needs $O(\log(j_2 - j_1 + 1)R_d(j_2 - i_1 + 1))$ time. Doing this for all creating simplices of $P_{i_1,i_2}^{j_1,j_2}$ can be done in $O(\mu_{i_1,i_2}^{j_1,j_2} \log(j_2 - j_1 + 1)R_d(j_2 - i_1 + 1))$. We summarize the results of this section with the following theorem.

Theorem 5. *For $i_1 \le i_2 \le j_1 \le j_2$, computing $P_{i_1,i_2}^{j_1,j_2}$ has a complexity of*

$$O((1 + \mu_{i_1,i_2}^{j_1,j_2} \log(j_2 - i_1 + 1))R_d(j_2 - i_1 + 1)).$$

4. Computing Γ-persistence: Inessential classes

Our goal is to compute points with persistence at least Γ for a fixed $\Gamma > 0$. There are two types of such classes, essential classes (whose persistence is ∞ by definition) and inessential classes whose birth and death time are at least Γ apart. We will split the discussion into two sections, starting with the inessential classes and postponing the essential case to Section 5.

Definition 6. *We say that a persistence pair (σ_i, σ_j) is* Γ-persistent *if $f_j - f_i \ge \Gamma$. We also set*

$$\begin{aligned} P(\Gamma)_{i_1,i_2}^{j_1,j_2} &:= \{(\sigma_i, \sigma_j) \in P_{i_1,i_2}^{j_1,j_2} \mid f_j - f_i \ge \Gamma\}, \\ \mu(\Gamma)_{i_1,i_2}^{j_1,j_2} &:= \operatorname{Card} P(\Gamma)_{i_1,i_2}^{j_1,j_2} \end{aligned}$$

We compute all the Γ-persistent pairs with a divide-and-conquer strategy: Let $[a, b]$ denote an index range, initially set to $[1, n]$. To compute all Γ-persistent pairs living within the index range $[a, b]$ (namely, created and destroyed within $[a, b]$), we set $m := \lfloor \frac{a+b}{2} \rfloor$ and first compute the set $P(\Gamma)_{a,m}^{m,b}$, that is, all Γ-persistent pairs created in the range $[a, m]$ and destroyed in the range $[m, b]$. Then, we recursively compute the Γ-persistent pairs living in the ranges $[a, m]$ and $[m, b]$. In each recursion step, the persistence diagram of a certain rectangular area of $\mathbb{R}^2$ is explored; see Figure 3 for an illustration of the areas that are explored in the recursion steps, numbered by $B1, \ldots, B7$. There are at most $2n - 1$ such areas explored by the algorithm.

Computing $P(\Gamma)_{a,m}^{m,b}$ in an output-sensitive fashion. The main challenge is to ignore pairs with small persistence during computation. We achieve that by computing persistent pairs within two (or more) different boxes. See Figure 4 for an illustration of the idea: We want to compute $P(\Gamma)_{a,m}^{m,b}$, that is, all dots in the dark shaded polygon in Figure 4. Instead, we compute the dots within the two boxes in Figure 5. Note that their union contains all of $P(\Gamma)_{a,m}^{m,b}$, whereas each box is at least $\Gamma/2$ away from the diagonal.

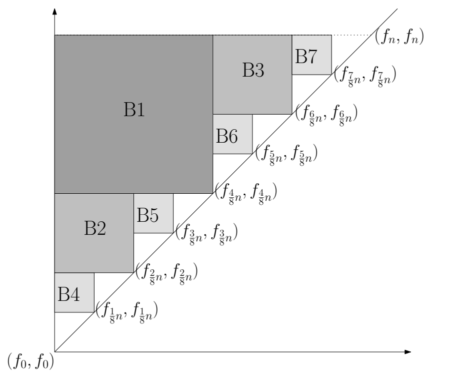
To formalize this idea, let $m^- < m$ be the maximal index such that $f_m - f_{m^-} \geq \Gamma/2$. Likewise, let $m^+ > m$ be defined as the minimal index with $f_{m^+} - f_m \geq \Gamma/2$. The two boxes in Figure 5 correspond to $P_{a,m^-}^{m,b}$ and $P_{m^-+1,m}^{m^+,b}$ and we have:

Lemma 7. *Each pair in $P(\Gamma)_{a,m}^{m,b}$ is created in $[a, m^-]$ or destroyed in $[m^+, b]$. More precisely,*

$$P(\Gamma)_{a,m}^{m,b} \subseteq P_{a,m^-}^{m,b} \sqcup P_{m^-+1,m}^{m^+,b} \subseteq P(\Gamma/2)_{a,m}^{m,b}.$$

Proof. For the first inclusion, assume for a contradiction that a pair in $P(\Gamma)_{a,m}^{m,b}$ is created in $(m^-, m]$ and destroyed in $[m, m^+)$. Then, its persistence is obviously bounded by $f_{m^+-1} - f_{m^-+1} = f_{m^+-1} - f_m + f_m - f_{m^-+1} < \Gamma$, a contradiction. The second inclusion is obvious from the definition of m^- and m^+. $\square$

By the previous lemma, it suffices to compute all persistence pairs created in $[a, m^-]$ and destroyed in $[m, b]$ and all persistence pairs created in $[m^- + 1, m]$ and destroyed in $[m^+, b]$. Among these, we only output the ones with persistence at least Γ.



B1
B2
B3
B4
B5
B6
B7
(f_n, f_n)
$(f_{\frac{7}{8}n}, f_{\frac{7}{8}n})$
$(f_{\frac{6}{8}n}, f_{\frac{6}{8}n})$
$(f_{\frac{5}{8}n}, f_{\frac{5}{8}n})$
$(f_{\frac{4}{8}n}, f_{\frac{4}{8}n})$
$(f_{\frac{3}{8}n}, f_{\frac{3}{8}n})$
$(f_{\frac{2}{8}n}, f_{\frac{2}{8}n})$
$(f_{\frac{1}{8}n}, f_{\frac{1}{8}n})$
(f_0, f_0)

Figure 3: Compute persistence pairs in a divide and conquer fashion.

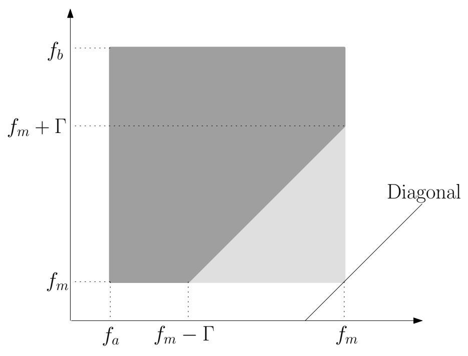


Figure 4: The dark gray polygon contains all the Γ-persistence pairs created in $[a, m]$ and destroyed in $[m, b]$.

By Theorem 5, the computation of $P^{m,b}_{a,m^-}$ and $P^{m^+,b}_{m^-+1,m}$ is bounded in total by

$$O\left(\left(1 + (\mu^{m,b}_{a,m^-} + \mu^{m^+,b}_{m^-+1,m}) \log(b-a+1)\right) R_d(b-a+1)\right).$$

By the second inclusion of Lemma 7, we have that $\mu^{m,b}_{a,m^-} + \mu^{m^+,b}_{m^-+1,m} \leq \mu(\Gamma/2)^{m,b}_{a,m}$. So, the running time simplifies to

$$O\left(\left(1 + \mu(\Gamma/2)^{m,b}_{a,m} \log(b-a+1)\right) R_d(b-a+1)\right).$$

We further improve the bound by replacing $\mu(\Gamma/2)^{m,b}_{a,m}$ by $\mu((1-\delta)\Gamma)^{m,b}_{a,m}$ for any $\delta \in (0,1)$. For that, we generalize Lemma 7 as follows: Fix $t := \lceil \frac{1}{\delta} \rceil$ and define a monotonously increasing sequence of subdivision points $m^-_1, \ldots, m^-_{t-1}$ as follows: $m^-_i < m$ is the maximal index such that $f_m - f_{m^-_i} \geq \Gamma \cdot (1 - \frac{i}{t})$. Similarly, we define a monotonously decreasing sequence $m^+_1, \ldots, m^+_{t-1}$ with $m^+_i > m$ as the minimal index such that $f_{m^+_i} - f_m \geq \Gamma \cdot (1 - \frac{i}{t})$. Note that this indeed generalizes the previous construction, where we had $t = 2$ and $m^-_1 = m^-$ and $m^+_1 = m^+$. We also define $m^-_t := m^+_t := m$ and $m^-_0 := a - 1$. In this situation, we have that

$$P(\Gamma)^{m,b}_{a,m} \subseteq \bigsqcup_{i=1}^{t} P^{m^+_{t-i+1},b}_{m^-_{i-1}+1,m^-_i} \subseteq P((1-1/t)\Gamma)^{m,b}_{a,m} \subseteq P((1-\delta)\Gamma)^{m,b}_{a,m}.$$

See Figure 6 for a case when $t = 4$. Here $f_{m^-_1}$, $f_{m^-_2}$, and $f_{m^-_3}$ equal to $f_m - 3\Gamma/4$, $f_m - \Gamma/2$ and $f_m - \Gamma/4$, respectively. Whereas $f_{m^+_1}$, $f_{m^+_2}$, and $f_{m^+_3}$ equal to $f_m + 3\Gamma/4$, $f_m + \Gamma/2$ and $f_m + \Gamma/4$, respectively.

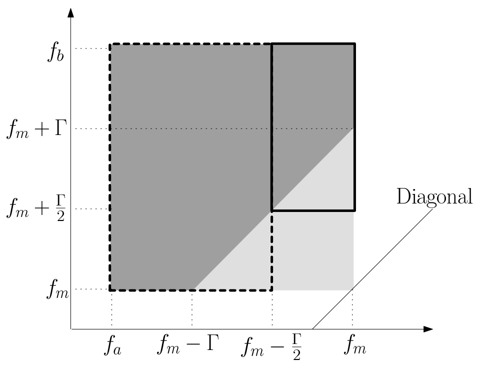


Figure 5: The two rectangles contain all the Γ-persistence dots we want to compute, and some extra Γ/2-persistence pairs.

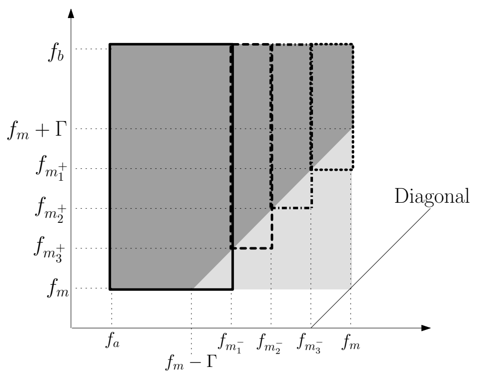


Figure 6: The four rectangles contain all the Γ-persistence dots we want to compute, and some extra 3Γ/4-persistence pairs.

For computing a single $P^{m^+_{t-i+1},b}_{m^-_{i-1}+1,m^-_i}$, we have a bound of

$$O\left(\left(1+\mu((1-\delta)\Gamma)^{m^+_{t-i+1},b}_{m^-_{i-1}+1,m^-_i}\log(b-a+1)\right)R_d(b-a+1)\right).$$

Since all the P-sets are disjoint and their cardinality adds up to $\mu((1-\delta)\Gamma)^{m,b}_{a,m}$, we obtain a total complexity bound of

$$O\left(\left(\frac{1}{\delta}+\mu((1-\delta)\Gamma)^{m,b}_{a,m}\log(b-a+1)\right)R_d(b-a+1)\right).$$

for the computation of $P(\Gamma)^{m,b}_{a,m}$.

The main algorithm. Having established that bound, the analysis of the divide-and-conquer algorithm to compute all Γ-persistent pairs follows from standard methods: Let $C_{(1-\delta)\Gamma}$ denote the total number of homology classes in the filtration with persistence at least $(1-\delta)\Gamma$. By definition, $\mu((1-\delta)\Gamma)^{m,b}_{a,m}\le C_{(1-\delta)\Gamma}$ for every $a,b\in[1,n]$, therefore the complexity of one divide-and-conquer step is $O((\frac{1}{\delta}+C_{(1-\delta)\Gamma}\log(b-a+1))R_d(b-a+1))$.

Let Cost(n) denote the cost of computing the Γ-persistent pairs in an index range $[a,b]$ of size n. We have the recurrence relation

$$\mathrm{Cost}(n)=\underbrace{\left(\frac{1}{\delta}+C_{(1-\delta)\Gamma}\log n\right)R_d(n)}_{\text{cost for }P(\Gamma)^{m,b}_{a,m}}+\underbrace{2\mathrm{Cost}(n/2)}_{\text{cost for pairs in }[a,m]\text{ and }[m,b]}\tag{4}$$

This recurrence equation solves to $O((\frac{1}{\delta}+C_{\Gamma(1-\delta)}\log n)R_d(n))$ by the Master theorem [33].[1] We can also see immediately that the space consumption is proportional to the size of the boundary matrix of the complex, since that matrix is essentially the only object that needs to be stored. However, computing the rank might require additional memory. We can thus summarize:

Theorem 8. *For a filtered simplicial complex of size n, computing the Γ-persistent pairs has a time complexity of*

$$O\left(\left(\frac{1}{\delta}+C_{(1-\delta)\Gamma}\log n\right)R_d(n)\right)$$

and a space complexity of $O(dn+R_d^S(n))$, where $R_d^S(n)$ is the space complexity for computing the rank of an $n\times n$-matrix with $O(dn)$ nonzero entries.

We will show in the next section that the same complexity bound also holds for computing the essential homology classes of the complex.

Necessity of δ. An unpleasant property of our bound is the presence of δ because it prevents the bound from being fully output-sensitive. However, we were not able to avoid that factor; a perhaps natural proposal is to use the divide-and-conquer scheme illustrated in Figure 3 directly on the dark shaded region from Figure 4 to find all dots in the diagram above the line which is Γ away from the diagonal. That is, one may first compute all points created in $[a,m^-]$ and destroyed in $[m^+,b]$ (they all have persistence at least Γ), and then recursively compute points living in $[a,m^+)$ and $(m^-,b]$ respectively. The overlapping of the intervals is due to the fact that Γ-persistence pairs created or destroyed in (m^-,m^+) may not be captured in the first step. Such an overlapping, however, leads to a recurrence relation of the form

$$\text{Cost}(n)=\cdots+2\text{Cost}(n-1),$$

because the interval $[a,m^+)$ can contain up to $b-a$ indices in the worst case.

5. Essential classes

We are left with the case of detecting those simplices whose addition creates an essential homology class. We compute them with similar methods, but we have to double the size of the complex:

[1]We assume here that $R_d(n)\in\Omega(n^{1+\epsilon})$ for some $\epsilon>0$ which seems to be a realistic assumption.

For a filtered complex K whose simplices are sorted in the order $(\sigma_1, \ldots, \sigma_n)$, we choose a new vertex $v_0 \notin K$, and define $\sigma'_i := \sigma_i \cup \{v_0\}$. We extend the filter function by setting $f(v_0) = -\infty$, and $f(\sigma) = \infty$ whenever $v_0 \subsetneq \sigma$. Then, the *coned complex* is defined as $K' = K \cup \{v_0\} \cup \{\sigma'_i \mid i \in [1, n]\}$, with the sorted simplex ordering $(v_0, \sigma_1, \ldots, \sigma_n, \sigma'_1, \ldots, \sigma'_n)$. Note that, indeed, K' arises from K by attaching each simplex to v_0. Moreover, K' has only trivial homology classes, except for one connected component which is created when v_0 is inserted.

Observe that the persistence pairs of K appear also in K' because the filtration of K equals the filtration of K' in the first $n + 1$ steps, except the unrelated vertex v_0. It follows that the homology class created at σ_i is essential in K if and only if there exists some j such that (σ_i, σ'_j) is a persistence pair of K'. Therefore, we just have to compute the set $P_{2,n+1}^{n+2,2n+1}$ of the coned complex K'. By Section 3, this can be achieved in $O(\beta R_d(2n) \log(2n))$, where β is the number of essential homology classes (which is the sum of the Betti numbers of the simplicial complex K). Finally, since $\beta \leq C_\Gamma$ for any $\Gamma > 0$, this computation is asymptotically not more expensive than the computation of inessential classes as described in the previous section. We conclude

Theorem 9. *For a filtered simplicial complex of size n, computing the homology classes of persistence at least Γ has a time complexity of*

$$O\left(\left(\frac{1}{\delta} + C_{(1-\delta)\Gamma} \log n\right) R_d(n)\right)$$

and a space complexity of $O(dn + R_d^S(n))$, where $R_d^S(n)$ is the space complexity for computing an $n \times n$-matrix with $O(dn)$ nonzero entries.

Furthermore, we compute an upper bound of the number of rank computations, which will be useful in Section 6.

Lemma 10. *The number of rank computation in the algorithm is not larger than*

$$\rho := \rho(n, \left\lceil \frac{1}{\delta} \right\rceil) := 4n(2 \left\lceil \frac{1}{\delta} \right\rceil + \log n + 1).$$

Proof. We let p denote the number of persistence pairs and e be the number of essential classes and note that $2p+e = n$. For convenience, we set $t := \lceil \frac{1}{\delta} \rceil$. Recall the divide-and-conquer strategy to find persistence points. The size of the recursion tree is obviously bounded by $2n$, and we require at least t μ-queries per node. If some query yields a non-zero value in a node, we need more μ-queries to identify births and deaths of persistence pairs. Note that every persistence pair causes at most $2\log n$ additional μ-queries in the algorithm, because the pair appears in only one P-set of only one divide-and-conquer step, requiring $\log n$ steps both to identify the birth and the death simplex. Since there are p persistence pairs, at most $2p\log n + 2tn$ μ-queries are needed in total for finding the inessential classes. For each essential class, $\log(2n) = 1 + \log n$ μ-queries are necessary. Therefore, the total number of μ-queries is bounded by

$$\begin{aligned} & 2p\log n + 2tn + e(1+\log n) \\ \leq\; & 2tn + n + (2p+e)\log n = n(2t + \log n + 1). \end{aligned}$$

Since a μ-query requires 4 rank computations, the bound follows. $\square$

Extended persistence. Note that in the coned complex defined above, the simplices $\sigma'_1, \ldots, \sigma'_n$ in the second half of the filtration can be arbitrarily interchanged without changing the result, as long as the complex remains simplicial when adding the simplices in order. By choosing a meaningful order in the second half, it is straight-forward to harvest more information from the filtration:

Assume that the original filtration $(\sigma_1, \ldots, \sigma_n)$ is a *lower star filtration* of a simplicial complex (that is, the function value at every simplex equals the maximal function value at the vertices of its convex hull). We set $f(\sigma'_i)$ to $f(\sigma_i)$ if σ_i is a vertex; otherwise, we set $f(\sigma'_i)$ to be the minimal function value of the convex hull vertices of σ_i. We sort the σ'_i in decreasing order, breaking ties by dimension and otherwise arbitrarily. In this way, we get a filtration $(\tau_1, \ldots, \tau_n)$ that equals the *upper star filtration* of the original complex, except that the additional vertex v_0 appears in the vertex list of every simplex. It is well-known that $(v_0, \sigma_1, \ldots, \sigma_n, \tau_1, \ldots, \tau_n)$ corresponds to the *extended filtration* that is obtained by first filtrating a topological space with respect to its sublevel sets $\mathsf{H}(\mathbb{X}_t)$, for t from $-\infty$ to $+\infty$, and then extending the filtration by taking relative homology groups $\mathsf{H}(\mathbb{X}, \mathbb{X}^t)$ with t from $+\infty$ to $-\infty$, where $\mathbb{X}^t := f^{-1}[t, +\infty)$ is a superlevel set of $\mathbb{X}$. This concept has been introduced as *extended persistence* in [10] and has been proven as a useful concept in the theory of persistent homology.

We can compute the extended persistence pairs in three steps: We first compute *ordinary* pairs of the form (σ_i, σ_j) by computing $P(\Gamma)^{2,n+1}_{2,n+1}$. Second, we compute *relative* pairs of the form (τ_i, τ_j) by $P(\Gamma)_{n+2,2n+1}^{n+2,2n+1}$. Finally, we compute *extended* pairs of the form (σ_i, τ_j) by $P_{2,n+1}^{n+2,2n+1}$. We obtain the same complexity bound as for the non-extended case.

Note that we interpret extended pairs to have infinite persistence; this makes sense if we define the persistence of a class to be the range of t-values where the class is alive, which is conformal to our definition of persistence for ordinary pairs. Alternatively, if we define the persistence of an extended class to be the difference of the birth and death value, we have to compute $P(\Gamma)_{2,n+1}^{n+2,2n+1}$ instead. This can be computed with the same divide-and-conquer approach as for ordinary and relative classes, but the complexity analysis does not carry over because the input parameter of the cost function in (4) does not decrease to 1. We pose it as an open question whether the algorithm can be extended to ignore extended pairs with small difference in function values.

6. Instantiating rank algorithms

We consider three possible choices of finite field rank algorithm, obtaining different complexity bounds:

Deterministic algorithm. The best known *deterministic* rank-computation algorithm [28] has a complexity of $O(n^\omega)$ operations in $\mathbb{Z}_2$ where ω is the matrix-multiplication exponent. The currently best known upper bound for ω is 2.376 [20]. Thus, we can state

Corollary 11. *There is a deterministic algorithm to compute Γ-persistence with time complexity*

$$O\left(\left(\frac{1}{\delta} + C_{(1-\delta)\Gamma} \log n\right) n^\omega\right) = O\left(\left(\frac{1}{\delta} + C_{(1-\delta)\Gamma}\right) n^{2.376}\right).$$

Note that according to [19], the whole persistence diagram can be computed in $O(n^\omega)$ instead.

Randomized algorithms. The randomized rank algorithms that we use require computations in a base field with sufficiently many elements, even though the rank over $\mathbb{Z}_2$ is computed. We let $\mathbb{K}$ denote an extension field of $\mathbb{Z}_2$ with at least $(n^2 \log n)$ elements (according to [27] and [26], a field of that size is sufficient for our purposes). We express the complexity in the number of arithmetic operations (i.e., addition, multiplications, and divisions) in $\mathbb{K}$. One operation in $\mathbb{K}$ has a bit complexity of $O(\log n \log\log n \log\log\log n)$ by Schönhage-Strassen multiplication [34], so we obtain the bit complexity by multiplying with this factor.

Furthermore, the used algorithms exploit the sparsity of the input matrix. Recall from (1) that the boundary matrix has up to $O(dn) = O(n \log n)$ nonzero entries.

Let $\widetilde{O}$ denote a complexity bound with logarithmic factors omitted. For Las-Vegas type (correct result, but running time depends on random choices), there is a rank-computation algorithm with expected $\widetilde{O}(n^{3-1/(\omega-1)}) = O(n^{2.28})$ arithmetic operations in $\mathbb{K}$, for matrices with $O(n \log n)$ nonzero entries [27].

Theorem 12. *There is a Las-Vegas algorithm to compute* Γ*-persistence with an expected number of*

$$\widetilde{O}\left(\left(\frac{1}{\delta} + C_{(1-\delta)\Gamma} \log n\right) n^{3-1/(\omega-1)}\right) = O\left(\left(\frac{1}{\delta} + C_{(1-\delta)\Gamma}\right) n^{2.28}\right)$$

arithmetic operations in $\mathbb{K}$ *in the worst case.*

By our preceeding remark, the arithmetic operations in $\mathbb{K}$ only add a logarithmic factor to the bit complexity, so the Theorem 12 also bounds the bit complexity of the algorithm.

Finally, we look at randomized algorithms of Monte-Carlo type, that means, the result is correct only with a certain probability. The following theorem follows from Theorem 3 of Kaltofen and Saunders [26][2] who improve on Wiedemann's seminal paper [35]. We give the details on how to get this bound in Appendix A.

Theorem 13 (Kaltofen-Saunders)**.** *Given a matrix* $M \in \mathbb{Z}_2^{n\times n}$ *with* $O(dn)$ *nonzero entries, there is an algorithm to compute an integer* $r \leq \operatorname{rank}(M)$ *such that* $r = \operatorname{rank}(M)$ *with a probability of at least* 0.94. *This algorithm performs* $O(n^2 \log n \log\log n)$ *arithmetic operations in* $\mathbb{K}$.

This results suggests to use $R_d(n) = O(n^2 \log n \log\log n)$. However, our goal is a Monte-Carlo algorithm that returns the correct persistence pairs

[2]The authors of [26] refer to their algorithm as Las-Vegas algorithm, although it is Monte-Carlo according to the usual definition

with some constant probability q. Note that a single wrong rank value leads to a wrong μ-query, and thus a unpredictable output of our algorithm. Since the number of rank computations increases with n, the probability of a wrong rank result goes to 1 for large instances. On the other hand, we can rerun a rank computation k times in order to achieve a higher success probability of $(1-0.06^k)$ (by taking the maximum among the k return values). We analyze next how often we have to repeat each rank computation such that, with probability q, every rank is computed without error.

Let $p \geq 0.94$ denote the success probability of the Monte-Carlo method from [26]. Assume that every rank computation in our algorithm is performed by calling that algorithm k times, and taking the maximal return value as rank. According to Lemma 10, to guarantee a success probability of q for our algorithm, we have to satisfy

$$(1-(1-p)^k)^\rho \geq q,$$

with $\rho := \rho(n, \lceil \frac{1}{\delta} \rceil)$ being an upper bound for the number of rank computations as defined in Lemma 10. We solve for k and get

$$k \geq \frac{1}{\log \frac{1}{1-p}} \cdot \log \frac{1}{1-q^{\frac{1}{\rho}}} \tag{5}$$

Note that the second factor of this expression depends on n and δ and thus affects the time complexity of the algorithm. However,

$$\log \frac{1}{1-q^{\frac{1}{\rho}}} = \Theta(\rho),$$

which follows in a straight-forward way from the fact that for any constant $q < 1$, $x(1-q^{1/x})$ converges to $-\log q$ for $x \to \infty$. This implies that choosing the smallest k satisfying (5), every rank computation requires $O(\log \rho) = O(\log n + \log \frac{1}{\delta})$ applications of the Monte-Carlo algorithm. Therefore, we have that $R_d(n) = O(n^2 \log n(\log n + \log \frac{1}{\delta}) \log\log n)$. Moreover, the space consumption for computing a rank using Theorem 13 is bounded by $O(dn)$, as stated in [26]. Therefore, we get

Theorem 14. *For any fixed success probability $q < 1$, there is a Monte-Carlo algorithm to compute Γ-persistence with*

$$O\left(\left(\frac{1}{\delta} + C_{(1-\delta)\Gamma} \log n\right) n^2 \log n(\log n + \log \frac{1}{\delta}) \log\log n\right)$$

arithmetic operations in $\mathbb{K}$ and with a space complexity of $O(dn)$.

If δ is considered as a constant, the time complexity simplifies to

$$O(C_{(1-\delta)\Gamma} n^2 \log^3 n \log\log n)$$

as stated in Table 1. We are able to bound the bit complexity of the algorithm by multiplication with $O(\log n \log\log n \log\log\log n)$:

Corollary 15. *For any fixed success probability $q < 1$, there is a Monte-Carlo algorithm to compute Γ-persistence with bit complexity*

$$O\left(\left(\frac{1}{\delta} + C_{(1-\delta)\Gamma} \log n\right) n^2 (\log n)^2 (\log n + \log \frac{1}{\delta}) (\log\log n)^2 \log\log\log n\right),$$

and with a space complexity of $O(dn)$.

7. Concluding remarks

We have presented the first output-sensitive analysis of an algorithm to compute persistent homology that ignores homology classes of low persistence. If ranks are computed using the Monte-Carlo approach, the algorithm requires only $O(dn)$ space, even for computing the whole persistence diagram (we remark that this is not true for the deterministic and Las-Vegas algorithms presented in this work) Although our complexity results do not improve the worst-case time complexity of the problem, we think that the approach of using state-of-the-art methods from symbolic computation (rank computation in our case) can lead to more efficient algorithms in this research context and should be investigated further.

Generalized complexes. Our analysis readily applies for persistence on cubical complexes. In more generality, we can state the result of Theorem 9 also with $R_d(n)$ replaced by $R(n, e)$, which stands for the cost of computing a rank of an $n \times n$-matrix with at most e nonzero entries. The determistic variant of the algorithm does not require any sparseness condition and therefore remains valid for any e. As long as $e \in \widetilde{O}(n)$, the statments of Theorem 12 and Theorem 14 will also remain true (the latter possibly with a few additional logarithmic factors).

Generalized fields. We concentrated on $\mathbb{Z}_2$-homology for the sake of simplicity, and because it is the most commonly used base field for persistence computation. However, Theorem 9 remains true for any base field (finite or infinite), with the interpretation that it counts the number of arithmetic operations within the given field. For high characteristics, such an operation can take non-constant time.

Computing representative cycles. Birth and death times are sufficient for computing the persistence diagram. However, one may want to compute representative cycles of each persistent homology class. The following additional steps achieve this without increasing our complexity bound. For each computed Γ-persistence pair or essential class created by the simplex σ_i, we solve a linear system $Ax = b_i$, where b_i is the column corresponding to σ_i, and A consists of all columns on b_i's left. The solution yields a cycle representing a class created by σ_i. The linear equation system can be solved in time $R_d(n)$ (again, cf. [28], [27], [26] for deterministic, Las-Vegas, and Monte-Carlo algorithms), and thus, the computation takes $O(C_\Gamma R_d(n))$ which does not worsen the overall bound. Note that the generated cycle represents one out of the whole coset of classes created by σ_i, not necessarily the one which becomes trivial when the corresponding destroyer σ_j enters the filtration.

Experimental results. The *LinBox* library[3] [36] provides an implemented variant of asymptotically fast Monte-Carlo algorithm for rank computation that we use for our complexity bound. Note that our divide-and-conquer algorithm computes the rank of the whole boundary matrix in the first step, so the Monte-Carlo approach has to compute ranks significantly faster than Gaussian elimination to turn our persistence algorithm practical.

Our preliminary experiments, however, do not show this outcome. On all tested boundary matrices, Gaussian elimination performs better than the randomized rank algorithm. This is affirmative to the experimental results from [37] where Gaussian elimination is also observed to be generally faster than the randomized version. The reason is that during Gaussian elimination, the columns only fill up slowly in the beginning, and usually stay rather sparse until the end of the algorithm (although one can construct worst-case examples where the reduced matrix becomes dense quickly). However, for very large instances, it is demonstrated in [37] that Gaussian elimination

[3]`http://www.linalg.org`

can fail due to this fill-up because the algorithm runs out of memory. Since our rank algorithm only requires $O(n \log n)$ space, we conjecture that it can compute persistence diagrams of complexes where the reduction algorithm fails. Moreover, our restrictions to simplicial complexes and to $\mathbb{Z}_2$ homology favor the Gaussian elimination because they both reduce the speed of the fill-up process during elimination.

Acknowledgements. The authors thank Herbert Edelsbrunner for many helpful discussions and suggestions. Moreover, they are grateful for the careful reviews that helped to improve the quality of the paper.

References

[1] G. Carlsson, T. Ishkhanov, V. de Silva, A. Zomorodian, On the local behavior of spaces of natural images, International Journal of Computer Vision 76 (2008) 1–12.

[2] A. Cerri, M. Ferri, D. Giorgi, Retrieval of trademark images by means of size functions, Graphical Models 68 (2006) 451–471, special Issue on the Vision, Video and Graphics Conference 2005.

[3] C. Chen, H. Edelsbrunner, Diffusion runs low on persistence fast, in: Proceedings of the Thirteenth International Conference on Computer Vision, IEEE, 2011.

[4] M. Chung, P. Bubenik, P. Kim, Persistence diagrams of cortical surface data, in: Information Processing in Medical Imaging, Vol. 5636 of LNCS, 2009, pp. 386–397.

[5] V. de Silva, R. Ghrist, Coverage in sensor networks via persistent homology, Algebraic and Geometric Topology 7 (2007) 339–358.

[6] Y. Wang, P. Agarwal, P. Brown, H. Edelsbrunner, J. Rudolph, Coarse and reliable geometric alignment for protein docking, in: Proceedings of the Pacific Symposium on Biocomputing 2005, 2005, pp. 65–75.

[7] P. Bendich, H. Edelsbrunner, D. Morozov, A. Patel, The robustness of level sets, in: Proceedings of the 18th European Symposium on Algorithms, 2010, pp. 1–10.

[8] F. Chazal, D. Cohen-Steiner, M. Glisse, L. Guibas, S. Oudot, Proximity of persistence modules and their diagrams, in: Proceedings of the 25th Annual Symposium on Computational Geometry, ACM, 2009, pp. 237–246.

[9] D. Cohen-Steiner, H. Edelsbrunner, J. Harer, Stability of persistence diagrams, Discrete and Computational Geometry 37 (2007) 103–120.

[10] D. Cohen-Steiner, H. Edelsbrunner, J. Harer, Extending persistence using Poincaré and Lefschetz duality, Foundations of Computational Mathematics 9 (2009) 79–103.

[11] V. De Silva, M. Vejdemo-Johansson, Persistent cohomology and circular coordinates, in: Proceedings of the 25th Annual Symposium on Computational Geometry, ACM, 2009, pp. 227–236.

[12] A. Zomorodian, G. Carlsson, Computing persistent homology, Discrete and Computational Geometry 33 (2005) 249–274.

[13] A. Zomorodian, G. Carlsson, The theory of multidimensional persistence, Discrete and Computational Geometry 42 (2009) 73–93.

[14] H. Edelsbrunner, J. Harer, Computational Topology, An Introduction, American Mathematical Society, 2010.

[15] H. Edelsbrunner, D. Letscher, A. Zomorodian, Topological persistence and simplification, Discrete and Computational Geometry 28 (2002) 511–533.

[16] G. Carlsson, V. De Silva, D. Morozov, Zigzag persistent homology and real-valued functions, in: Proceedings of the 25th Annual Symposium on Computational Geometry, ACM, 2009, pp. 247–256.

[17] D. Morozov, Persistence algorithm takes cubic time in the worst case, in: BioGeometry News, Duke Computer Science, Durham, NC, 2005.

[18] C. Chen, M. Kerber, Persistent homology computation with a twist, in: 27th European Workshop on Computational Geometry, 2011.

[19] N. Milosavljević, D. Morozov, P. Škraba, Zigzag persistent homology in matrix multiplication time, in: Proceedings of the 27th Annual Symposium on Computational Geometry, 2011, pp. 216–225.

[20] D. Coppersmith, S. Winograd, Matrix multiplication via arithmetic progresssions, Journal of Symbolic Computation 9 (1990) 251–280.

[21] W. Harvey, Y. Wang, R. Wenger, A randomized $O(m \log m)$ time algorithm for computing Reeb graphs of arbitrary simplicial complexes, in: Proceedings of the 26th Annual Symposium on Computational Geometry, 2010, pp. 267–276.

[22] A. Zomorodian, The tidy set: a minimal simplicial set for computing homology of clique complexes, in: Proceedings of the 26th Annual Symposium on Computational Geometry, 2010, pp. 257–266.

[23] B. Hudson, G. Miller, S. Oudot, D. Sheehy, Topological inference via meshing, in: Proceedings of the 2010 Annual Symposium on Computational Geometry, 2010, pp. 277–286.

[24] H. Wagner, C. Chen, E. Vuçini, Efficient computation of persistent homology for cubical data, in: Workshop on Topology-based Methods in Data Analysis and Visualization, 2011.

[25] P. Bendich, H. Edelsbrunner, M. Kerber, Computing robustness and persistence for images, IEEE Transactions on Visualization and Computer Graphics 16 (2010) 1251–1260.

[26] E. Kaltofen, B. David Saunders, On Wiedemann's method of solving sparse linear systems, in: Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, Vol. 539, Springer, 1991, pp. 29–38.

[27] W. Eberly, M. Giesbrecht, P. Giorgi, A. Storjohann, G. Villard, Faster inversion and other black box matrix computations using efficient block projections, in: Proceedings of the 2007 international symposium on Symbolic and algebraic computation (ISSAC 2007), ACM, 2007, pp. 143–150.

[28] O. H. Ibara, S. Moran, R. Hui, A generalization of the fast LUP matrix decomposition algorithm and applications, Journal of Algorithms 3 (1982) 45–56.

[29] J. Munkres, Elements of algebraic topology, Westview Press, 1984.

[30] C. Chen, D. Freedman, Quantifying homology classes, in: Proceedings of the 25th Annual Symposium on Theoretical Aspects of Computer Science (STACS 2008), 2008, pp. 169–180.

[31] C. Chen, D. Freedman, Measuring and computing natural generators for homology groups, Computational Geometry: Theory and Applications 43 (2010) 169–181.

[32] D. Cohen-Steiner, H. Edelsbrunner, D. Morozov, Vines and vineyards by updating persistence in linear time, in: Proceedings of the 22nd Annual Symposium on Computational Geometry, 2006, pp. 119–126.

[33] T. Cormen, C. Leiserson, R. Rivest, C. Stein, Introduction to algorithms, The MIT press, 2001.

[34] A. Schönhage, V. Strassen, Schnelle Multiplikation grosser Zahlen, Computing 7 (1971) 281–292, in German.

[35] D. H. Wiedemann, Solving sparse linear equations over finite fields, IEEE Transactions on Information Theory 32 (1986) 54–62.

[36] J.-G. Dumas, T. Gautier, M. Giesbrecht, P. Giorgi, B. Hovinen, E. Kaltofen, B. Saunders, W. Turner, G. Villard, Linbox: A generic library for exact linear algebra, in: Mathematical Software: ICMS 2002 (Proceedings of the first International Congress of Mathematical Software), 2002, pp. 40–50.

[37] J.-G. Dumas, G. Villard, Computing the rank of large sparse matrices over finite fields, in: Proceedings of the Fifth International Workshop on Computer Algebra in Scientific Computing, 2002, pp. 47–62.

Appendix A: Details on the Monte-Carlo method

We explain how our Theorem 13 follows from Theorem 3 in [26]. For that, we first state their Theorem in full generality. In their paper, they assume that $\mathbb{K}$ is a finite field with sufficiently many elements, and state in the introduction that $50n^2 \log n$ elements suffice (for an $n \times n$ matrix).

Theorem 16 (Kaltofen-Saunders, Thm.3)**.** *Let $A \in \mathbb{K}^{n\times n}$ and $\mathbb{S} \subset \mathbb{K}$. Using $5n-2$ random elements from $\mathbb{S}$, we may probabilistically determine the rank of A by $O(n)$ multiplications of A by vectors and $O(n^2 \log n \log\log n)$ arithmetic operations in $\mathbb{K}$. The algorithm returns an integer that is with probability no less than*

$$1 - \frac{3}{2}\frac{n(n+1)}{\operatorname{Card}\mathbb{S}}$$

the rank of A.

We simplify their theorem in several respects:

- First of all, there is no restriction on $\mathbb{S}$, so we can simply choose $\mathbb{S} = \mathbb{K}$. It follows that we can bound the success probability by

$$1 - \frac{3}{2}\frac{n(n+1)}{\operatorname{Card}\mathbb{K}} \geq 1 - \frac{3}{2}\frac{n(n+1)}{50n^2} \geq 1 - \frac{3}{2}\frac{2n^2}{50n^2} = 0.94.$$

- Moreover, the algorithm that they define always returns a value which is at most the rank of A.
- Furthermore, we observe that, if A has $O(dn) = O(n \log n)$ nonzero entries, a matrix-vector multiplication needs $O(n \log n)$ operations in $\mathbb{K}$, so that the $O(n^2 \log n \log\log n)$ operations are dominant.
- Finally, we consider our case of a matrix $A \in \mathbb{Z}_2^{n\times n}$. The base field has not enough elements, but we can pass to an algebraic extension field $\mathbb{K}$ of $\mathbb{Z}_2$ with a sufficient number of elements. It is well-known that the rank of A over $\mathbb{Z}_2$ equals the rank of A over $\mathbb{K}$ (because the rank is defined by the maximal non-vanishing minor of A, and the property of a minor being zero or not does not change when passing to an extension field).

Putting everything together yields Theorem 13.