An Output-Sensitive Algorithm for Persistent Homology

Chao Chen[*]
Institute of Science and Technology Austria
Klosterneuburg, Austria
& PRIP, Vienna University of Technology,
Vienna, Austria
chao.chen@ist.ac.at

Michael Kerber
Institute of Science and Technology Austria
Klosterneuburg, Austria
mkerber@ist.ac.at

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 $O(C_{(1-\delta)\Gamma} R(n) \log n)$, 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, and $R(n)$ is the complexity of computing the rank of an $n \times n$ matrix with $O(n)$ 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 $\epsilon > 0$.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and problems—*Computations on discrete structures*; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Geometric algorithms, languages and systems*

General Terms

Algorithms

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 has turned it into a successful tool for the analysis of various types of data; see [4, 5, 8, 26, 28] 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. [2, 6, 9, 10, 14, 31, 32]. See also [16] for a recent textbook covering both theory and applications.

We are concentrating on the computation of persistent homology in this work. The first algorithm by Edelsbrunner et al. [17] as well as other related algorithms [3, 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 size of the given complex, and Morozov [23] 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 recently presented in [7]. Milosavljević et al. [22] recently presented an algorithm to compute persistence in matrix multiplication time $O(n^\omega)$, where the best estimation of ω is currently 2.376 [12]. 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 [16]. Numerous efficient solution, in theory and practice, have been presented recently for related problems such as computing the Reeb graph of a simplicial complex [18] and computing persistence on special structures like clique complexes [30], point clouds in higher dimensions [19], regular cubical grids [27], or dual complexes of cubical subdivisions in $\mathbb{R}^3$ [1].

In this paper, we present the first output-sensitive algorithm for computing homology classes above a predefined threshold: Given some filtration, namely, a 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 any constant $\delta > 0$, our algorithm has a running time of $O(C_{(1-\delta)\Gamma} R(n) \log n)$, where $R(n)$ is the complexity of computing the rank of an $n \times n$ binary matrix with $O(n)$ nonzero entries. For simplicity and clarity of presentation, the dimension d of the simplicial complex is considered to be a constant throughout the paper, and factors in d and δ are generally ignored. We refer to Section 7 for more de-

[*]Partially supported by the Austrian Science Fund (FWF) grant No. P20134-N13.

Type	Ref.	Rank Complexity	Persistence Computation
Monte-Carlo	[21]	$O(n^2 \log^2 n \log\log n)$	$O(C_{(1-\delta)\Gamma} n^2 \log^3 n \log\log n)$
Las-Vegas	[15]	$O(n^{3-1/(\omega-1)}) = O(n^{2.28})$	$O(C_{(1-\delta)\Gamma} n^{2.28})$
Deterministic	[20]	$O(n^\omega)$	$O(C_{(1-\delta)\Gamma} n^\omega \log n)$

Table 1: Time bounds of our algorithm using different types of rank computation algorithms.

tailed bounds which additionally depend on these quantities. 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 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 a predefined constant $q < 1$. Moreover, the bounds of the randomized methods refer to the number of arithmetic operations in a finite field of $O(n^2 \log n)$ elements. When considering bit complexities, another factor of $O(\log n \log\log n \log\log\log n)$ appears in the bound. 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.

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 extend 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 [16, 24] 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 injective mapping $f^{s,t} : \mathsf{H}(\mathbb{X}^s) \to \mathsf{H}(\mathbb{X}^t)$ 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 class dies at some $t \geq s$, if t is the smallest value such that $f^{s,t}(\alpha) \in \operatorname{img} f^{s-\epsilon,t}$ 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, $\emptyset = K_0 \subseteq K_1 \subseteq \ldots \subseteq K_n = K$. 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. Assuming that d is a fixed constant, ∂ has therefore $O(n)$ nonzero entries. It also holds that any $k \times k$ submatrix of ∂ has $O(k)$ nonzero entries.

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 $\operatorname{low}(i)$. The standard persistence algorithm [16] 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 exists a reduced column j, $j < i$, such that $\operatorname{low}(j) = \operatorname{low}(i)$, subtract j from i. This is repeated until either the i-th column becomes zero, or $\operatorname{low}(i)$ 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 [16, §VII.1]. Moreover, let the i-th column of $\mathcal{R}$ be nonzero, and let $j = \text{low}(i)$ 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*, $\text{Dgm}(f)$, which is a multiset of points embedded 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 (of some constant 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 $i_1 \le i_2 \le j_1 \le j_2$,*

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

In other words, $P_{i_1,i_2}^{j_1,j_2}$ is the set of dots of the persistence diagram within the box $[f_{i_1}, f_{i_2}] \times [f_{j_1}, f_{j_2}] \subseteq \mathbb{R}^2$, except that dots on the boundary of the box are counted only by a fraction of their multiplicity. We show next that $\mu_{i_1,i_2}^{j_1,j_2}$ is determined by the ranks of 4 submatrices of ∂ of size at most $j_2 - i_1 + 1$. The result generalizes the "Paring Uniqueness Lemma" from [11] which handles the special case of $i_1 = i_2$, $j_1 = j_2$.

For any matrix A, denote $A_{a_1,a_2}^{b_1,b_2}$ 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 $b_1, b_1 + 1, \cdots, b_2$.

Theorem 2 (μ-query).

$$\begin{aligned} \mu_{i_1,i_2}^{j_1,j_2} \quad = \quad & \text{rank}(\partial_{i_1,j_2}^{i_1,j_2}) - \text{rank}(\partial_{i_1,j_1-1}^{i_1,j_1-1}) - \\ & \text{rank}(\partial_{i_2+1,j_2}^{i_2+1,j_2}) + \text{rank}(\partial_{i_2+1,j_1-1}^{i_2+1,j_1-1}). \qquad (1) \end{aligned}$$

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, $\mu_{i_1,i_2}^{j_1,j_2}$ is equal to the number of 1's in $\mathcal{S}_{i_1,i_2}^{j_1,j_2}$. By inclusion-exclusion,

$$\begin{aligned} \mu_{i_1,i_2}^{j_1,j_2} = & \left(\#\ \text{1's in}\ \mathcal{S}_{i_1,n}^{1,j_2}\right) - \left(\#\ \text{1's in}\ \mathcal{S}_{i_1,n}^{1,j_1-1}\right) - \\ & \left(\#\ \text{1's in}\ \mathcal{S}_{i_2+1,n}^{1,j_2}\right) + \left(\#\ \text{1's in}\ \mathcal{S}_{i_2+1,n}^{1,j_1-1}\right). \qquad (2) \end{aligned}$$

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, the lower-left submatrices of ∂ and of $\mathcal{S}$ of equal dimension have the same rank. Therefore we can replace the first term in the right hand side of (2) with $\text{rank}(\partial_{i_1,n}^{1,j_2})$, which is equal to $\text{rank}(\partial_{i_1,j_2}^{i_1,j_2})$ because ∂ is upper-triangular. This also applies to the other three terms and thus leads to (1). $\square$

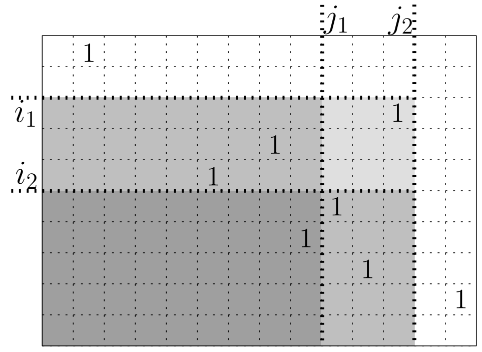

Figure 1: Part of the completely reduced matrix $\mathcal{S}$. $\mu_{i_1,i_2}^{j_1,j_2}$ counts the number of 1's in the light gray submatrix.

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

Computing $P_{i_1,i_2}^{j_1,j_2}$. Next, we compute $P_{i_1,i_2}^{j_1,j_2}$ 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 $P_{i_1,i_2}^{j_1,j_2}$, 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 $i_1 \le i \le i_2$ appears as a leaf in the tree. We call such a tree the *bisection tree* of $[i_1, i_2]$ (Figure 2).

For any node $[a, b]$ of the bisection tree, we define its *μ-value* to be $\mu_{a,b}^{j_1,j_2}$. 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 $P_{i_1,i_2}^{j_1,j_2}$ 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

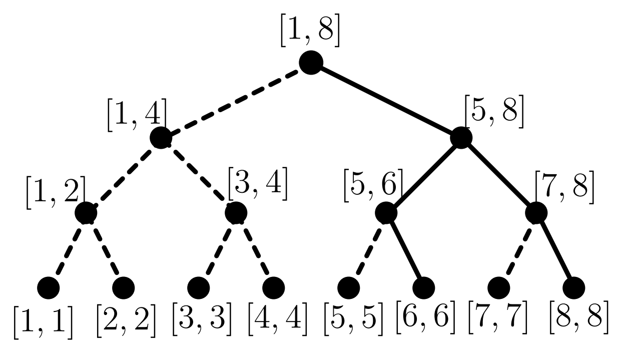

Figure 2: An illustration of the bisection tree of an interval $[i_1, i_2] = [1, 8]$, which contains two creators, σ_6 and σ_8. Only the nodes on the two solid paths (and their siblings) are explored by the algorithm.

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 which part of the tree is explored by the algorithm.

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

$$O((1 + \mu^{j_1,j_2}_{i_1,i_2} \log(i_2 - i_1 + 1))R(j_2 - i_1 + 1))$$

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 $\mu^{j_1,j_2}_{i_1,i_2}$ such paths, and thus at most $(1 + \mu^{j_1,j_2}_{i_1,i_2} \log(i_2 - i_1 + 1))$ 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 $\mu^{j_1,j_2}_{i,i} = 1$, and the index of its destroying counterpart is the unique integer $j \in [j_1, j_2]$ with $\mu^{j,j}_{i,i} = 1$. Clearly, we can find j by a binary search on $[j_1, j_2]$ which needs $O(\log(j_2 - j_1 + 1)R(j_2 - i_1 + 1))$ time. Doing this for all creating simplices of $P^{j_1,j_2}_{i_1,i_2}$ can be done in $O(\mu^{j_1,j_2}_{i_1,i_2} \log(j_2 - j_1 + 1)R(j_2 - i_1 + 1))$. We summarize the results of this section with the following theorem.

THEOREM 5. *For $i_1 \leq i_2 \leq j_1 \leq j_2$, computing $P^{j_1,j_2}_{i_1,i_2}$ has a complexity of*

$$O((1 + \mu^{j_1,j_2}_{i_1,i_2} \log(j_2 - i_1 + 1))R(j_2 - i_1 + 1))$$

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 $f_j - f_i \geq \Gamma$. We also set*

$$P(\Gamma)^{j_1,j_2}_{i_1,i_2} := \{(\sigma_i, \sigma_j) \in P^{j_1,j_2}_{i_1,i_2} \mid f_j - f_i \geq \Gamma\},$$
$$\mu(\Gamma)^{j_1,j_2}_{i_1,i_2} := \text{Card}\, P(\Gamma)^{j_1,j_2}_{i_1,i_2}$$

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 $P(\Gamma)^{m,b}_{a,m}$, 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 $P(\Gamma)^{m,b}_{a,m}$ 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 $P(\Gamma)^{m,b}_{a,m}$, 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 $P(\Gamma)^{m,b}_{a,m}$, 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 $P^{m,b}_{a,m^-}$ and $P^{m^+,b}_{a,m}$ and we have:

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

$$P(\Gamma)^{m,b}_{a,m} \subseteq P^{m,b}_{a,m^-} \cup P^{m^+,b}_{a,m} \subseteq P(\Gamma/2)^{m,b}_{a,m}$$

PROOF. For the first inclusion, assume for a contradiction that such a pair 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 $[a, m]$ and destroyed in $[m^+, b]$. Among these, we only output the ones with persistence at least Γ, eliminating duplicates.[1]

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

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

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

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

[1] The final step of filtering out duplicates could be avoided by computing the sets $P^{m,b}_{a,m^-}$ and $P^{m^+,b}_{m^-+1,m}$ instead. Although slightly more efficient, this does not affect the complexity of the algorithm, thus we consider the symmetric version for simplicity.

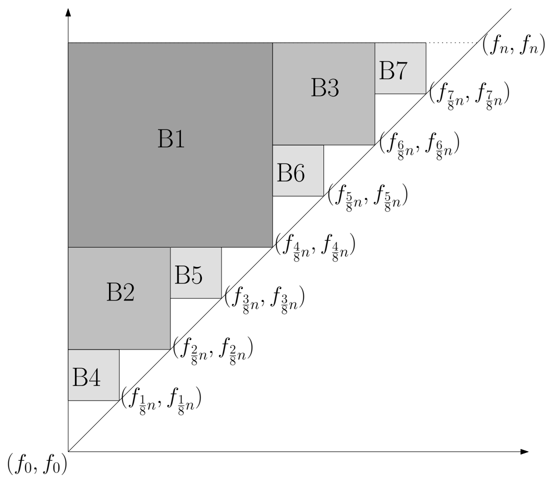
B1
B2
B3
B4
B5
B6
B7
(f_0, f_0)
(f_n, f_n)
$(f_{\frac{7}{8}n}, f_{\frac{7}{8}n})$
$(f_{\frac{6}{8}n}, f_{\frac{6}{8}n})$
$(f_{\frac{5}{8}n}, f_{\frac{5}{8}n})$
$(f_{\frac{4}{8}n}, f_{\frac{4}{8}n})$
$(f_{\frac{3}{8}n}, f_{\frac{3}{8}n})$
$(f_{\frac{2}{8}n}, f_{\frac{2}{8}n})$
$(f_{\frac{1}{8}n}, f_{\frac{1}{8}n})$

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

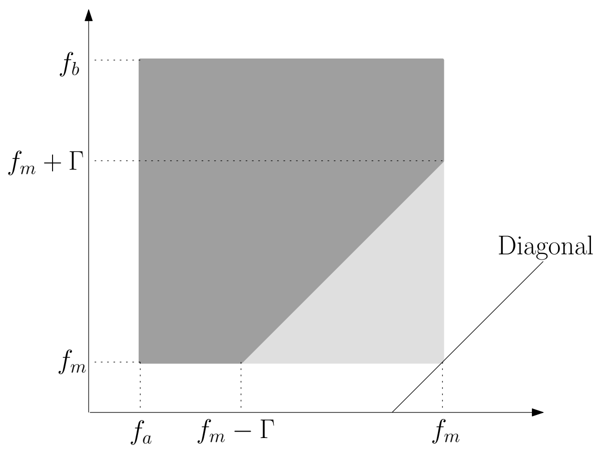

Figure 4: The dark gray polygon contains all the Γ-persistence pairs created in $[a, m]$ and destroyed in $[m, b]$.

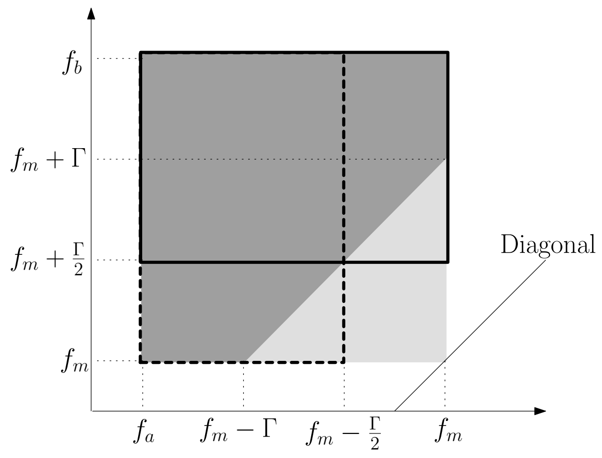

Figure 5: The two rectangles contain all the Γ-persistence dots we want to compute, and some extra $\Gamma/2$-persistence pairs.

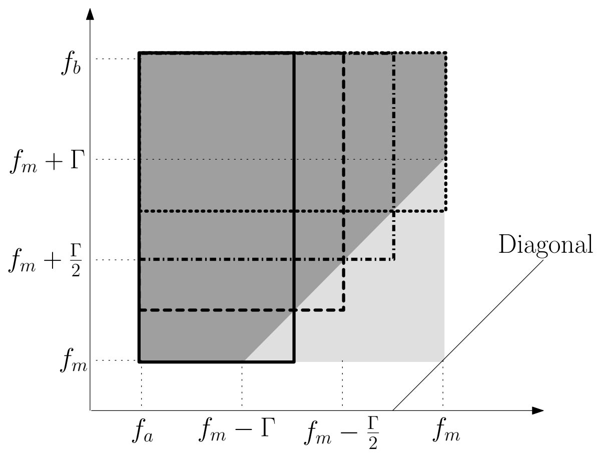

Figure 6: The four rectangles contain all the Γ-persistence dots we want to compute, and some extra $3\Gamma/4$-persistence pairs.

We can further improve the bound by replacing $\mu(\Gamma/2)^{m,b}_{a,m}$ by $\mu((1-\delta)\Gamma)^{m,b}_{a,m}$ for any constant $\delta \in (0,1)$. For that, we generalize Lemma 7 as follows: Fix $t := \lceil \frac{1}{\delta} \rceil$ and define 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 $m^+_1, \ldots, m^+_{t-1}$ 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 $m^-_t := m^+_t := m$. In this situation, we have that

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

See Figure 6 for a case when $t = 4$. It follows the bound of

$$O((1 + \mu((1-\delta)\Gamma)^{m,b}_{a,m} \log(b-a+1))R(b-a+1))$$

for computing every single $P^{m^+_{t-i+1},b}_{a,m^-_i}$ and since this must be only done t times for the computation of $P(\Gamma)^{m,b}_{a,m}$, the total complexity is bounded by the same formula.

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 $(1-\delta)\Gamma$. Since there is at least one essential class (a connected component), $1 + \mu(\Gamma(1-\delta))^{m,b}_{a,m} \leq C_{(1-\delta)\Gamma}$ for every $a, b \in [1, n]$. Therefore the complexity of one divide-and-conquer step is $O(C_{(1-\delta)\Gamma} \log(b-a+1)R(b-a+1))$.

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

$$\mathrm{Cost}(n) = \underbrace{C_{(1-\delta)\Gamma} R(n) \log n}_{\text{cost for } P(\Gamma)^{m,b}_{a,m}} + \underbrace{2\mathrm{Cost}(n/2)}_{\text{cost for pairs in } [a,m] \text{ and } [m,b]} \qquad (3)$$

By the Master theorem [13], this recurrence equation solves to $O(C_{\Gamma(1-\delta)}R(n)\log n)$.[2] We can also see immediately that the space consumption is linear in the number of simplices, since the boundary matrix of the complex 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 $O(C_{(1-\delta)\Gamma}R(n)\log n)$ and a space complexity of $O(n+R_S(n))$, where $R_S(n)$ is the space complexity for computing a $n\times n$-matrix with $O(n)$ 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 $(1-\delta)$ term in the index of C because it prevents the first factor from being exactly the output size of the algorithm. 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

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

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 first. The idea described here is a simplified version of the *extended filtration* from [10].

For a filtered complex K whose simplices is 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 $O(\beta R(2n)\log(2n))$, where β is the number of essential homology classes (which is the sum of the Betti numbers of the simplicial complex K). Finally, since $\beta\le C_\Gamma$ for any $\Gamma>0$, this computation is asymptotically not more expensive than the computation of inessential classes as described in the previous section. We can thus conclude

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

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

$$O(C_{(1-\delta)\Gamma}R(n)\log n)$$

and a space complexity of $O(n+R_S(n))$, where $R_S(n)$ is the space complexity for computing a $n\times n$-matrix with $O(n)$ 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 bounded by*

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

PROOF. We let p denote the number of persistence pairs and e be the number of essential pairs 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 nonzero 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 $2t\log n$ additional μ-queries in the algorithm, because the pair appears in only one divide-and-conquer step, and the point might be detected up to t times, each time requiring $\log n$ steps both to identify the birth and the death simplex. Since there are p persistence pairs, at most $2pt\log n+2tn$ 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

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

Since a μ-query requires 4 rank computations, the bound follows. □

6. INSTANTIATING RANK ALGORITHMS

We consider three possible choices of finite field rank algorithm that gives different complexity bounds:

The best known *deterministic* rank-computation algorithm [20] 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 [12]. Thus, we can state

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

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

Note that according to [22], the whole persistence diagram can be computed in $O(n^\omega)$ instead.

We consider randomized algorithms next. The algorithms that we use require a base field with sufficiently many elements. We let $\mathbb{K}$ denote an extension field of $\mathbb{Z}_2$ with at least $(n^2 \log n)$ elements (which suffices for our purposes). We express the complexity in the number of arithmetic operations in $\mathbb{K}$. Observe that one operation in $\mathbb{K}$ has a bit complexity of $O(\log n \log\log n \log\log\log n)$ by Schönhage-Strassen multiplication [25], so we obtain the bit complexity by multiplying by this factor.

For Las-Vegas type (correct result, but running time depends on random choices), there is a rank-computation algorithm with expected $O(n^{3-1/(\omega-1)}) = O(n^{2.28})$ arithmetic operations in $\mathbb{K}$ [15].

Corollary 12. *There is a Las-Vegas algorithm to compute Γ-persistence with expected worst-case complexity*

$$O(C_{(1-\delta)\Gamma} n^{3-1/(\omega-1)} \log n) = O(C_{(1-\delta)\Gamma} n^{2.28}).$$

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 [21][3] who improved on Wiedemann's seminal paper [29]. 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 $O(n)$ nonzero entries, there is an algorithm to compute an integer $r \le \operatorname{rank}(M)$ 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 $R(n) = O(n^2 \log n \log\log n)$ in our bound. However, our goal is a Monte-Carlo algorithm that returns the correct persistence pairs 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 must repeat each rank computation such that with probability q every rank is computed without error.

Let $p \ge 0.94$ denote the success probability of the Monte-Carlo method from [21]. 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

$$(1 - (1-p)^k)^{\rho_n} \ge q,$$

with ρ_n defined as in Lemma 10. We solve for k and get

$$k \ge \frac{1}{\log \frac{1}{1-p}} \cdot \log \frac{1}{1 - q^{\frac{1}{\rho_n}}} \tag{4}$$

Note that this expression depends on n and thus affects the time complexity of the algorithm. However, the next lemma shows that the term grows the same as $\log \rho_n$.

Lemma 14. *For any constant $q < 1$:*

$$\log \frac{1}{1 - q^{\frac{1}{x}}} \in \Theta(\log x)$$

[3]The authors of [21] refer to their algorithm as Las-Vegas algorithm, although it is Monte-Carlo according to the usual definition

Proof. It suffices to show that

$$\lim_{x\to\infty} \frac{\log \frac{1}{1-q^{\frac{1}{x}}}}{\log x}$$

is a positive constant. We first substitute x by $\frac{1}{x}$ and then use L'Hopital's rule twice to obtain:

$$\lim_{x\to\infty} \frac{\log \frac{1}{1-q^{\frac{1}{x}}}}{\log x} = \lim_{x\to 0} \frac{\log \frac{1}{1-q^x}}{\log \frac{1}{x}} = \lim_{x\to 0} \frac{x q^x \ln(1/q)}{1-q^x}$$

$$= \ln(\frac{1}{q}) \lim_{x\to 0} \frac{x}{1-q^x} \cdot \underbrace{\lim_{x\to 0} q^x}_{=1} = \ln(\frac{1}{q}) \lim_{x\to 0} \frac{1}{q^x \ln(1/q)} = 1$$

□

It follows that, choosing the smallest k satisfying (4), every rank computation requires $O(\log \rho_n) = O(\log n)$ applications of the Monte-Carlo algorithm. Therefore, we have that $R(n) = O(n^2 \log^2 n \log\log n)$ and thus

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

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

arithmetic operations in $\mathbb{K}$.

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. Although our complexity results do not improve the worst-case 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. We also think that output-sensitive analysis should be the key to better understand the behavior of reduction-based algorithms, in particular why they often show a almost linear performance on practical data (as observed, for instance, in [1]) although their worst-case complexity is cubic.

Generalizations. The result of this paper can be generalized in various ways with only moderate extra effort; we have restricted the problem for simplicity of presentation only. We talk about several possibilities which work independent of each other; we postpone a more detailed discussion to a full version of the paper.

- We have concentrated on ordinary persistent homology in this work. The generalization to extended persistence [10] is straightforward through a more careful filtration of the coned complex (Section 5).

- With a more careful analysis of the algorithm, we can determine the constant hidden in the O-notation of Theorem 8. It is not difficult to see that this constant is of the form $\frac{c}{\delta}$, where c is a constant that does not depend on δ. However, we can further improve by changing the algorithm to compute $P(\Gamma)_{a,m}^{m,b}$, according to Footnote 1 in the text; with that, we avoid the overhead for duplicates of output classes and decrease the influence of δ on the constant.

- In the paper, the boundary matrix contains only $O(n)$ non-zero entries, because we assumed a simplical complex with a constant dimension. More generally, we can define $R(n, e)$ to be cost of a rank computation of an $n \times n$-matrix with at most e entries. This generalization does not affect the bound of Theorem 8, but it changes the bounds for computing the ranks. For instance, if the dimension d of the simplicial complex is not assumed to be constant, we have that $e \leq (d+1)n$, and Theorem 16 from the appendix yields that a rank can be computed with high probability in

  $$O(n^2(d + \log n \log \log n)).$$

  This generalized setup also allows an analysis of the algorithm for cubical or even generalized CW-complexes.

Computing representative cycles. Birth and death times are sufficient for computing the persistence diagram. However, one may want to compute a 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 generated 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 would give a cycle representing a class created by σ_i. The linear equation system can be solved in time $R(n)$ (again, cf. [20], [15], [21] for deterministic, Las-Vegas, and Monte-Carlo algorithms), and thus, the computation takes $O(C_\Gamma R(n))$ 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.

8. REFERENCES

[1] P. Bendich, H. Edelsbrunner, M. Kerber: "Computing Robustness and Persistence for Images". *IEEE Transactions on Visualization and Computer Graphics* **16** (2010) 1251–1260.

[2] P. Bendich, H. Edelsbrunner, D. Morozov, A. Patel: "The Robustness of Level Sets". In: *Proceedings of the 18th European Symposium on Algorithms*, 2010 1–10.

[3] 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 247–256.

[4] 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.

[5] 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.

[6] 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 237–246.

[7] C. Chen, M. Kerber: "Persistent Homology Computation with a Twist". In: *27th European Workshop on Computational Geometry*, 2011 .

[8] M. Chung, P. Bubenik, P. Kim: "Persistence Diagrams of Cortical Surface Data". In: J. Prince, D. Pham, K. Myers (eds.) *Information Processing in Medical Imaging, LNCS*, vol. 5636, 386–397, 2009.

[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] D. Cohen-Steiner, H. Edelsbrunner, D. Morozov: "Vines and vineyards by updating persistence in linear time". In: *Proceedings of the twenty-second Annual Symposium on Computational Geometry.* SCG '06, 2006 119–126.

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

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

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

[15] 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 143–150.

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

[17] H. Edelsbrunner, D. Letscher, A. Zomorodian: "Topological Persistence and Simplification". *Discrete & Computational Geometry* **28** (2002) 511–533.

[18] 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 2010 Annual Symposium on Computational Geometry*, 2010 267–276.

[19] B. Hudson, G. Miller, S. Oudot, D. Sheehy: "Topological inference via meshing". In: *Proceedings of the 2010 Annual Symposium on Computational Geometry.* ACM, 2010 277–286.

[20] 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.

[21] E. Kaltofen, B. David Saunders: "On Wiedemann's method of solving sparse linear systems". In: H. Mattson, T. Mora, T. Rao (eds.) *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, vol. 539, 29–38. Springer, 1991.

[22] N. Milosavljević, D. Morozov, P. Škraba: "Zigzag Persistent Homology in Matrix Multiplication Time". In: *Proceedings of the twenty-seventh Annual Symposium on Computational Geometry*, 2011 .

[23] D. Morozov: "Persistence Algorithm Takes Cubic Time in the Worst Case". In: *BioGeometry News.* Duke Computer Science, Durham, NC, 2005.

[24] J. Munkres: *Elements of algebraic topology.* Westview Press, 1984.

[25] A. Schönhage, V. Strassen: “Schnelle Multiplikation grosser Zahlen”. *Computing* **7** (1971) 281–292. In German.

[26] V. de Silva, R. Ghrist: “Coverage in Sensor Networks via Persistent Homology”. *Algebraic and Geometric Topology* **7** (2007) 339–358.

[27] 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 .

[28] 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 65–75.

[29] D. H. Wiedemann: “Solving sparse linear equations over finite fields”. *IEEE Transactions on Information Theory* **32** (1986) 54–62.

[30] A. Zomorodian: “The tidy set: a minimal simplicial set for computing homology of clique complexes”. In: *Proceedings of the 2010 Annual Symposium on Computational Geometry*, 2010 257–266.

[31] A. Zomorodian, G. Carlsson: “Computing persistent homology”. *Discrete and Computational Geometry* **33** (2005) 249–274.

[32] A. Zomorodian, G. Carlsson: “The Theory of Multidimensional Persistence”. *Discrete and Computational Geometry* **42** (2009) 73–93.

Appendix A: Details on the Monte-Carlo method

We explain how our Theorem 13 follows from Theorem 3 in [21]. 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 a $n \times n$ 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 $O(n)$ nonzero entries, a matrix-vector multiplication needs $O(n)$ 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.