

WORST-CASE EXAMPLES FOR THE COMPUTATION OF PERSISTENT HOMOLOGY

UZAY ÇETIN AND ERGÜN YALÇIN

ABSTRACT. We construct worst-case examples for the standard reduction algorithm for computing persistent homology. Our constructions are similar to the worst-case examples introduced by Morozov, but we replace the single-triangle arrangement with a strip of base and fin triangles. This structure allows us to give an explicit algorithm for their construction and to perform experiments comparing the runtimes of different variants of the reduction algorithm. We further show that, after suitable edge and triangle subdivisions, these strip examples remain worst-case and can be realized as clique complexes of filtered graphs, and hence as Vietoris–Rips complexes of finite point clouds for a sequence of scale parameters.

1. INTRODUCTION

Due to the population increase and advances in internet and digital technologies, data has become one of the most important aspects of our lives. While classical mathematical methods such as statistical and clustering techniques are useful in data analysis, Topological Data Analysis (TDA) has emerged as an important new tool for analyzing data in the last 20 years [3]. TDA is currently used in many areas, including genomics, robotics, medical imaging, cancer research, the spread of infectious diseases, financial networks, and natural language analysis (see, for example, [5], [7], [12], and [6]).

Topological Data Analysis uses methods from algebraic topology to analyze data. One of the main methods in TDA is persistent homology, which was developed by Edelsbrunner, Letscher, Zomorodian, Carlsson, and others (see [2], [15]). In most of the applications, TDA via persistent homology takes as input a finite metric space embedded in $\mathbb{R}^m$ and produces persistent diagrams as output. One of the reasons that makes persistent homology so effective is that it can be computed by a simple algorithm called the reduction algorithm. There are many computer packages based on different variants of the reduction algorithm that compute the persistent diagram of a filtered simplicial complex (see [13] for a comparison of the runtime performance of different packages).

In [10], Morozov constructed a family of filtered simplicial complexes that provide worst-case examples for the reduction algorithm, exhibiting a runtime of $\Omega(N^3)$, where N denotes the number of simplices. The main purpose of this paper is to give a new family of worst-case examples similar to Morozov’s examples, but they are constructed by successively adding base and fin triangles to a strip of triangles instead of adding triangles inside a single triangle. We refer to this new family of worst-case examples as *strip examples*. Because of their strip structure, it is easier to give an explicit algorithm for their construction and to perform experiments comparing the runtime of different variants of the reduction algorithm.

In Theorem 3.2, we show that the runtime of the reduction algorithm on the strip worst-case examples is again $\Omega(N^3)$, where N is the number of simplices. In Appendix A, we present

Date: April 16, 2026.

2020 Mathematics Subject Classification. 55N31 (primary), 68W40, 68R10 (secondary).

Key words and phrases. Topological Data Analysis, Persistent Homology, Reduction Algorithm, Algorithmic Complexity, Clique Complexes.

a simple pseudo-code for constructing the strip examples for any number of triangles. In Section 4 we use this code to perform computational experiments and report on the results comparing the runtimes of different variants of the reduction algorithm such as the standard reduction algorithm [2], the reduction algorithm with a twist [4], and the Look-ahead variant of the reduction algorithm [11]. Throughout the paper, we measure computational complexity in terms of the number of field additions on nonzero entries.

In Section 5, we show that after modifying the strip examples by subdividing some of the triangles and edges, they can be realized as clique complexes of filtered graphs. Then we show that these filtered clique complexes can be realized as Vietoris–Rips complexes of a finite point cloud in some Euclidean space. As a consequence, we obtain that these modified strip examples can be realized as Vietoris–Rips complexes of finite point clouds for some sequence of real numbers.

Complexity questions for Erdős–Rényi and Vietoris–Rips filtration models were also studied by Giunti, Houry, Kerber, and Söls in [9]. In that paper, they construct worst-case examples for the standard reduction algorithm performed on the 2-dimensional boundary matrix for the Erdős–Rényi filtration model (see Remark 5.7).

2. MATRIX REDUCTION ALGORITHMS

We refer the reader to [7], [12], and [6] for basic definitions on persistent homology. Here we only give the definitions related to filtered simplicial complexes that will be used in the paper to explain the reduction algorithm.

Definition 2.1. A *simplicial complex* K consists of a pair (V, S) where V is the set of vertices of K , and S is a set of subsets of V , called the simplices of K , satisfying the following properties:

- (1) for all $v \in V$, $\{v\} \in S$,
- (2) if $\sigma \in S$ and $\tau \subseteq \sigma$, then $\tau \in S$.

A simplex σ in K is called a *k-simplex* or a *k-dimensional simplex* if $|\sigma| = k + 1$. For a simplex σ in K , a subset $\tau \subseteq \sigma$ is called a *face* of σ . If $\dim \tau = \dim \sigma - 1$, then τ is called a *facet* of σ . We say the simplicial complex K is finite if its vertex set V is finite. Throughout the paper we assume all the simplicial complexes are finite. The dimension of K is the largest d such that K has a d -dimensional simplex. We say that the simplicial complex $K' = (V', S')$ is a subcomplex of $K = (V, S)$ if $V' \subseteq V$ and $S' \subseteq S$. In this case we write $K' \subseteq K$.

Definition 2.2. A nested sequence of simplicial complexes

$$\emptyset \subseteq K_1 \subseteq K_2 \subseteq \cdots \subseteq K_m = K$$

is called a *filtration* of K . A simplicial complex with a filtration is called a *filtered simplicial complex*.

A filtration function on a simplicial complex $K = (V, S)$ is a function $f : S \rightarrow \mathbb{R}$ satisfying $f(\tau) \leq f(\sigma)$ whenever $\tau \subseteq \sigma$. For each filtered complex K , we can define a filtration function $f : K \rightarrow \mathbb{Z}_{>0}$ such that for each simplex σ , $f(\sigma)$ is the smallest $i \in \mathbb{Z}_{>0}$ such that $\sigma \in K_i$. Conversely, given a filtration function f on a complex K , we can choose finitely many real numbers $r_1, \dots, r_m$ and define the filtration on K by taking $K_i \subseteq K$ to be the subcomplex consisting of simplices $\sigma \in K$ such that $f(\sigma) \leq r_i$.

Let $K = (V, S)$ be a filtered simplicial complex $\emptyset \subseteq K_1 \subseteq K_2 \subseteq \cdots \subseteq K_m = K$ with associated filtration function $f : K \rightarrow \mathbb{Z}^+$. Consider the chain complex $C_*(K)$ of K in $\mathbb{F}_2$ -coefficients. We can consider the chain complex $C_*(K)$ as a graded $\mathbb{F}_2$ -vector space $C_*(K) =$

$\bigoplus_{p=0}^d C_p(K)$, where $d = \dim(K)$, and consider the family of boundary maps $\partial_* = \{\partial_p : C_p(K) \rightarrow C_{p-1}(K)\}_{p=1}^d$ as a linear transformation of degree -1 between graded vector spaces.

To define a matrix representing the boundary map $\partial_* : C_*(K) \rightarrow C_*(K)$, we assume that the simplices in K are totally ordered, i.e. sorted as a list $\sigma_1, \dots, \sigma_N$, in such a way that it satisfies the following properties:

- (1) If σ_i is a face of σ_j , then $i < j$.
- (2) If $f(\sigma_i) < f(\sigma_j)$, then $i < j$.

Such an ordering of simplices of K is called *ordering compatible* with the filtration. Because of condition (1), every face of a simplex appears earlier in the ordering, hence the boundary matrix is strictly upper triangular.

Definition 2.3. Let K be a filtered simplicial complex and $S = \{\sigma_1, \sigma_2, \dots, \sigma_N\}$ be the set of simplices of K ordered with an ordering compatible with the filtration. The *boundary matrix* of K is the $N \times N$ -matrix D with entries in $\mathbb{F}_2$ such that

$$D_{i,j} = \begin{cases} 1 & \text{if } \sigma_i \text{ is a facet of } \sigma_j, \\ 0 & \text{otherwise.} \end{cases}$$

For each j , the j -th column of the boundary matrix D is denoted by D_j .

2.1. The Standard Algorithm for Reduction. It is shown in [2] that the computation of the persistent diagram can be accomplished by means of a simple algorithm that brings the boundary matrix into a reduced form. To define this algorithm we need to introduce a function called the low function.

Definition 2.4. For every nonzero vector $\mathbf{v}$ with entries in $\mathbb{F}_2$, the *low* of $\mathbf{v}$ is defined by

$$\text{low}(\mathbf{v}) = \max(\{i \mid \mathbf{v}[i] = 1\})$$

where $\mathbf{v}[i]$ denotes the i -th entry of $\mathbf{v}$. We set $\text{low}(\mathbf{0}) = 0$.

The standard reduction algorithm is similar to the Gaussian elimination but it is performed on the columns. For each j running from 1 to N , the j -th column D_j is modified as follows: If $\text{low}(D_i) = \text{low}(D_j)$ for some $i < j$, then the vector D_j is replaced by $D_i + D_j$. Here the addition is performed in the field $\mathbb{F}_2$. Pseudocode for standard algorithm can be given in Algorithm 1.

Algorithm 1 Standard Reduction Algorithm

Input: Boundary Matrix $D = [D_1 | D_2 | \dots | D_N] \in \mathbb{F}_2^{N \times N}$

Output: Reduced Matrix $R \in \mathbb{F}_2^{N \times N}$

$R \leftarrow D$;

for $j = 1$ to N **do**

$\triangleright O(N)$

while $\exists i < j$ with $\text{low}(R_i) = \text{low}(R_j) \neq 0$ **do**

$\triangleright O(N)$

$R_j \leftarrow R_i + R_j$

$\triangleright O(N)$

end while

end for

For every $k = 1, \dots, d$, the persistent diagram $\text{dgm}_k(K)$ for a filtered complex K of dimension d is the set of all persistent pairs $(f(\sigma_i), f(\sigma_j))$ such that $f(\sigma_i)$ is the filtration level where a homology class at dimension k is born, and $f(\sigma_j)$ is the level where it dies (see [7] for details). Note that if R is the reduced matrix obtained from the boundary matrix of K after

applying the reduction algorithm, then the pair $(f(\sigma_i), f(\sigma_j))$ is in the persistent diagram $\text{dgm}_k(K)$ if and only if σ_i is a k -dimensional simplex and $i = \text{low}(R_j) \neq 0$.

For all the columns of R with $R_i = 0$, we can conclude that there is a birth at $f(\sigma_i)$. If $R_i = 0$ for some i and there is no column R_j such that $\text{low}(R_j) = i$, then $(f(\sigma_i), \infty)$ is an infinite persistence interval.

Example 2.5. Let K be the filtered simplicial complex whose simplices in the order of appearance are given as follows:

$$K = \{[1], [2], [3], [1, 2], [1, 3], [2, 3], [1, 2, 3]\}.$$

The boundary matrix of K is:

$$D = \begin{array}{c} \begin{matrix} [1] \\ [2] \\ [3] \\ [1, 2] \\ [1, 3] \\ [2, 3] \\ [1, 2, 3] \end{matrix} \end{array} \begin{bmatrix} \begin{matrix} [1] & [2] & [3] & [1, 2] & [1, 3] & [2, 3] & [1, 2, 3] \end{matrix} \\ \begin{matrix} 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix} \end{bmatrix}$$

Then, the standard reduction algorithm is applied to the boundary matrix D to obtain the reduced matrix R as follows:

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & \color{red}{1} & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & \color{red}{1} & \color{red}{1} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & \color{red}{1} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \xrightarrow{C5+C6 \rightarrow C6} \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & \color{red}{1} & 0 & \color{red}{1} & 0 \\ 0 & 0 & 0 & 0 & \color{red}{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & \color{red}{1} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \xrightarrow{C4+C6 \rightarrow C6} \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & \color{red}{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \color{red}{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & \color{red}{1} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

If we take the filtration function to be the function that assigns σ_i to the index i , then from the indices of the lows in the reduction matrix, we can conclude that the set of persistent pairs for the filtered simplicial complex K is

$$\text{dgm}_0(K) = \{(1, \infty), (2, 4), (3, 5)\} \text{ and } \text{dgm}_1(K) = \{(6, 7)\}.$$

2.2. Reduction Algorithm with a Twist. Chen and Kerber [4] introduced a variation of the reduction algorithm called the reduction algorithm with a twist. The idea behind this new algorithm is that if we do the reduction by first applying it to higher dimensional simplices, we will find a persistent pair $(f(\sigma_i), f(\sigma_j))$ after reducing the j -column before processing the i -th column. Since this pair tells us that the simplex σ_i creates a homology class, we must have the i -th column in the reduced matrix equal to zero. So we can clear this column, i.e. set to zero without doing any column additions. This avoids many unnecessary column operations and makes the reduction algorithm run much faster.

To introduce the pseudocode for this algorithm, first note that the boundary matrix can be decomposed to smaller boundary matrices for linear maps $\partial_p : C_p(K) \rightarrow C_{p-1}(K)$ for $p = 1, \dots, d$. The reduction algorithm can be described using these smaller boundary matrices. Pseudocode for the reduction algorithm with a twist is given in Algorithm 2.

Example 2.6. Let D denote the boundary matrix of the filtered simplicial complex as in Example 2.5. Then, the reduced matrix R is obtained using Chen and Kerber's algorithm after applying only one step:

Algorithm 2 Reduction Algorithm with a Twist

Input: Boundary Matrix $D = [D_1 | D_2 | \dots | D_N] \in \mathbb{F}_2^{N \times N}$
Output: Reduced Matrix $R \in \mathbb{F}_2^{N \times N}$
 $R \leftarrow D$; $L : [0, 0, \dots, 0]$
for $\delta = d, \dots, 1$ **do**
 for $j = N, \dots, 1$ **do**
 if $\dim \sigma_j = \delta$ **then**
 while $R_j \neq 0$ and $L[\text{low}(R_j)] \neq 0$ **do**
 $R_j \leftarrow R_j + R_{L[\text{low}(R_j)]}$
 end while
 if $R_j \neq 0$ **then**
 $i \leftarrow \text{low}(R_j)$
 $L[i] \leftarrow j$
 $R_i \leftarrow 0$
 end if
 end if
 end for
end for

$$D = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & \color{red}{1} & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & \color{red}{1} & \color{red}{1} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & \color{red}{1} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \xrightarrow{\text{clear the 6-th column}} \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & \color{red}{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \color{red}{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & \color{red}{1} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} = R$$

The twist algorithm is a very efficient algorithm, and it is used in all the current persistent homology calculation packages such as Gudhi, Phat, and Ripser. We discuss the runtime of the twist algorithm in Section 2.4.

2.3. Reduction Algorithm with Look-ahead. In a recent paper, Morozov and Skraba [11] introduced an algorithm for persistent homology in matrix multiplication time. In this paper they also discuss a look-ahead variant of the standard reduction algorithm. The main idea of this algorithm is that if the low of the column R_j is $i \neq 0$, then for every column $R_{j'}$ with $j' > j$ and $R_{i,j'} = 1$, we add column R_j to $R_{j'}$ to make $R_{i,j'} = 0$. Pseudocode for this algorithm is given in Algorithm 3.

Example 2.7. Let D denote the boundary matrix as in Example 2.5. Then, with the look-ahead reduction algorithm, the reduced matrix R is obtained as follows:

$$D = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & \color{red}{1} & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & \color{red}{1} & \color{red}{1} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & \color{red}{1} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \xrightarrow{C4+C6 \rightarrow C6} \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & \color{red}{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \color{red}{1} & \color{red}{1} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & \color{red}{1} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \xrightarrow{C5+C6 \rightarrow C6} \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & \color{red}{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \color{red}{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & \color{red}{1} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} = R$$

Note that in this example the reduced matrix is obtained using exactly the same number of operations as in the standard algorithm. This is not true in general. There are examples where the look-ahead reduction algorithm performs better than the standard reduction algorithm.

Algorithm 3 Reduction Algorithm with Look-ahead

Input: Boundary Matrix $D = [D_1 | D_2 | \dots | D_N] \in \mathbb{F}_2^{N \times N}$
Output: Reduced Matrix $R \in \mathbb{F}_2^{N \times N}$
 $R \leftarrow D;$
for $j = 1$ to N **do** $\triangleright O(N)$
 if $\text{low}(R_j) \neq 0$ **then**
 $i \leftarrow \text{low}(R_j)$
 for $j' > j$ and $R[i, j'] = 1$ **do** $\triangleright O(N)$
 $R[\cdot, j'] \leftarrow R[\cdot, j'] + R[\cdot, j]$ $\triangleright O(N)$
 end for
 end if
end for

We discuss the runtime of the look-ahead variant of the reduction algorithm applied to the strip worst-case examples in Section 4.

2.4. The runtime of reduction algorithms. It is shown that for all three variants of the reduction algorithm, the runtime is $O(N^3)$ where N is the number of simplices.

For the standard reduction algorithm, we see this by calculating the runtime of the different steps of the algorithm. In this algorithm we have a **for** loop with N elements so it runs in $O(N)$ time. The **while** loop also runs in $O(N)$. Inside these loops the summation of two vectors of size N is performed in $O(N)$ time. So the runtime of the standard reduction algorithm is $O(N^3)$.

The runtime calculation for the reduction algorithm with a twist is similar. In this algorithm, the first two **for** loops each run in $O(N)$, the **while** loop runs in $O(N)$, and the additions are performed in $O(N)$ time. So the complexity of the reduction algorithm with a twist is also $O(N^3)$.

It is shown that in practice the twist algorithm is much more efficient because the boundary matrix is a sparse matrix in general. There is an explicit calculation for the runtime of this algorithm in terms of the number of persistent pairs in the reduced matrix (see [4]). In the experiments that we performed with random data set consisting of 10 points, we observed that the number of vector additions that are performed goes down drastically by using the reduction algorithm with twist. Our experiments with the worst case strip examples also support this claim (see Section 4).

In the look-ahead variant of the standard algorithm, we have two **for** loops each having runtime $O(N)$, and there is a vector addition with runtime $O(N)$. Hence the look-ahead algorithm also runs in $O(N^3)$. Similar to the twist algorithm we can perform some calculations to compare the efficiency of the look-ahead reduction algorithm with other algorithms. In our experiments with a random dataset consisting of 10 points, we obtained that the look-ahead reduction algorithm performs better than the standard reduction algorithm.

3. AN ALTERNATIVE WORST-CASE EXAMPLE: STRIP

In this section we construct a worst-case example for the reduction algorithm, called the strip example, similar to the worst-case examples introduced by Morozov in [10].

For each positive integer n , the *strip worst-case complex* $X(n)$ is a filtered simplicial complex consisting of n base triangles sitting on a plane and n fin triangles with one edge lying on the plane and one vertex lying outside the plane. We place these base triangles and fin triangles as shown in Figure 1. The base triangles are labeled $\textcircled{i}$ and the fin triangles are labeled

$\textcircled{-i}$ for $i = 1, \dots, n$. The vertices of these triangles are labeled $v_1, \dots, v_{n+2}$ and $f_1, \dots, f_n$. The labeling of the base triangles starts from the center and then alternates left and right.

We label the horizontal edges of the base triangles $1', \dots, n'$ such that i' is the horizontal edge of the base triangle $\textcircled{i}$. The vertical base edges are labeled $1, \dots, n+1$ so that 1 is the edge between v_1 and v_2 , and for all $i = 2, \dots, n+1$, the edge i is between v_{i-1} and v_{i+1} as shown in Figure 1. The base edge with label $n+1$ is also labeled as $0'$ to make it easier to justify its place in the filtration.

The edges of the fin triangles are labeled $-1, \dots, -n$ and $-1^*, \dots, -n^*$. The edge -1 is between the vertices f_1 and v_2 , and the edge -1^* is between f_1 and v_1 . For all $i = 2, \dots, n$, the edge $-i$ is between the vertices f_i and v_{i+1} , and the edge $-i^*$ is between f_i and v_{i-1} . Note that there is one fin triangle above each vertical base edge except the edge with label $n+1 = 0'$.

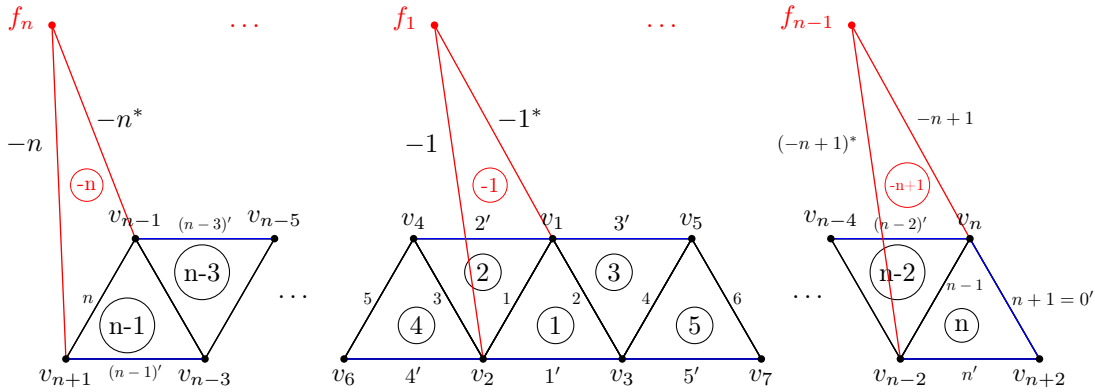


FIGURE 1. The simplicial complex $X(n)$ for $n \equiv 1 \pmod{4}$.

The strip worst-case examples grow inductively by adding triangles. To reach from $X(n)$ to $X(n+1)$ we just add a fin triangle above the edge $n+1$ and a base triangle next to the edge with label n . Note that the simplicial complex $X(n)$ has $2n+2$ vertices, $4n+1$ edges, and $2n$ triangles. The number of simplices of $X(n)$ is $N = 8n+3$. The Euler characteristic of $X(n)$ is equal to 1 since it has homology isomorphic to the homology of a point. To define the filtration on $X(n)$, we order the simplices of the complex using the following ordering:

- (1) Fin vertices $f_1, \dots, f_n$,
- (2) Base vertices $v_1, \dots, v_{n+2}$,
- (3) Horizontal base edges that destroy components $n', (n-1)', \dots, 1'$,
- (4) Base edge $n+1$ which is also labeled as $0'$,
- (5) Fin edges that destroy fin components $-n^*, \dots, -1^*$,
- (6) Fin edges that create cycles $-n, -n+1, \dots, -1$,
- (7) Vertical base edges $n, n-1, \dots, 1$,
- (8) Base triangles $\textcircled{1}, \textcircled{2}, \dots, \textcircled{n}$, and
- (9) Fin triangles $\textcircled{-1}, \textcircled{-2}, \dots, \textcircled{-n}$.

To simplify calculations, in the proof of Theorem 3.2 we apply the reduction algorithm only on the boundary matrix for $\partial_2 : C_2(X(n)) \rightarrow C_1(X(n))$. We denote this matrix by $B_2(n)$ and the associated reduced matrix by $R_2(n)$.

Example 3.1. For $n = 2$, the boundary matrix $B_2(2)$ and the reduced matrix $R_2(2)$ are as follows:

$$B_2(2) = \begin{matrix} & \textcircled{1} & \textcircled{2} & \textcircled{-1} & \textcircled{-2} \\ \begin{matrix} 2' \\ 1' \\ 0' \\ -2^* \\ -1^* \\ -2 \\ -1 \\ 2 \\ 1 \end{matrix} & \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} & \xrightarrow{\begin{matrix} C_1+C_2 \rightarrow C_2 \\ C_1+C_3 \rightarrow C_3 \\ C_2+C_3 \rightarrow C_3 \\ C_2+C_4 \rightarrow C_4 \end{matrix}} & \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} & = R_2(2) \end{matrix}$$

The processing of the columns of the matrix $B_2(2)$ can be explained using the sparse matrix notation:

$$\textcircled{1} \rightarrow (1, 2, 1')$$

Note that the low of the column for $\textcircled{1}$ is the row corresponding to the edge 1. In this case we say that $\textcircled{1}$ is paired with the edge 1 under the reduction. For the triangle $\textcircled{2}$, we have

$$\textcircled{2} \rightarrow (1, 0', 2')$$

$$\textcircled{1} + \textcircled{2} \rightarrow (1, 2, 1') + (1, 0', 2') = (2, 0', 1', 2')$$

This pairs the triangle $\textcircled{2}$ with the edge 2. For the fin triangles we have a similar pairing process.

$$\textcircled{-1} \rightarrow (1, -1, -1^*)$$

$$\textcircled{-1} + \textcircled{1} \rightarrow (1, -1, -1^*) + (1, 2, 1') = (2, -1, -1^*, 1')$$

$$\textcircled{-1} + \textcircled{1} + ((\textcircled{1}) + \textcircled{2}) \rightarrow (2, -1, -1^*, 1') + (2, 0', 1', 2') = (-1, -1^*, 0', 2')$$

This makes the fin triangle $\textcircled{-1}$ pair with the edge -1 . Finally for the fin triangle $\textcircled{-2}$, we have

$$\textcircled{-2} \rightarrow (2, -2, -2^*)$$

$$\textcircled{-2} + ((\textcircled{1}) + \textcircled{2}) \rightarrow (2, -2, -2^*) + (2, 0', 1', 2') = (-2, -2^*, 0', 1', 2')$$

This pairs the triangle $\textcircled{-2}$ with the edge -2 . So the persistent pairs for the 1-dimensional homology H_1 are

$$\text{dgm}_1(X(2)) = \{(1, \textcircled{1}), (2, \textcircled{2}), (-1, \textcircled{-1}), (-2, \textcircled{-2})\}.$$

Here we listed the persistent pairs using the simplices since in the above matrices we used the simplices for labeling the columns and rows. Note that these pairs correspond to the indices of the lows in the reduced matrix $R_2(2)$.

Now we prove our main result in this section. In the following calculations, we measure complexity in terms of the number of field additions on nonzero entries.

Theorem 3.2. *Let $B_2(n)$ denote the matrix for the boundary map $\partial_2 : C_2(X(n)) \rightarrow C_1(X(n))$. The runtime of the standard reduction algorithm applied to $B_2(n)$ has complexity $\Omega(n^3)$. As a consequence, the complexity of the standard reduction algorithm for the boundary matrix of $X(n)$ is $\Omega(N^3)$ where $N = 8n + 3$ is the number of simplices of $X(n)$.*

Proof. Extending the pairing algorithm given above for $n = 2$, we see that the base triangles $\textcircled{i}$ will be paired by the base edges i with the following list of representatives:

$$\begin{aligned} \textcircled{1} &\rightarrow (1, 2, 1') \\ \textcircled{1} + \textcircled{2} &\rightarrow (1, 2, 1') + (1, 3, 2') = (2, 3, 1', 2') \\ \textcircled{1} + \textcircled{2} + \textcircled{3} &\rightarrow (2, 3, 1', 2') + (2, 4, 3') = (3, 4, 1', 2', 3') \\ &\vdots \\ \textcircled{1} + \textcircled{2} + \cdots + \textcircled{n-1} &\rightarrow (n-1, n, 1', 2', \dots, (n-1)') \\ \textcircled{1} + \textcircled{2} + \cdots + \textcircled{n} &\rightarrow (n, 0', 1', 2', \dots, n') \end{aligned}$$

This table should be read in the following way. The boundary of $\textcircled{1}$ is $(1, 2, 1')$ as an ordered 3-tuple of edges. So $\textcircled{1}$ is paired with 1. The boundary of the triangle $\textcircled{2}$ is $(1, 3, 2')$. Since 1 is already paired with $\textcircled{1}$, we need to add the column of $\textcircled{1}$ to the column of $\textcircled{2}$. This creates a chain $\textcircled{1} + \textcircled{2}$ with boundary $(2, 3, 1', 2')$. So the triangle $\textcircled{2}$ is paired with 2. We continue this way until for all i , the triangle $\textcircled{i}$ is paired with edge i . From the table we see that in the reduced matrix $R_2(n)$, the column for $\textcircled{i}$ has $i + 2$ nonzero entries. For each i , let us denote the chain $\textcircled{1} + \cdots + \textcircled{i} = (i, i + 1, 1', 2', \dots, i')$ by c_i .

The complexity of processing the base triangles can be calculated as follows: Since the complexity of adding two vectors with a_1 and a_2 nonzero entries is equal to $a_1 + a_2$, the complexity of performing all the above additions is

$$(3 + 3) + (4 + 3) + (5 + 3) + \cdots + ((n + 1) + 3) = \frac{1}{2}(n^2 + 9n - 10).$$

Now we calculate the complexity of processing the fin triangles. For each i , the processing of the fin triangle $\textcircled{-1}$ is illustrated in the following table:

$$\begin{aligned} \textcircled{-1} &\rightarrow (1, -1, -1^*) \\ \textcircled{-1} + c_1 &\rightarrow (1, -1, -1^*) + (1, 2, 1') = (2, -1, -1^*, 1') \\ (\textcircled{-1} + c_1) + c_2 &\rightarrow (2, -1, -1^*, 1') + (2, 3, 1', 2') = (3, -1, -1^*, 2') \\ (\textcircled{-1} + c_1 + c_2) + c_3 &\rightarrow (3, -1, -1^*, 2') + (3, 4, 1', 2', 3') = (4, -1, -1^*, 1', 3') \\ (\textcircled{-1} + c_1 + c_2 + c_3) + c_4 &\rightarrow (4, -1, -1^*, 1', 3') + (4, 5, 1', 2', 3', 4') = (5, -1, -1^*, 2', 4') \\ &\vdots \\ (\textcircled{-1} + c_1 + c_2 + \cdots + c_{n-1}) + c_n &\rightarrow \begin{cases} (-1, -1^*, 0', 1', 3', 5', \dots, n') & \text{if } n \text{ is odd} \\ (-1, -1^*, 0', 2', 4', 6', \dots, n') & \text{if } n \text{ is even.} \end{cases} \end{aligned}$$

This makes the fin triangle $\textcircled{-1}$ pair with the edge -1 . There is a similar process for each of the fin triangles. If we assume that the processing of the fin triangle $\textcircled{-1}$ is done by performing

the additions $\textcircled{-1} + c_1 + \dots + c_n$, then the total number of field additions on nonzero entries is equal to

$$3 + \sum_{i=1}^n (i + 2) = \frac{1}{2}(n^2 + 5n + 6).$$

In a similar way if we process the j -th fin triangle $\textcircled{-j}$, we see that we will need to perform the addition $\textcircled{-j} + c_j + \dots + c_n$, hence the total number of additions is

$$3 + \sum_{i=j}^n (i + 2) = \frac{1}{2}(n^2 + 5n - j^2 - 3j + 10).$$

Summing these over all $j = 1, \dots, n$, we obtain that the number of additions is equal to

$$\frac{1}{6}n(2n^2 + 9n + 25).$$

Adding this to the complexity of processing the base triangles, we find the total number of additions is

$$S(n) = \frac{1}{3}(n^3 + 6n^2 + 26n - 15).$$

In particular, $S(n) = \Omega(n^3)$. Since the number of simplices in the complex is $N = 8n + 3$, it follows that the worst-case running time of the standard reduction algorithm is $\Omega(N^3)$. $\square$

Remark 3.3. In our implementation, additions are carried out at each intermediate step, so the actual number of additions performed is usually larger than $S(n)$.

4. EXPERIMENTS WITH THE STRIP EXAMPLE

One of our motivations for introducing an alternative worst-case example is to be able to write computer programs that use worst-case examples for performance tests. The linear structure of the strip example allows us to give a simple explicit algorithm for producing the complex $X(n)$ for every $n \geq 1$. This algorithm can be found in Appendix A. The Python code written using this algorithm and the results of the experiments that we ran can be found on the GitHub page: https://github.com/Ergun1234/Morozov_type_Examples.

After running the Python code for constructing the strip example $X(n)$ and running the standard reduction algorithm for the boundary matrix of $X(n)$, we obtained the persistent diagrams for the strip examples. Figure 2 shows the filtered simplicial complexes $X(n)$ for some values of n , and Figure 3 illustrates the corresponding persistence diagrams.

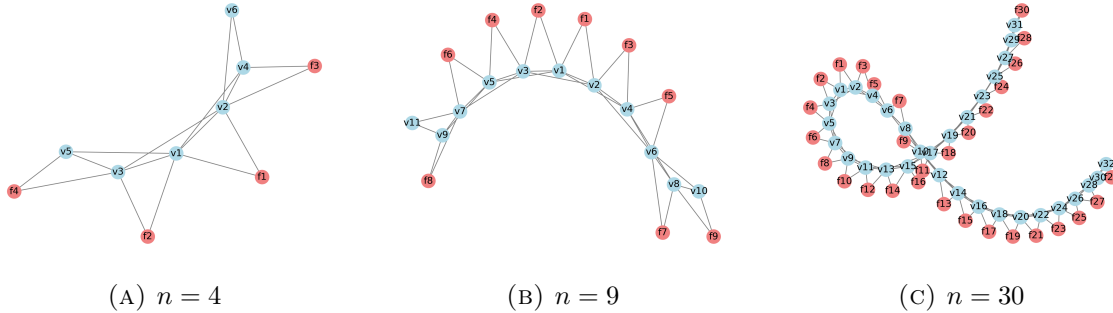


FIGURE 2. The 1-skeletons of the strip worst-case simplicial complexes $X(n)$ for different values of n .

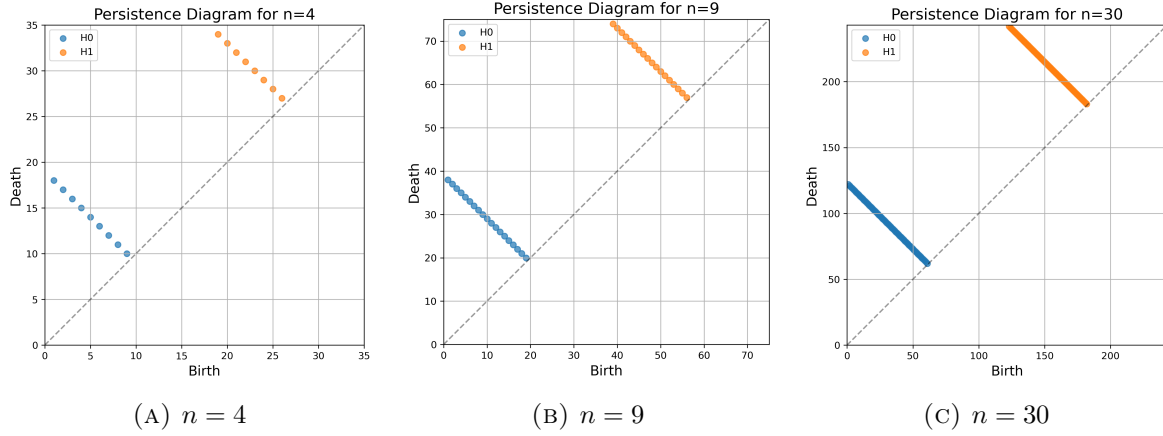


FIGURE 3. Persistence diagrams for strip worst-case examples $X(n)$ for different values of n . Blue points correspond to H_0 features and orange points correspond to H_1 features. The dashed line indicates the diagonal $y = x$.

We performed further experiments with the strip worst-case examples and calculated the runtime of the reduction algorithms, the standard reduction algorithm, the reduction algorithm with a twist, and the reduction algorithm with look-ahead. Figure 4 shows the running times of these three algorithms where the runtime is calculated using a counter that tracks the number of additions performed during the calculation of the reduced boundary matrix. Figure 5 shows the elapsed running times of the algorithms on a MacBook with an M5 processor.

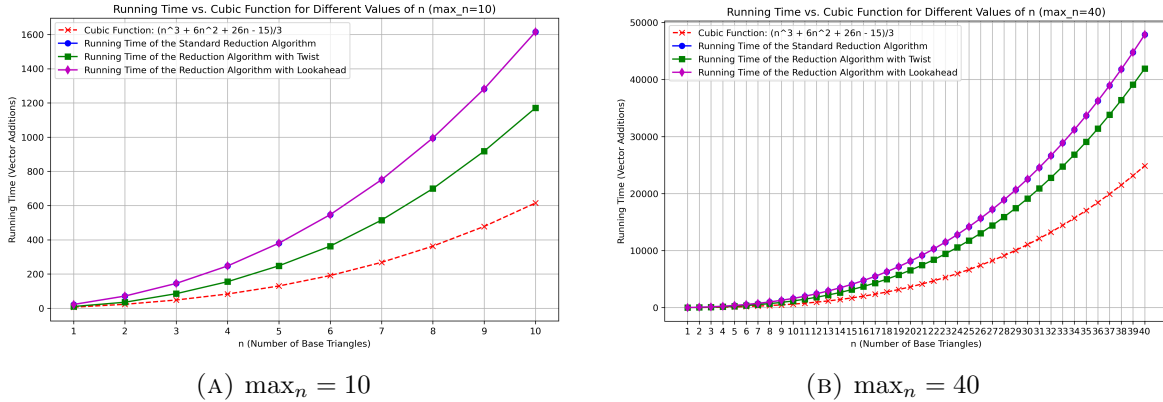


FIGURE 4. Running times of the variants of the reduction algorithm compared with a cubic function, for two values of $\max_n$ (the maximum value of n used in the experiments).

These graphs show that the running times for the standard reduction algorithm and look-ahead reduction algorithm are the same. The runtime of the reduction algorithm with twist is smaller than the other two algorithms. We explain the reason for this in Example 4.1. The elapsed running times for these three algorithms show that both twist and look-ahead variants of the reduction algorithm perform much better than the standard algorithm when they are applied to the strip examples.

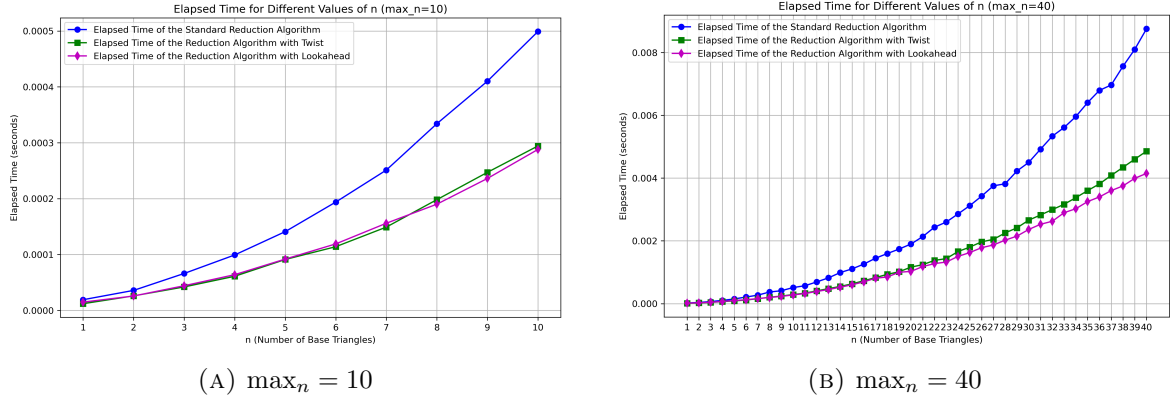


FIGURE 5. Elapsed running time of different variants of the reduction algorithm as a function of the parameter n , for two values of $\max_n$.

Example 4.1. For the strip example $X(2)$, consider the matrix $B_1(2)$ for the boundary map $\partial_1 : C_1(X(2)) \rightarrow C_0(X(2))$.

$$B_1(2) = \begin{matrix} & \begin{matrix} 2' & 1' & 0' & -2^* & -1^* & -2 & -1 & 2 & 1 \end{matrix} \\ \begin{matrix} f_1 \\ f_2 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{matrix} & \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{matrix}$$

To reduce the first five columns of the boundary matrix $B_1(2)$, we perform the column operations $C_1 + C_3 \rightarrow C_3$ and $C_4 + C_5 \rightarrow C_5$, requiring 8 field additions. Then we carry out 9 more column additions to clear the last four columns. These operations require 36 additional field additions. If we instead apply the reduction algorithm with twist, since we first process the matrix $B_2(2)$, we set the last four columns of $B_1(2)$ to zero. This reduces the number of field additions from 44 to 8. Since the processing of $B_2(2)$ requires 27 field additions, the total number of field additions performed for reducing the boundary matrix decreases from 71 to 35. This is the drop observed for $n = 2$ in the first graph in Figure 4. The following table shows the number of field additions performed by the variants of the reduction algorithm for $n \leq 5$.

TABLE 1. Number of field additions performed by the standard reduction algorithm and its variants.

n	Standard reduction	With twist	Look-ahead
1	22	10	22
2	71	35	71
3	145	85	145
4	247	155	247
5	380	248	380

5. A WORST-CASE EXAMPLE THAT IS A FLAG COMPLEX

In this section we show that after modifying the strip complex by subdividing some edges and triangles, the resulting filtered complex is the flag complex of a filtered graph. At the end of the section we also show that these modified strip examples can be realized as the Vietoris–Rips complex of a finite point cloud for some increasing sequence of real numbers.

Definition 5.1. Let $G = (V, E)$ be an undirected loop-free graph. The *clique complex* of G is a simplicial complex $\text{Cl}(G)$ whose vertex set is V and a subset of vertices $\{v_0, \dots, v_k\}$ is a simplex in $\text{Cl}(G)$ if and only if for every $i < j$, $\{v_i, v_j\}$ is an edge in G .

Note that if G is a filtered graph with filtration $\emptyset \subseteq G_1 \subseteq G_2 \subseteq \dots \subseteq G_m$, the associated sequence of clique complexes $\emptyset \subseteq \text{Cl}(G_1) \subseteq \text{Cl}(G_2) \subseteq \dots \subseteq \text{Cl}(G_m)$ gives a filtered simplicial complex. As defined, the strip worst-case examples cannot be realized as clique complexes of filtered graphs since in the filtration the triangles appear several steps after its boundary edges. To realize them as clique complexes, we alter the strip examples as follows:

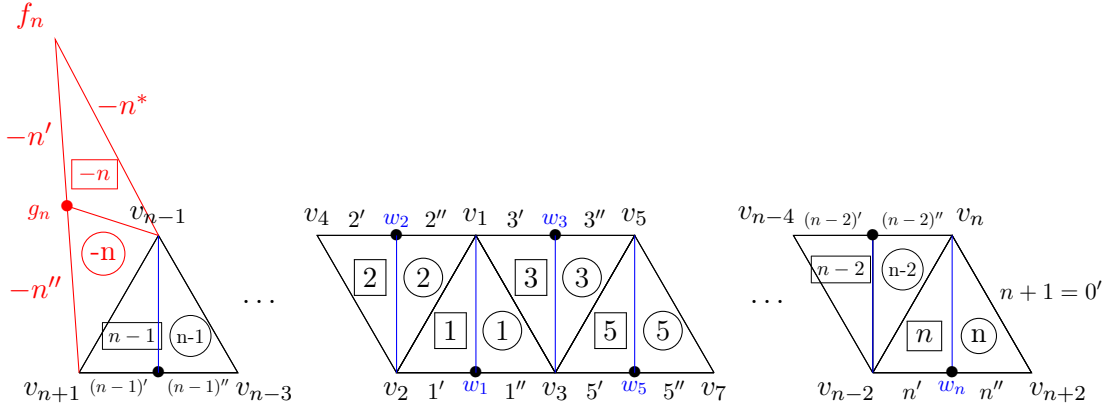


FIGURE 6. The simplicial complex $Y(n)$ for $n \equiv 1 \pmod{4}$.

The picture is similar to the previous example; the difference is that we add vertices $w_1, \dots, w_n$ as the midpoints of the horizontal base edges $1', \dots, n'$, and denote the resulting half edges as i' and i'' for $i = 1, \dots, n$. Similarly, we add vertices $g_1, \dots, g_n$ at the midpoints of the edges $-1, \dots, -n$ of the fin triangles and denote the resulting half edges as $-i'$ and $-i''$ for $i = 1, \dots, n$.

We also add new edges that divide the base and fin triangles into half. For base triangles, these are denoted by $e_1^b, \dots, e_n^b$ and for the fin triangles, by $e_1^f, \dots, e_n^f$. They are indexed by the triangles that they subdivide. For the divided triangles, we denote the left one as boxed i and the right one as the circled i , for base triangles. They are denoted by boxed $-i$ and circled $-i$ for fin triangles. We have the following filtration for this modified simplicial complex:

- (1) Fin vertices $f_1, \dots, f_n$, and $g_1, \dots, g_n$,
- (2) Base vertices $v_1, \dots, v_{n+2}$, and $w_1, \dots, w_n$,
- (3) Horizontal base edges $n'', \dots, 1''$, and $n', \dots, 1'$,
- (4) Base edge $n+1$ which is also labeled as $0'$,
- (5) Fin edges that destroy fin components $-n^*, \dots, -1^*$,
- (6) Fin edges that create cycles $-n'', \dots, -1''$, and $-n', \dots, -1'$
- (7) Vertical base edges $n, n-1, \dots, 1$,
- (8) Base midlines and base triangles $e_1^b, \boxed{1}, \textcircled{1}, \dots, e_n^b, \boxed{n}, \textcircled{n}$,

(9) Fin midlines and fin triangles $e_1^f, \boxed{-1}, \textcircled{-1}, \dots, e_n^f, \boxed{-n}, \textcircled{-n}$.

Note that for all i , the edges e_i^b and the base triangles $\boxed{i}$ and $\textcircled{i}$ appear at the same time in the filtration. Similarly for the fin triangles, the edges e_i^f and the fin triangles $\boxed{-i}$ and $\textcircled{-i}$ appear at the same filtration level. We call this modified example the *modified strip example* and denote it by $Y(n)$.

For each $k = 0, 1, 2$, let $B'_k(n)$ denote the boundary matrix for $\partial_k : C_k(Y(n)) \rightarrow C_{k-1}(Y(n))$. For the boundary matrix, we use the linear ordering as given in the above list.

Example 5.2. In this example we illustrate how the boundary matrix $B'_2(2)$ for the modified strip example $Y(2)$ is reduced using the reduction algorithm. We explain the processing of the columns using the sparse matrix notation. We write the boundary of a simplex as the list of edges where the boundary vector has nonzero entries.

$$\begin{aligned} \boxed{1} &\rightarrow (e_1^b, 1, 1') \\ d_1 = \boxed{1} + \textcircled{1} &\rightarrow (e_1^b, 1, 1') + (e_1^b, 2, 1'') \rightarrow (1, 2, 1', 1'') \end{aligned}$$

So the triangle $\boxed{1}$ is paired with e_1^b , and the triangle $\textcircled{1}$ is paired with edge 1. For the triangles $\boxed{2}$ and $\textcircled{2}$, we have

$$\begin{aligned} \boxed{2} &\rightarrow (e_2^b, 0', 2') \\ \boxed{2} + \textcircled{2} &\rightarrow (e_2^b, 0', 2') + (e_2^b, 1, 2'') \rightarrow (1, 0', 2', 2'') \\ d_2 = (\boxed{2} + \textcircled{2}) + d_1 &\rightarrow (1, 0', 2', 2'') + (1, 2, 1', 1'') = (2, 0', 1', 1'', 2', 2''). \end{aligned}$$

This shows that the triangle $\boxed{2}$ is paired with e_2^b and the triangle $\textcircled{2}$ is paired with 2. For the fin triangles we have a similar pairing process.

$$\begin{aligned} \boxed{-1} &\rightarrow (e_1^f, -1', -1^*) \\ \boxed{-1} + \textcircled{-1} &\rightarrow (e_1^f, -1', -1^*) + (e_1^f, 1, -1'') \rightarrow (1, -1', -1'', -1^*) \\ (\boxed{-1} + \textcircled{-1}) + d_1 &\rightarrow (1, -1', -1'', -1^*) + (1, 2, 1', 1'') \rightarrow (2, -1', -1'', -1^*, 1', 1'') \\ (\boxed{-1} + \textcircled{-1} + d_1) + d_2 &\rightarrow (2, -1', -1'', -1^*, 1', 1'') + (2, 0', 1', 1'', 2', 2'') \\ &\rightarrow (-1', -1'', -1^*, 0', 2', 2''). \end{aligned}$$

This pairs the fin triangle $\boxed{-1}$ with e_1^f and the fin triangle $\textcircled{-1}$ with the edge $-1'$. Finally we have

$$\begin{aligned} \boxed{-2} &\rightarrow (e_2^f, -2', -2^*) \\ \boxed{-2} + \textcircled{-2} &\rightarrow (e_2^f, -2', -2^*) + (e_2^f, 2, -2'') \rightarrow (2, -2', -2'', -2^*) \\ (\boxed{-2} + \textcircled{-2}) + d_2 &\rightarrow (2, -2', -2'', -2^*) + (2, 0', 1', 1'', 2', 2'') \\ &\rightarrow (-2', -2'', -2^*, 0', 1', 1'', 2', 2''). \end{aligned}$$

This pairs the triangle $\boxed{-2}$ with e_2^f and the triangle $\textcircled{-2}$ with the edge $-2'$. So the persistent pairs for $Y(2)$ for the 1-dimensional homology H_1 are

$$\{(e_1^b, \boxed{1}), (1, \textcircled{1}), (e_2^b, \boxed{2}), (2, \textcircled{2}), (e_1^f, \boxed{-1}), (-1', \textcircled{-1}), (e_2^f, \boxed{-2}), (-2', \textcircled{-2})\}.$$

Since some of the pairs consist of simplices that appear at the same filtration level, they will appear as points on the diagonal line in the persistent diagram.

Now we are ready to prove the following:

Proposition 5.3. *The modified strip example $Y(n)$ is the clique complex of a filtered graph and it has time complexity $\Omega(M^3)$ where M is the number of simplices.*

Proof. At each filtration level the modified strip example is the clique complex of its 1-skeleton because all the triangles in the complex appear at the same filtration level as the boundary edges of the triangles. Moreover, no additional 3-cliques arise beyond the prescribed triangles, and no higher-dimensional cliques occur. Hence the complex coincides with the clique complex of its 1-skeleton at each filtration level.

To see that the reduction algorithm applied to $B'_2(n)$ has cubic runtime complexity, observe that the complex $Y(n)$ has $4n + 2$ vertices, $8n + 1$ edges, and $4n$ triangles. So the number of simplices in $Y(n)$ is $M = 16n + 3$. Note that in the reduction algorithm, the boxed triangles are always paired with the edge that divides the triangle into two. To obtain the chains $t_i = \boxed{i} + \textcircled{i}$ and $t_{-i} = \boxed{-i} + \textcircled{-i}$ we perform $2n$ vector additions, requiring $12n$ field additions.

The processing of the chains t_i and t_{-i} are very similar to the processing of the triangles in the original strip example. In fact we use the same number of vector additions to process these chains. The only difference is that the vectors added have more nonzero entries since more edges are introduced in the modified case. This increases the number of field additions in the reduction process. We now calculate exact number of field additions used to process columns for the boundary matrix $B'_2(n)$.

Let $d_i = t_1 + \dots + t_i$. Note that the columns for d_i have $2i + 2$ nonzero entries. Since the column for d_i is obtained by adding the columns t_i and d_{i-1} , the number of field additions to process all the columns for $t_1, \dots, t_n$ is

$$(4 + 4) + (6 + 4) + (8 + 4) + \dots + (2n + 4) = n^2 + 5n - 6.$$

To process the column for t_{-j} , we perform the additions $t_{-j} + d_j + \dots + d_n$ which require

$$4 + \sum_{i=j}^n (2i + 2) = n^2 + 3n - j^2 - j + 6$$

field additions. Adding these over all $j = 1, \dots, n$, we see that to process the columns for fin triangles, we perform $2n(n^2 + 3n + 8)/3$ additions. Hence the total number of field additions to reduce the matrix $B'_2(n)$ is

$$S'(n) = \frac{1}{3}(2n^3 + 9n^2 + 37n - 18).$$

Since $M = 16n + 3$, we have $S'(n) = \Omega(M^3)$. Note that both the number of simplices and the number of field additions in the reduction process increases roughly by a factor of 2 for the modified complex $Y(n)$ compared to the reduction process for the strip example $X(n)$. $\square$

Next we show that the modified strip examples can be realized as the Vietoris–Rips complex of a finite point cloud for a sequence of real numbers $0 < r_1 < \dots < r_s$ as scale parameters. We first recall the definition of a Vietoris–Rips complex of a data cloud.

Definition 5.4. Let X be a finite set of points in $\mathbb{R}^l$. For a real number $r \geq 0$, the *Vietoris–Rips complex* $VR(X, r)$ is defined as the simplicial complex whose vertex set is X , and whose n -simplices are the subsets $\{x_0, \dots, x_n\}$ of X such that $d(x_i, x_j) \leq r$ for every i, j .

For an increasing sequence $0 < r_1 < r_2 < \dots < r_m$ of positive real numbers, we can define a filtered simplicial complex by taking the simplicial complex K_i to be the Vietoris–Rips complex $VR(X, r_i)$. The following proposition is implicit in classical distance geometry and

the theory of Vietoris–Rips complexes, but we were unable to find a direct reference. We include a proof here for completeness.

Proposition 5.5. *Let $\emptyset \subseteq G_1 \subseteq \dots \subseteq G_m$ be a filtered graph on a finite vertex set $V = \{1, \dots, n\}$, and let $\text{Cl}(G_t)$ denote the clique complex of G_t for $t = 1, \dots, m$. Then there exist points*

$$X = \{p_1, \dots, p_n\} \subset \mathbb{R}^l$$

and increasing sequence of real numbers $0 < r_1 < \dots < r_m$ such that for each $t = 1, \dots, m$, there is an isomorphism of simplicial complexes

$$VR(X, r_t) \cong \text{Cl}(G_t).$$

Proof. It is well-known that Vietoris–Rips complexes are flag complexes, hence they are clique complexes of underlying 1-skeletons (see for example [8, Section III.2]). Therefore it suffices to realize the filtered graphs G_i as nested distance–threshold graphs.

Since the $n = 1$ case is trivial, we can assume $n \geq 2$. For each unordered pair $\{i, j\}$, define its *appearance level*

$$\ell(i, j) = \min\{t : \{i, j\} \in E(G_t)\},$$

with $\ell(i, j) = \infty$ if the edge never appears. Note that $\ell(i, j) \geq 1$ for all i, j . Fix

$$\varepsilon := \frac{1}{n-1}, \quad \delta_t := 2^{-t}\varepsilon$$

for $t = 1, \dots, m$, and let $\delta_{m+1} := 0$. Then we have $\frac{1}{2} \geq \delta_1 > \dots > \delta_m > \delta_{m+1} = 0$ and for each i ,

$$\sum_{j \neq i} \delta_{\ell(i, j)} \leq (n-1)\delta_1 = (n-1)\frac{\varepsilon}{2} = \frac{1}{2}.$$

Define an $n \times n$ symmetric matrix $K = (K_{ij})$ by

$$K_{ii} = 1, \quad K_{ij} = \delta_{\ell(i, j)} \text{ for all } i \neq j.$$

By Gershgorin’s circle theorem (see, for example, [14]), every eigenvalue of K lies in the interval $[1 - \frac{1}{2}, 1 + \frac{1}{2}]$. In particular K is positive definite.

By a standard result in linear algebra, every real symmetric positive definite matrix is a Gram matrix. Thus there exist vectors $p_1, \dots, p_n \in \mathbb{R}^l$ such that

$$K_{ij} = \langle p_i, p_j \rangle$$

(see [1, Theorem 7.35]). In particular, $\|p_i\| = 1$ for all i , and

$$\|p_i - p_j\|^2 = \|p_i\|^2 + \|p_j\|^2 - 2\langle p_i, p_j \rangle = 2 - 2\delta_{\ell(i, j)}.$$

For each $t = 1, \dots, m+1$, define

$$d_t := \sqrt{2 - 2\delta_t}.$$

For all i, j , we have $\|p_i - p_j\| = d_{\ell(i, j)}$. Since the sequence δ_t is strictly decreasing, the sequence d_t is strictly increasing. We have

$$1 \leq d_1 < d_2 < \dots < d_m < d_{m+1} = \sqrt{2}.$$

For each $t = 1, \dots, m$, choose a radius r_t such that

$$d_t < r_t < d_{t+1}.$$

Then

$$\|p_i - p_j\| \leq r_t \iff d_{\ell(i, j)} \leq d_t \iff \ell(i, j) \leq t \iff \{i, j\} \in E(G_t).$$

Hence the distance–threshold graph at scale r_t is exactly G_t . This completes the proof. $\square$

We finish the paper with the following corollary.

Corollary 5.6. *The modified strip example $Y(n)$ can be realized as the Vietoris–Rips complex of a finite point cloud for some sequence of real numbers $0 < r_1 < \dots < r_m$ as scale parameters.*

Proof. This follows from Propositions 5.3 and 5.5. $\square$

Remark 5.7. In [9], Giunti, Houry, Kerber, and Söls study the complexity of the persistent homology calculation for Čech, Vietoris–Rips, and Erdős–Rényi filtration models. The Erdős–Rényi filtration model is obtained by taking a complete graph with n vertices, ordering its $n(n-1)/2$ edges uniformly at random, and considering the induced filtration of clique complexes. In [9], a worst-case example for the Erdős–Rényi model is also constructed with n vertices for which the standard reduction algorithm, applied to the matrix of boundary map ∂_2 , has runtime $\Theta(n^7)$. Since this boundary matrix has $\Theta(n^2)$ rows (edges) and $\Theta(n^3)$ columns (triangles), this runtime complexity of $\Theta(n^7)$ is the worst-case complexity for the standard reduction algorithm.

ACKNOWLEDGEMENTS AND FUNDING DECLARATION

The second author is supported by TÜBİTAK 2219-International Postdoctoral Research Fellowship Program (2023, 2nd term). We gratefully acknowledge TÜBİTAK for its support of this research. We thank Matthew Gelvin, Barış Coşkunüzer, Caroline Yalçın, Koray Karabina, and Barbara Giunti for their helpful comments on earlier versions of the paper. Part of the paper was written while the second author was on sabbatical leave visiting University of Waterloo. The second author thanks University of Waterloo for hospitality during his sabbatical leave.

REFERENCES

- [1] S. Axler, *Linear Algebra Done Right*, 3rd ed., Springer, 2015.
- [2] H. Edelsbrunner, D. Letscher, and A. Zomorodian, Topological persistence and simplification, *Discrete Comput. Geom.* **28** (2002), 511–533.
- [3] G. Carlsson, Topology and data, *Bull. Amer. Math. Soc.* **46** (2009), 255–308.
- [4] C. Chen and M. Kerber, Persistent homology computation with a twist, in *Proc. 27th European Workshop on Computational Geometry (EuroCG 2011)*, 2011, pp. 1–4.
- [5] F. Chazal and B. Michel, An introduction to topological data analysis: Fundamental and practical aspects for data scientists, *Front. Artif. Intell.* **4** (2021), 667963.
- [6] B. Coskunüzer and C. Gürçan Akçora, Topological methods in machine learning: A tutorial for practitioners, arXiv:2409.02901, 2024.
- [7] T. K. Dey and Y. Wang, *Computational Topology for Data Analysis*, Cambridge Univ. Press, 2022.
- [8] H. Edelsbrunner and J. Harer, *Computational Topology: An Introduction*, Amer. Math. Soc., 2010.
- [9] B. Giunti, G. Houry, M. Kerber, and M. Söls, Expected complexity of barcode reduction, *J. Appl. Comput. Topol.* **9** (2025), 29.
- [10] D. Morozov, Persistence algorithm takes cubic time in the worst case, BioGeometry News, Dept. of Computer Science, Duke Univ., 2005.
- [11] D. Morozov and P. Skraba, Persistent (co)homology in matrix multiplication time, arXiv:2412.02591, 2024.
- [12] R. Rabadán and A. J. Blumberg, *Topological Data Analysis for Genomics and Evolution*, Cambridge Univ. Press, Cambridge–New York, 2020.
- [13] N. Otter, M. A. Porter, U. Tillmann, P. Grindrod, and H. A. Harrington, A roadmap for the computation of persistent homology, *EPJ Data Sci.* **6** (2017), 17.
- [14] T. S. Shores, *Applied Linear Algebra and Matrix Analysis*, 2nd ed., Springer, 2018.
- [15] A. Zomorodian and G. Carlsson, Computing persistent homology, *Discrete Comput. Geom.* **33** (2005), 249–274.

APPENDIX A. AN ALGORITHM FOR CONSTRUCTING THE STRIP EXAMPLES

An algorithm to construct the filtered simplicial complex $X(n)$ for $n \geq 2$ can be given as follows. The complex $X(1)$ can be constructed directly by inserting the list of simplices from the picture.

Algorithm 4 Building ordered lists of vertices, edges, and triangles of the simplicial complex $X(n)$ for $n \geq 2$

```

1: function BUILDVERTICES( $n$ )
2:   return  $[f_1, \dots, f_n, v_1, \dots, v_{n+2}]$ 
3: end function
4: function BUILDEDGES( $n$ )
5:    $E \leftarrow []$ 
6:   for  $i = n, n-1, \dots, 3$  do
7:     append  $(v_{i-2}, v_{i+2})$  to  $E$ 
8:   end for
9:   append  $(v_1, v_4)$  to  $E$ 
10:  append  $(v_2, v_3)$  to  $E$ 
11:  append  $(v_n, v_{n+2})$  to  $E$ 
12:  for  $i = n, n-1, \dots, 2$  do
13:    append  $(f_i, v_{i-1})$  to  $E$ 
14:  end for
15:  append  $(f_1, v_1)$  to  $E$ 
16:  for  $i = n, n-1, \dots, 1$  do
17:    append  $(f_i, v_{i+1})$  to  $E$ 
18:  end for
19:  for  $i = n, n-1, \dots, 2$  do
20:    append  $(v_{i-1}, v_{i+1})$  to  $E$ 
21:  end for
22:  append  $(v_1, v_2)$  to  $E$ 
23:  return  $E$ 
24: end function
25: function BUILDTRIANGLES( $n$ )
26:    $T \leftarrow []$ 
27:   append  $(v_1, v_2, v_3)$  to  $T$ 
28:   append  $(v_1, v_2, v_4)$  to  $T$ 
29:   for  $i = 3, \dots, n$  do
30:     append  $(v_{i-2}, v_i, v_{i+2})$  to  $T$ 
31:   end for
32:   append  $(f_1, v_1, v_2)$  to  $T$ 
33:   for  $i = 2, \dots, n$  do
34:     append  $(f_i, v_{i-1}, v_{i+1})$  to  $T$ 
35:   end for
36:   return  $T$ 
37: end function

```

$\triangleright$ Horizontal base edges $n', \dots, 1', 0'$
 $\triangleright$ edge i'
 $\triangleright$ edge $2'$
 $\triangleright$ edge $1'$
 $\triangleright$ edge $0'$
 $\triangleright$ Fin edges $-n^*, \dots, -1^*$
 $\triangleright$ edge $-i^*$
 $\triangleright$ edge -1^*
 $\triangleright$ Fin edges $-n, \dots, -1$
 $\triangleright$ Vertical base edges $n, \dots, 1$
 $\triangleright$ edge i
 $\triangleright$ edge 1
 $\triangleright$ Base triangles $\textcircled{1}, \textcircled{2}, \dots, \textcircled{n}$
 $\triangleright$ triangle $\textcircled{1}$
 $\triangleright$ triangle $\textcircled{2}$
 $\triangleright$ triangle $\textcircled{i}$
 $\triangleright$ Fin triangles $\textcircled{-1}, \textcircled{-2}, \dots, \textcircled{-n}$
 $\triangleright$ triangle $\textcircled{-1}$
 $\triangleright$ triangle $\textcircled{-i}$

Algorithm 5 Constructing the filtered simplicial complex $X(n)$ for $n \geq 2$

```

1: function CREATEWORSTCASECOMPLEX( $n$ )
2:    $V \leftarrow \text{BUILDVERTICES}(n)$ 
3:    $E \leftarrow \text{BUILDEDGES}(n)$ 
4:    $T \leftarrow \text{BUILDTRIANGLES}(n)$ 
5:    $\mathcal{K} \leftarrow []$ 
6:    $t \leftarrow 1$ 
7:   for each  $v \in V$  do
8:     append  $(\{v\}, t)$  to  $\mathcal{K}$ 
9:      $t \leftarrow t + 1$ 
10:  end for
11:  for each  $e \in E$  do
12:    append  $(\text{sort}(e), t)$  to  $\mathcal{K}$ 
13:     $t \leftarrow t + 1$ 
14:  end for
15:  for each  $\sigma \in T$  do
16:    append  $(\text{sort}(\sigma), t)$  to  $\mathcal{K}$ 
17:     $t \leftarrow t + 1$ 
18:  end for
19:  return  $\mathcal{K}$ 
20: end function

```

UZAY ÇETİN, DEPARTMENT OF MATHEMATICS, THE OHIO STATE UNIVERSITY, COLUMBUS, OH, 43210-1174, USA

Email address: cetin.16@osu.edu

ERGÜN YALÇIN, DEPARTMENT OF MATHEMATICS, BILKENT UNIVERSITY, 06800 BILKENT, ANKARA, TÜRKİYE

Email address: yalcine@fen.bilkent.edu.tr