Zigzag Persistent Homology in Matrix Multiplication Time

Nikola Milosavljević
Max Planck Institute
Saarbrücken, Germany
nikolam@mpi-inf.mpg.de

Dmitriy Morozov
Stanford University
Stanford, California
dmitriy@mzrv.org

Primož Škraba
INRIA-Saclay
Saclay, France
primoz.skraba@ijs.si

ABSTRACT

We present a new algorithm for computing zigzag persistent homology, an algebraic structure which encodes changes to homology groups of a simplicial complex over a sequence of simplex additions and deletions. Provided that there is an algorithm that multiplies two $n \times n$ matrices in $M(n)$ time, our algorithm runs in $O(M(n) + n^2 \log^2 n)$ time for a sequence of n additions and deletions. In particular, the running time is $O(n^{2.376})$, by result of Coppersmith and Winograd. The fastest previously known algorithm for this problem takes $O(n^3)$ time in the worst case.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems; G.2.1 [**Discrete Mathematics**]: Combinatorics

General Terms

Algorithms, Theory

Keywords

Zigzag persistent homology, matrix multiplication.

1. INTRODUCTION

Motivation. Since its introduction a decade ago, *persistent homology* [1] has become an important tool in such wide-ranging domains as computational biology, geometric processing, machine learning, scientific visualization, and sensor networks. Its success rests on two pillars: a solid theoretical foundation [2], including the celebrated stability result [3], and the availability of fast algorithms [4].

A more recent development is *zigzag persistence* [5], which is a generalization of ordinary persistence built on algebraic insights. Together with an efficient algorithm in the homological setting, zigzag persistence has already resolved open questions in the theoretical study of persistence [6]. In addition to the algebraic generality, it brings an appealing property for algorithm design: one is not constrained to study growing families of spaces as in ordinary persistence, but instead is free to choose whether to grow or shrink the space in question on demand. This flexibility has led to a technique for computing ordinary persistent homology of a real-valued function using space that depends only on the size of the largest levelset, rather than the entire domain [6].

Despite the growing interest in persistence, the complexity of its computation is not well understood. The best currently known algorithms for ordinary and zigzag persistence run in time $\mathrm{O}(M(n))$ and $\mathrm{O}(n^3)$, respectively, in the worst case, where $M(n)$ is the time to multiply two $n \times n$ matrices. In this paper, we present an algorithm that computes zigzag persistent homology (and therefore also ordinary persistent homology) over a finite field in time $\mathrm{O}(M(n) + n^2 \log^2 n)$. It is not known if any of these algorithms are optimal, and we leave this question open.

We also do not consider the problem of computing homology over infinite fields. In this case it cannot be assumed that arithmetic operations take constant time, but it is known that standard homology can still be computed in $\mathrm{O}(M(n))$ time [7].

Related Work. The computation of homology through Gaussian elimination has been known for some time [8, 9]. This algorithm has the worst case running time of $\mathrm{O}(n^3)$. Persistent (and standard) homology is essentially Gaussian elimination with known column order and row pivoting. Therefore, it can be computed in time $\mathrm{O}(M(n))$ (see [10] for details) over finite fields, using the algorithm for PLU factorization of Bunch and Hopcroft [11], with minor modifications. In the case of zigzag persistent homology, *neither row nor column order is known in advance, i.e. both row and column pivoting is required.* Some elimination steps are specified by column, others by row, but never both. Furthermore, the two kinds of elimination steps can be arbitrarily interleaved. Due to these complications, it is not clear how to apply [11] and other similar ideas [12, 13] to this problem.

The first sub-cubic algorithm to appear incrementally computes the Betti numbers of subcomplexes of triangulations of the three sphere $\mathbb{S}^3$ [14]. The running time is $n\alpha(n)$, where n is the number of simplices and $\alpha(\cdot)$ is the inverse Ackermann function. An approach to computing Betti numbers using combinatorial Laplacians appears in [15]. It uses the power method on the Laplacian matrix, hence operating via matrix-vector multiplication. However, the number of multiplications depends on the eigenvalues of the matrix and hence is not easily bounded.

The introduction of persistent homology [1, 2], zigzag persistence [5, 6], and other variants [16, 17] came with algorithms based on the sequential Gaussian elimination and so had a running time of $O(n^3)$. In [18], an example complex is given where the sequential persistence algorithm takes cubic time. However, in practice, experimental results show close to linear behavior, most often attributed to the sparsity of the involved matrices.

In the literature, most work to speed up homology computation has had a different flavor than our approach, with efforts being made to reduce the size of the input complex using combinatorial operations which preserve the homology [19], most often simplicial collapses. In certain cases, notably 2-manifolds, it is possible to create an optimal order of collapses to obtain the homology [20]. In the end of this procedure we end up only with critical cells, allowing us to simply read off the homology. However, it was also shown that in the general case, there may not exist a way to collapse the complex to the minimal number of cells [21]. Furthermore, it was shown to be NP-hard to find an order which will result in the minimum number of cells [22]. More recently, a method [23] was given to reduce the size of the complex in the special case of clique complexes. Such complexes are completely determined by their graphs, and the work tries to extract a minimal representation based on maximal cliques. As with the collapsing techniques, it does not come with any guarantees on the size of the output complex.

In this work we use fast matrix multiplication as a blackbox, but we briefly recount the work which began with the celebrated result of Strassen [24], showing that the matrix inversion can be done with an exponent of $\omega \approx 2.8$ rather than 3. This has been followed by numerous improvements, with the currently best known algorithm of Coppersmith–Winograd [25]. The current best bound for the exponent is $\omega = 2.376$. Finally, we note that this is still an active area of research, where [26] proves that if a group with certain properties exists, so must a quadratic time algorithm for matrix multiplication.

Outline. We review the necessary background in Section 2. We recast the zigzag persistent homology algorithm of [6] in terms of matrix multiplications in Section 3. By batching some of the operations together in Section 4, we obtain the claimed running time.

2. BACKGROUND

In this section, we recount the necessary mathematical and algorithmic background. Since the emphasis of this paper is algorithmic, we begin with a simplicial complex $\mathbb{X}$ rather than the usual topological space. Furthermore, everything will be done with simplicial homology over a field $\mathbb{F}$. While mathematically it is often more convenient to work in singular homology, in practice, we always compute in a combinatorial setting.

We now give a brief overview of homology and zigzag persistence. We refer the reader to Munkres [8] or Hatcher [27] for background on homology, Edelsbrunner and Harer [4] for persistence, and Carlsson and de Silva [5] for zigzag persistence. Our primary object is a simplicial complex, which is a set of simplices such that all faces of a simplex are in the complex and the intersection of two simplices is a (possibly empty) face of both.

We define the *chain group* C_p as an Abelian group on the set of oriented p-simplices in the simplicial complex. A p-chain is a linear combination of simplices with coefficients in a group. We restrict ourselves to the case where the coefficients lie in a field, therefore each C_p is a vector space. We also define the boundary operator $\partial_p : \mathsf{C}_p \to \mathsf{C}_{p-1}$. In a simplicial complex, the boundary operator is defined on a simplex σ as $\partial_p \sigma = \sum_i (-1)^i [v_0, v_1, \ldots, \hat{v}_i, \ldots, v_p]$, where $\hat{v}_i$ is deleted from the sequence. Since the operator is linear, the boundary of a chain is the linear combination of the boundaries of the simplices. The boundary operator connects the chain groups into the *chain complex*: $\ldots \to \mathsf{C}_{p+1} \xrightarrow{\partial_{p+1}} \mathsf{C}_p \xrightarrow{\partial_p} \mathsf{C}_{p-1} \to \ldots$ The boundary operator is a map between vector spaces and is most naturally represented as a matrix where the rows represent $(p-1)$-simplices and the columns represent p-simplices.

To define homology, we require two subgroups: the *cycle group* Z and the *boundary group* B. The cycle group Z_p is the kernel of the ∂_p, which is the null space of the boundary matrix. The boundary group B_p is the image of ∂_{p+1}. The *homology* is the quotient group of the two subspaces: $\mathsf{H}_p = \mathsf{Z}_p/\mathsf{B}_p$. By the property that $\partial_p \partial_{p+1} = 0$, it is not difficult to see that $\mathsf{B}_p \subseteq \mathsf{Z}_p \subseteq \mathsf{C}_p$, meaning the above quotient is well defined. Since these are vector spaces, standard Gaussian elimination can be used to find a basis for each space.

Persistent homology and its recently-introduced generalization, *zigzag persistence*, are two powerful extensions of the classical homology theory. Instead of working with a fixed simplicial complex, both study a parameterized family of complexes and examine the evolution of homology classes in the induced sequences of homology groups. The study is motivated by data that simultaneously contains features at multiple scales — a common behavior in practice — and, therefore, does not justify a single choice of the scale parameter. In contrast, by examining a data set across all scales, we extract information about meaningful parameter values as well as detect the prominent features in the input.

The starting point of zigzag persistent homology is a sequence of spaces connected with maps

$$\mathbb{X}_1 \leftrightarrow \mathbb{X}_2 \leftrightarrow \ldots \leftrightarrow \mathbb{X}_n. \tag{1}$$

The above notation means that the map between spaces can point in either direction. For each space we can form a chain complex; the maps between topological spaces induce maps between the chain complexes, $\mathsf{C}(\mathbb{X}_1) \leftrightarrow \mathsf{C}(\mathbb{X}_2) \leftrightarrow \ldots \leftrightarrow \mathsf{C}(\mathbb{X}_n)$. Finally, by passing to homology, we obtain a sequence of vector spaces connected by homomorphisms,

$$\mathsf{H}(\mathbb{X}_1) \leftrightarrow \mathsf{H}(\mathbb{X}_2) \leftrightarrow \ldots \leftrightarrow \mathsf{H}(\mathbb{X}_n). \tag{2}$$

This sequence is a zigzag module, denoted by $\mathbb{V}$. As shown in [5], $\mathbb{V}$ has a unique decomposition into a direct sum of interval modules, $\mathbb{V} = \bigoplus \mathbb{I}_{[b,d]}$. Each interval module $\mathbb{I}_{[b,d]}$ represents a homology class which exists in all the spaces from $\mathsf{H}(\mathbb{X}_b)$ to $\mathsf{H}(\mathbb{X}_d)$ inclusive. Therefore, we say this class persists from b to d. Note that persistent homology is a special case of zigzag persistence where all the maps point one way.

To compute this decomposition, we require compatible bases in all of the individual homology groups and track how they change as we apply the maps. Fortunately, a single filtration encodes all the required information (see [5] for details).

Definition. *The* right filtration *of a space* $\mathsf{H}(\mathbb{X}_i)$ *is the collection of its subspaces taking the form*

$$R^i = (R_0, R_1, R_2, \ldots, R_i),$$

satisfying inclusion relations $R_k \subseteq R_l$ *for* $k \leq l$. *The filtration is then defined inductively.* $R^0 = (0)$. *If the map points forward:* $\mathsf{H}(\mathbb{X}_i) \xrightarrow{f} \mathsf{H}(\mathbb{X}_{i+1})$, *then the filtration is updated with the images of the map,*

$$R^{i+1} = (f(R_0), f(R_1), \ldots, f(R_i), \mathsf{H}(\mathbb{X}_{i+1})).$$

If the map points in the other direction: $\mathsf{H}(\mathbb{X}_i) \xleftarrow{g} \mathsf{H}(\mathbb{X}_{i+1})$, *then the preimages are added,*

$$R^{i+1} = (0, g^{-1}(R_0), g^{-1}(R_1), \ldots, g^{-1}(R_i)).$$

By keeping track of the changes in the right filtration as we process the zigzag in (1), we obtain the interval decomposition. In the algorithmic setting, without loss of generality, we can assume that in the sequence of simplicial complexes (1) consecutive complexes differ by a single simplex, i.e. either $\mathbb{X}_{i+1} = \mathbb{X}_i \cup \sigma_i$, or $\mathbb{X}_i = \mathbb{X}_{i+1} \cup \sigma_i$.

3. SEQUENTIAL ALGORITHM

In this section we express the algorithm of [6] in terms of matrix update operations.

Representation. We start with a sequence of simplicial complexes (1), where consecutive spaces differ by a single simplex. Enumerating all the simplices $\sigma_1, \ldots, \sigma_k$ that appear in the sequence, we ignore multiplicities: if the same simplex enters, leaves, and then re-enters, we treat it as two simplices in the enumeration. We associate to each simplex σ_i its interval $\mathbb{I}_{[b_i,d_i]}$ in the chain complex zigzag, i.e. $\sigma_i \in \mathbb{X}_j$ iff $j \in [b_i, d_i]$.

As in [6], we maintain the right filtration of the zigzag (2), and represent it after $j-1$ steps via the matrices Z^j, B^j as well as the matrix C^j, which maintains the bounding chains. Denoting the boundary matrix of the simplicial complex with D, we have $0 = DZ^j$, $Z^j B^j = DC^j$. Furthermore, the right filtration of the group $\mathsf{H}(\mathbb{X}_j) = (R^j_0, R^j_1, \ldots)$ is represented via

$$R^j_k = \operatorname{span}\left(\{z + \mathsf{B} \mid z \in \mathsf{Z}^j_k \text{ and } \mathsf{B} = \operatorname{span}(Z^j B^j)\}\right),$$

where Z^j_k denotes the subset of the cycle group spanned by the first k columns of the matrix Z^j. We also implicitly maintain the birth vector associated to the right filtration, which allows us to output the persistence intervals; however, this detail is irrelevant to our argument, and we ignore it from now on.

In addition to the matrices D, Z^j, C^j, B^j of [6], we introduce matrices E^j, F^j. The matrices E^j and F^j keep track of the "past" and "future" simplices, respectively. Their columns correspond to the simplices that have been removed in the case of E^j, or not yet added in the case of F^j. Their boundaries are expressed using the auxiliary matrices $G^j, H^j, K^j, L^j, R^j, S^j$, as defined by Equation (3) — for instance, $Z^j_{p-1} H^j_p$ is the contribution of the existing cycles to the boundaries of the "future" simplices, fully expressed as,

$$D_p F_p = Z_{p-1} H_p + C_{p-1} K_p + F_{p-1} L_p.$$

Similarly, for the boundaries of the "past" simplices,

$$D_p E_p = E_{p-1} G_p + Z_{p-1} R_p + C_{p-1} S_p.$$

We say that the matrix H^j is *independent* of the matrix B^j if any non-zero row in matrix B^j is zero in matrix H^j.

Sequential Invariant. *After step* $j-1$ *of the zigzag, the following conditions must hold:*

1. $$D_p \left[\begin{array}{c|c|c|c} E^j_p & 0 & 0 & 0 \\ \hline 0 & Z^j_p & C^j_p & F^j_p \end{array}\right] =$$
   $$\left[\begin{array}{c|c|c|c} E^j_{p-1} & 0 & 0 & 0 \\ \hline 0 & Z^j_{p-1} & C^j_{p-1} & F^j_{p-1} \end{array}\right] \left[\begin{array}{c|c|c|c} G^j_p & 0 & 0 & 0 \\ \hline R^j_p & 0 & B^j_p & H^j_p \\ \hline S^j_p & 0 & 0 & K^j_p \\ \hline 0 & 0 & 0 & L^j_p \end{array}\right] \tag{3}$$
   For convenience, we abbreviate the matrices in (3) as $D\Phi^j_p = \Phi^j_{p-1}\Gamma^j_p$.
2. *Matrix* B^j_p *has exactly one non-zero element per column, and at most one per row. Matrix* H^j_p *is independent of* B^j_p.
3. *The columns of matrix* F^j_p *are linearly independent. Matrix* Z^j_p *forms a basis for the p-cycles* $\mathsf{Z}_p(\mathbb{X}_j)$. *The boundary* DC^j_p *forms a basis for the* $(p-1)$*-boundaries* $\mathsf{B}_p(\mathbb{X}_j)$.
4. $R^j_k = \operatorname{span}\left(\{z + \mathsf{B} \mid z \in \mathsf{Z}^j_k \text{ and } \mathsf{B} = \operatorname{span}(Z^j B^j)\}\right)$.

Additionally, we choose a particular ordering for the rows and columns of our matrices.

Ordering. *The rows of the matrices* Z^j, C^j, E^j, *and* F^j *correspond to the individual simplices in the complex, and we order them by the* removal *time of those simplices. Consequently, the rows and the columns of the boundary matrix* D *are also ordered by the removal time. The columns of the matrix* F^j *represent future simplices, and we order them by the* addition *time. The columns of the matrix* E^j *represent past (removed) cycles and chains, and we order them by the* removal *time.*

From the invariant and ordering requirements it follows that, initially, L^0_p is the boundary matrix with rows and columns ordered by addition, while F^0_p is a permutation matrix, its columns and rows are indexed by simplices ordered by addition and removal, respectively. The above ordering is insignificant in the rest of this section, however, it becomes critical in Section 4 when we describe how to divide and conquer the necessary computation.

Four cases. In the remainder of the section, we express the update performed to the matrices after each step as matrix multiplication.

$$\left[\begin{array}{c|c|c|c} E^{j+1}_p & 0 & 0 & 0 \\ \hline 0 & Z^{j+1}_p & C^{j+1}_p & F^{j+1}_p \end{array}\right] = \left[\begin{array}{c|c|c|c} E^j_p & 0 & 0 & 0 \\ \hline 0 & Z^j_p & C^j_p & F^j_p \end{array}\right] M^j_p$$

$$\left[\begin{array}{c|c|c|c} G_p^{j+1} & 0 & 0 & 0 \\ \hline R_p^{j+1} & 0 & B_p^{j+1} & H_p^{j+1} \\ \hline S_p^{j+1} & 0 & 0 & K_p^{j+1} \\ \hline 0 & 0 & 0 & L_p^{j+1} \end{array}\right] =$$

$$(M_{p-1}^j)^{-1} \left[\begin{array}{c|c|c|c} G_p^{j} & 0 & 0 & 0 \\ \hline R_p^{j} & 0 & B_p^{j} & H_p^{j} \\ \hline S_p^{j} & 0 & 0 & K_p^{j} \\ \hline 0 & 0 & 0 & L_p^{j} \end{array}\right] M_p^j$$

Step j of the zigzag can be one of the two possibilities: an addition of a simplex, $\mathbb{X}_{j+1} = \mathbb{X}_j \cup \sigma_i$, or a removal of a simplex, $\mathbb{X}_j = \mathbb{X}_{j+1} \cup \sigma_i$. Each of the two cases can have one of two outcomes: a birth or a death of a homology class. Before we describe the four possibilities, we establish an equivalence between the matrices.

Lemma 1 (Forward Step Equivalence Lemma). *If the j-th step of the zigzag is the addition of the simplex σ_i, then $L_p^j[i] = 0$, $K_p^j[i] = 0$, and $DF_p^j[i] = Z_{p-1}^j H_p^j[i]$; in particular, $DF_p^j[i] = 0 \Leftrightarrow H_p^j[i] = 0$.*

Proof. By the first condition of the sequential invariant, $DF_p = Z_{p-1}H_p + C_{p-1}K_p + F_{p-1}L_p$. The last term, $F_{p-1}^j L_p^j[i]$, is zero when we are adding simplex σ_i, since it represents the boundary of σ_i among the "future" simplices. The assumption that every step of our zigzag is a homology group of a simplicial complex implies that the boundary of a simplex cannot occur in the future. From the invariant, the columns of the matrix F_{p-1}^j are linearly independent, therefore, $L_p^j[i] = 0$. Taking the boundary of the remaining equation (and recalling that the boundary matrix squared is 0), we get $0 = DDF_p^j[i] = DZ_{p-1}^j H_p^j[i] + DC_{p-1}^j K_p^j[i] = DC_{p-1}^j K_p^j[i]$, where the last equality follows since the boundary of every cycle in the matrix Z_{p-1}^j is zero, by definition. Therefore, $K_p^j[i] = 0$ since DC_{p-1}^j is a basis by the third condition of the invariant.

We get $DF_p^j[i] = Z_{p-1}^j H_p^j[i]$. Since the columns of matrix Z_p^j are linearly independent, $DF_{p-1}^j[i] = 0$ iff $H_p^j[i] = 0$. □

Birth after addition. This case is characterized by $DF^j[i] = 0$, or, by Lemma 1 equivalently, $H^j[i] = 0$. In particular, $F^j[i]$ is the newly born cycle (indeed, it contains the new simplex σ_i). We append it to the cycle matrix Z^j. The update matrix for this operation is

$$M_p^j = \left[\begin{array}{c|c|c|c|c} I & 0 & 0 & 0 & 0 \\ \hline 0 & I & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & I & 0 \\ \hline 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & I \end{array}\right]. \tag{4}$$

Death after addition. This case is characterized by $DF_p^j[i] \neq 0$, or, equivalently by Lemma 1, $H_p^j[i] \neq 0$. Moreover, $DF_p^j[i] = Z_{p-1}^j H_p^j[i]$ represents the cycle that gets killed by the addition of σ_i. Indeed, since $H_p^j[i]$ is independent of B_p^j and non-zero, it means that $Z_{p-1}^j H_p^j[i]$ is not a boundary before step j. On the other hand, by definition, $DF_p^j[i]$ is a boundary after the addition of simplex σ_i. Let a be the last non-zero entry in the column $H_p^j[i]$. We split the column using this entry into $H_p^j[i] = [\mathbf{h}^T \ a \ \mathbf{0}^T]^T$, where $\mathbf{h}$ is a vector. Suppose a is in the row k. We split the row as follows $H_p^j[k,\cdot] = [a \ \mathbf{g}^T]$, where $\mathbf{g}$ is a vector. $F_p^j[i]$ becomes a bounding chain, and therefore moves to C_p^j. Its boundary replaces the column k of Z_{p-1}^j (which is a change of basis in Z_{p-1}^j). We insert a column with a single 1 in row k into the matrix B_p^j to reflect the death of the cycle $Z_{p-1}^j[k]$. We use this column to zero out row k of H_p^j, thus keeping H_p^j independent of B_p^j. The necessary update operations are

$$M_{p-1}^j = \left[\begin{array}{c|ccc|c|c} I & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & I & \mathbf{h}/a & 0 & 0 & 0 \\ 0 & 0 & 1/a & 0 & 0 & 0 \\ 0 & 0 & 0 & I & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & I & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & I \end{array}\right],$$
$$M_p^j = \left[\begin{array}{c|c|cc|c} I & 0 & 0 & 0 & 0 \\ \hline 0 & I & 0 & 0 & 0 \\ \hline 0 & 0 & I & 0 & 0 \\ \hline 0 & 0 & 0 & 1 & -\mathbf{g}^T \\ 0 & 0 & 0 & 0 & I \end{array}\right]. \tag{5}$$

Remark 1. *A subtle, but important consequence of the matrix H^j being independent of the matrix B^j is that the column $F^j[i]$ arrives "ready": we do not need to reduce it at the beginning of an individual step. This fact is crucial for the speed-up explained in the next section.*

Removing a simplex σ_i corresponds to nullifying the rows i in the matrices Z^j, C^j, and F^j.

Birth after removal. This case is characterized by $Z_p^j[i,\cdot] = 0$, i.e., no cycle contains simplex σ_i. Let a be the rightmost non-zero entry in row i of matrix C_p^j, and suppose it lies in column k, we have $C_p^j[i,\cdot] = [\mathbf{c}^T \ a \ \mathbf{0}^T]$; specifically, $C_p^j[i, 1..k-1] = \mathbf{c}^T$, $C_p^j[i,k] = a$. Let $B_{\mathbf{c}}^j$ be the part of the matrix B_p^j corresponding to the columns $\mathbf{c}^T$ in Equation (3), i.e. $B_{\mathbf{c}}^j = B_p^j[\cdot, 1..k]$. Let $\mathbf{f}^T$ be the i-th row of the matrix F_p^j.

1: The boundary $DC_p^j[k] = Z_{p-1}^j B_p^j[k]$ is the newly born cycle. We prepend it to the matrix Z^j. In the operations below, suppose the 1 in the column $B_p^j[k]$ is in row l.
2: We subtract column $C_p^j[k]/a$ from the other columns in C_p^j and all columns of F_p^j.
3: To keep track of the "past", we move column k from C_p^j to E_p^j (with the corresponding update in Γ_p).
4: The term $-(\mathbf{c}^T/a)(B_{\mathbf{c}}^j)^T$ below undoes the effect of Step 2 on matrix B_p^j.

The operations are

$$M_{p-1}^j = \left[\begin{array}{c|ccc|c|c} I & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & I & 0 & 0 & 0 \\ 0 & 1 & -(\mathbf{c}^T/a)(B_{\mathbf{c}}^j)^T & 0 & 0 & 0 \\ 0 & 0 & 0 & I & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & I & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & I \end{array}\right] \text{(row } l\text{)}$$
$$M_p^j = \left[\begin{array}{cc|c|cc|c} I & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & I & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & I & 0 & 0 \\ 0 & 1 & 0 & -\mathbf{c}^T/a & 0 & -\mathbf{f}^T/a \\ 0 & 0 & 0 & 0 & I & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & I \end{array}\right]. \tag{6}$$

Death after removal. This case is characterized by $Z_p^j[i,\cdot] \neq 0$, i.e. there is a cycle that contains σ_i. Let a be the leftmost non-zero entry in this row, we have $Z_p^j[i,\cdot] = [\mathbf{0}^T\ a\ \mathbf{z}^T]$. Furthermore, let $C_p^j[i,\cdot] = \mathbf{c}^T$ be the i-th row of matrix C_p^j and let $F_p^j[i,\cdot] = \mathbf{f}^T$ be the i-th row of matrix F_p^j. We zero out these rows by adding multiples of the column containing element a to the other columns, while moving that column into the "past" matrix E^j.

Operations

$$M_p^j = \left[\begin{array}{cc|cc|c|c} I & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & I & 0 & 0 & 0 \\ 0 & 1 & 0 & -\mathbf{z}^T/a & -\mathbf{c}^T/a & -\mathbf{f}^T/a \\ 0 & 0 & 0 & I & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & I & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & I \end{array}\right]. \quad (7)$$

In both removal cases, the purpose of zeroing out $\mathbf{f}^T$ is to satisfy statement 1 of the Invariant, specifically, to make sure that the row of the freshly removed simplex σ_i is zero in the future matrix F_p^j.

REMARK 2. *In the addition cases, we only inspect the column of the matrix H_p^j corresponding to the j-th step of the zigzag, both to make a decision about birth or death, and to construct the update matrices. Similarly, in the removal cases we only inspect the rows of the matrices Z_p^j or C_p^j corresponding to the j-th step of the zigzag.*

In the next section, we divide our matrices into smaller blocks, always ensuring that the necessary parts are available to the procedure responsible for the j-th step.

Correctness. The above operations are almost the same as in [6]; in Appendix A, we focus on verifying the new parts of the invariant.

4. HIERARCHICAL ALGORITHM

In this section we take advantage of the Remarks 1 and 2 above, and construct a divide-and-conquer algorithm for processing a zigzag of simplicial complexes. We emphasize that our algorithm is simply an efficient implementation of the algorithm from Section 3 by changing the order in which various matrix updates are applied.

For simplicity, we assume that the number of steps in the zigzag is a power of two. Throughout this section, $M(n)$ denotes time to multiply two $n \times n$ matrices.

Restrictions. The main idea of the hierarchical algorithm is that the elementary updates M_p^j are not applied immediately, but *piecewise*. Each M^j is split into submatrices, which are applied *at different times* to corresponding submatrices of Φ^j and Γ^j. Recall that rows and columns of M^j, Φ^j, Γ^j are associated with the sequence of insertions and removals (cf. Section 3, ORDERING). Not surprisingly, the submatrices of interest to the hierarchical algorithm are associated with contiguous *subsequences* of operations. We refer to these submatrices as *restrictions*, and we define them as follows.

Let $i \in [a, b+1]$ be a step. Restriction of matrix Φ_p^i to steps $[a,b]$, denoted by $\Phi_p^{i;[a,b]}$, is the submatrix of Φ_p^i obtained by removing:

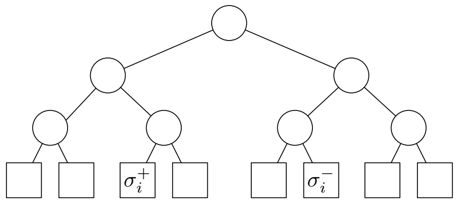
- rows corresponding to p-simplices removed outside steps $[a,b]$,
- columns corresponding to p-simplices added after step b, and
- columns corresponding to p-cycles/chains removed before step a.

In other words (as it follows from ORDERING), to get $\Phi_p^{i;[a,b]}$ from Φ_p^i, we remove a few topmost and bottommost rows, and a few leftmost and rightmost columns.

We define restrictions of other matrices similarly. Restriction of matrix Γ_p^i to steps $[a,b]$, denoted by $\Gamma_p^{i;[a,b]}$, is the submatrix of Γ_p^i obtained by removing rows corresponding to $(p-1)$-simplices added after step b, rows corresponding to $(p-1)$-cycles/chains removed before step a, columns corresponding to p-simplices added after step b, and columns corresponding to p-cycles/chains removed before step a.

Let $i \in [a,b]$ be a step. Restriction of update matrix M_p^i to steps $[a,b]$, denoted by $M_p^{i;[a,b]}$, is the submatrix obtained by removing rows and columns corresponding to p-simplices added after step b, rows and columns corresponding to p-cycles/chains removed before step a.

Operation tree. We can view the hierarchical algorithm below as an in-order traversal of a binary tree. The leaves of the tree correspond to individual steps of the zigzag (simplex additions and removals). Each tree vertex represents an invocation of H-ZZPH.



For the rest of this section we assume, unless noted otherwise, that v is a generic tree node whose leaf descendants correspond to steps $[a,b]$. Also, we will often use a tree node as a shorthand for its set of leaf descendants. For example, if $i \in [a,b]$, then we can write $i \in \mathrm{v}$, $\Phi_p^{i;\mathrm{v}}$, $\Gamma_p^{i;\mathrm{v}}$, $M_p^{i;\mathrm{v}}$, etc.

The main recursive procedure H-ZZPH appears below. Lemma 2 formally specifies what H-ZZPH(v) does. In a nutshell, it solves the problem for a subsequence of steps given by v, with all matrices restricted to v. More precisely, given initial matrices Φ_p^a and Γ_p^a, it computes final matrices Φ_p^{b+1} and Γ_p^{b+1}, and all the intermediate elementary updates M_p^i, $i \in \mathrm{v}$, where all the input and output matrices are restricted to v. Since the root of the tree corresponds to all zigzag steps, top level of recursion returns unrestricted matrices, as desired. H-ZZPH is implemented as follows.

```
H-ZZPH(v)
  if v is leaf then
    Compute $M_p^{a;a}$, $\Phi_p^{a+1;a}$, $\Gamma_p^{a+1;a}$ as in Section 3, but restricting to suitable $1 \times 1$, $1 \times n$, or $n \times 1$ submatrices
  if v is internal then
    H-ZZPH(l)
    LEFT-TO-RIGHT
    APPLY-LEFT
    H-ZZPH(r)
    RIGHT-TO-LEFT
    APPLY-RIGHT
```

Unless v is a leaf, H-ZZPH(v) calls itself recursively on the two children of v. Generically, the left child of v is l, its right child is r, and the number of leaf descendents of l and r (i.e., half the number of leaf descendents of v) is k. We denote by c the last step of l, i.e., $c = a + k - 1$. Procedure Left-to-Right loosens the restriction of the result of the first recursive call; it computes $M_p^{i;\mathrm{v}}$ from $\{M_p^{i;\mathrm{l}} : i \in \mathrm{l}\}$. Procedure Right-to-Left does the symmetric operation for the second recursive call; it computes $M_p^{i;\mathrm{v}}$ from $\{M_p^{i;\mathrm{r}} : i \in \mathrm{r}\}$. Procedure Apply-Left simply applies the result of Left-to-Right to $\Phi_p^{a;\mathrm{v}}$ and $\Gamma_p^{a;\mathrm{v}}$ to compute $\Phi_p^{c+1;\mathrm{v}}$ and $\Gamma_p^{c+1;\mathrm{v}}$; this establishes the the precondition for the second recursive call.

Structure of matrices. Our algorithm exploits the structure of Φ, Γ, and M matrices in order to multiply them quickly. To describe this structure, we introduce a notion of *width.*

Definition. *An $n \times n$ matrix has* width k*, for some $k \le n$, if it can be written as $P + UV^T$, where U, V are $n \times k$ matrices, and P is an $n \times n$ matrix with at most one non-zero entry in any row or column. The triple (P, U, V) is called the* width-k representation *of the matrix.*

In other words, a width-k matrix is a sum of a sparse matrix and a rank-k matrix. The key property is that matrices can be multiplied in time proportional to their width (cf. Lemma 5). We assume that matrices are also stored in their appropriate low-width representation, if they have one.

It is obvious that elementary update matrices M_p^i have width-1 representation $P + uv^T$, where P is a permutation matrix, so that $P^{-1} = P^T$ (see Section 3). Moreover, due to the structure of Γ_p^i, any restriction of Γ_p^i to k steps has width $O(k)$. Finally, any restriction of Φ_p^i to k steps is a $k \times n$ ("skinny") matrix. We use all these facts for efficient computation (cf. Lemma 3, 4, 5).

Notice that width-k representation is not unique. Product of k width-1 matrices is a width-k matrix (cf. Lemma 5). In this case, two particular width-k representations are of special interest; we define them by showing how to compute them. Let $\{(P_i, u_i, v_i) : i = 1, \dots, k\}$ be width-1 representations for k width-1 matrices. The Prefix *and* Suffix *representation* of $\prod_{i=1}^k (P_i + u_i v_i^T)$ are defined by

```
Prefix($P_i, u_i, v_i \mid i = 1, \dots, k$)
  **if** $k = 1$ **then**
    **return** $P_1, u_1, v_1$
  $Q_1, X_1, Y_1 \leftarrow$ Prefix($P_i, u_i, v_i \mid i \le \lfloor k/2 \rfloor$)
  $Q_2, X_2, Y_2 \leftarrow$ Prefix($P_i, u_i, v_i \mid i > \lfloor k/2 \rfloor$)
  **return** $Q_1Q_2, \begin{bmatrix} X_1 & (Q_1 + X_1Y_1^T)X_2 \end{bmatrix}, \begin{bmatrix} Q_2^TY_1 & Y_2 \end{bmatrix}$

Suffix($P_i, u_i, v_i \mid i = 1, \dots, k$)
  **if** $k = 1$ **then**
    **return** $P_1, u_1, v_1$
  $Q_1, X_1, Y_1 \leftarrow$ Suffix($P_i, u_i, v_i \mid i \le \lfloor k/2 \rfloor$)
  $Q_2, X_2, Y_2 \leftarrow$ Suffix($P_i, u_i, v_i \mid i > \lfloor k/2 \rfloor$)
  **return** $Q_1Q_2, \begin{bmatrix} X_1 & Q_1X_2 \end{bmatrix}, \begin{bmatrix} (Q_2 + X_2Y_2^T)^TY_1 & Y_2 \end{bmatrix}$
```

These two forms are significant because from the output of Prefix (resp. Suffix) we can easily compute a low-width representation of any *prefix* $\prod_{i=1}^l (P_i + u_i v_i^T)$ (resp. *suffix* $\prod_{i=l}^k (P_i + u_i v_i^T)$), $1 \le l \le k$, of the product by zeroing out columns and permuting. This property will be useful in implementing algorithms Left-to-Right and Apply-Left. In particular, we exploit it in equations (8) and (9) below.

To see that the output of the two algorithms has the claimed form, consider the following operation. First, zero out the second half of the second matrix in the output of Prefix. Then, undo the permutation corresponding to the second half of the product by right-multiplying the output expression of Prefix by Q_2^T,

$$(Q_1Q_2 + [X_1\ 0][Q_2^TY_1\ Y_2]^T)Q_2^T = Q_1 + X_1Y_1^T .$$

The result is precisely the first half of the product. Symmetrically, in Suffix, zeroing the first half of the second matrix and left-multiplying with Q_1^T yields $Q_2 + X_2Y_2^T$.

Left-to-Right. The goal of this procedure is to extend the restriction of $\{M_p^i : i \in \mathrm{l}\}$, from l to v. Definition of restrictions of M_p^i implies that the only rows/columns in $M_p^{i;\mathrm{v}}$ which are not present in $M_p^{i;\mathrm{l}}$ are those corresponding to p-simplices added in the steps of r. In other words, $M_p^{i;\mathrm{v}}$ is missing fewer bottommost rows and rightmost columns compared to $M_p^{i;\mathrm{l}}$.

For $i \in \mathrm{l}$, let $M_p^{i;\mathrm{l}} = P_i + u_i v_i^T$, hence

$$M_p^{i;\mathrm{v}} = \begin{bmatrix} P_i & 0 \\ 0 & I \end{bmatrix} + \begin{bmatrix} u_i \\ 0 \end{bmatrix} \begin{bmatrix} v_i \\ w_i \end{bmatrix}^T$$

where the unknown w_i expresses how to apply update i to the leaves of r. The Prefix form of $\prod_{i=a}^c M_p^{i;\mathrm{l}}$ is

$$\prod_{i=a}^{c} M_p^{i;\mathrm{l}} = \prod_{i=a}^{c} P_i + \begin{bmatrix} u'_a & \cdots & u'_c \end{bmatrix} \begin{bmatrix} v'_a & \cdots & v'_c \end{bmatrix}^T .$$

The Prefix form of $\prod_{i=a}^c M_p^{i;\mathrm{v}}$ is

$$\prod_{i=a}^{c} M_p^{i;\mathrm{v}} = \begin{bmatrix} \prod_{i=a}^c P_i & 0 \\ 0 & I \end{bmatrix} + \begin{bmatrix} u'_a & \cdots & u'_c \\ 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} v'_a & \cdots & v'_c \\ w_a & \cdots & w_c \end{bmatrix}^T .$$

Note that using Prefix is crucial for $w_a, \dots, w_c$ to stay unchanged. We will need prefixes $\prod_{j=a}^i M_p^{i;\mathrm{v}}$ of this product, and we can access them by zeroing out and permuting as claimed above.

$$\prod_{j=a}^{i} M_p^{j;\mathrm{v}} = \begin{bmatrix} \prod_{j=a}^i P_j & 0 \\ 0 & I \end{bmatrix} + \begin{bmatrix} u'_a & \cdots & u'_i & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} v'_a & \cdots & v'_c \\ w_a & \cdots & w_c \end{bmatrix}^T \begin{bmatrix} (\prod_{j=i+1}^c P_j)^T & 0 \\ 0 & I \end{bmatrix} \tag{8}$$

The algorithm proceeds in the order of increasing dimension p. Therefore, we can assume that $M_{p-1}^{i;\mathrm{v}} = Q_i + x_i y_i^T$ has already been computed for all $i \in \mathrm{l}$. In the base case $p = 0$, we have $M_{p-1}^{i;\mathrm{v}} = I$, which is already in the required form. The Suffix form of $(\prod_{i=a}^c M_{p-1}^{i;\mathrm{v}})^{-1}$ is

$$(\prod_{i=a}^{c} M_{p-1}^{i;\mathrm{v}})^{-1} = (\prod_{i=a}^{c} Q_i)^T + \begin{bmatrix} x'_a & \cdots & x'_c \end{bmatrix} \begin{bmatrix} y'_a & \cdots & y'_c \end{bmatrix}^T .$$

We will need suffixes $(\prod_{j=a}^i M^{j;\mathrm{v}})^{-1}$ of this product, and we use the same zero-and-permute strategy to access them.

$$(\prod_{j=a}^{i} M^{j;\mathrm{v}})^{-1} = (\prod_{j=a}^{i} Q_j)^T + \\ (\prod_{j=i+1}^{c} Q_j) \begin{bmatrix} x'_a \cdots x'_i & 0 \cdots 0 \end{bmatrix} \begin{bmatrix} y'_a \cdots y'_c \end{bmatrix}^T . \tag{9}$$

To find the unknown w_i, we express each one as a linear function of w_j, $j < i$: we compute column vectors α_i, β_i such that $w_i = \begin{bmatrix} w_a & \cdots & w_c \end{bmatrix} \alpha_i + \beta_i$, and only the leading $i - a$ entries of α_i can be non-zero. We then find unknown vectors $w_a, \ldots, w_c$ by solving the linear system. It remains to explain how to compute α_i and β_i.

First of all, note that $w_i = 0$ in case 1 (birth after addition), as can be verified from (4). Let's examine the three remaining possibilities. Suppose step i is removal of a p-simplex (cases 3 and 4). The removed simplex corresponds to a row of Φ^i_p. Let s_i be the indicator vector of that row. Then, by the description of the sequential algorithm in Section 3, to get w_i^T we need to apply to $\Phi_p^{a;\mathrm{v}}$ updates $a, \ldots, i-1$, select the row indicated by s_i and columns of r, and negate the resulting submatrix. Formally (after some calculation which we skip), we get

$$w_i^T = -s_i^T \Phi_p^{a;\mathrm{v}} \left(\begin{bmatrix} 0 \\ I \end{bmatrix} + \begin{bmatrix} u'_a \cdots u'_{i-1} & 0 \cdots 0 \\ 0 \cdots \ 0 & 0 \cdots 0 \end{bmatrix} \begin{bmatrix} w_a \cdots w_c \end{bmatrix}^T \right) ,$$

which implies

$$\alpha_i^T = -s_i^T \Phi_p^{a;\mathrm{v}} \begin{bmatrix} 0 \\ I \end{bmatrix}$$

$$\beta_i^T = -s_i^T \Phi_p^{a;\mathrm{v}} \begin{bmatrix} u'_a & \cdots & u'_{i-1} & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & \cdots & 0 \end{bmatrix} .$$

Now suppose that step i is addition of a p-simplex that kills a cycle (case 2). The killed cycle corresponds to a row in Γ^i_p. Let s_i be the indicator vector of that row. Again, using the description of the sequential algorithm from Section 3 and skipping some calculations,

$$w_i^T = -s_i^T \Bigg((\prod_{j=a}^{i-1} Q_j)^T + \\ (\prod_{j=i}^{c} Q_j) \begin{bmatrix} x'_a & \cdots & x'_{i-1} & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} y'_a & \cdots & y'_c \end{bmatrix}^T \Bigg) \\ \times \quad \Gamma_p^{a;\mathrm{v}} \left(\begin{bmatrix} 0 \\ I \end{bmatrix} + \begin{bmatrix} u'_a & \cdots & u'_{i-1} & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} w_a & \cdots & w_c \end{bmatrix}^T \right)$$

Letting

$$s'_i = (\prod_{j=a}^{i-1} Q_j) s_i ,$$

$$s''_i = \begin{bmatrix} x'_a & \cdots & x'_{i-1} & 0 & \cdots & 0 \end{bmatrix}^T (\prod_{j=i}^{c} Q_j)^T s_i ,$$

$$s'''_i = s'_i + \begin{bmatrix} y'_a & \cdots & y'_c \end{bmatrix} s''_i ,$$

we get

$$\alpha_i^T = -(s'''_i)^T \Gamma_p^{a;\mathrm{v}} \begin{bmatrix} 0 \\ I \end{bmatrix} ,$$

$$\beta_i^T = -(s'''_i)^T \Gamma_p^{a;\mathrm{v}} \begin{bmatrix} u'_a & \cdots & u'_{i-1} & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & \cdots & 0 \end{bmatrix} .$$

Appendix C gives pseudo-code for one possible efficient implementation of the above computations. Note that LEFT-TO-RIGHT also requires $\Phi_p^{a,\mathrm{v}}$, $\Gamma_p^{a,\mathrm{v}}$, $\{\Phi_p^{i;i}, \Gamma_p^{i;i} : i \in \mathrm{l}\}$, in addition to already mentioned $\{M_p^{i;\mathrm{l}} : i \in \mathrm{l}\}$. In the proof of correctness (Lemma 2) we will show that all this data is indeed available when LEFT-TO-RIGHT is invoked.

Batched application of updates. Next, we APPLY-LEFT updates $M^{i;\mathrm{v}}$ to matrices $\Phi^{a;\mathrm{v}}$ and $\Gamma^{a;\mathrm{v}}$ for all $i = a, \ldots, c$. The algorithm uses PREFIX to compute $(\prod_{i=a}^{c} M_{p-1}^{i;\mathrm{v}})^{-1}$ and $\prod_{i=a}^{c} M_p^{i;\mathrm{v}}$ as width-k matrices (SUFFIX works just as well). Then it multiplies $\Phi_p^{a;\mathrm{v}}$ and $\Gamma_p^{a;\mathrm{v}}$ appropriately, resulting in $\Phi_p^{c+1;\mathrm{v}}$, $\Gamma_p^{c+1;\mathrm{v}}$.

```
APPLY-LEFT
  for $p = 0, 1, \ldots,$ max. dimension do
    for $i \in \mathrm{l}$ do
      $P_i, u_i, v_i \leftarrow$ width-1 form of $M_p^{i;\mathrm{v}}$
      $Q_i, x_i, y_i \leftarrow$ width-1 form of $M_{p-1}^{i;\mathrm{v}}$
    $P, U', V' \leftarrow$ PREFIX$(P_i, u_i, v_i \mid i = a, \ldots, c)$
    $Q, X', Y' \leftarrow$ PREFIX$(Q_i^T, \frac{Q_i^T x_i}{y_i^T Q_i^T x_i - 1}, Q_i y_i \mid i = c, \ldots, a)$
    $\Phi_p^{c+1;\mathrm{v}} \leftarrow \Phi_p^{a;\mathrm{v}}(P + U'(V')^T)$
    $\Gamma_p^{c+1;\mathrm{v}} \leftarrow \Gamma_p^{a;\mathrm{v}}(P + U'(V')^T)$
    $\Gamma_p^{c+1;\mathrm{v}} \leftarrow (Q + X'(Y')^T)\Gamma_p^{c+1;\mathrm{v}}$
```

Right-to-Left. RIGHT-TO-LEFT extends the restriction of $\{M_p^i : i \in \mathrm{r}\}$ from r to v. Definition of restrictions of M_p^i implies that the only rows/columns in $M_p^{i;\mathrm{v}}$ which are not present in $M_p^{i;\mathrm{r}}$ are those corresponding to p-cycles and p-chains removed in steps of l. In other words, $M_p^{i;\mathrm{v}}$ is missing fewer topmost rows and leftmost columns compared to $M_p^{i;\mathrm{r}}$. But updates $\{M_p^i : i \in \mathrm{r}\}$ have no effect on p-cycles and p-chains removed in steps of l; this can be verified by inspecting update matrices in Section 3. Hence, we simply pad each $\{M^{i;\mathrm{r}} : i \in \mathrm{r}\}$, with an identity matrix of suitable size.

```
RIGHT-TO-LEFT
  for $i \in \mathrm{r}$ and all $p$ do
    $M_p^{i;\mathrm{v}} \leftarrow \begin{bmatrix} I & 0 \\ 0 & M_p^{i;\mathrm{r}} \end{bmatrix}$
```

APPLY-RIGHT is implemented like APPLY-LEFT, except $\{M_p^{i;\mathrm{v}} : i \in \mathrm{l}\}$ are replaced by $\{M_p^{i;\mathrm{v}} : i \in \mathrm{r}\}$, and $\Phi_p^{a,\mathrm{v}}$, $\Gamma_p^{a,\mathrm{v}}$ are replaced by $\Phi_p^{c+1;\mathrm{v}}$, $\Gamma_p^{c+1;\mathrm{v}}$.

Correctness. We prove correctness of the hierarchical algorithm by applying the following more general statement to the root of the operation tree.

LEMMA 2. *$\Phi^{a;\mathrm{v}}$, $\Gamma^{a;\mathrm{v}}$ are correct when* H-ZZPH(v) *is invoked and thereafter.* $\{M^{i;\mathrm{v}}, i \in \mathrm{v}\}$, $\Phi^{b+1;\mathrm{v}}$, $\Gamma^{b+1;\mathrm{v}}$ *are correct when* H-ZZPH(v) *terminates and thereafter.*

Proof. We prove the statement by induction on the order of events (invocations and terminations of H-ZZPH).

The first event is invocation where v is the root. In that case, $\Phi_p^{a;\mathrm{v}}$ and $\Gamma_p^{a;\mathrm{v}}$ are simply initial unrestricted matrices Φ^1_p and Γ^1_p, which are correct initially, because they are initialized in the same way as in the sequential algorithm.

Now consider some other event — an invocation or a termination at a node u — assuming that the claim is true for all the preceding events.

Suppose that the event is an invocation of H-ZZPH(u). Noting that u is not a root, let v be the *parent* of u. If $u = \mathrm{l}$, since the restrictions of the matrices to a child are a subset of the restrictions to a parent, we are done by the inductive hypothesis (i.h.) on the invocation of H-ZZPH(v). Otherwise, by i.h., when H-ZZPH(l) terminates, $\{M^{i;\mathrm{l}} : i \in \mathrm{l}\}$, $\Phi^{c+1;\mathrm{l}}$, $\Gamma^{c+1;\mathrm{l}}$ are correct. By i.h. for the invocation on each leaf $i \in \mathrm{l}$, $\{\Phi_p^{i;i}, \Gamma_p^{i;i} : i \in \mathrm{l}\}$ are correct. Hence the vectors s_i in Left-to-Right can be correctly read off from $\Phi_p^{i;i}$ and $\Gamma_p^{i;i}$. Since all input data for Left-to-Right is correct, $\{M^{i;\mathrm{v}} : i \in \mathrm{l}\}$ are correct when Left-to-Right terminates. By design of Apply-Left, at its termination, $\Phi^{c+1,\mathrm{v}}$, $\Gamma^{c+1;\mathrm{v}}$, and, therefore, $\Phi^{c+1,\mathrm{r}}$, $\Gamma^{c+1;\mathrm{r}}$ are correct, which completes this part of the proof.

Now suppose that the event is termination of H-ZZPH(v). If v is a leaf, then at termination, $M^{a;\mathrm{v}}$, $\Phi^{a+1;\mathrm{v}}$, $\Gamma^{a+1;\mathrm{v}}$ are correct because they are computed using the same rules as in the sequential algorithm, and starting from correct input ($\Phi^{a;\mathrm{v}}$, $\Gamma^{a;\mathrm{v}}$). Now suppose v is internal. As before, combining i.h. on termination of H-ZZPH(l), i.h. on the invocation on the leaf descendents of l, and design of Left-to-Right and Apply-Left, we prove that when Apply-Left terminates, $\{M^{i;\mathrm{v}} : i \in \mathrm{l}\}$, $\Phi^{c+1;\mathrm{v}}$, $\Gamma^{c+1;\mathrm{v}}$ are correct. By i.h., when H-ZZPH(r) terminates, $\{M^{i;\mathrm{r}} : i \in \mathrm{r}\}$, $\Phi^{b+1;\mathrm{r}}$, $\Gamma^{b+1;\mathrm{r}}$ are correct. When H-ZZPH(v) terminates, $\{M^{i;\mathrm{v}} : i \in \mathrm{r}\}$ are correct by design of Right-to-Left, and $\Phi^{b+1;\mathrm{v}}$, $\Gamma^{b+1;\mathrm{v}}$ are correct by design of Apply-Right. □

Running time. We show that the hierarchical algorithm runs on an n-step zigzag in time $O(M(n) + n^2 \log^2 n)$. The key step is showing that H-ZZPH(v), excluding its recursive calls, can be implemented to run in time proportional to the number of leaf descendents of v. The missing proofs of lemmas below appear in Appendix B.

Theorem 1. H-ZZPH(v) *runs in* $O(\frac{n}{k}M(k) + \frac{n}{k}k^2 \log^2 k)$ *time.*

From this we easily get the main result of the paper.

Corollary 1. *Zigzag persistent homology for a sequence of n steps can be computed in time $O(M(n) + n^2 \log^2 n)$.*

Before proving Theorem 1, we provide tools for efficient computation that will be invoked repeatedly.

Lemma 3. *Let U, V be $n \times k$ matrices, $k \le n$, and let W be a $k \times k$ matrix. One can compute $U^T V$ and UW in $O(\frac{n}{k}M(k))$ time.*

Lemma 4. *Let B be an $n \times k$ matrix. Given a width-k representation of A, AB can be computed in $O(\frac{n}{k}M(k))$ time.*

Lemma 5. *If A has width k_1 and B has width k_2, then AB has width $k_1 + k_2$. Given low-width representations A and B, low-width representation of AB can be computed in $O(\frac{n}{k_1+k_2}M(k_1+k_2))$ time.*

Proof of Theorem 1. First assume that $M(n) \notin O(n^2)$.

Let $T_{\text{Prefix}}(t)$ be the worst-case running time of Prefix on an input consisting of t width-1 terms. Then $T_{\text{Prefix}}(t) \le 2T_{\text{Prefix}}(\frac{t}{2}) + O(\frac{n}{t}M(t))$, where the last term is determined by the time it takes to compute $Y_1^T X_2$ by Lemma 3. Solving this recursion, we get $T_{\text{Prefix}}(t) = O(\frac{n}{t}M(t))$. By similar analysis we can bound the running time of Suffix on t terms by $O(\frac{n}{t}M(t))$.

Apply-Left and Apply-Right run in time $2T_{\text{Prefix}}(k) + O(\frac{n}{k}M(k)) + O(\frac{n}{2k}M(2k)) + O(\frac{n}{3k}M(3k))$, where the terms correspond to two calls to Prefix, computing $\Phi_p^{a;\mathrm{v}}(P + U'(V')^T)$ by Lemma 4, computing $\Gamma_p^{a;\mathrm{v}}(P + U'(V')^T)$ by Lemma 5, and computing $(Q + X'(Y')^T)\Gamma_p^{c+1;\mathrm{v}}$ by Lemma 5.

We analyze the running time of Left-to-Right, as implemented in Appendix C. Let k_p be the number of nontrivial updates for dimension p. Clearly, $\sum_p k_p = O(k)$. We fix dimension p, and analyze its iteration of the **for** loop. Calls to Prefix and Suffix take $O(\frac{n}{k}M(k))$ time. Computing s_i' and s_i'', $i \in \mathrm{l}$ takes $O(nk)$ time, using any obvious implementation. Computing $Y' \begin{bmatrix} s_a'' & \cdots & s_c'' \end{bmatrix}$ takes $O(\frac{n}{k}M(k))$ time by Lemma 3. Computing $\begin{bmatrix} s_a''' & \cdots & s_c''' \end{bmatrix}$ takes $O(nk)$ time. Since $\Phi_p^{a;\mathrm{v}}$ has size at most $k \times n$, computing $\Phi_p^{a;\mathrm{v}}U'$ takes $O(\frac{n}{k}M(k))$ by Lemma 3. Since $\Gamma_p^{a;\mathrm{v}}$ has width k, computing $\Gamma_p^{a;\mathrm{v}}U'$ takes $O(\frac{n}{k}M(k))$ time by Lemma 4. Multiplying the result by $\begin{bmatrix} s_1''' & \cdots & s_k''' \end{bmatrix}^T$ takes $O(\frac{n}{k}M(k))$ time, by Lemma 3. Coefficient vectors α_i^T, β_i^T are computed in $O(k^2)$ time, because they appear as submatrices of already computed matrices T_1 and T_2. The $k \times k$ linear system can be solved in $O(M(k))$ time using Bunch and Hopcroft's algorithm [11]. We conclude that the **for** loop for dimension p runs in $O(\frac{n}{k}M(k))$ time. It can actually be implemented to run in $O(\frac{n}{k_p}M(k_p))$, where k_p is the number of updates that affect dimension p. Since this is superlinear in k_p, and $\sum_p k_p = O(k)$, we conclude that Left-to-Right runs in $O(\frac{n}{k}M(k))$ time.

Right-to-Left modifies k matrices by changing adding k nonzero entries to each. Clearly, it can be implemented to run in optimal $O(k^2)$.

We define $T_{\text{H-ZZPH}}(k)$ to be the worst-case running time of H-ZZPH(v) for any tree node v with k leaf descendents. It is easy to see that $T(1) = O(n)$, because if v is a leaf, the algorithm manipulates $O(1)$ rows and columns, by inspecting, moving or copying them to compute $M_p^{a;a}$, $\Phi_p^{a+1;a}$, $\Gamma_p^{a+1;a}$. Furthermore, from the above analysis it follows that $T_{\text{H-ZZPH}}(2k) \le 2T_{\text{H-ZZPH}}(k) + O(\frac{n}{k}M(k))$. The claim follows by solving the recurrence relation.

If $M(n) \in O(n^2)$ the analysis is similar. Solving each recurrent relation (the one for $T_{\text{Prefix}}/T_{\text{Suffix}}$, and the one for $T_{\text{H-ZZPH}}$) contributes one logarithmic factor. □

Acknowledgement

The authors would like to thank Leonidas Guibas, Gunnar Carlsson and Frédéric Chazal for their support. This work has been supported in part by NSF grants CCF 0634803, FODAVA 0808515, CCF 1011228, DARPA grant HR0011-05-1-0007 and ANR-09-BLAN-0331-01.

5. REFERENCES

[1] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. *Discrete Comput. Geom.*, 28:511–533, 2002.

[2] A. Zomorodian and G. Carlsson. Computing Persistent Homology. *Discrete Comput. Geom.*, 33(2):249–274, 2005.

[3] D. Cohen-Steiner, H. Edelsbrunner, and J. Harer. Stability of persistence diagrams. *Discrete and Computational Geometry*, 37(1):103–120, 2007.

[4] H. Edelsbrunner and J. Harer. Persistent homology — a survey. *Surveys on Discrete and Computational Geometry. Twenty Years*, pages 257–282, 2008.

[5] G. Carlsson and V. de Silva. Zigzag persistence. *Foundations of Computational Mathematics*, 10(4):367–405, August 2010.

[6] G. Carlsson, V. de Silva, and D. Morozov. Zigzag persistent homology and real-valued functions. In *Proceedings of the Annual Symposium on Computational Geometry*, pages 247–256, 2009.

[7] W. Eberly, M. Giesbrecht, and G. Villard. On computing the determinant and smith form of an integer matrix. In *Proceedings of Symposium on the Foundations of Computer Science*, pages 675–685, 2000.

[8] J.R. Munkres. *Elements of Algebraic Topology.* Westview Press, 1984.

[9] T. Kaczynski, K. Mischaikow, and M. Mrozek. *Computational Homology.* Springer, 2004.

[10] N. Milosavljevic, D. Morozov, and P. Skraba. Zigzag Persistent Homology in Matrix Multiplication Time. Research Report RR-7393, INRIA, 2010.

[11] J. R. Bunch and J.E. Hopcroft. Triangular Factorization and Inversion by Fast Matrix Multiplication. *Mathematics of Computation*, 28+(125):231–236, 1974.

[12] M. Mucha and P. Sankowski. Maximum matchings via gaussian elimination. In *Foundations of Computer Science, 2004. Proceedings. 45th Annual IEEE Symposium on*, pages 248 – 255, Oct. 2004.

[13] J. A. Harvey N. Algebraic structures and algorithms for matching and matroid problems. In *Foundations of Computer Science, 2006. FOCS '06. 47th Annual IEEE Symposium on*, pages 531–542, Oct 2006.

[14] C. J. A. Delfinado and H. Edelsbrunner. An incremental algorithm for Betti numbers of simplicial complexes. In *SCG '93: Proceedings of the ninth annual symposium on Computational geometry*, pages 232–239, New York, NY, USA, 1993. ACM.

[15] J. Friedman. Computing Betti numbers via combinatorial Laplacians. In *STOC '96: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 386–391, New York, NY, USA, 1996. ACM.

[16] D. Cohen-Steiner, H. Edelsbrunner, and J. Harer. Extending Persistence Using Poincaré and Lefschetz Duality. *Found. Comput. Math.*, 9(1):79–103, 2009.

[17] D. Cohen-Steiner, H. Edelsbrunner, J. Harer, and D. Morozov. Persistent homology for kernels, images, and cokernels. In *SODA '09: Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1011–1020, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics.

[18] D. Morozov. *Homological Illusions of Persistence and Stability.* PhD thesis, Duke University, August 2008.

[19] M. Mrozek, P. Pilarczyk, and N. Zelazna. Homology algorithm based on acyclic subspace. *Comput. Math. Appl.*, 55(11):2395–2412, 2008.

[20] T. Lewiner, H. Lopes, and G. Tavares. Optimal discrete Morse functions for 2-manifolds. *Comput. Geom. Theory Appl.*, 26(3):221–233, 2003.

[21] Ö. Eğecioğlu and T.F. Gonzalez. A computationally intractable problem on simplicial complexes. *Comput. Geom. Theory Appl.*, 6(2):85–98, 1996.

[22] M. Joswig and M.E. Pfetsch. Computing Optimal Morse Matchings. *SIAM J. Discret. Math.*, 20(1):11–25, 2006.

[23] A. Zomorodian. The Tidy Set: A Minimal Simplicial Set for Computing Homology of Clique Complexes. In *Proc. ACM Symposium of Computational Geometry*, pages 257–266, 2010.

[24] V. Strassen. Gaussian Elimination is not Optimal. *Numer. Math.*, 13:354–356, 1969.

[25] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.*, 9(3):251–280, 1990.

[26] H. Cohn, R. Kleinberg, B. Szegedy, and C. Umans. Group-theoretic Algorithms for Matrix Multiplication. In *FOCS '05: Proceedings of the Symposium on Foundations of Computer Science*, pages 379–388, Washington, DC, USA, 2005. IEEE Computer Society.

[27] A. Hatcher. *Algebraic Topology.* Cambridge Univ. Press, 2001.

APPENDIX

A. CORRECTNESS OF THE SEQUENTIAL ALGORITHM

Since the operations in Section 3 are almost the same as in [6], we focus our proof of correctness on verifying the new parts of the invariant.

1: $D\Phi_p^j = \Phi_{p-1}^j \Gamma_p^j$. The equality is satisfied in Equation (3) because we apply the same updates to both sides of the equation. However, we must still ensure that the specific forms prescribed to matrices Φ_p^j and Γ_p^j remain intact: namely, that the blocks marked zero in (3) remain zero.

If there is a *birth after an addition*, the product $\Phi_p^j M_p^j$ moves the newly born cycle from submatrix F_p^j into Z_p^{j+1}; its boundary is zero by the case assumption. $\Gamma_p^j M_p^j$ moves the zero column (zero by Lemma 1) into the portion of Γ_p^{j+1} corresponding to the cycle matrix Z_p^j. Pre-multiplying Γ_{p+1} by $(M_p^j)^{-1}$ performs the corresponding row update, safely moving a row from the fourth to the second section of Γ_{p+1}. If there is a *death after an addition*, M_{p-1}^j performs a basis change in Z_{p-1}^j (with a matching update in B_p^j). It does not affect the structure of the matrices, only ensures that the leftmost column of B_p^{j+1} has a single 1. M_p^j does two things. First it moves the bounding chain from F_p^j into C_p^{j+1}, and the corresponding boundary from H_p^j to B_p^{j+1}. Then it subtracts a multiple of the new column in B_p^{j+1} from H_p^j; these subtractions only introduce zeros. The effect of $(M_p^j)^{-1}$ on Γ_{p+1}^j is adding rows of L_{p+1}^j to K_{p+1}^j, a benign update as far as zeros are concerned.

If a simplex is *removed*, multiplying by matrix M_p^j zeroes out the outgoing row of matrix C_p^j (and also Z_p^j in case of a death), as well as F_p^j. As a result, once the row is moved to the past, all but its first portion (now belonging to matrix E_p^{j+1}) is zero. The effect of the matrix $(M_{p-1}^j)^{-1}$ (from Equation (6)) in the product $(M_{p-1}^j)^{-1}\Gamma_p^j$ does not cross the partition boundaries.

2: *B_p^j has exactly one non-zero element per column, and at* *most one per row. H_p^j is independent of B_p^j.* In the removal cases, the only change to the matrix H_p^j is that it loses a row, and so does B_p^j; this loss does not affect the invariant. It's important to note that in case of the change of basis in Z_p^j after a death, B_p^j cannot stop satisfying the invariant, since the columns of Z_p^j that have a matching 1 in B_p^j cannot contain the outgoing simplex (the 1 would indicate that the cycles were boundaries, but the outgoing simplex has no cofaces). In case of a birth after removal, M_{p-1}^j performs a change of basis in Z_{p-1}^{j+1} such that its inverse updates B_p^{j+1} to once again satisfy the invariant.

In the case of addition, if birth occurs, matrix H_p^j loses a zero column, which again requires no update. If there is a death, B_p^j acquires a single column. Pre-multiplication of Γ_p^j by $(M_{p-1}^j)^{-1}$ ensures that B_p^{j+1} has a single 1 in that column, which is in the row of the lowest non-zero entry in the outgoing column of H_p^j; therefore, B_p^{j+1} remains a permutation matrix. We explicitly zero this row out via matrix M_p^j. Therefore, in all cases statement 2 of the Invariant is maintained.

3, 4: Matrix M_p^j perform performs additions from left to right in F_p^j, and possibly removes a column from it (in the case of an addition of a simplex). Therefore, the columns of F_p^{j+1} remain linearly independent.

In the case of a removal of a simplex, the main updates performed to matrices Z and C are the same as in [6], with the exception of a few additional changes of basis in Z via left to right operations (which respect the right filtration). Therefore, the last two parts of the invariant are maintained.

In the case of an addition, the only difference from [6] is that the first column of the matrix F_p^j already has the correct form, and does not require any additional processing.

In the birth case, this is easy to see: the column $F_p^0[i]$ corresponding to the simplex σ_i contains a 1 in the row corresponding to this simplex. None of the operations ever remove this element from this column. Therefore, when we add σ_i we have a new cycle (recorded as the column $F_p^j[i]$), which we append to the matrix Z_p^j.

In the death case, the boundary of the column $F_p^j[i]$ is in the kernel of the map on the homology. If the lowest element in the matching column $H_p^j[i]$ is in the row l, then this cycle is in the span of the first l elements of the right filtration given by the matrix Z_{p-1}^j. Furthermore, since the matrix H_p^j is reduced with respect to B_p^j, the row l in B_p^j is zero, by definition. Therefore, there does not exist a cycle in the span of the first $l-1$ elements of the right filtration that's in the kernel of the homology map. Therefore, our update is correct — it adds to the matrix B_p^j a column with a 1 in row l, indicating that the updated cycle Z_{p-1}^{j+1} dies after the addition of simplex σ_i. (This analysis reflects the Reduction Lemma in [6].) In summary, the operations in Section 3 closely mimic the original zigzag persistent homology algorithm [6], and satisfy the additional Sequential Invariant requirements.

B. PROOFS OF RUNNING TIME LEMMAS

Proof of Corollary 1. This is a special case for $k = n$ (i.e. root of the tree), with Z_p, C_p, B_p, H_p and K_p "empty" (Z_p, C_p, B_p with zero columns, H_p, K_p with zero rows), with F_p equal to a permutation matrix that maps addition order to removal order, and $L_p = F_{p-1}^T D_p F_p$. □

Proof sketch of Lemma 3. Split matrices into $k \times k$ blocks, and treat blocks as entries. Use a fast $k \times k$ matrix multiplication algorithm to multiply blocks. □

Proof sketch of Lemma 5. Let P, U, V, and Q, X, Y be low-width representations of A and B, respectively. It is easy to check that

$$PQ\,, \qquad \begin{bmatrix} U & PX + U(V^T X) \end{bmatrix}\,, \qquad \begin{bmatrix} Q^T V & Y \end{bmatrix}\,.$$

is a width-$(k_1 + k_2)$ representation of AB. The running time follows by applying Lemma 3 to the above expressions. □

Proof sketch of Lemma 4. Let P, U, V be the width-k representation of A. Then one can compute AB as $PB + U(V^T B)$, using Lemma 3. □

C. LEFT-TO-RIGHT IMPLEMENTATION

The following is one possible implementation of the Left-to-Right computation, described in Section 4, expressed in pseudo-code.

```
Left-to-Right(v)
  for $i \in \mathrm{l}$ do
    $Q_i \leftarrow I\,, \quad x_i \leftarrow 0\,, \quad y_i \leftarrow 0$
  for $p = 0, 1, \ldots,$ maximum dimension do
    for $i = a, \ldots, c$ do
      $P_i, u_i, v_i \leftarrow$ width-1 form of $M_p^{i;\mathrm{l}}$
      $s_i \leftarrow$ pivot row of $M_p^{i;\mathrm{l}}$ (if any)
    $P, U', V' \leftarrow \textsc{Prefix}(P_i, u_i, v_i \mid i = a, \ldots, c)$
    $Q, X', Y' \leftarrow \textsc{Suffix}(Q_i^T, \frac{Q_i^T x_i}{y_i^T Q_i^T x_i - 1}, Q_i y_i \mid i = c, \ldots, a)$
    $T_1 \leftarrow I\,, \quad T_2 \leftarrow Q$
    for $i = a, \ldots, c$ do
      $s'_i \leftarrow T_1 s_i$
      $s''_i \leftarrow \begin{bmatrix} e_1 \cdots e_{i-a} & 0 \cdots 0 \end{bmatrix}^T (X')^T (T_2 s_i)$
      $T_1 \leftarrow T_1 Q_i\,, \quad T_2 \leftarrow T_2 Q_i$
    $T_1 \leftarrow Y' \begin{bmatrix} s''_a & \cdots & s''_c \end{bmatrix}$
    $\begin{bmatrix} s'''_a & \cdots & s'''_c \end{bmatrix} \leftarrow \begin{bmatrix} s'_a & \cdots & s'_c \end{bmatrix} + T_1$
    $T_1 \leftarrow \Phi_p^{a;\mathrm{v}} U'\,, \quad T_2 \leftarrow \Gamma_p^{a;\mathrm{v}} U'$
    $T_2 \leftarrow \begin{bmatrix} s'''_a & \cdots & s'''_c \end{bmatrix}^T T_2$
    for $i \in \mathrm{l}$ do
      $[\alpha_i^T \mid \beta_i^T] \leftarrow 0$ or $s_i^T T_1$ or $e_{i-a+1}^T T_2$
      $\beta_i^T \leftarrow \beta_i^T \begin{bmatrix} e_1 \cdots e_{i-a} & 0 \cdots 0 \end{bmatrix}$
    $\begin{bmatrix} w_a \cdots w_c \end{bmatrix} \leftarrow -\begin{bmatrix} \beta_a \cdots \beta_c \end{bmatrix} \left( \begin{bmatrix} \alpha_a \cdots \alpha_c \end{bmatrix} - I \right)^{-1}$
    for $i = a, \ldots, c$ do
      $Q_i \leftarrow \begin{bmatrix} P_i & 0 \\ 0 & I \end{bmatrix}$
      $x_i \leftarrow \begin{bmatrix} u_i \\ 0 \end{bmatrix}\,, \quad y_i \leftarrow \begin{bmatrix} v_i \\ w_i \end{bmatrix}$
      $M_p^{i;\mathrm{v}} \leftarrow Q_i + x_i y_i^T$
```