Incremental Matrix Reduction

Anindya Moitra

1 The Background

This document addresses one of the primary challenges of computing persistent homology on streaming data by a fully incremental approach. As a data stream is a potentially infinite sequence of data objects, the entire stream cannot be stored in the memory typically available to a computer. Therefore, the computation of persistent homology on streaming data requires an incremental approach. A couple of computational models for applying persistent homology on data streams are being developed as two separate projects that involve partially incremental approaches. In particular, consistent with the standard computational paradigm [1] for processing data streams, each of those two models consists of two principal components: (i) *online*, and (ii) *offline*. However, yet another (*i.e.*, a third) computational model is developed by a *fully online* or *fully incremental* approach in this project.

A key requirement for developing such a fully incremental model for persistent homology is the ability to perform the *Gaussian elimination* (also called the *reduction*) of the boundary matrix [2, 3] by an incremental algorithm. The Gaussian elimination step is performed during the offline component (*i.e.*, as a *batch processing* mechanism) in the previous two models for computing persistent homology on streaming data. This document develops the theoretical foundation for performing the *Gaussian elimination* by an incremental algorithm.

It is worth mentioning that while computing persistent homology on data streams is one of the primary target applications of the incremental Gaussian elimination algorithm, it will have other important applications as well. For example, due to the large size of the *complex* constructed on a point cloud, the dimension of the boundary matrix increases exponentially with the number of data objects on which persistent homology is being computed. As a result, reducing the boundary matrix by a batch processing algorithm becomes prohibitive even for 'static' (*i.e.*, non-streaming) data sets of moderate size (such as, data sets with up to a few thousands of objects, depending on the memory available to the computer). The incremental algorithm developed by this project will help in the Gaussian elimination of the boundary matrix where this is not possible by batch processing mechanisms due to the size of the matrix.

2 The Problem

The standard algorithm [2,3], as described in [4–7] among others, computes the persistence of a filtration [8] by *reducing* its boundary matrix ∂ to a column-echelon form R. Usually, the entire boundary matrix or a simplified data structure thereof is processed in the memory while the algorithm is executed. This approach is not desirable when computing persistent homology on streaming data by a fully incremental mechanism. When working with data streams processed by a fully incremental model, one would want to add a simplex σ to the already reduced matrix R without having to recompute the reduction of all other columns due to the addition of σ.

Kerber *et al.* [7] introduced a streaming algorithm for reducing the boundary matrix based on optimized versions of the standard algorithm [2, 3]. However, their algorithm assumes that the entire data set or

filtration is available on disk. Therefore, the total ordering of all the simplices in the filtration is known *a priori*. The next simplex added to R has a higher weight than any of the previously added simplices.

In a real streaming application, the entire point cloud or filtration is not available at any time. Hence, every time a new simplex σ is added to the filtration, the indices of those simplices that have weights higher than that of σ are incremented by one. In other words, in a real streaming environment, the column corresponding to a new simplex does not necessarily get added to the right of R. Therefore, the algorithm of [7] can not be applied to real-world streaming applications.

In the next section, we provide two algorithms to incrementally reduce a boundary matrix after the addition of simplices. The goal of both of the algorithms is to compute the updated reduction of the boundary matrix ∂ when new columns are added to R, the existing reduction of ∂.

3 Addition of Simplices: Two Solutions

3.1 Algorithm 1

Let us assume that R is the reduction of a boundary matrix ∂ associated to a filtration K that has n simplices. When a new simplex σ needs to be added to R, we compute the index j and the unreduced column ∂_j of σ with respect to the total ordering of the filtration. ∂_j specifies the facets[1] of σ.

We then apply an optimization technique called *compression* [7] on the column ∂_j: scan through the non-zero entries of ∂_j; If a row index i corresponds to a negative simplex (*i.e.*, if the i-th column of R is not zero), remove the non-zero entry from the column ∂_j. We then insert the compressed column ∂_j in R as the new j-th column. If, due to the addition of ∂_j, R becomes unreduced (*i.e.*, if the pivots of the columns of R are no longer in unique rows), scan the columns from j to n, and reduce them: while the column $k \geq j$ is non-empty, and its pivot is the pivot of another column $l < k$, add column l to column k.

Let R' be the reduction returned by the above algorithm. If the simplex σ was present in the filtration K to begin with, the reduction obtained from that filtration is denoted by R_{org}.

Theorem 3.1. *R' and R_{org} produce the same persistence intervals.*

Proof. It has already been established that the compression step does not alter the pivots of a reduction [7,9]. Since the columns are reduced by left-to-right column additions, the insertion of a new j-th column in R does not impact any column $i < j$.

As the new simplex σ is added to the filtration K, an existing simplex $\tau \in K$ can become a facet of σ. However, τ can never be a cofacet[2] of σ. In other words, an existing simplex of K can be a facet of a new simplex, but the new simplex can not be a facet of an existing simplex. This is due to the definition of the simplicial complex K: every face of an existing simplex τ must already be contained in K. It means that no column $k > j$ can have a non-zero entry in the j-th row.

Since any reduced column is a *linear combination of itself and the columns to its left*, the order of column operations does not alter the reduction of a column.

Therefore, R' and R_{org} have the same pivots and produce the same persistence intervals.

□

[1] A facet is a co-dimension one face of a simplex.

[2] A cofacet is a co-dimension one coface of a simplex.

3.2 Algorithm 2

This algorithm represents a more time-efficient version of Algorithm 1. In particular, the addition and reduction of one simplex at a time might not result in an optimal implementation. As such, we formulate a variant of Algorithm 1 based on the strategy of the reduction of chunks as suggested in [7]. Instead of adding one column at a time, we insert a chunk of C columns at a time into R. An additional advantage of inserting columns in chunks is that we can exploit the popular *clearing optimization* strategy. When a new data point arrives from the stream, we build all the simplices around the new point using an iterative algorithm [8]. Therefore, in a streaming environment, every new data point naturally creates a chunk of simplices. Even though the chunk size C represents a trade-off between time and space efficiency, Kerber *et al.* [7] recommend that one should choose a large value of C to rip higher benefits from the clearing optimization. In a real streaming application, we can adjust C based on the stream speed (the number of incoming data points per time unit) and the update frequency of the persistence intervals required by the application. The algorithm is described below.

We insert C columns into R at a time, and compress them as before. Let j be the lowest index of the newly inserted columns. Then, we start the reduction of the columns j, $j + 1$, ..., n using the clearing optimization. That is, we reduce the columns j, $j + 1$, ..., n in decreasing dimension and set a column to zero as soon as its index becomes the pivot of another column [6].

References

[1] J. A. Silva, E. R. Faria, R. C. Barros, E. R. Hruschka, A. C. P. L. F. de Carvalho, and J. ao Gama, "Data stream clustering: A survey," *ACM Computing Surveys*, vol. 46, no. 1, pp. 3.1–13.31, Oct. 2013.

[2] H. Edelsbrunner, D. Letscher, and A. Zomorodian, "Topological persistence and simplification," in *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, ser. FOCS '00. Washington, DC, USA: IEEE Computer Society, 2000.

[3] A. Zomorodian and G. Carlsson, "Computing persistent homology," *Discrete & Computational Geometry*, vol. 33, no. 2, pp. 249–274, Feb. 2005.

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

[5] N. Otter, M. A. Porter, U. Tillmann, P. Grindrod, and H. A. Harrington, "A roadmap for the computation of persistent homology," *EPJ Data Science*, vol. 6, no. 1, Aug. 2017.

[6] C. Chen and M. Kerber, "Persistent homology computation with a twist," in *Proceedings 27th European Workshop on Computational Geometry (EuroCG'11)*, 2011, pp. 197–200.

[7] M. Kerber and H. Schreiber, "Barcodes of towers and a streaming algorithm for persistent homology," *Discrete & Computational Geometry*, Oct. 2018. [Online]. Available: https://doi.org/10.1007/s00454-018-0030-0

[8] A. Zomorodian, "Fast construction of the vietoris–rips complex," *Computer and Graphics*, pp. 263–271, 2010.

[9] U. Bauer, M. Kerber, and J. Reininghaus, "Clear and compress: Computing persistent homology in chunks," in *Topological Methods in Data Analysis and Visualization III*, P. T. Bremer, I. Hotz, V. Pascucci, and R. Peikert, Eds. Springer International Publishing, Mar. 2014, pp. 103–117.