A BRIEF DESCRIPTION OF THE JAVAPLEX SOFTWARE PACKAGE

ANDREW TAUSZ

Abstract. We give a brief description of the internals of the JavaPlex software package.

1. Introduction

In this chapter, we discuss the JavaPlex software package. The reason for its exists is due to the developments in topological data analysis and computational topology over the past few years. It is an open source package, and its repository is located at `http://code.google.com/p/javaplex`.

2. Architecture

We give a brief overview of the chosen architecture of JavaPlex: which classes inherit from which, which classes carry which functionality, and how the entire package sticks together.

2.1. **Overall Structure.** In a nutshell, the two main capabilities of JavaPlex are:

- the computation of the persistent homology of filtered chain complexes of finite vector spaces
- the automated construction of filtered complexes from geometric data

While it is expected that most of the use of JavaPlex will be for computing the simplicial homology of complexes through constructions such as the Vietoris-Rips and witness definitions, its internal structure is built in a way that anticipates further extensions. We will discuss such such extensions towards the end of the section.

3. Fundamental Interfaces and Classes

3.1. **Basis Elements and Filtered Chain Complexes.** Let us start with the essential core: the notion of a basis element in a chain complex. The two main data attached to such an object are its homological dimension (where it fits into the complex) and its boundary. The `PrimitiveBasisElement` interface defines the behavior of such an object by mandating the implementation of the `getDimension` function, as well as the `getBoundaryArray` and `getBoundaryCoefficients` functions. Of course, we require any implementation of such a class to satisfy the condition that the boundary iterated twice must yield zero.

The two main implementations of the `PrimitiveBasisElement` interface are the `Simplex` and `Cell` classes. The first provides the functionality of a standard n-simplex where the boundary is the oriented sum of its faces. Note that unlike in jPlex, JavaPlex has no restrictions on the dimensionality of a simplex. The `Cell` class implements the functionality of a cell within a CW complex. In general, the prescription of the boundary of a cell is more cumbersome since it may require the specification of the degree of each attaching map. Nevertheless, cell complexes provide a significantly more compact representation of various

Date: October 8, 2011.

topological spaces than simplicial complexes. Also note that at the moment, there are no automated methods for constructing cell complexes out of geometric data as there are with simplicial complexes.

It is also important to note that the filtration information about a simplex or cell is not held by the simplex itself, but is held within the filtered complex, which is discussed next

The next essential interface is the `AbstractFilteredStream`. This abstracts the notion of a filtered chain complex of finite dimensional vector spaces. As with the mathematical notion of a chain complex, it defines both the terms in the complex (the vector spaces) as well as the connecting morphisms (the boundary maps) as follows:

- Access to the basis elements for the terms in the filtered complex are provided through an iterator. In particular, this interface extends the `Iterable` interface which allows for convenient iteration. Implementing classes must provide the basis elements for the filtered complex in increasing order, where the ordering is lexicographical first on the filtration index, and then on the dimension.
- The boundary operators are provided by the `getBoundary` and `getBoundaryCoefficients` functions, which return the basis elements in the boundary as well as the coefficients.
- An implementation of the `AbstractFilteredStream` interface must also provide methods for obtaining the filtration index and homological dimension of a given basis element.

At this point, one may ask why the operations such as the homological dimension, boundary, etc. must be provided by filtered complex, and not by the basis elements themselves. The answer to this question is that these operations may depend on the context in which they are being evaluated. However, there is an abstract subclass of `AbstractFilteredStream` which provides these.

The `PrimitiveStream` class inherits from `AbstractFilteredStream`, and provides the default values for the dimension and boundary functionality based on the geometric properties of the given basis element.

However, this need not always be the case. For example, there are all kinds of homological constructions (tensor products, hom, shifting, direct sums, algebraic mapping cylinders) in which the notions of dimension and boundary do not have direct geometric meanings, but are derived from the constituent complexes. Examples of implemented operations are the tensor product of complexes, hom complexes, and dual complexes.

3.2. **Persistent Homology.** The main interfaces of importance here are `AbstractPersistenceAlgorithm` and `AbstractPersistenceBasisAlgorithm`. The first one requires the implementation of a function called `computeIntervals` which takes in a `AbstractFilteredStream` and returns a barcode collection. The second one has an additional function, `computeAnnotatedIntervals`, that returns generators along with each interval.

There are a number of different implementations of the above interfaces. The main two sources of which are the algorithms presented in the papers [ZC05, dSMVJ10]. Furthermore, each of them are implemented over different coefficient types, with specialized implementations for $\mathbb{Z}/2\mathbb{Z}$ coefficients. In general, it is recommended that the user obtains the default algorithm by calling the `getDefaultSimplicialAlgorithm` function (or its equivalent) from the class `Plex4`. This returns the algorithm that performs most efficiently overall.

3.3. **Streams Derived from Metric Spaces.** In order for the package to be useful in application settings, there are a number of classes available for the construction of filtered simplicial complexes from geometric data. The available implementations are the Vietoris-Rips, witness and lazy-witness constructions, which are provided by the `VietorisRipsStream`, `WitnessStream` and `LazyWitnessStream` classes. These classes can be constructed from metric spaces which implement the `AbstractSearchableMetricSpace` interface, but the accessor functions in the `Plex4` class also allow the user to construct these directly from Euclidean point-sets.

The `AbstractSearchableMetricSpace` interface basically defines the functionality of a finite metric space in which one can perform queries for nearest neighbors, k-nearest neighbors, and fixed-radius neighborhoods. There is a default implementation called `EuclideanMetricSpace` which performs all of these operations using the standard Euclidean distance on arrays of doubles. There is also an implementation which is based on KD-trees, however, due to the storage overhead one only experiences a speed-up for very large datasets. It is likely that more work will be put into optimizing it in the future. There is also a class called `ExplicitMetricSpace` which requires the user to supply a distance matrix.

The classes `ExplicitSimplexStream` and `ExplicitCellStream` implement the `AbstractSearchableMet` interface as well, and provide the user with convenient functions for building their own custom filtered complexes.

4. Tests and Verification

In order to ensure that JavaPlex performs adequately in terms of correctness and efficiency, there are a number of tests implemented. We do not talk about all of them here (an interested reader may consult the src_test folder), but we describe the most important one. The class `PersistenceAlgorithmEqualityTest` verifies the integrity of the persistent homology computations as follows. It generates several random test spaces and constructs various filtered simplicial complexes from them. It then runs all of the available persistent homology algorithms, including the one from the previous version (jPlex), on the datasets. Then it compares the resulting barcodes to make sure that they are all identical. The examples range from the trivial (a triangle) to complexes with several hundred thousand simplices.

5. A Brief Word on Efficiency

In this section we present a few notes and caveats regarding efficiency. Although efficiency is not one of the main goals of JavaPlex, good algorithms were selected according to the latest advances in the research literature. With this in mind, we remind the user of a few points:

- Simplicial complexes are larger than you think! For example, a minimal simplicial 16-sphere (the boundary of a standard 17-simplex) contains 262142 simplices.
- We invite the user to be aware of the fact that careless parameter choices may create extremely large complexes. For example, if you are asking JavaPlex to compute the persistent homology of a Vietoris-Rips filtration on a 1000 point dataset, it may be the case that there are on the order of one million simplices without you realizing it.
- One important recommendation is to use small values for the maximum filtration value, and maximum dimension and scale them up gradually. The witness and lazy-witness constructions are also very helpful in capturing the homological features of a dataset with far fewer simplices.

- Since JavaPlex is implemented in Java, it comes with the associated overhead costs.

Developing software that is reliable, extensible and efficient is difficult. Furthermore, once a program is working it is possible to put virtually unlimited amount of effort to making it run as efficiently as possible. The current version of JavaPlex in fact scales better for computing persistent homology than its predecessor jPlex. Nevertheless, improving efficiency is an ongoing process.

6. Extension Examples

The fact that the design of JavaPlex is centered around a few key interfaces makes extension relatively easy. Suppose for example, that we wanted to implement a shifted chain complex. There are two easy ways of doing this. One method is to create a subclass of `AbstractFilteredStream` which has a member variable also of type `AbstractFilteredStream` which is set through the constructor. This member variable will be the "underlying" complex. For all of the member functions, we just perform the exact same function on the underlying complex, with the exception of the `getDimension` function in which we return the value of `getDimension` on the underlying complex with the shift added.

Another example is that one may be interested in implementing other filtered stream constructions from point clouds or metric spaces, perhaps the alpha-shape complex. This can be easily accomplished by inheriting from the `PrimitiveStream` interface.

7. Key Differences from jPlex

In this section we list some key differences between jPlex and JavaPlex

- In JavaPlex, none of the filtered complexes (Vietoris-Rips, etc.) are specified by the parameter delta. In jPlex, the maximum filtration value and the increment (delta) were used. However, this made it cumbersome when the data changed since both quantities had to be updated. In JavaPlex, the relevant parameters are the maximum filtration value, and the number of divisions, which has a global default value of 20. This means that one can set the number of divisions (or just use the default), and forget about it.
- Persistent homology computations in JavaPlex return intervals with filtration indices and not filtration values. These intervals can be converted to filtration value intervals as shown in the matlab examples. The reason for this choice is that there are situations where floating point filtration values are not always relevant.
- JavaPlex is completely 0-based. jPlex did some converting between 0-based and 1-based arrays, however, this is not necessary.
- The dimension of simplices in jPlex was limited to 7. There is no such limitation in JavaPlex.
- Persistent homology in JavaPlex can be computed over arbitrary fields. Additionally, it has specialized implementations for $\mathbb{Z}/p\mathbb{Z}$ and $\mathbb{Z}/2\mathbb{Z}$.
- JavaPlex does not display barcodes in GUI windows. Instead it just saves them directly to file in png format.
- Filtration data in JavaPlex is not stored with the simplex (or basis element), but rather it is stored in the stream (the filtered complex). The reason for this choice is that this allows one to do homological operations on chain complexes that would not be possible otherwise.

References

[dSMVJ10] Vin de Silva, Dmitriy Morozov, and Mikael Vejdemo-Johansson, *Dualities in persistent (co)homology*, Unpublished manuscript, 2010.

[ZC05] Afra Zomorodian and Gunnar Carlsson, *Computing persistent homology*, Discrete Comput. Geom **33** (2005), 249–274.

Stanford University, Stanford, CA, 94305
E-mail address: atausz@stanford.edu