Implementation of Strong Collapses

Generated by Doxygen 1.8.11

Contents

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

Class Documentation

2.1 MsMatrix Class Reference

Class MsMatrix.

`#include <MsMatrix.h>`

Public Member Functions

- MsMatrix ()
  Default Constructor.
- MsMatrix (Simplex_tree st)
  Main Constructor.
- ~MsMatrix ()
  Destructor.
- void strong_collapse ()
  Function for performing strong collapse.
- Simplex_tree collapsed_tree ()
  Function for computing the Simplex_tree corresponding to the core of the complex.
- Map reduction_map ()
  Function for returning the ReductionMap.

Private Member Functions

- void init_lists ()
  Initialisation of vertices' and maximal simplices' lists.
- bool check (int index1, int index2, bool which)
  Checks if index1 contains index2.
- Vector analyse_list (bool which)
  Analyses a list for possible collapses/contractions.
- void turn_off (Vector indices_to_off, bool which)
  Deactivates rows/columns that were identified by analyse_list().
- void modify (Vector indices_to_off, bool which)
  Modifies the other list because of changes identified by analyse_list().
- bool one_step_collapse ()
  Performs one step of the entire strong collapse.
- void fully_compact_this_vertex (Map::iterator iter)
  Function to fully compact a particular vertex of the ReductionMap.
- void fully_compact ()
  Function to fully compact the Reduction Map.

Private Attributes

- typeVectorVertex vertex_list

  Stores the vertices of the original Simplicial Complex.
- MapVertexToIndex reverse_map

  Stores the Reverse Map between indices and values of the vector ***vertex_list****.*
- int rows

  Stores the number of vertices in the original Simplicial Complex.
- boolVector * MxSimplices

  Stores the Matrix of bool values representing the Original Simplicial Complex.
- int cols

  Stores the number of Maximal Simplices in the original Simplicial Complex.
- bool * active_rows

  Stores true for active rows and false for deactivated rows.
- bool * active_cols

  Stores true for active columns and false for deactivated columns.
- List vert_indices

  Stores the list of vertices in an "appropriate" manner convenient for collapse.
- List simp_indices

  Stores the list of simplices in an "appropriate" manner convenient for collapse.
- Map ReductionMap

  Map that stores the Reduction / Collapse of vertices.

2.1.1 Detailed Description

Class MsMatrix.

The class for storing the Vertices v/s MaxSimplices Matrix and doing collapse operations using that matrix.

2.1.2 Constructor & Destructor Documentation

2.1.2.1 MsMatrix::MsMatrix ( ) [inline]

Default Constructor.

Only initialises all Data Members of the class to empty/Null values as appropriate. One *WILL* have to create the matrix using the Constructor that has an object of the Simplex_tree class as argument.

2.1.2.2 MsMatrix::MsMatrix (Simplex_tree st) [inline]

Main Constructor.

Argument is an instance of Simplex_tree.
This is THE function that initialises all data members to appropriate values.
vertex_list, **reverse_map**, **rows**, **cols**, **MxSimplices**, **active_rows** and **active_cols** are initialised here. **vert_indices** and **simp_indices** are initialised by init_lists() function which is called at the end of this.
What this does:

1. Populate **vertex_list** and **reverse_map** by going over through the vertices of the Simplex_tree and assign the variable **rows** = no. of vertices
2. Initialise the variable **cols** to zero and allocate memory from the heap to **MxSimplices** by doing *MxSimplices = new boolVector[rows];*
3. Iterate over all simplices [Depth-First-Search fashion] (using Gudhi's *complex_simplex_range()*) and for all leaf nodes of the tree [candidates for Maximal Simplices] :
   Check if there is already a maximal simplex inserted into the matrix that is a coface of the current simplex in concern.
   If not, insert this simplex as a maximal simplex into the Matrix [candidacy confirmed] and increment the variable **cols** by one.
   Else, don't, because it is confirmed that this is not a maximal simplex.
4. Initialise **active_rows** to an array of length equal to the value of the variable **rows** and all values assigned true. [All vertices are there in the simplex to begin with]
5. Initialise **active_cols** to an array of length equal to the value of the variable **cols** and all values assigned true. [All maximal simplices are maximal to begin with]
6. Calls the private function init_lists().

2.1.2.3 MsMatrix::~MsMatrix ( ) [inline]

Destructor.

Frees up memory locations on the heap. Specifically, does delete[ ] on :

1. **active_rows**
2. **active_cols**
3. **MxSimplices**

2.1.3 Member Function Documentation

2.1.3.1 Vector MsMatrix::analyse_list (bool which) [inline], [private]

Analyses a list for possible collapses/contractions.

Assumes that the list is sorted on the first integer values of the tuples in decreasing order.#
Does this for the list of vertices (**vert_indices**) if the value of the argument **which** is *true*, otherwise does this for the list of simplices (**simp_indices**).
Iterates over the list and for each tuple (say *tup*) where the bool value is *true* (meaning that this is a

possible candidate for removal), it goes over all the entries that have appeared before *tup* in the list (because only entries before this *tup* are candidates for dominating it : sorted condition), and calls the function check() to see if there is actually a dominating-dominated pair.

If yes, then :
(1) Remove *tup* from the list. ** *This was the reason of using a list : constant time removal of entries* **
(2) Append the row number of *tup* (as contained in the second *int* element of the tuple) to the vector that we will return.
(3) If we are operating on **vert_indices** (ie, if **which** is *true*) : Add this pair to the **ReductionMap**.

If no, then change that *true* value to *false* in *tup*.

2.1.3.2 bool MsMatrix::check (int index1, int index2, bool which) [inline], [private]

Checks if index1 contains index2.

Checks if the *ticks* (true bool values) of row/column[1] numbered **index2** are a subset of those of row/column[1] numbered **index1**.
[1] Does this for rows if the value of argument **which** is *true* and for columns if value of the argument **which** is *false*.
Returns *true* if they are indeed a subset, else *false*.

2.1.3.3 Simplex_tree MsMatrix::collapsed_tree ( ) [inline]

Function for computing the Simplex_tree corresponding to the core of the complex.

First calls strong_collapse(), and then computes the Simplex_tree of the core using the Matrix that we have. How does it compute the simplex tree ?
Goes over all the columns (remaining MaximalSimplices) and for each of them, inserts that simplex ['that simplex' means the maximal simplex with all the (remaining) vertices] with all subfaces using the *insert_simplex_and_subfaces()* function from Gudhi's Simplex_tree.

2.1.3.4 void MsMatrix::fully_compact ( ) [inline], [private]

Function to fully compact the Reduction Map.

While doing strong collapses, we store only the immediate collapse of a vertex. Which means that in one round, vertex x may collapse to vertex y. And in some later round it may be possible that vertex y collapses to z. In which case our map stores :
x -> y and also y -> z. But it really should store : x -> z and y -> z. This function achieves the same.
It basically calls fully_compact_this_vertex() for each entry in the map.

2.1.3.5 void MsMatrix::fully_compact_this_vertex (Map::iterator iter) [inline], [private]

Function to fully compact a particular vertex of the ReductionMap.

It takes as argument the iterator corresponding to a particular vertex pair (key-value) stored in the ReductionMap.
It then checks if the second element of this particular vertex pair is present as a first element of some other key-value pair in the map. If no, then the first element of the vertex pair in consideration is fully compact. If yes, then recursively call fully_compact_this_vertex() on the second element of the original pair in consideration and assign its resultant image as the image of the first element of the original pair in consideration as well.

2.1.3.6 void MsMatrix::init_lists ( ) [inline], [private]

Initialisation of vertices' and maximal simplices' lists.

Assumption : Assumes that the 2D matrix is formed.#
Initialises the two lists of vertices and maximal simplices.
These lists are important because the collapse operations are mainly performed on these only.
Lists are of 3-tuples of the type *(int , int , bool)*.
For vertices :
(no. of MxSimp of which this vertex is a part of , row number in the matrix , true)
For MaximalSimplices :
(no. of vertices that are a part of this MxSimp , column number in the matrix , false)

2.1.3.7 void MsMatrix::modify (Vector indices_to_off, bool which) [inline], [private]

Modifies the other list because of changes identified by analyse_list().

Does this for **vert_indices** if the value of argument **which** is *true* and for **simp_indices** if value of the argument **which** is *false*.
Essentially, analyse_list() identifies and does some deletions from one of the lists. This potentially causes changes in the other list. This function does those changes.
Let's say analyse_list() analysed the list of vertices(**vert_indices**) and removed some of them. Now, MaxSimplices that contained one or more of those vertices (which were removed) would be affected.
So this function would go over all the vertices (say *v*) that were removed (as contained in the argument vector **indices_to_off**) and go over all MaxSimplices (say *mxs*) (contained in the data member **simp_indices**) and if **MxSimplices**[*v*][*mxs*] {ie, this MaxSimplex named *mxs* contains *v*}, then change the tuple corresponding to this *mxs* from :
(num , index , true/false) to (num-1 , index , true).

Similarly, if the list of simplices(**simp_indices**) would have been changed by analyse_list(), this function would appropriately modify the list of vertices(**vert_indices**) {and the argument **which** would be *true*}.

2.1.3.8 bool MsMatrix::one_step_collapse ( ) [inline], [private]

Performs one step of the entire strong collapse.

init_lists() should be done before this.#
Returns *true* if this one step actually caused some reduction. Else, returns *false*, which indicates that we have reached the core of the complex.
What it does :

1. Sort **vert_indices** on the first integer values of the tuples in decreasing order.
2. Apply analyse_list() function on the **vert_indices**.
3. If no reduction happens (ie analyse_list() returns an empty vector), then it means that we have reached the core of the complex. Hence, return *false*.
4. Otherwise, some reduction has happened (and analyse_list() returns a non-empty vector of indices to remove >> call this vector *vec_vert*).
5. So then, apply turn_off() on **vert_indices** using this *vec_vert* and apply modify() on **simp_indices** using this *vec_vert*.
6. Sort **simp_indices** on the first integer values of the tuples in decreasing order.
7. Apply analyse_list() function on the **simp_indices** ; say it returns a vector *vec_simp* that it removed.
8. Then, apply turn_off() on **simp_indices** using this *vec_simp* and apply modify() on **vert_indices** using this *vec_simp*.

2.1.3.9 Map MsMatrix::reduction_map ( ) [inline]

Function for returning the ReductionMap.

This is the (stl's unordered) map that stores all the collapses of vertices.
It is simply returned.

2.1.3.10 void MsMatrix::strong_collapse ( ) [inline]

Function for performing strong collapse.

While one step collapses are possible, it does them and stops when the matrix has reached to the core of the initial complex.
Then, it completes the ReductionMap by calling the function fully_compact().

2.1.3.11 void MsMatrix::turn_off (Vector indices_to_off, bool which) [inline], [private]

Deactivates rows/columns that were identified by analyse_list().

Does this for rows if the value of argument **which** is *true* and for columns if value of the argument **which** is *false*.
Deactivation means for all elements (say *elem*) of the vector **indices_to_off**, it does :
active_rows[*elem*] = *false* or **active_cols**[*elem*] = *false* as indicated by the value of the argument variable **which**.

2.1.4 Member Data Documentation

2.1.4.1 bool* MsMatrix::active_cols [private]

Stores *true* for active columns and *false* for deactivated columns.

Initialised to an array of length equal to the value of the variable **cols** with all *true* values. Subsequent removal of Maximal Simplices (caused by removal of vertices) is reflected by concerned entries changing to *false* in this array.

2.1.4.2 bool* MsMatrix::active_rows [private]

Stores *true* for active rows and *false* for deactivated rows.

Initialised to an array of length equal to the value of the variable **rows** with all *true* values. Subsequent removal of dominated vertices is reflected by concerned entries changing to *false* in this array.

2.1.4.3 int MsMatrix::cols [private]

Stores the number of Maximal Simplices in the original Simplicial Complex.

This stores the count of Maximal Simplices (which is also the number of columns in the Matrix).

2.1.4.4 boolVector* MsMatrix::MxSimplices [private]

Stores the Matrix of bool values representing the Original Simplicial Complex.

```
boolVector = std::vector<bool>
```

So after counting the number of rows, this is initialised as :

```
MxSimplices = new boolVector[rows];
```

And filled with columns by the Constructor with a Simplex tree as an argument.

2.1.4.5 Map MsMatrix::ReductionMap [private]

Map that stores the Reduction / Collapse of vertices.

```
Map = std::unordered_map<Vertex_handle,Vertex_handle>
```

This is empty to begin with. As and when collapses are done (let's say from dominated vertex v to dominating vertex v') :
ReductionMap$[v] = v'$ is entered into the map.
This does not store uncollapsed vertices. What it means is that say vertex x was never collapsed onto any other vertex. Then, this map ***WILL NOT*** *have any entry like x -> x. Basically, it will have no entry corresponding to vertex x at all.*

2.1.4.6 MapVertexToIndex MsMatrix::reverse_map [private]

Stores the Reverse Map between indices and values of the vector **vertex_list**.

```
MapVertexToIndex = std::unordered_map<Vertex_handle,int>
```

So, if the original simplex tree had vertices 0,1,4,5
vertex_list would store :

```
Values =  | 0 | 1 | 4 | 5 |
Indices =   0   1   2   3
```

And **reverse_map** would be a map like the following :

```
0 -> 0
1 -> 1
4 -> 2
5 -> 3
```

2.1.4.7 int MsMatrix::rows [private]

Stores the number of vertices in the original Simplicial Complex.

This stores the count of vertices (which is also the number of rows in the Matrix).

2.1.4.8 List MsMatrix::simp_indices [private]

Stores the list of simplices in an "appropriate" manner convenient for collapse.

```
List = std::list<Tuple>
Tuple = std::tuple<int,int,bool>
```

Stores the list of simplices that is essential for executing the collapse. The tuple represents :
(no. of vertices that are a part of this MaxSimplex , column number in the matrix , does this have to be checked in this step)
The bool values are all assigned to *false* in the initialisation steps. (All MaxSimplices are not candidates for collapse to begin with because by definition they are maximal)

2.1.4.9 List MsMatrix::vert_indices [private]

Stores the list of vertices in an "appropriate" manner convenient for collapse.

```
List = std::list<Tuple>
Tuple = std::tuple<int,int,bool>
```

Stores the list of vertices that is essential for executing the collapse. The tuple represents :
(no. of MxSimp of which this vertex is a part of , row number in the matrix , does this have to be checked in this step)
The bool values are all assigned to *true* in the initialisation steps. (All vertices are candidates for collapse to begin with)

2.1.4.10 typeVectorVertex MsMatrix::vertex_list [private]

Stores the vertices of the original Simplicial Complex.

```
typeVectorVertex = std::vector< Vertex_handle >
```

So basically this is a vector that stores all the vertices of the Original Simplicial Complex.
So, if the original simplex tree had vertices 0,1,4,5
This would store :

```
Values =  | 0 | 1 | 4 | 5 |
Indices =   0   1   2   3
```

The documentation for this class was generated from the following file:

- MsMatrix.h

Index