DOCUMENTATION: FLAGSER-COUNT

JASON P. SMITH

This is an adapted version of Daniel Lütgehetmann's FLAGSER-COUNT (available here) for computing the number of directed cliques in a directed graph, or equivalently the simplex counts of the directed flag complex of the graph. The main differences from the original flagser-count are: all persistent homology code removed to reduce unnecesary memory usage, a compressed option where the graph is stored by flagser-count in sparse format, the ability to read in the graph in sparse format, the functionality to print the simplices in normal form or in a condensed memory efficient form, the functionality to print the number of directed cliques each vertex belongs to, and the ability to select which vertices to consider as source vertices of the clique.

Important: the input graphs can not contain self-loops, i.e. edges that start and end in the same vertex.

1. Requirements

FLAGSER-COUNT requires a C++11 compiler (such as g++) and the sparsehash package available here.

2. Compiling the source code

To compile using g++: Open the command line, change into the FLAGSER main directory and compile with:

```
g++ src/flagser-count.cpp -o flagser-count -std=c++11 -pthread -lz -I<path_to_sparsehash_src>
```

3. Usage

After building FLAGSER-COUNT it can be run as follows:

```
./flagser-count [options]
```

For example:

```
./flagser-count --in-format flagser --in ./test/a.flag --out a.out --threads 8
```

The following `[options]` exist:

- **--in-format *format*:** the input format, can be *flagser*, *edge-list*, *csc* or *coo*
- **--out *filename*:** the simplex counts and Euler characteristic to *filename*, when omitted the results are printed to the terminal
- **--max-dim *dim*:** the maximal dimension to be computed
- **--print *filename*:** prints all the simplices to a text file where each line gives a simplex, a file called filename$i$.simplices is created for $i = 0, ..., number\_of\_threads - 1$
- **--binary *filename*:** prints all the simplices in a condensed binary format, a file called filename$i$.binary is created for $i = 0, ..., number\_of\_threads - 1$.
- **--max-print *dim*:** the maximum dimension to be printed (inclusive) when –print or –binary are also called
- **--min-print *dim*:** the minimum dimension to be printed (inclusive) when –print or –binary are also called, by default set to 2
- **--size *n*:** the number of vertices in the graph, must be given for edge-list, csc and coo formats
- **--threads *t*:** the number of parallel threads to be used, if not specified $t = 8$ is used
- **--containment *filename*:** prints the number of directed cliques each vertex belongs, where line i in *filename* corresponds to vertex i with the number of cliques it belongs to given for each dimension

--vertices-todo *filename*: select only certain vertices to be considered as source neurons of cliques, input a 32bit numpy array containing the vertices to be considered.
--compressed: stores the graph in memory in sparse format, slower performance but significantly less memory required
--transpose: transposes the graph
--progress: prints the progress of the computation, every time a vertex is finished being considered as a source it prints the current count of that thread and the vertex considered.

4. Formats

4.1. **flagser.** The *flagser* format takes as input a flag file by including --in *filename*, which must have the following shape:

```
dim 0:
0 0 ... 0
dim 1:
first_vertex_id_of_edge_0 second_vertex_id_of_edge_0
first_vertex_id_of_edge_1 second_vertex_id_of_edge_1
...
first_vertex_id_of_edge_m second_vertex_id_of_edge_m
```

The edges are oriented to point from the first vertex to the second vertex.

Example. The full directed graph on three vertices is described by the following input file:

```
dim 0:
0.2 0.522 4.9
dim 1:
0 1
1 0
0 2
2 0
1 2
2 1
```

Note the number of vertices on line 1 of the flag file will be the number of vertices used, so --size has no effect when used with flagser format.

4.2. **edge-list.** The *edge-list* format takes as input a text file by including --in *filename*, where every line of the file has two integers separated by whitespace which reprsents an edge from the first integer to the second. This is equivalent to flagser format with the first three lines removed.

4.3. **csc.** Uses scipy csc format, also requires

```
--indices filename1 --indptr filename2
```

where the two files are numpy arrays in int32 format corresponding to the indices and indptr lists in scipy csc format. Note you can get use csr by using csc and –transpose. WARNING: it is important int32 format is used, this is not checked by flagser-count, and if a different format is used the program will run but may give incorrect results.

4.4. **coo.** Uses scipy coo format, also requires

```
--row filename1 --col filename2
```

where the two files are numpy arrays in int32 format corresponding to the row and column lists in scipy coo format, i.e two lists were (row[i],column[i]) is an edge of the graph, for all i. WARNING: It is important int32 format is used, this is not checked by flagser-count, and if a different format is used the program will run but may give incorrect results.

5. Binary output

When using the --binary the simplices are printed in the following way: Each simplex is represented as a sequence of 64 bit integers. With each 64 bit integer representing 3 vertex id's stored in 21bit format. The simplices are ordered from right to left. If the leading bit is 0 this indicates the start of a new simplex. So if b is the binary representation of a 64 bit int, then $b[0]$ indicates whether we start a new simplex, $b[43:64]$, $b[22:43]$ and $b[1:22]$ are the binary representations of the vertex id of the simplex, in that order. So in a 2 dimensional simplex $b[43:64]$ is the source and $b[1:22]$ is the sink and $b[0] = 0$. See the included python script binary2simplex.py for code to extract the simplices. The number of vertices must be less than 2^{21}

The size (in bytes) required to print the simplices can be computed, when the simplex counts are known, using the following formula:

$$\sum_{i=2}^{d} C_i \left\lceil \frac{i}{3} \right\rceil 8$$

where C_i is the number of simplices in dimension i. Assuming we don't print the vertices and edges, which is the default as these are already known.

University of Aberdeen, Aberdeen, United Kingdom
Email address: jason.smith@abdn.ac.uk