DOCUMENTATION: TOURNSER

JASON P. SMITH

The software TOURNSER is designed for the computation of persistent homology for tournaplexes with different filtrations, see for here for the background. TOURNSER is an adaptation of FLAGSER which in turn is an adaptation of Ulrich Bauer's RIPSER. An online version of tournser is available here.

1. Compiling the source code

TOURNSER requires a C++14 compiler. For Linux or Mac (Windows is currently not supported) open the terminal, change into the TOURNSER main directory and execute

```
c++ tournser.cpp -o tournser -std=c++14 -pthread -O3
```

2. Usage

After building TOURNSER, you can compute persistent homology as follows:

```
./tournser in_file_address out_file_address [options]
```

For example:

```
./tournser ./Examples/test.trn ./Examples/test.homology --filtration relative
```

The following `[options]` exist:

--filtration: One of the following which gives filtration value $F(\sigma)$ to simplex σ:

- *local*: $F(\sigma) = 2\binom{n}{3} + \sum_{v\in\sigma}(\mathrm{sd}_\sigma(v))^2$
- *global*: $F(\sigma) = \sum_{v\in\sigma}(\mathrm{sd}_\mathcal{G}(v))^2$
- *3cycle*: $F(\sigma)$ = Number of directed 3-cycles in σ
- *vertexMax*: $F(\sigma) = \max_{v\in\sigma}(F(v))$
- *vertexSum*: $F(\sigma) = \sum_{v\in\sigma} F(v)$
- *natural*: $F(\sigma) = \sum_{v\in\sigma}(\mathrm{sd}_\sigma(v))^2 + \max_{b\in bdry(\sigma)} F(b)$
- *natural-max*: $F(\sigma) = \max(\sum_{v\in\sigma}(\mathrm{sd}_\sigma(v))^2, \max_{b\in bdry(\sigma)} F(b))$

where $sd_X(v) = indeg_X(v) - outdeg_X(v)$ and $indeg_X(v)$ is the indegree of vertex v in the digraph X.

--max-dim *dim*: the maximal homology dimension to be computed

--min-dim *dim*: the minimal homology dimension to be computed

--print_dist true: Prints the distribution of the filtration values

--count_only true: Computes only the simplex counts and skips all homology computations

--print print_address: Prints the whole tournaplex to a file, see below for format.

--approximate *n*: skip all cells creating columns in the reduction matrix with more than n entries. Use this for hard problems, a good value is often 100000. Increase for higher precision, decrease for faster computation. This function is taken from FLAGSER, see for this article more information.

2.1. **Input Format.** The input file defines the directed graph and must have the following shape:

```
dim 0:
weight_vertex_0 weight_vertex_1 ...  weight_vertex_n
dim 1:
first_vertex_id_of_edge_0 second_vertex_id_of_edge_0 [weight_edge_0]
first_vertex_id_of_edge_1 second_vertex_id_of_edge_1 [weight_edge_1]
...
first_vertex_id_of_edge_m second_vertex_id_of_edge_m [weight_edge_m]
```

The edges are oriented to point from the first vertex to the second vertex, the weights of the edges are optional. The weights should be rational numbers. Important: the input graphs can not contain self-loops, i.e. edges that start and end in the same vertex.

Example. The full directed graph on three vertices without explicit edge weights 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
```

2.2. **Print Tournaplex.** Using the option **--print print_address** will print the whole tournaplex to a file, where each simplex is written as a line with the following information:

- *Vrts*: This is the vertices of the simplex
- *bdry*: This is the boundary of the simplex, where each number corresponds to the simplex in the dimension below which has that "loc" value
- *cbdry*: This is the coboundary of the simplex, where each number corresponds to the simplex in the dimension above which has that "loc" value
- *filt*: The filtration value of the simplex
- *ort*: The orientation of the edges of the tournament represented as a lower triangular matrix M, where the first entry is $M_{1,0}$ and each row is separated by :, where

$$M_{i,j} = \begin{cases} 0, & \text{if } i \to j \\ 1, & \text{if } i \leftarrow j \end{cases}$$

- *loc*: The location (or index) of the simplex, i.e all simplices in dimension k are numbered 0 to n_k.

For example:

```
Vrts = 0 1 3 | bdry = 5 2 3 | cbdry = 1 3 | ort = 0 : 1 1 : | filt = 4 | loc = 5
```

Corresponds to the simplex in dimension 2 which is given by the tournament

and this simplex has three faces in its boundary and two in its coboundary, has filtration value 4 and is the 5'th face of dimension 2

2.3. **pytournser.** Also included here is a python wrapper for tournser. To use *pytournser*, first ensure it is installed (see README), then open python and load the package with:

```
from pytournser import *
```

Then simply call:

```
X=tournser(V,M)
```

Where M is the adjacency matrix of the graph to consider as a numpy array (with entries the filtration values of the edges, which cannot be 0), and V is a vector of the vertex filtration values. The optional entries are:

```
tournser(vertex_values, adjacency_matrix, filtration='vertexMax', approx=False,
approx_val=100000, count_only=False)
```

where filtration, approx_val and count_only are the same as the C++ version of tournser, and approx indicates whether or not to use the approx value. The output of the tournser function is a dictionary with entries: 'cell_counts', 'finite_pairs', 'infinte_pairs' and 'bettis'. Where the i'th entry of each corresponds to the i'th dimension. And an entry of infinite pairs is a single number denoting the birth time of the pair.

Also available is tournser_file and tournser_edge which have the arguments:

```
tournser_file(in_file, filtration='vertexMax', approx=False, approx_val=100000, count_only=False)
tournser_edges(vertex_values, edge_list, filtration='vertexMax', approx=False,
approx_val=100000, count_only=False)
```

where edge list is a $m \times 2$ or $m \times 3$ numpy array, where the third entry of each row is the optional filtration value. And in_file is the address of a .trn file.

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