Spatial graph embeddings and coupler curves Documentation

Release 1.0

Jan Legerský

Apr 30, 2018

CONTENTS

README:

CHAPTER

ONE

SPATIAL GRAPH EMBEDDINGS AND COUPLER CURVES

This program implements a method for obtaining edge lengths of a minimally rigid graph with many real spatial embeddings. The method is based on sampling over two parameter family that preserves so called coupler curve. See project website for the details.

Moreover, it includes Qt application for plotting coupler curves of the 7-vertex minimally rigid graph with the maximal number of embeddings, G48.

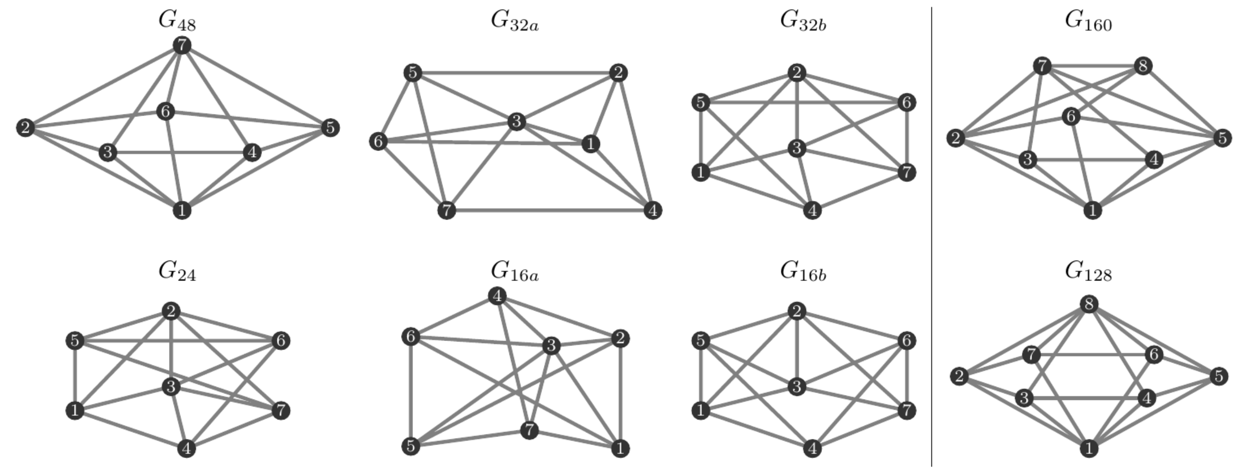
The main functionality is provided by the package *graphEmbeddings3D*, see Documentation.

1.1 Requirements and installation

- Python 2.7
- For solving the system of equations corresponding to graph embeddings, polynomial homotopy continuation by the package `phcpy` is used (homepages.math.uic.edu/~jan/phcpy_doc_html/).
- In the sampling heuristic, clustering is done by `DBSCAN` from the package `sklearn` (scikit-learn.org).
- For GUI application for plotting coupler curves of G48, `PyQt5` (pypi.python.org/pypi/PyQt5) and `matplotlib` (matplotlib.org/) are needed.
- For installation, just clone or download from github.com/Legersky/SpatialGraphEmbeddings.

1.2 Supported graphs

- 6 vertices: octahedron/cyclohexane (the unique 6-vertex graph with the maximal number of embeddings)
- 7 vertices: G16a, G16b, G24, G32a, G32b, G48 (all 7-vertex graphs requiring the last Henneberg step being H2, the number corresponds to the number of embeddings)
- 8 vertices: G128, G160



If you want to compute the number of embeddings for another minimally rigid graph, please, provide a method `constructEquations_YOUR_GRAPH` with sphere equations in *graphEmbeddings3D.graphEmbedding*, and modify the constructor accordingly. For the sampling method, subgraphs suitable for sampling must be added to constructor of *graphEmbeddings3D.algRealEmbeddings*. We appreciate if you share your changes in the GitHub repository.

1.3 Tests

`python test_6vert.py` runs the sampling method for octahedron

`python test_7vert.py` verifies that there are edge lengths for G16a, G16b, G24, G32a, G32b and G48 such that all embeddings are real

`python test_8vert.py` verifies that there are edge lengths G128 and G160 have 128 real embeddings

1.4 Sampling

The scripts in the folder `sampling_scripts` use the proposed method for various graphs and starting edge lengths.

1.5 Coupler curves of G48

This Qt program is launched by `python CouplerCurveG48.py`.

Functionality:

- loading and saving edge lengths
- plotting coupler curve of G48
- computing number of real embeddings of G48 by PHC
- sampling of parameters for specific subgraphs
- iterative method for increasing the number of real embeddings
- export to Axel

1.6 Warning

The program strongly depends on PHC computation - this fails sometimes that might cause failure of the program.

1.7 License

Package documentation:

CHAPTER

TWO

GRAPHEMBEDDINGS3D

2.1 Module AlgRealEmbeddings

class graphEmbeddings3D.algRealEmbeddings.**AlgRealEmbeddings**(*graph_type*, *num_phi=20*, *num_theta=20*, *factor_second=4*, *choice_from_clusters='center'*, *window=None*, *name=None*)

This class implements the sampling procedure for obtaining edge lengths with many real embeddings.

The supported graphs given by *graph_type* can be found in `graphEmbeddings3D.graphEmbedding.GraphEmbedding`.

num_phi and *num_theta* determine the number of samples of φ and θ in the first phase of sampling. In the second phase, *num_phi/factor_second* and *num_theta/factor_second* values are sampled around the ones from the first phase with the highest number of real embeddings.

After sampling, edge lengths are selected from clusters by *choice_from_clusters*:

- *'center'*: center of the (φ, θ) cluster
- *'closestToAverageLength'*: average of lengths in cluster

name is used for temporary and results files.

For implementing a new graph, *self._numAllSol* must be set in the constructor to the maximal number of complex embeddings of the graph, and *self._combinations* contains all subgraphs suitable for sampling.

findMoreEmbeddings(*starting_lengths*, *required_num=None*, *combinations=None*, *allowed_repetition=1*)

Edge lengts with *required_num* real embeddings are searched by linear approach from *starting_lengths*, namely, subgraphs given by *combinations* are used one by one.

If *required_num==None*, then *self._numAllSol* is used. Similarly, if *combinations==None*, then *self._combinations* are used.

The parameter *allowed_repetition* determines, how many times can be the whole list *combinations* used without increase of the number of real embeddings.

Results are saved in `../results`.

findMoreEmbeddings_tree(*starting_lengths*, *required_num=None*, *onlyOne=True*, *combinations=None*)

Edge lengts with *required_num* real embeddings are searched by tree approach from *starting_lengths*, namely, trying all combinations of subgraphs given by *combinations*.

If *required_num==None*, then *self._numAllSol* is used. Similarly, if *combinations==None*, then *self._combinations* are used.

If *onlyOne* is *True*, then the algorithm stops if the first edge lengths with *required_num* real embeddings are found. Otherwise, the whole tree is traversed (extremely long!!!).

Results are saved in `../results`.

2.2 Module GraphEmbedding

class `graphEmbeddings3D.graphEmbedding.`**`GraphEmbedding`**(*lengths*, *graph_type*, *tmpFileName=None*, *window=None*)

This class implements the computation of spatial embeddings for a graph G with edge lengths

Supported graphs:

- G_{16}
  - *graph_type='Max6vertices'*
  - edges: *{(1, 3), (5, 6), (2, 6), (2, 3), (3, 5), (1, 2), (4, 6), (1, 5), (4, 5), (1, 6), (3, 4), (2, 4)}*
- G_{48}
  - *graph_type='Max7vertices'*
  - edges: *{(2, 7), (4, 7), (1, 3), (4, 5), (1, 4), (5, 6), (2, 6), (1, 6), (3, 7), (1, 2), (6, 7), (5, 7), (1, 5), (2, 3), (3, 4)}*
- G_{32a}
  - *graph_type='7vert32a'*
  - edges: *{(4, 7), (1, 3), (5, 6), (1, 4), (1, 6), (3, 7), (2, 5), (3, 5), (1, 2), (6, 7), (5, 7), (3, 6), (2, 3), (3, 4), (2, 4)}*
- G_{32b}
  - *graph_type='7vert32b'*
  - edges: *{(2, 7), (4, 7), (2, 6), (4, 5), (1, 4), (5, 6), (1, 3), (2, 3), (3, 7), (2, 5), (1, 2), (6, 7), (1, 5), (3, 6), (3, 4)}*
- G_{24}
  - *graph_type='7vert24'*
  - edges: *{(2, 7), (4, 7), (2, 6), (5, 6), (1, 4), (1, 3), (2, 3), (3, 7), (2, 5), (1, 2), (4, 6), (5, 7), (1, 5), (3, 6), (3, 4)}*
- G_{16a}
  - *graph_type='7vert16a'*
  - edges: *{(4, 7), (1, 3), (5, 6), (1, 6), (3, 7), (2, 5), (3, 5), (1, 2), (4, 6), (5, 7), (3, 6), (1, 7), (2, 3), (3, 4), (2, 4)}*
- G_{16b}
  - *graph_type='7vert16b'*
  - edges: *{(2, 7), (4, 7), (2, 6), (4, 5), (1, 4), (1, 3), (2, 3), (3, 7), (2, 5), (3, 5), (1, 2), (6, 7), (4, 6), (1, 5), (3, 6)}*
- G_{160}

  - *graph_type='Max8vertices'*, or *'Max8vertices_distSyst'* for using distance system instead of sphere equations (faster but often inaccurate)
  - edges: *{(2, 7), (3, 2), (2, 6), (6, 8), (7, 8), (6, 1), (3, 1), (2, 8), (4, 7), (2, 1), (5, 8), (4, 3), (5, 1), (5, 4), (3, 7), (4, 1), (6, 5), (5, 7)}*
- G_{128}
  - *graph_type='Ring8vertices'*
  - edges: *{(1, 2), (2, 7), (5, 6), (1, 3), (6, 7), (6, 8), (4, 8), (4, 5), (2, 8), (7, 8), (1, 4), (3, 8), (1, 5), (1, 6), (1, 7), (2, 3), (3, 4), (5, 8)}*

Inputs:

- *lengths* is a dictionary with edge lengths of graph given by *graph_type*
- *tmpFileName* is used for temporary files used during computations. If *None*, random hash is used.

For implementation of a new graph, method **constructEquations_newGraph** must be implemented and *__init__* modified accordingly. Moreover, constructor of `AlgRealEmbeddings` must be modified.

Optionally, method **getEmbedding** and function **getEdgeLengthsByEmbedding** should be implemented.

findEmbeddings(*tolerance=1e-15*, *errorMsg=True*, *usePrev=True*)
Compute embeddings of the graph compatible with the current edge lengths and fixed triangle and return them as dictionary *{['real']: listRealEmbeddings, ['complex']: listComplexEmbeddings}*. Embeddings are considered real if the imaginary part of all coordinates is smaller than *tolerance*.

Package `phcpy` is used for the computation. If *usePrev=True*, then the solutions are tracked from ones from the previous call, if there was any.

getEdgeLength(*u*, *v=None*)
Return length of edge *uv*.

getEmbedding()
Return one of the real embeddings comaptible with the current edge lengths.

getEquations()
Return sphere equations of the graph corresponding to current edge lengths.

getLengths()
Return dictionary of edge lengths.

getPhiTheta(*uvwpc*)
Return angles ϕ and θ in the subgraph (u, v, w, p, c) given by 5-tuple *uvwpc*.

setEdgeLength(*Luv*, *u*, *v*)
Set length of edge *uv* to *Luv*.

setLengths(*lengths*)
Set edge lengths to *lengths*.

setPhiTheta(*uvwpc*, *phi*, *theta*)
Set edge lengths so that the angles ϕ and θ in the subgraph (u, v, w, p, c) given by 5-tuple *uvwpc* are *phi* and *theta*.

exception `graphEmbeddings3D.graphEmbedding.`**TriangleInequalityError**(*errorMsg*)
Exception raised if a tringle inequality is violated.

`graphEmbeddings3D.graphEmbedding.`**getEdgeLengthsByEmbedding**(*graph_type*, *vert_coordinates*)
Return edge lengths for *graph_type* obtained by taking corresponding distances of vertices given by *vert_coordinates*.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

g

INDEX