Documentation for linkcalculator.py

David Cushing

April 15, 2016

Abstract

This documentation gives an overview of the file linkcalculator.py.

1 Background

A *polyhedron* is a two-dimensional complex which is obtained from several oriented p-gons by identification of corresponding sides.

Consider a point of the polyhedron and take a sphere of a small radius at this point. The intersection of the sphere with the polyhedron is a graph, which is called the *link* at this point.

The union of all the distinct links of the polyhedron is called the *stargraph* of the complex.

Given a list of words which describe a complex the programme can do the following;

1. Give a list of the vertices used in the stargraph;
2. Gives the edges in the in the stargraph;
3. Lists the vertices used in the stargraph and the neighbours of each vertex;
4. Generate the connected components of the stargraph(the links);
5. Calculate the number of links.
6. Give the adjacency matrix of the star graph.

2 The Programme

To label the edges of a polygon we use lower and upper case letters. The lower case letters represent to edges with clockwise orientation. The upper case letters represent an edge with an anti-clockwise orientation.

For example you would define a Torus(see Figure 1 for the polygon) as follows.

```
>>> TORUS=["abAB"]
```

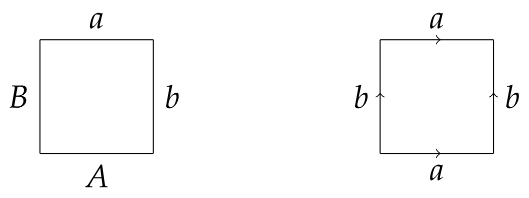

Figure 1: A Torus

To enter a complex with more than one polygon(the one in Figure 2) you input it as follows.

```
>>> EX2=["abcd", "efcl", "fgdi", "ghaj", "hebk",
"ijch", "jkde", "klaf", "libg"]
```

The polygons are glued along corresponding edges in the following way: All edges with the same letter(lower case and upper case) are identified such that their orientations agree. For example the lower and upper sides and the left and right sides in Figure 1 are identified to generate topologically a torus. Note that, in EX2, after the identification, each side bounds three quadrilaterals.

We will now go through each command in the program.

The command `linkvertices` computes all the vertexes used in the stargraph. Note that each letter used in the original labelling contributes two vertices to the stargraph. If the label "a" was used then the vertices "a1" and "a2" are generated corresponding to the start and end of a side labelled "a", respectively. Similarly "a1" and "a2" correspond to the end and start of a side labelled "A", respectively.

```
>>> linkvertices(TORUS)
['a1', 'a2', 'b1', 'b2']
```

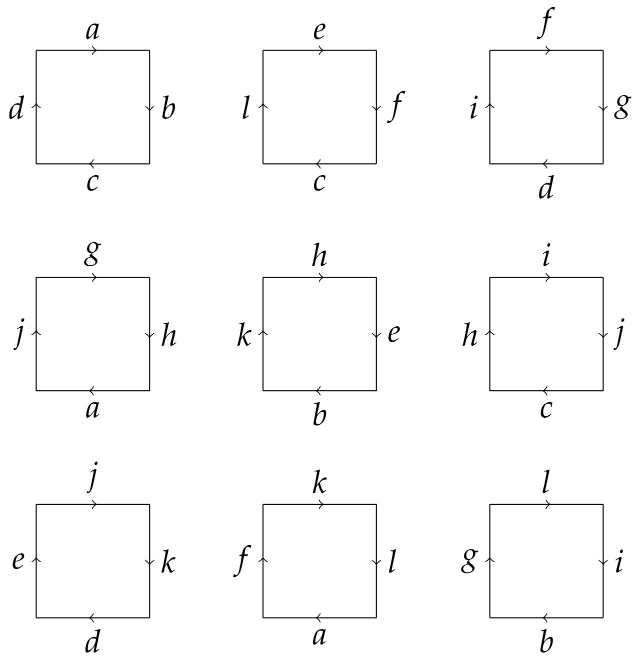

Figure 2: EX2

```
>>> linkvertices(EX2)
['a1', 'a2', 'b1', 'b2', 'c1', 'c2', 'd1', 'd2', 'e1', 'e2', 'f1', 'f2',
 'l1', 'l2', 'g1', 'g2', 'i1', 'i2', 'h1', 'h2', 'j1', 'j2', 'k1', 'k2']
```

The command `linkedges` computes every edge in the stargraph listed as pairs of vertices.

```
>>> linkedges(TORUS)
[['a2', 'b1'], ['b2', 'a2'], ['a1', 'b2'], ['b1', 'a1']]

>>> linkedges(EX2)
[['a2', 'b1'], ['b2', 'c1'], ['c2', 'd1'], ['d2', 'a1'], ['e2', 'f1'],
 ['f2', 'c1'], ['c2', 'l1'], ['l2', 'e1'], ['f2', 'g1'], ['g2', 'd1'],
 ['d2', 'i1'], ['i2', 'f1'], ['g2', 'h1'], ['h2', 'a1'], ['a2', 'j1'],
 ['j2', 'g1'], ['h2', 'e1'], ['e2', 'b1'], ['b2', 'k1'], ['k2', 'h1'],
 ['i2', 'j1'], ['j2', 'c1'], ['c2', 'h1'], ['h2', 'i1'], ['j2', 'k1'],
 ['k2', 'd1'], ['d2', 'e1'], ['e2', 'j1'], ['k2', 'l1'], ['l2', 'a1'],
 ['a2', 'f1'], ['f2', 'k1'], ['l2', 'i1'], ['i2', 'b1'], ['b2', 'g1'],
 ['g2', 'l1']]
```

The command `link_components` list the vertices in each connected component of the stargraph and `number_of_vertices` counts the number of them.

```
>>> link_components(TORUS)
[['a1', 'b2', 'b1', 'a2']]

>>> link_components(EX2)
[['a1', 'd2', 'h2', 'l2', 'i1', 'e1'],
['a2', 'b1', 'j1', 'f1', 'e2', 'i2'],
['b2', 'c1', 'k1', 'g1', 'f2', 'j2'],
['c2', 'd1', 'l1', 'h1', 'g2', 'k2']]

>>> number_of_vertices(TORUS)
1

>>> number_of_vertices(EX2)
4
```

Given a vertex of the stargraph you can calculate all of its neghbours using `neighbours`.

```
>>> neighbours("a1", linkedges(EX2))
['d2', 'h2', 'l2']
```

If an arbitrary graph is given by a list of edges, like in the output of `linkedges(EX2)`, this procedure is still giving the neighbours in this general graph.

Finally we give two functions to produce the stargraph. `stargraph` calculates the stargraph in a way which is simple to read.

```
>>> stargraph(TORUS)
The vertices are ['a1', 'a2', 'b1', 'b2']
a1 has neighbours ['b2', 'b1']
a2 has neighbours ['b1', 'b2']
b1 has neighbours ['a2', 'a1']
b2 has neighbours ['a2', 'a1']

>>> stargraph(EX2)
The vertices are ['a1', 'a2', 'b1', 'b2', 'c1', 'c2', 'd1', 'd2',
'e1', 'e2', 'f1', 'f2', 'l1', 'l2', 'g1', 'g2', 'i1', 'i2', 'h1',
'h2', 'j1', 'j2', 'k1', 'k2']
a1 has neighbours ['d2', 'h2', 'l2']
a2 has neighbours ['b1', 'j1', 'f1']
b1 has neighbours ['a2', 'e2', 'i2']
b2 has neighbours ['c1', 'k1', 'g1']
c1 has neighbours ['b2', 'f2', 'j2']
c2 has neighbours ['d1', 'l1', 'h1']
d1 has neighbours ['c2', 'g2', 'k2']
d2 has neighbours ['a1', 'i1', 'e1']
e1 has neighbours ['l2', 'h2', 'd2']
e2 has neighbours ['f1', 'b1', 'j1']
f1 has neighbours ['e2', 'i2', 'a2']
f2 has neighbours ['c1', 'g1', 'k1']
l1 has neighbours ['c2', 'k2', 'g2']
l2 has neighbours ['e1', 'a1', 'i1']
g1 has neighbours ['f2', 'j2', 'b2']
g2 has neighbours ['d1', 'h1', 'l1']
i1 has neighbours ['d2', 'h2', 'l2']
i2 has neighbours ['f1', 'j1', 'b1']
h1 has neighbours ['g2', 'k2', 'c2']
h2 has neighbours ['a1', 'e1', 'i1']
j1 has neighbours ['a2', 'i2', 'e2']
j2 has neighbours ['g1', 'c1', 'k1']
k1 has neighbours ['b2', 'j2', 'f2']
k2 has neighbours ['h1', 'd1', 'l1']
```

`stargraph_adjacency_matrix` gives the adjency matrix of the stargraph.

```
>>> stargraph_adjacency_matrix(TORUS)
[[0, 0, 1, 1], [0, 0, 1, 1], [1, 1, 0, 0], [1, 1, 0, 0]]
```

See Figures 3 and 4 below for the star graphs of the torus and EX2 respectively.

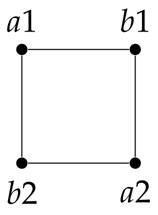

Figure 3: Star graph of the torus.

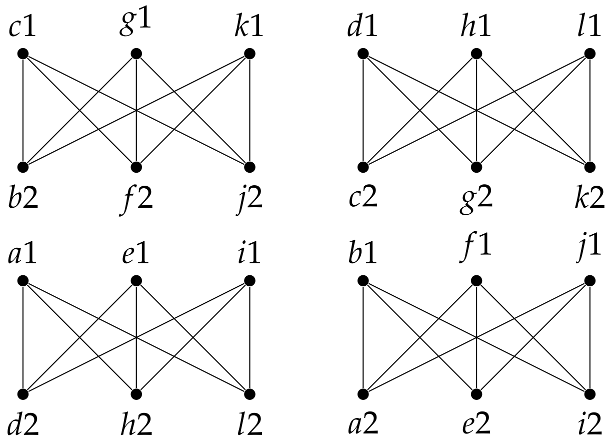

Figure 4: Star graph of EX2.