Introduction to python-igraph

Soroosh Nazem

November 22, 2021

In this tutorial, we are going to introduce python-igraph that is packages for network analysis. igraph provides a huge amount of facilities for those who want to do any analysis on networks, from elementary aspects to advanced ones like shortest path, community detection and clustering, network traffic analysis and so forth. In this tutorial, we are going to introduce igraph by a real-life example which is finding a route in Australian road network between two Australian cities with minimum duration.

Contents

1 The Problem

In figure **??**, a road network between major Australian cities are shown. The numbers written on the roads are the amount of hours between the two cities. Here for each city, we are going to find a path that minimize the spent hours between any two cities. In graph theory, this problem is called **shortest path problem**.

For finding the shortest paths in a graph, there are different algorithms. Depending on the type of graph and our objective, one algorithm may work better than others. For example, some algorithms work better on weighted graph, some others on unweighted, some work better on directed graphs, some on undirected and so forth. Some well-known algorithms are Dijkstra algorithm, Bellman-Ford, Breadth first search. In order to see the list of algorithm and their comparison with respect to complexity and application, see here.

Here we are not going to explain the algorithms in detail, we just want to use the built-in methods in igraph.

2 Introduction to python-igraph

For solving the shortest path problem in igraph, at first we should import igraph package:

Input:

```
import igraph as ig
```

Now, by `Graph()` method, we initiate a graph that here we call it g.

Input:

```
g = ig.Graph()
print g
```

Output:

```
IGRAPH U--- 0 0 --
```

By `print` g, we can get some general information about g. As you see above, graph g is undirected with no vertex and no edge.

Then, by `add_vertices()` method, we can request to add vertices to the graph. Here, since the number of cities is 9, we add 9 vertices to the g.

Input:

```
g.add_vertices(9)
print g
```

Output:

```
IGRAPH U--- 9 0 --
```

As you see, right now g has 9 vertices and 0 edge. now, it is time to add some edges to g. Here, we could have a more concise solution which was giving the number of vertices when initiating the graph:

Input:

```
g = ig.Graph(9)
```

In the next step, we want to give label to the edges

Input:

```
g.vs["label"]=["Sydney","Melbourne","Canberra","Brisbane","Adelaide",
               Alice Springs","Cairns","Darwin","Perth"]
```

Now, we have to add the edges to g. By `add_edge()`, we just add one edge to the graph. By `add_edges()`, we add a list of edges to the graph. Edges are defined by their end-points indices.

Input:

```
g.add_edge(0,1)
g.add_edges([(0,2),(0,3),(1,2),(1,4),(2,5),(3,5),(3,6),(4,5),(4,8),
             (5,6),(5,7),(6,7),(7,8)])
print g
```

Output:

```
IGRAPH U--- 9 14 --
+ edges:
0 -- 1 2 3       3 -- 0 5 6       6 -- 3 5 7
1 -- 0 2 4       4 -- 1 5 8       7 -- 5 6 8
2 -- 0 1 5       5 -- 2 3 4 6 7   8 -- 4 7
```

Right now, we introduce `summary()` method. As you see in the following, the difference between `print g.summary()` is that in the `summary()` the list of edges are not shown anymore. This method can be useful when the graph is very big with a lot of edges.

Input:

```
print g.summary()
```

Output:

```
IGRAPH U--- 9 14 --
```

Now, we have to add the weights of edges to the graph:

Input:

```
g.es["weight"]=[12, 4, 9, 6, 1, 8, 15, 31, 22, 8, 15, 32,
                24, 15, 30, 48]
```

There are three useful methods for checking is the graph is directed, weighted and connected. The result is a boolean value. Besides by `degree()` method, we can have the degrees of vertices.

Input:

```
print g.degree()
print "Number of vertices and edges in g are ",g.vcount(), " and ", g.ecount()
print g.is_directed()
print g.is_weighted()
print g.is_connected()
```

Output:

```
[3, 3, 3, 3, 3, 5, 3, 3, 2]
Number of vertices and edges in g are  9  and  14
False
True
True
```

At this point, all data are imported into igraph and we can start to do our analysis

3 Network Analysis

We are going to find the shortest path between all these cities, for this, we use `shortest_paths_dijkstra()` method in the following way:

Input:

```
shortestLength=g.shortest_paths_dijkstra(weights=g.es["weight"])
print shortestLength
```

Output:

```
[[0.0, 10.0, 4.0, 9.0, 18.0, 19.0, 31.0, 34.0, 50.0], [10.0, 0.0, 6.0, 19.0,
8.0, 21.0, 41.0, 36.0, 40.0], [4.0, 6.0, 0.0, 13.0, 14.0, 15.0, 35.0, 30.0, 46.0
], [9.0, 19.0, 13.0, 0.0, 27.0, 28.0, 22.0, 43.0, 59.0], [18.0, 8.0, 14.0, 27.0,
0.0, 15.0, 39.0, 30.0, 32.0], [19.0, 21.0, 15.0, 28.0, 15.0, 0.0, 24.0, 15.0,
47.0], [31.0, 41.0, 35.0, 22.0, 39.0, 24.0, 0.0, 30.0, 71.0], [34.0, 36.0, 30.0
, 43.0, 30.0, 15.0, 30.0, 0.0, 48.0], [50.0, 40.0, 46.0, 59.0, 32.0, 47.0, 71.0
, 48.0, 0.0]]
```

In the output, we have a list of lists that shows the length of shortest distance between cities. The minimum distance of each city from itself is correctly zero. But we are interested as well to find the routes that lead us to the shortest paths. Let us find the shortest paths from Sydney to other cities. In graph g, the index of Sydney is 0:

Input:

```
shortestPaths=g.get_shortest_paths(0,weights=g.es["weight"],output="vpath")[1:]
print shortestPaths
```

Output:

```
[[0, 2, 1], [0, 2], [0, 3], [0, 2, 1, 4], [0, 2, 5], [0, 3, 6], [0, 2, 5, 7],
[0, 2, 1, 4, 8]]
```

For Sydney we could find the shortest paths to other Australian cities. Here, `output="vpath"` means that we are interested in having the list of paths by vertices not by edges. If we want to

have the path by edges we have to put `output="epath"`. We are interested more in the name of cities instead of their indices, so we can get the names by the following code:

Input:

```
[g.vs[i]["label"] for i in shortestPaths]
```

Output:

```
[['Sydney', 'Canberra', 'Melbourne'],
 ['Sydney', 'Canberra'],
 ['Sydney', 'Brisbane'],
 ['Sydney', 'Canberra', 'Melbourne', 'Adelaide'],
 ['Sydney', 'Canberra', 'Alice Springs'],
 ['Sydney', 'Brisbane', 'Cairns'],
 ['Sydney', 'Canberra', 'Alice Springs', 'Darwin'],
 ['Sydney', 'Canberra', 'Melbourne', 'Adelaide', 'Perth']]
```

4 Some More Points

In this section we introduce a concept that can be very informative about the paths in the network. This concept is called `betweenness centrality`. Betweenness centrality can be defined for both vertices and edges. The betweenness centrality of a vertex v is given by the following formula:

$$g(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

where σ_{st} is the total number of shortest paths from node s to node t and $\sigma_{st}(v)$ is the number of those paths that pass through v, while the betweenness centrality for an edge e is:

$$g(e) = \sum_{s \neq t \in V} \frac{\sigma_{st}(e)}{\sigma_{st}}$$

where σ_{st} is the total number of shortest paths from node s to node t and $\sigma_{st}(e)$ is the number of those paths that pass through e.In fact, betweenness centrality tells us in what percentage of shortest paths a vertex or an edge is present. Let us return to our example and see the betweenness centrality of all cities and roads. For cities, we have:

Input:

```
g.vs.betweenness(weights=g.es["weight"])
```

Output:

```
[8.0, 6.0, 13.0, 3.0, 6.0, 7.0, 0.0, 0.0, 0.0]
```

As you see, Canberra has the maximum presence in the shortest paths, while Cairns, Darwin and Perth do not exist in any shortest path among the cities. On the other hand, for roads the centralities are:

Input:

```
g.es.edge_betweenness(weights=g.es["weight"])
```

Output:

```
[0.0, 14.0, 10.0, 12.0, 8.0, 8.0, 0.0, 4.0, 5.0, 7.0, 3.0, 6.0, 1.0, 1.0]
```

There are two roads that are absent in all shortest path, even the cities that are connected directly by that road. These two roads are "Sydney-Melbourne" and "Brisbane-Alice Springs" roads. In fact, we can claim these two direct roads are useless.

5 Conclusion

igraph is a very powerful package for network analysis. You can find the complete documentation of python-igraph in its webpage.Here; I tried to give you an introduction to igraph by a simple practical example. There are much more complicated examples on network analysis that can be imported to igraph like social networks, networks of diseases and so forth.