Sage Project

Clara Moore

June 10, 2013

1 Background

Sage already has many different types of graph to choose from, but only one of those graphs is data taken from the real world, the rest are purely mathematical. While these graphs my provide interesting and useful results when graph algorithms are applied to them they are not as useful in the development of graph algorithms for sage.

1.1 To make my point

For example sage contains:

1. DegreeSequenceConfigurationModel
2. KneserGraph
3. RandomNewmanWattsStrogatz

1.2 Real world data

But the only real world data graph contained in sage is a world map. Simply one of many (see section **??**). And even that is a relatively small graph with only 166 vertices and with only 323 edges.

2 Why this is important

There are graph algorithms that are meaningless on purely mathematical graphs. To do testing or benchmarking of these algorithms graphs are needed on which the algoritms make sense/are usefull. Also, generated graphs can be very regular, or may not have been seeded well, and using collected data lowers the risk of that kind of bias.

For instance a graph where most vertices have around the same number of edges will behave differently for algorithms like breath first search or connected components than a graph with some irregular structure where a few vertices have many, many edges.

And this is not to even mention that such a difference might be completely unexpected. In this case benchmarking over varying types of graph might reveal oddities not expected, and in attempting to account for and understand these oddities bugs or inefficiencies in the code might be found.

These uses for the graphs will be useful not only for developers of sage attempting to add to the functionality of graphs in sage, but also to users of sage who implement any graph algorithm of their own in sage specifically to suit their needs.

3 Format

I've chosen to make .mtx graph files readable by sage. An .mtx graph is stored in a text file in this form:

```
%comment
number_of_starting_vertices number_of_ending_vertices number_of_edges
starting_vertex ending_vertex edge_weight
starting_vertex ending_vertex edge_weight
starting_vertex ending_vertex edge_weight
starting_vertex ending_vertex edge_weight
starting_vertex ending_vertex edge_weight
.
.
.
```

Though if there is no weight on the edges then each edge line will contain only the starting vertex and the ending vertex.

Reading such a file in and creating a graph from it involves reading through each of lines in the file and parsing and adding it to the edge list if it's an edge.

Because of how large these files are though they are (on the sites I've found) stored compressed. If I created simply a function that could read in a .mtx file it would be up to the user to download and uncompress the file before they could use it. This (from personal experience) is rather annoying, so I also created a function that given a url will download the file, use it be build a graph, and remove the file after it is used. This allows the user to simply specify a url, and not end up with extra files clogging up their directory.

4 Places to aquire graphs

For my code to be useful it requires that there are easily accessible graphs in the right format. Fortunately I chose this format so that there were.

4.1 University of Florida Sparse Matrix Collection

The University of Florida Sparse Matrix Collection holds 2,650 matrices each accesable in a .mtx.tar.gz format. These can be found at

`http://www.cise.ufl.edu/research/sparse/matrices/index.html`

and searched through in many ways including by:

1. dimension
2. symmetry
3. group

4.2 Matrix Market

There is also the site

`http://math.nist.gov/MatrixMarket/`

which seems to store its matrices as .mtx.gz served through an ftp server which is marginally easier to read in but allowing urls from both sites means that the function needs to have parsing capability for the given url and split on what the file extension is.

5 End result

It would be easy to create graphs in sage from actual data, and there are datasets of different sizes available.

The differing sizes are extremely important because the graphs currently in sage are rather small

1. World Graph: 323 edges

while the graphs in the University of Florida Sparse Matrix Collection can get quite large

1. Bates/Chem97Zt: 62,044 edges
2. Andrianov/ex3sta1: 678,998 edges
3. Andrianov/mip1: 10,352,819 edges

The graphs created from data can be orders of magnitude larger than those already in sage, allowing not only for better benchmarking, but also making more apparent ineffecient or unscalable algorithms in the graph portion of the sage library.

5.1 Files created or modified

There was only one file created and one file modified to make this project.

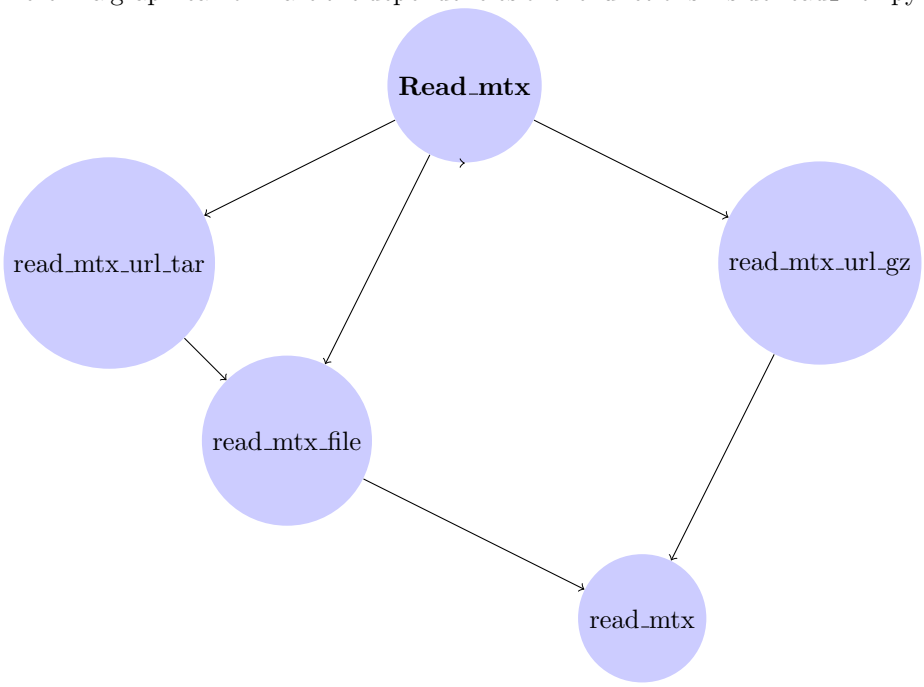
5.1.1 graph_generators.py

The file graph_generators.py was modified so that the name Read_mtx would be visible as a possible option for graphs. Having this layer of abstraction allows Read_mtx to be easily accessable while leaving users of graphs not knowing about the various helper functions implemented.

5.1.2 read_mtx.py

The code written for this project is inside the read_mtx.py file. This file contains the fuction Read_mtx and its various helper functions. Read_mtx takes in a string and does the parsing to tell if that string is a file path or a url, and if it is a url what the extensions are. Once Read_mtx has parsed the input string it calls the appropriate helper function to create a sage graph, and then returns that graph.

Here in a graphical form are the dependencies of the functions inside read_mtx.py.



The helper functions are:

1. read_mtx_url_tar which takes as input a url and the filename of the url path, the function then downloads a .mtx.tar.gz file, untars it, passes the filename to read_mtx_file, does the neccessary clean up of the download, and returns the graph

2. read_mtx_url_gz which takes as input a url and the filename of the url path, the function then downloads a .mtx.gz file, then opens the file and

pases the file descriptor to read_mtx, does the neccessary clean up of the download, and returns the graph

3. read_mtx_file which takes as input a filename of a .mtx file, opens the file, passes the file descriptor to read_mtx, and returns the graph that it gets back

4. read_mtx takes in a file descriptor, reads in the lines, creates a list of edges from those lines, creates a graph with those edges, and returns the created graph