**Q1**

**Graph 1: Metabolic**

Nodes: 1039

Total degree: 9482

Average degree: 9.126

a)

b)

c)

d) There is only 1 GCC containing 100% of the nodes.

e)

f)

g)

**Graph 2: Powergrid**

Nodes: 4941

Total degree: 13188

Average degree: 2.67

a)

b)

c)

d) There is only 1 GCC containing 100% of the nodes.

e)

f)

g)

**Graph 3: Protein**

Nodes: 5410

Total degree: 5410

Average degree: 2.68

**OR**

**Graph 3: Phonecalls**

Nodes: 36595

Total degree: 113706

Average degree: 3.11

a)

b)

c)

d) There are 2643 connected components. 83% of nodes are in the largest one.

e)

f)

g)

**Q2: Complexity**

Loading the graphs.

Time: O(E)

In make\_symmetric() we iterate over the edges in the edgelist to remove self-edges. We then convert the edgelist to a set of frozensets which will remove duplicates of the form (j, k), (k, j). We then convert back to a list of lists. All these operations are linear in the number of edges.

Next, we create a COO matrix, convert is to CSR and add it’s transpose to it to obtain a symmetric matrix. Creating the COO matrix is O(E), converting a COO to CSR is also linear [1], transposing is linear since under the hood they perform a conversion from CSC to CSR [2] and operations on a CSR are said to be “efficient”[3] which we take to mean linear in time.

Space: O(E)

* numpy.loadtxt() is O(E) since file is an edgelist
* make\_symmetric takes O(E) space when we convert between lists and sets.
* O(E) again when building COO matrix.

a) Degree distribution: O(N2)

Getting the degrees of a matrix is O(N2) since we have to iterate over the rows and columns of the adjacency matrix. In practice, we used csgraph.laplacian() which was very fast. The rest of the manipulations are O(n).

b) Clustering coefficients O(N3)

The bottleneck will be raising A to the power of 3. According to the scipy source code [4], the algorithm used for

CSR matrix multiplication runs in O(N\_row \* K2 + max(N\_row, N\_col)), where K is the **maximum nnz** in a row of A and column of B. We use square matrices only so that becomes O(N\*K2 + N).

In a simple undirected graph, K is N-1 for a fully connected node, so the worst-case complexity O(N3), though in practice it might feel much quicker since not all nodes have lots of connections. In fact, given the power law distribution of degrees, we know very few of them do.

c) Shortest Paths O(N2 log N)

We used Djistra’s algorithm with Fibonacci heaps, which is the quickest according to the documentation.

Time complexity is given as “approximately O[N(N\*k + N\*log N)], where k is the average number of connected edges per node.” [5]

Since our average degrees are around much smaller than N this can be approximated by O(N2 + N2 log N ) which reduces to O(N2 log N).

d) Connected Components Θ(N + E)

We used csgraph.connected\_components() whose documentation refers [6] which gives the algorithm Θ(N + E).

e) Eigenvalues O(N3)

We used the scipy.sparse.linalg.eigsh() function, which according to the source code [7] uses the Implicitly Restarted Lanczos Method to find the eigenvalues.

According to [8], the algorithm is O(d\*N2) where d is the average number of nonzero elements in a row.

For the Laplacian, which has all non-zero values, this will be O(N3)

f) Degree correlations O(N2)

We iterate over all rows and all columns to gather each node’s degree.

g) Clustering Coefficient to Degree Relation O(N3)

We need to get the clustering coefficients and we said in b) that it was O(N3). In practice however it was very quick.

**Q3**

We had a first version of the BA model that only attempted to make m edges for each new node. As the graph grew bigger and the probabilities smaller however, more and more nodes did not make edges. We would end up with a graph of n nodes but a big portion of them would have degree 0.

Our second and final version, the one used underneath, ensured that each new node made m edges. This took considerably longer to build. The 30k node to approximate Phonecalls took 6h15min to build and analyze.

**Imitation of Metabolic: AB Model with 1039 nodes, m = 5**

Total degree: 10362

Average degree: 9.97

a)

b)

c)

d) There is 1 connected component. 100% of nodes are in the largest one.

e)

f)

g)

**Imitation of Powergrid: AB Model with 4941 nodes, m = 1**

Total degree: 9880

Average degree: 2.00

Another attempt to imitate had m=2. This gave us:

Total degree: 19758

Average degree: 4.00

a)

b) Zero. M=2 had something here. Maybe change?

c)

d) There is 1 connected component. 100% of nodes are in the largest one.

e)

f)

g) Zero. M=2 had something here. Maybe change?

**Imitation of Protein: AB Model with 5400 nodes, m = 1**

Total degree: 10798

Average degree: 2.00

Another attempt to imitate used m = 2 but we got results too far away

Total degree: 21594 – about 4 times what protein had

Average degree: 4.00 – significantly higher

a)

b) Zero, plot is empty. The m=2 model had something here, maybe switch?

c)

d) There is 1 connected component. 100% of nodes are in the largest one.

e)

f)

g) Zero because clustering coefficients are zero. The m=2 model had something here, maybe switch?

**Bonus: Preferential Unattachment**

We changed the attachment probability calculation of the BA model to 1 – Poriginal BA model, thus inverting the mechanism and making edges more likely with less connected nodes.

This produced some noticeable differences in almost every graph of this assignment. In the following we will compare a normal AB Model graph with n=1039, m=5 (Model A) to our “preferentially less connected” AB Model (Model B) with the same parameters.

Firstly, the degree distribution of the Model A follows a power law much more closely than Model B. This is not all that surprising, as according to the BA textbook [9]: “The origin of the power law and the associated hubs is a *rich-gets-richer phenomenon*”. As Model B does not have such a phenomenon, it might be expected that it doesn’t follow the power law as closely as Model A.

<insert pic>

The average clustering coefficient of Model A is 0.027, which is reasonably close to the theoretical value of

On the other hand, the average for Model B is about 1.385. This is somewhat unexpected, as intuitively Model A generates hubs with closely interconnected nodes.

It is equally surprising that the average shortest path for Model A is much greater than that of Model B. As nodes added to the original BA graph are likely to connect to a hub, it seems strange that the shortest path length is 4.25 compared to 2.00 for Model B.

The plot of the degree vs clustering coefficient suggests that Model B is more disassortative than model A. This makes sense, as nodes added to Model B are more likely to connect to existing nodes of small degree. In the case that an edge from a new node connects to a hub, the next edge from this node is likely to be to a relatively unconnected node. In Model A, all edges from a new node are likely to be incident on high-degree nodes that form hubs.

**References**

[1] Scipy Source Code gives the complexity of COO to CSR conversion as O(nnz(A) + max(n\_row,n\_col))

<https://github.com/scipy/scipy/blob/3b36a574dc657d1ca116f6e230be694f3de31afc/scipy/sparse/sparsetools/coo.h#L31>

[2] CSR and CSC call each other’s constructors when converting.

While COO inverts it’s column and row arrays and it’s shape.

<https://github.com/scipy/scipy/blob/v1.7.1/scipy/sparse/csr.py#L135-L145>

<https://github.com/scipy/scipy/blob/v1.7.1/scipy/sparse/csc.py#L108-L118>

<https://github.com/scipy/scipy/blob/v1.7.1/scipy/sparse/coo.py#L291-L299>

[3] “All conversions among the CSR, CSC, and COO formats are efficient, linear-time operations.” [https://docs.scipy.org/doc/scipy/reference/sparse.html#usage-information\](https://docs.scipy.org/doc/scipy/reference/sparse.html#usage-information%5C)

[4] Scipy Source Code <https://github.com/scipy/scipy/blob/701ffcc8a6f04509d115aac5e5681c538b5265a2/scipy/sparse/sparsetools/csr.h#L542-L544>

[5] Scipy documentation <https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csgraph.shortest_path.html>

[6] D. J. Pearce, “An Improved Algorithm for Finding the Strongly Connected Components of a Directed Graph”, Technical Report, 2005

[7] <https://github.com/scipy/scipy/blob/v1.7.1/scipy/sparse/linalg/eigen/arpack/arpack.py#L1351-L1692>

[8] <https://en.wikipedia.org/wiki/Lanczos_algorithm>

[9] Barabási, A.-L. (n.d.). *Barbasi Model*. Retrieved from Network Science Book: http://networksciencebook.com/chapter/5#barabasi-model