Network Analysis

Abstract

In this assignment I have analysed random graphs such as ring group graphs and preferential attachment networks as well as a coauthorship network. Some of the properties of these networks I analyse are the degree distribution, diameter and brilliance. Another focus is searching algorithms to traverse this graph. In my report I only consider undirected networks, using python, storing them as dictionaries. The key is the node id (from 0 to N-1, where N is the total number of nodes), and the value is a set containing all the nodes ids it is connected to. This means that if eg. node 0 is connected to node 1 the value for 0 would contain 1 and for node 1 it would contain 0. The graphs are generated considering each node once, so for a random graph the expected degree, $\overline{d} = Np$.

1 Ring Group Graphs

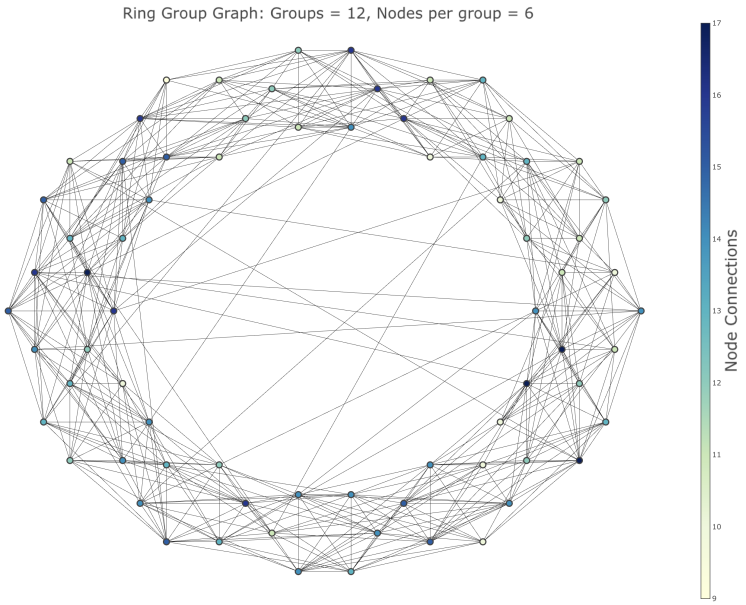

Figure 1: Ring Group Graph positioned with group 0 at the top and going clockwise to group 11 (m - 1) to the left of group 0, p = 0.495, q = 0.005.

I will consider random Ring Group Graphs, these are graphs partitioned into groups with the nodes in each group more likely to connect to nodes in its own group or the groups adjacent. Define the number of groups by m, and group size k. To create the graph I consider each of the N nodes individually and each of the possible N-1 edges it could have. If they are distinct and there is no edge already, I then place an edge between the two nodes with probability p if it is local or q if not (local meaning in the same or adjacent group). I will investigate Ring Group Graphs for $p + q = 0.5, p > q$. Using a program I wrote and with the help of Plotly[1] to visualise Ring Group Graphs, here is one I produced with $m = 12$, $k = 6$, $p = 0.495$, $q = 0.005$ shown in Figure(1). This strongly illustrates the underlying structure, with many edges linking adjacent groups together and few crossing to non-neighbouring groups.

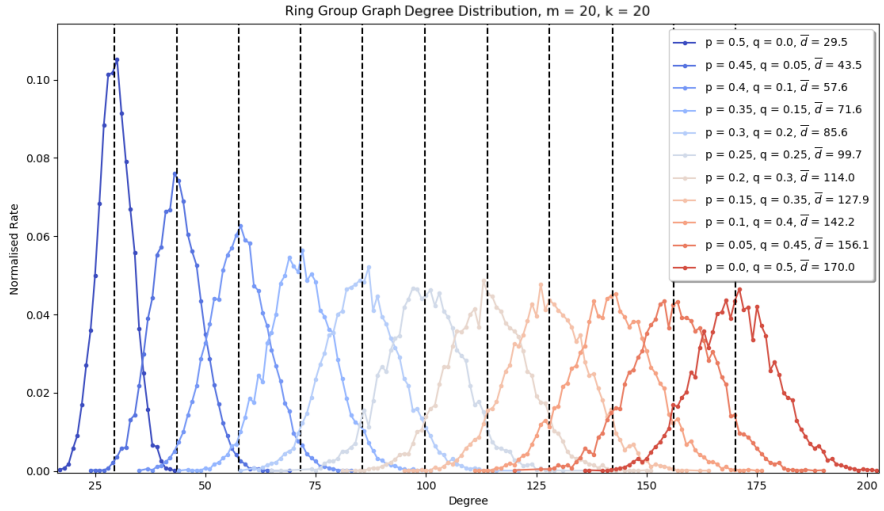

Figure 2: Ring Group Graphs with m = k = 20, p + q = 0.5, p > q for the blue plots and p < q for the red plots, for varying p, q ≤ 0.5 (averaged over 20 graphs each).

1.1 Investigating Degree Distributions

Figure(2) illustrates the degree distribution of different p and q for a typical large Ring Group Graph. However note q ≥ p displays very similar behaviour to random graphs. I will henceforth only consider p > q. Note the y-axis, normalised rate corresponds to the proportion of nodes of that degree.

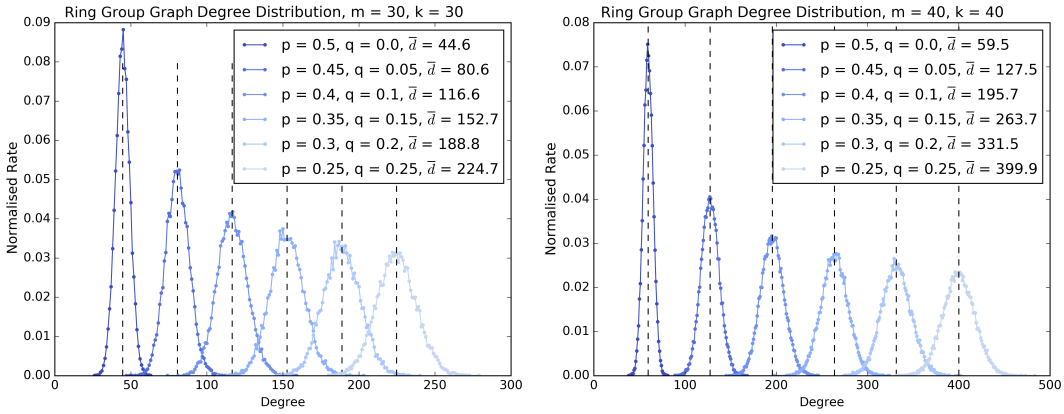

Figure 3: Ring Group Graphs with m = k = 30 (left) and m = k = 40 (right), p + q = 0.5, p > q, for varying p, q (averaged over 20 graphs each).

Similar distributions are found in Figure(3) showing plots for graphs of size 900 (left) and 1600 (right) across all values of p and q. Note the expectations (dotted lines), $\overline{d}$ are approximately evenly spaced with separation ≈ 36 and ≈ 68 degrees. Observe the variation in degree is also small, the degrees don't overlap for graphs produced with p and q differing by 0.1, the degrees are well localised around the average. See for example on the left in Figure(3) the maximum degree of the graph with p = 0.5, q = 0 is about 65 and the minimum degree for p = 0.4, q = 0.1 is about 75. To investigate the ranges I wrote a program to calculate the degree of the lower 0.05 and upper 0.95 quantile of the leftmost 6 distributions shown in Figure(2). I then

[1]Plotly

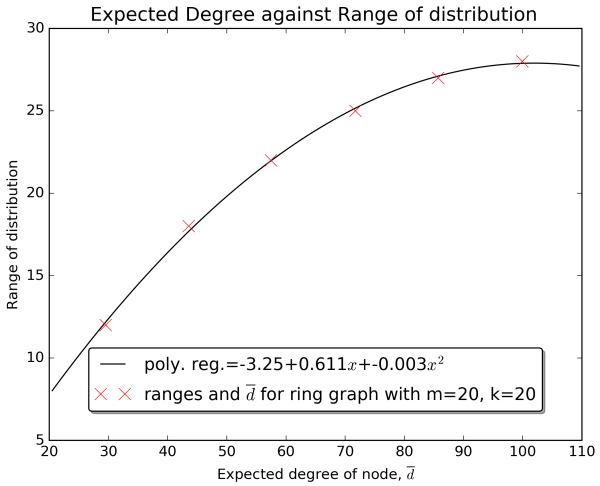

Figure 4: The expected degree for a node for the first 6 plots in Figure(2) against their ranges red x's, fitted with 2^{nd} order polynomial regressor (the black line).

approximated the degree range by the upper 0.95 quantile minus the lower 0.05 quantile. To visualise the relationship I plotted the expected degree against the range and fitted it with a second order polynomial regressor which matches the data quite well, see Figure(4). For p $\leq$ q (plots with expected degree 100 and above) the ranges become approximately constant shown in Figure(2), similar to varying probability of connection for a random graph. For p $\geq$ q the plots are more likely to have many connections to one of 3 groups (the group the node is in, and the two neighbouring ones) than the other groups and hence the standard deviation around the average degree was smaller.

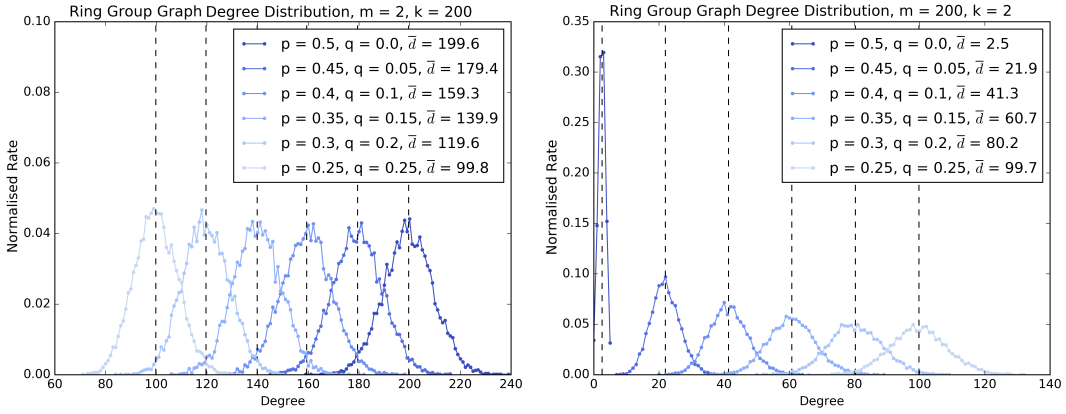

Figure 5: Ring Group Graphs with m = 2, k = 200 (left) and m = 200, k = 2 (right), p + q = 0.5, p > q, for varying p, q (averaged over 20 graphs each).

I then considered extreme cases for m and k, namely m = 2, k = 200 and m = 200, k = 2 shown in Figure(5) we can see that the distribution is very similar to the Ring Group Graph we have explored, so it would be reasonable to assume this distribution holds for Ring Group Graphs in general. Notice that the plots for m = 2, k = 200 (shown on the left) are in reverse order of what we have seen before for Ring Group Graphs. This is because there are only two groups so q has no effect, hence the distributions behave exactly as a random graph with parameter p; similar distribution around average degree, np (= mkp). Note the plots on the left in Figure(5) and Figure(6) are more or less identical. However in the right hand plots in Figure(5) show typical Ring Group Graph behaviour we have seen before. These graphs have just two

nodes in each group, which means the degree of each node will only have a maximum of five neighbours just due to p, the remaining structure will be due to q connecting the rest of the graph (same as a random graph with parameter q). This is evident for q = 0 since there are no nodes greater than five.

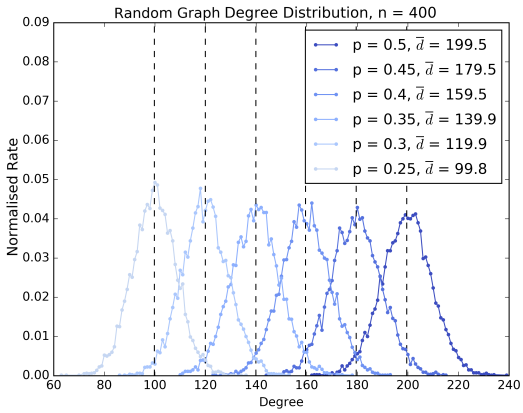

Figure 6: Random Graphs with N = 200, p $\leq$ 0.5, for varying p (averaged over 20 graphs each).

1.2 Investigating Diameter of Ring Group Graphs and p (for fixed q, p > q)

Consider the set of shortest distances between every node in a graph, the diameter of a graph is the longest of these. Figure(7) shows the diameters of Ring Group Graphs against p between 0.2 to 0.8, repeated for 3 values of q (q < p). For each of the q values, diameter is approximately constant across all values of p. What this means is that not only are no nodes/sub-graphs disconnected but the shortest distance between every pair of nodes is no longer than 11 for q = 0, and 3 for q = 0.1! This shows that the graphs are very well connected. Otherwise there is not much interesting information to take from Figure(7).

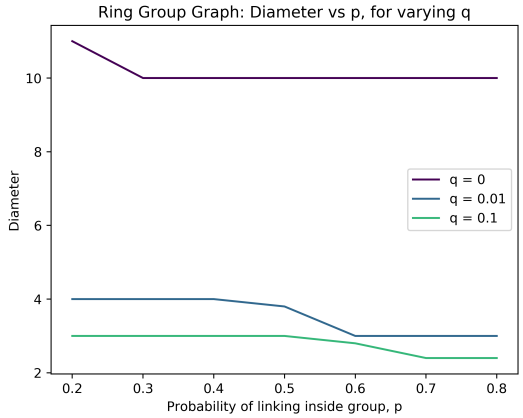

Figure 7: Ring Group Graphs diameters with m = k = 20 against probability, p, with different values of q (averaged over 10 graphs each).

2 Vertex Brilliance

Brilliance of a vertex is defined as the maximum set size of the subset of neighbouring nodes such that no nodes are neighbours of one another, we can investigate this property for the nodes in different networks. To calculate Brilliance I used a function belonging to the Networkx library[2].

2.1 Coauthorship Network

Using citation data for a set of papers the brilliance values and their frequency can be plotted, see Figure(8). The plot shows that the papers with very high brilliance, 100-130 are very rare, however it is very common for papers to have a brilliance between 0-20, showing less popular ones are generally cited less as there are a lot of them.

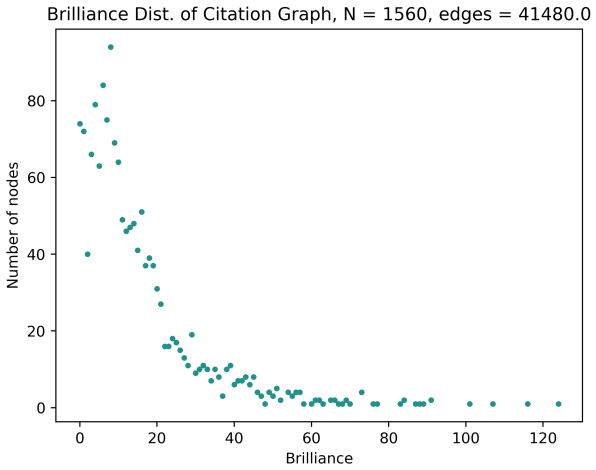

Figure 8: Citation Graph Brilliance.

2.2 Preferential Attachment Graph

Preferential Attachment Graphs are generated by making a complete graph and then adding nodes by making a random choice of nodes with higher probability given to nodes which are more connected, this causes some nodes to become more and more connected and leaves a large amount of less connected ones. Figure(9) shows the Frequencies of nodes of certain Brilliance for three Preferential Attachment Graphs with number of edges similar to the Citation Graph. The trend shown in the preferential attachment plot is very similar to the citation plot, possibly because their structure is generated in a similar way, like PA Graphs, Citation networks may be created by some small set of papers being very popular and becoming more popular because they are cited more.

2.3 Ring Group Graphs

Ring Group Graphs show Gaussian distribution for Brilliance and frequency. This is because each node is symmetric with regard to its probability distribution from linking to every other node, so considering all the nodes in the graph a Normal distribution is produced.

[2]Networkx

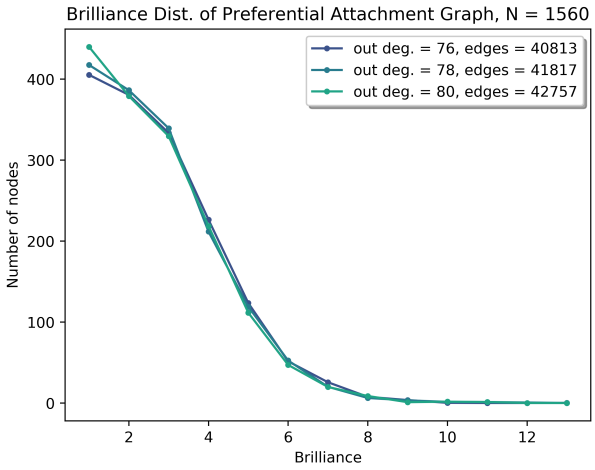

Figure 9: PA Graph Brilliance.

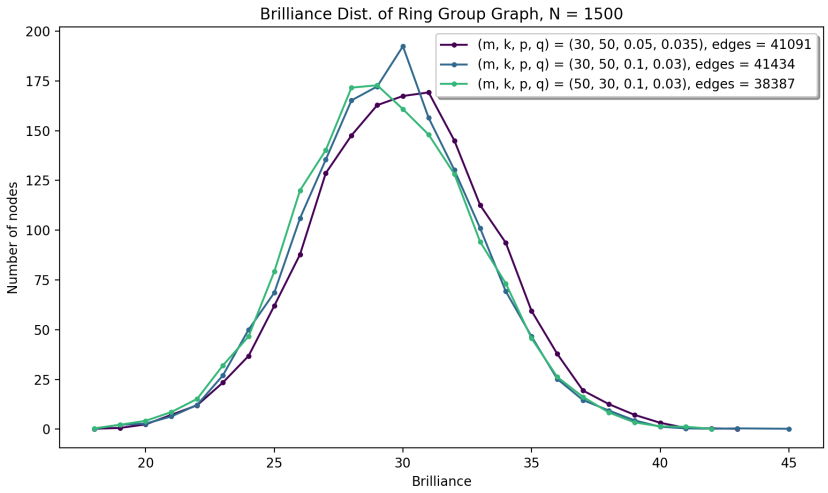

Figure 10: Ring Group Graph Brilliance.

3 Graph Search

Consider the problem of searching random networks not recording the path taken, only the current node and target node. Define the search time to get from node u to distinct node v as the number of queries made, where a query is getting the degree of a node from the neighbours of a current node.

3.1 Searching Random Graphs

To search a Random Graph there aren't many techniques we can use to improve the search time. Searching more of the neighbours for a node with above average neighbours may be a good idea as it is more likely to be connected to random target node v. Going to a neighbour with a higher degree with greater probability may also be a good idea to explore more of the network in less queries. I will explore these two features in an algorithm to search Random Graphs.

3.2 Searching Algorithm

I made an algorithm with the following structure: function search_random_graph(starting_node, target_node, graph, average_degree):
while current_node not equal to target_node:
choose_next current_node, update number of queries made.

return number of queries
function choose_next(current_node, target_node, graph, average_degree):
view each neighbour one-by-one,
judge when to move to one of the neighbours based on their degree, and which one.
return next_node and number of queries made to choose that node.

In choose_next after each new neighbours degree is viewed a judgement is made whether this node is a suitable candidate to move to, and based on how many we've already searched. For each decision it computes the difference between the current degree and average degree, normalised by the range (range being my estimate for the degree of the 0.95 quantile - average degree, or average degree - 0.05 quantile) my estimate is a quadratic function with coefficients learnt from the 2^{nd} order polynomial regressor I used for the expectations vs ranges, for example in Figure(4). Then the output goes through a sigmoid function to get a value between 0 and 1. The resulting probability has tuneable parameter alpha to the power of it (alpha > 0). The other factor I thought could affect the search time, how many of the neighbours we've seen, is implemented into the decision by computing p *= (num_neighbours-num_searched)/num_neighbours) this means if it hasn't searched many of the neighbours it has a higher probability of searching for longer, and if it has search many it is more likely to stop searching. The overall probability p that it makes this decision is tuneable by parameter beta as its exponent (beta > 0). The decision of which node to mode to is made by a roulette wheel with weights based on each of the nodes degree, tuned by parameter gamma as its exponent (gamma > 0).

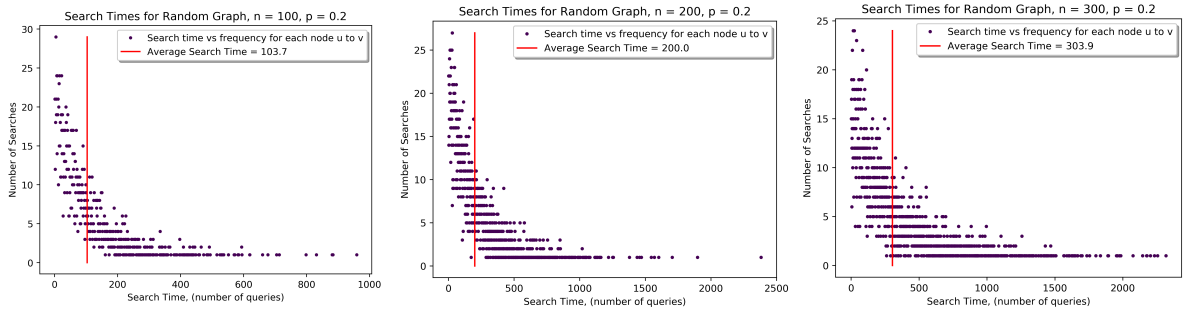

Figure 11: Random Graph Search, number of queries against how many searches took that long, for p = 0.2, n = 100, 200, 300.

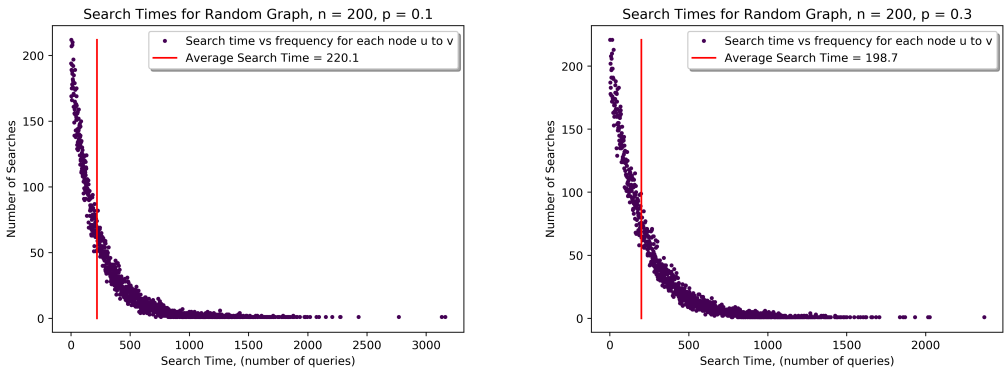

Figure 12: Random Graph Search, number of queries against how many searches took that long, for n = 200, p = 0.1, 0.3.

I tried many attempts of parameter tuning, reviewing results such as the queries per move, and of course the average search time across a random sample of all the nodes in the graph. However I consistently produced graphs similar to those in Figure(11) across a wide range of

parameter values (some parameter values produced much worse searches, but I struggled to find a combination that would get significantly better results than these). It led me to believe that with only this amount of information available when performing searches on random graphs it is not possible to achieve averages searches below a certain number of queries. Over the time I tested this, I couldn't seem to get average number of queries for a search of a random graph below the number of nodes in that graph, see Figure(11) (apart from the odd case with a search a small proportion of the graph).
In Figure(11) note that search time distributions are all very similar across the graphs of different sizes, suggesting this my be the case for larger values of n. In Figure(12) I fix the number of nodes and test two different probabilities of connection, for a more connected graph, p = 0.3 the search time is similar to what we have seen before, however for the less connected graph, p = 0.1, average search time has increased significantly by about 10%. The plots in Figure(12) appear more condensed, this is just because they were generated over a greater proportion of the graph, otherwise the distributions do not differ significantly from those in Figure(11). I have shown my random search algorithm can consistently produce search times appropriately equal to the number of nodes, n, in a random graph where n is of the order of 100 nodes, and p is sufficiently high (greater than approximately 0.1).

3.3 Searching Ring Group Graphs

Although I did not manage to find any reasonable heuristic for searching Random Graphs, Ring Group Graphs with p > q have the property that they are more likely to be connected to neighbouring groups. With the knowledge of which group each new neighbour is in we can use a metric (of how far in groups each neighbour is from the target groups) to judge which neighbour to move to. This will on average improve search time as once we are in one of the target groups each node is more likely to be connected to the target one.

3.4 Searching Algorithm

I made an algorithm with the following structure: curr, targ = (current_node_id, current_group), (target_node_id, target_group)
target_groups list of 3 groups, the group the node is in, and its adjacent groups.
time, curr_node = function search_for_target_group(curr, targ, target_groups, graph, m, k, p, q):
curr_node is a node in target groups, if curr_node == targ: done, else: extract the problem as a random graph, consisting of the current group and target group. Then perform random search on the extracted graph.
search_for_target_group views each neighbour one by one and makes a judgement to move to a group, more likely if close to target groups. The function only returns once it has reached a node in a target group.

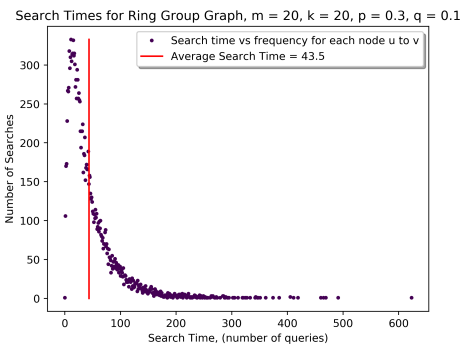

Figure 13: Ring Group Graph Searching.

Using the algorithm on a Ring Group Graph of 20 groups and 20 nodes per group (n = 400) achieved far shorter search times that for a random graph of that size, see Figure(13), with an average search time of 43.5. The shape of the distribution is similar aside from the region for search times below about 30 queries. This distribution looks very much like an inverse gaussian distribution.

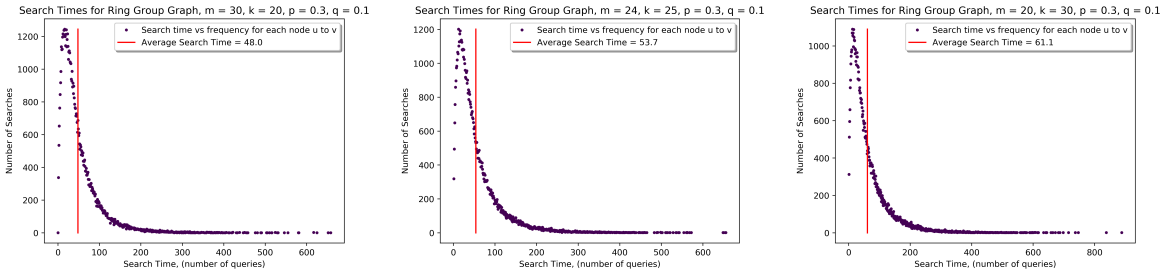

Figure 14: Ring Group Graph Searching.

To investigate the distribution as m and k change I produced the plots in Figure(14) (for 3 values of m and k, with mk = 600), clearly the distributions are very similar with average search time increasing as group size, k increases (approximately proportionally). This makes sense as the algorithm extracts a subgraph during the search treated as a random graph, consisting of the target group and possibly a neighbouring group. These extracted random graphs are obviously smaller than those I have looked at in the previous section, but are searched in the same way, with search time approximately proportional to n (in this case k).

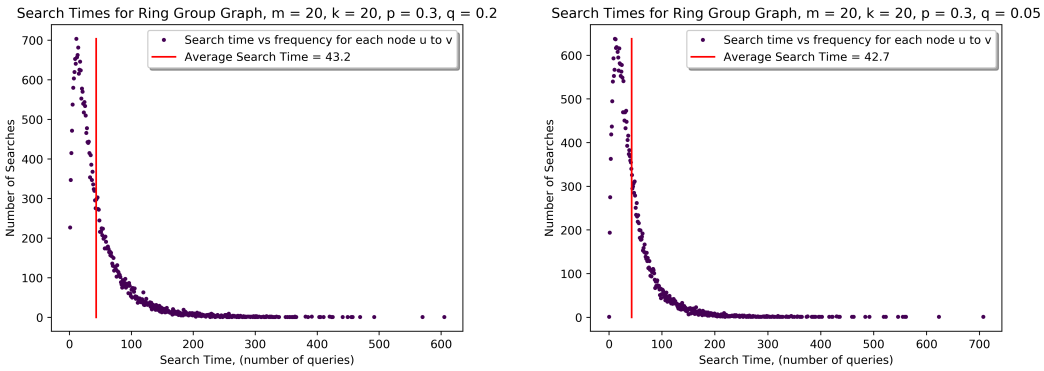

Figure 15: Random Graph Search, number of queries against how many searches took that long, for n = 200, p = 0.1, 0.3.

Investigating the effect of changing the parameter q, I produced the plots shown in Figure(15) these show negligible differences from Figure(13). This implies that the part of the search corresponding to searching for a target group (where q might be useful in connecting the graph) is relatively short compared with the remaining search over the extracted random graph.

Finally, when investigating the effect of parameter p on searching Ring Group Graphs I produced the plots in Figure(16). They show that for p = 0.4, the distribution and average search time is slightly shorter, but otherwise insignificantly different to Figure(13). When p = 0.2 however the average search time jumps up significantly and the distribution changes, producing some rally long searches with the longest lasting almost 4000 queries! Compared with almost 500 for p = 0.4. This will most likely be due to the target groups being much less connected.

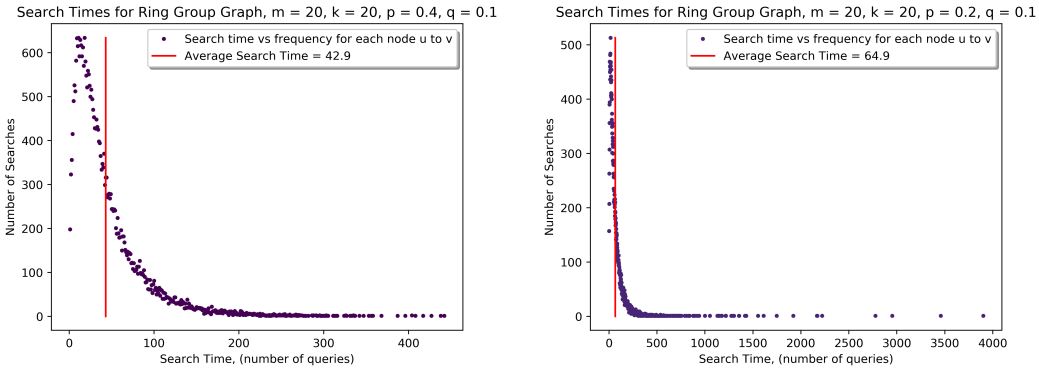

Figure 16: Random Graph Search, number of queries against how many searches took that long, for n = 200, p = 0.1, 0.3.

For the plot in Figure(13) with p = 0.2, my algorithm extracted random graphs that had nodes that had no neighbours in the extracted graph, producing disconnected graphs, to overcome this I allowed it to extract all of the target groups to perform random search on, this will also contribute to the increased search time. (Note if you run the code for 'q3_ring_search.py' when using small values of p, comment line 96 and uncomment line 97, to make the change mentioned in the previous sentence).

References

[1] Plotly network plotting. https://plot.ly/
[2] Networkx, maximum independent set algorithm. https://networkx.github.io/