Network Analysis Assignment

Jack Leyland

Computer Science L3 student

1 Question 1

1.1 Degree Distribution in Ring Group Graphs

To investigate the degree distribution of ring group graphs, we maintain a dictionary which stores the frequency of each discrete degree value and plot the results to visualise any trends that there might be.

Firstly, I took some arbitrary values of m, k, p and q to get an idea as to what kind of trends we might expect in the degree distribution of ring group graphs.

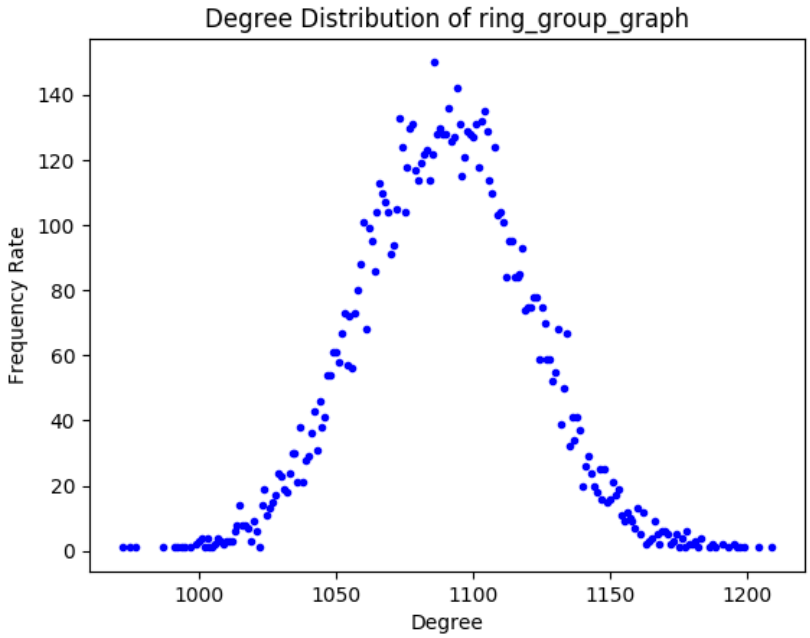

FIGURE 1: $m = 100, k = 100, p = 0.4, q = 0.1$

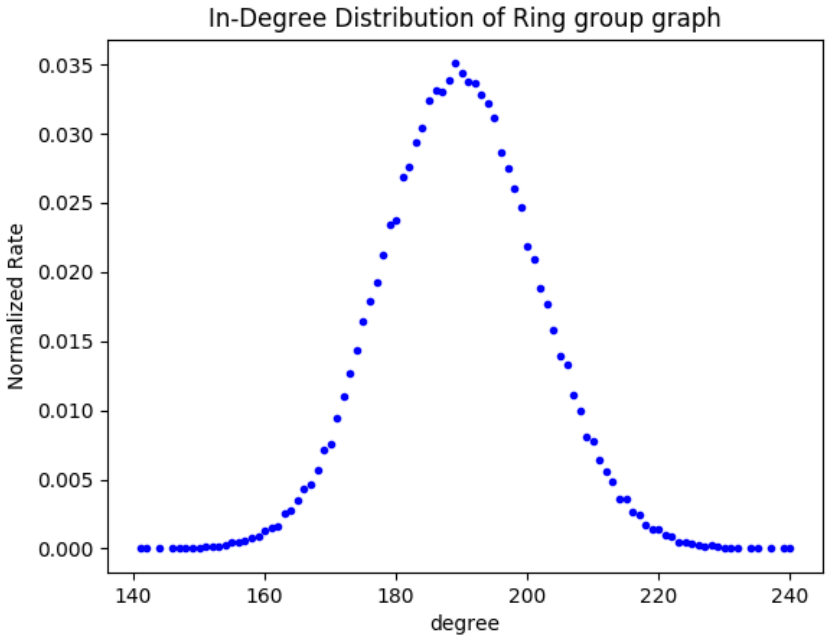

FIGURE 2: For a more reliable graph, I created 100 ring group graphs with the same m,k,p,q and averaged frequencies across the 100 trials. Note, different parameters used to in Figure 1

As you can see in Figures 1 and 2, we get a normal distribution bell-curve. We might expect this due to the element of randomness in the graph. Each node is considered in turn, and compared with a fixed number of other nodes (namely, $3k-1$ nodes) with probability p of creating an edge, and a fixed number of other nodes $(m*k-3k)$ with probability q of creating an edge.

From this, we can conclude the following:

$$E[\#edges] = p(3k-1) + q(k(m-3)) \tag{1}$$

Therefore, although we have 2 probabilities p and q, they are references a fixed number of times and so it is no surprise that random graphs show a degree-distribution of the same shape.

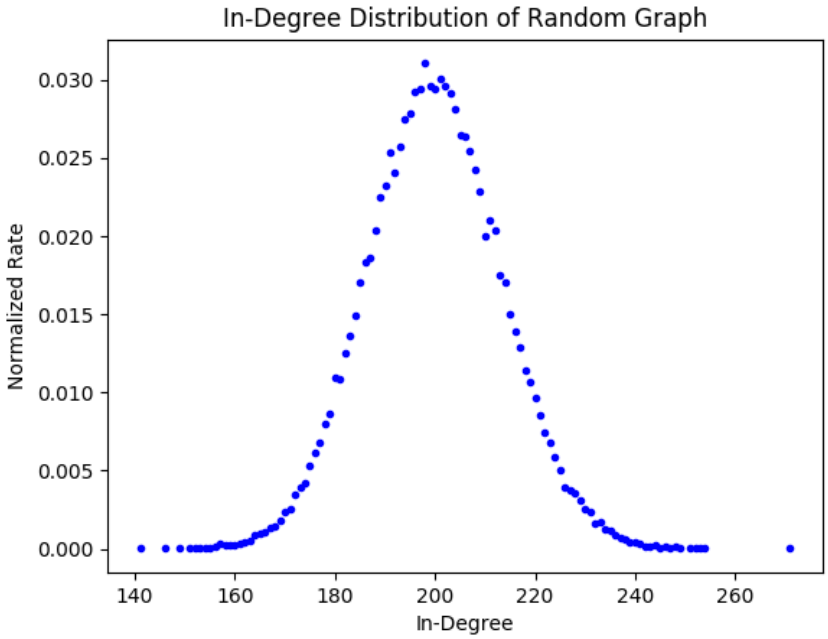

FIGURE 3: Random graph degree distribution with n=2000 and p=0.1

Finally, to get an overall idea of how the degree distribution changes as p and q vary (with $p > q, p + q = 0.5$), I plotted a set of degree distributions on the same graph. I kept a fixed m and k.

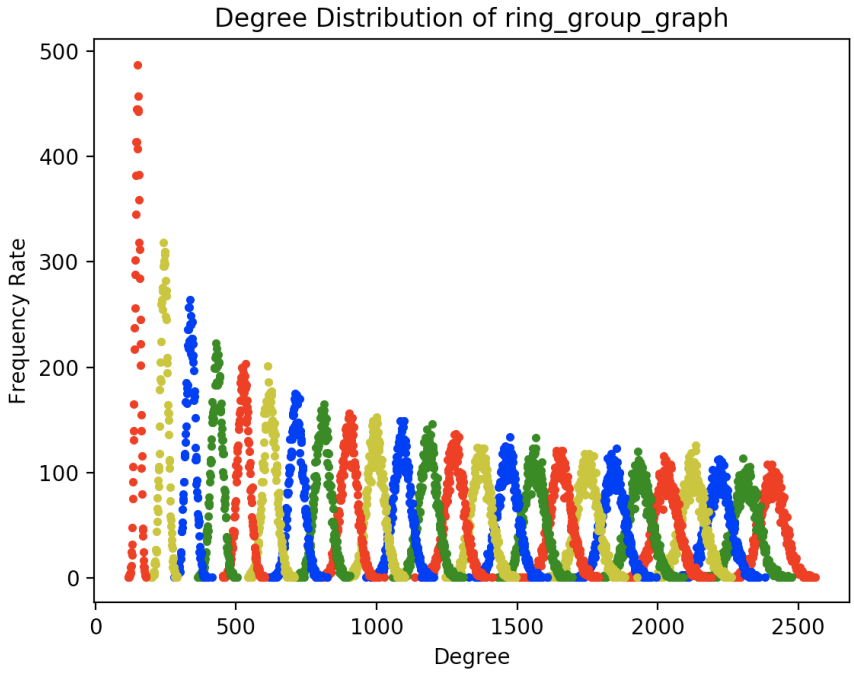

FIGURE 4: Degree distribution of ring group graphs as p and q vary ($m = 100, k = 100$). Each coloured bell-curve represents one graph. The leftmost has p=0.50,q=0.0. The rightmost has p=0.26,q=0.24

Since each of these curves represents a graph with a fixed number of nodes, we can directly compare the resulting curves. Where p is high (0.5) and q is low (0.0), we get a tall and narrow curve, the degrees are quite low and don't vary much. Where p is much lower (0.26) and q is higher (0.24), we get a much different bell curve. It is much wider and shorter. As p decreases and q increases, the degree distribution becomes more varied and the frequency of the mean degree is decreasing - the rate of this change decreasing.

Values of m, k, p and q values to investigate We want $p > q, p + q = 0.5$. This gives a range of p values up to 0.5. I felt that it was appropriate to investigate every p value between 0.5 and 0.26 (and q between 0.0 to 0.24) with 0.01 intervals. This creates 25 different graph structures (for fixed m and k), and it is clear that these values were appropriate in displaying a data trend. The bell curve transformation is made very clear in Figure 4.

m and k determine the number of groups we have and the size of these groups respectively. I predicted that lower values of m and k would give exactly the same trend as in Figure 4, just with lower degree values and lower frequencies, as we have less nodes pairs being considered when creating edges. This was confirmed by the following graph.

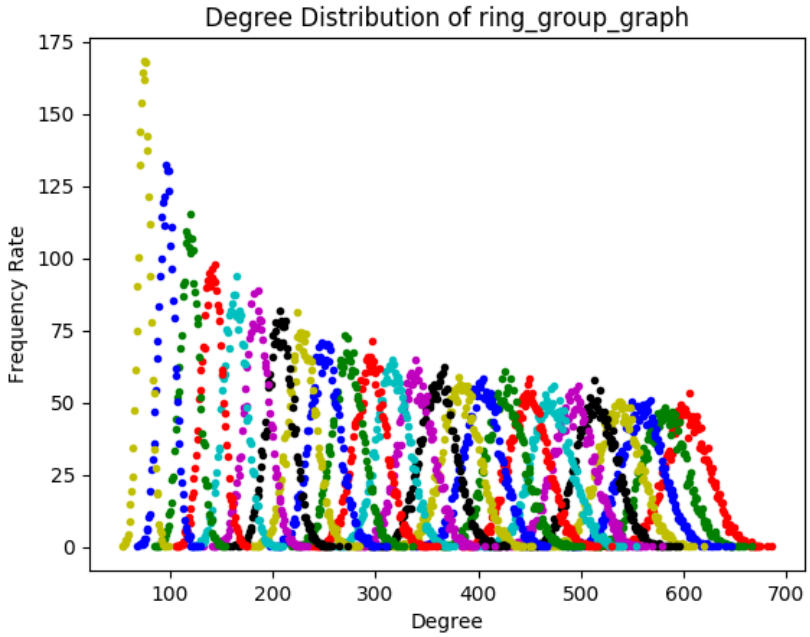

FIGURE 5: Degree distribution of ring group graphs as p and q vary ($m = 50, k = 50$). Each coloured bell-curve represents one graph. The leftmost has p=0.50,q=0.0. The rightmost has p=0.26,q=0.24

The plots have less clarity since there are less nodes (2,500 compared to 10,000), so the bell curves are less complete. However, we can see that the pattern remains the same. As p decreases (and q increases), the bell curves have a higher mean degree, and a much more varied degree which leads to lower frequencies of these. The rate at which this change occurs decreases.

The trend would be expected to change as m and k differ relatively to each other, however! A reminder of an earlier comment:

'Each node is considered in turn, and compared with a fixed number of other nodes (namely, $3k - 1$ nodes) with probability p of creating an edge, and a fixed number of other nodes ($m * k - 3k$) with probability q of creating an edge.'

With some rearranging, this means that if $m > 6 - 1/k$, there are more q-comparisons. If not, there are more p-comparisons. So, as m becomes larger, we have more groups and so q will become a more dominant probability (a higher proportion of the node pairs considered will be from different groups), whereas as k increases, the groups become larger and so p will become more dominant eventually. Also note, if m and k are the same, there will be many more q-comparisons.

Therefore, I should also consider both of these cases. For example, $m = 250$, $k = 10$ will have more q-comparisons, and $m = 5$, $k = 500$ will have more p-comparisons. The resulting graphs will be comparable too since these will both have 2500 nodes.

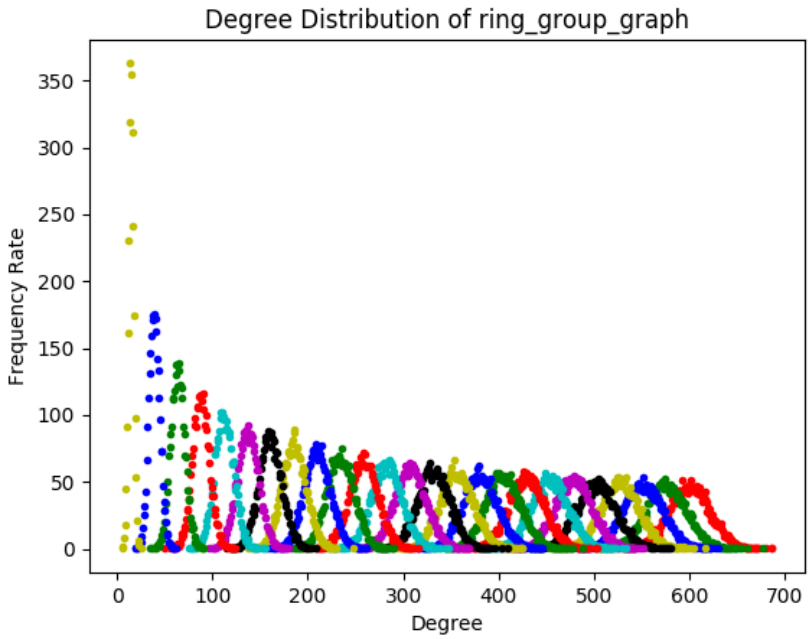

FIGURE 6: Degree distribution of ring group graphs as p and q vary ($m = 250, k = 10$). Each coloured bell-curve represents one graph. The leftmost has p=0.50,q=0.0. The rightmost has p=0.26,q=0.24

For the former case ($m = 250, k = 10$), we get a very similar degree distribution pattern to those already seen. We get bell-curves for each graph, and as p decreases we get higher and more varied degree values, with a lower frequency at the peak. The only difference here is that the bell-curves become smaller and wider more suddenly.

The left-most bell curve is extremely tall and thin where p=0.50 and q=0.00. This would make sense, as while q=0.0, we are only adding edges within groups. Since we have groups of size 10, each node is only being compared with 29 others, each with probability 0.5. This explains the narrow bell, possible degree values lie within the small range of 0 and 29.

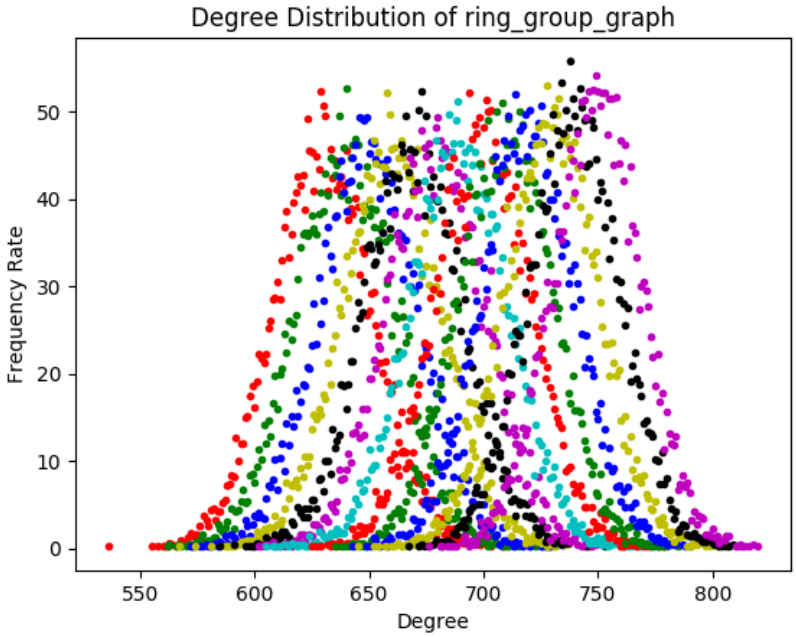

FIGURE 7: Degree distribution of ring group graphs as p and q vary ($m = 5, k = 500$). Each coloured bell-curve represents one graph. The leftmost has p=0.26,q=0.24. The rightmost has p=0.50,q=0.0. Note, p-intervals are 0.02 here (instead of 0.01) for graph clarity.

When $m = 5, k = 500$, we get a different pattern! We still have a bell-curve for each of the different graphs created, but actually the LEFT-most bell-curve here is where p is low, whereas previously this has been where p is high. So now, increasing p and decreasing q is actually increasing the mean degree in the degree distribution of the graphs.

This is because we have set m and k such that there are more in-group node pairs than out-of-group node pairs. So, when p increases, we are getting an increasing number of edges and thus an increasing mean degree.

However, the bell-curves shapes don't change much. The max degree value frequency is still the same.

1.2 Graph structure as p and q change

For fixed m and k, p and q varying does change the structure of the graph. From the earlier inequality, if m is greater than 6 then there will be more q-comparisons. Therefore, q often has a larger influence on the graph's structure. This is why we should make the value of q much smaller than p to obtain more of a ring structure as opposed to a completely random graph. In fact, if p and q

were equal, every pair of vertices would be compared with each other and so we would have an equivalent random graph.

Nonetheless, generally, if p is very high, we would expect there to be lots of edges within groups and between neighbouring groups and vice versa. If q is very high, we would expect a lot more interlinking of groups that aren't neighbouring, and vice versa. Since, if m is high enough, most groups will *not* be neighbouring to a given group, so if q is high we will have many more edges in the graph than if p is high. This is visualised in Figures 4 and 5.

1.3 Relationship between p and diameter

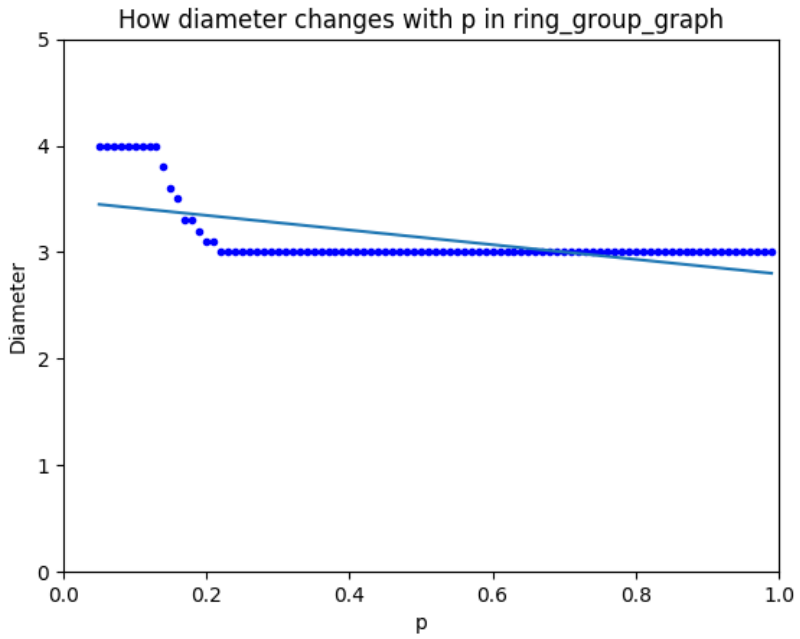

FIGURE 8: Graph showing diameter of ring group graphs with varying p, and fixed $m = 15, k = 15$ and $q = 0.05$

The diameter of a graph is the length of the longest path composed of unique vertices for any distinct vertex pair. In a ring-group graph, the diameter varies (slightly) based on our value of p. The values plotted are averaged over numerous trials, hence why there are some non-integer values.

As we can see, when p is extremely low we have a diameter of 4 in the graph. When p reaches a value of roughly 0.15, the diameter starts to drop between 3 and 4 and then once it reaches 3 when p=0.22, the diameter never changes from 3.

I theorise that this is because for any source-destination pair, we are able to use a q-edge between non-adjacent groups to 'home in' on the destination node,

and then a short succession of p-edges to finally reach it. Once p is high enough, this final step won't take as long as so diameter will become quite low.

When p=1, it will only require one step from any adjacent (or same) group to our destination node. So, since diameter never drops below 3, we must be limited finally by q being low (0.05).

2 Question 2

2.1 Coauthorship brilliance distribution

Firstly, let my method for calculating brilliance for a vertex v be noted. To do so, I created a sub-graph which consisted of the v's immediate neighbours and the edges between them, excluding v itself. Then, using python package NetworkX's maximum independent set method, I found the largest independent set in this sub-graph. This gives the largest set of vertices connected to v which are not adjacent to each other. Thus, the size of this independent set gives k, with v as the centre of a k star.

Below is the brilliance distribution of the coauthorship graph (Figure 9) and an average over numerous pa and ring group graphs (Figures 10 and 11).

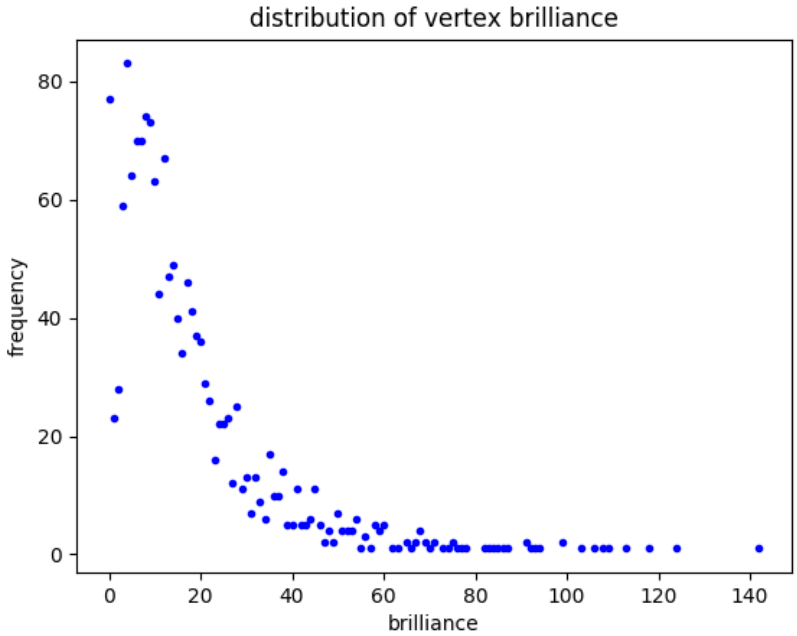

FIGURE 9: The brilliance of the coauthorship graph

Since this is a real-life graph, you can't say much about the pattern it shows in a mathematical sense. More appropriately, it makes sense to explain the given graph in context.

Other than a couple of outliers in the bottom-left of the graph, we have a decreasing curve. As brilliance increases, we generally get a lower frequency of authors with this brilliance.

So, why do authors generally have a low brilliance? Generally, I don't believe that an author will write many papers (perhaps < 20) in their life.

The size of the k-star (the brilliance of an author's vertex) is limited firstly, by how many papers the author has actually written - since the author needs to have co-written with each of these members of the k-star. Then, if either of these authors know each other (likely, if they are part of the same academic institution as the centre author), they could have co-authored together which further limits the brilliance of our author.

I believe that there would be less authors who have written more papers. Also, once you have more co-authors, it becomes increasingly unlikely that any of them are independent, thus limiting how big our k-star can get. This is reflected by the decreasing trend in our graph.

2.2 Ring group graphs brilliance distribution

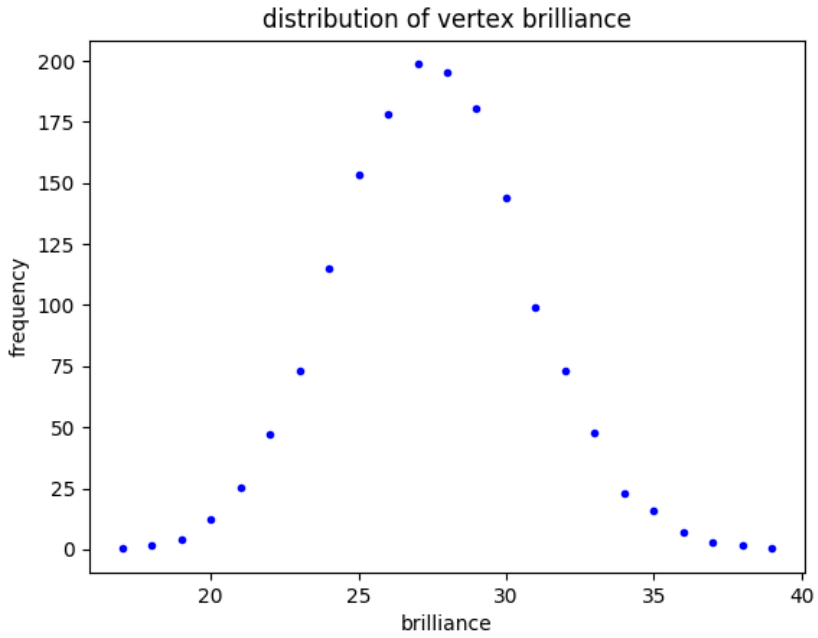

FIGURE 10: Ring group graph vertex brilliance distribution. Created with $m = 40, k = 40, p = 0.23, q = 0.02$. This gave a similar number of vertices and edges to the coauthorship graph.

Ring group graphs show a much more concrete pattern when it comes to their brilliance. Again, the same as the degree distribution, we have a bell-curve

(normal distribution) for ring group brilliance distribution. I would expect this, personally, since - as mentioned - each individual node is compared to the same number of other nodes with probability p, and the same number of other nodes with probability q.

Thus, the out-degree of every node will be **expected** to be the same. Then, when considering each node as the centre of a k-star, each of its neighbours will have the same **expected** out degree and so we would have a fixed **expected** brilliance for a vertex. Then, some vertices will have a greater brilliance, some will have a lower brilliance.

As a result, I think the normal distribution makes sense.

2.3 PA graphs brilliance distribution

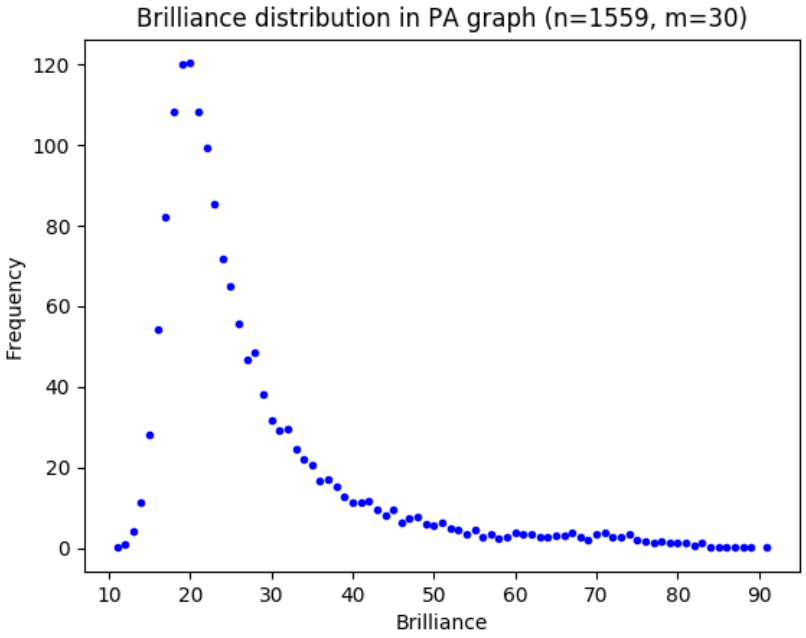

FIGURE 11: PA graph vertex brilliance distribution. Created with $n = 1559, m = 30$

The PA graph has a different distribution of vertex brilliance to both of the Figures 9 and 10. It appears to be a normal distribution but with quite a heavy positive skew.

The first m nodes will be created as a complete graph. So, when it comes to preferential attachment for the first few nodes after the initial m nodes, they are almost certain to connect to these first m nodes, but not each other. They will each end up with degree m.

Further nodes will be connected to the original m, but with an increasing chance be connecting to other nodes outside of the original m, since more and more of these are developing degree m.

So, our original m nodes will have a very high brilliance since initially, lots of nodes connected to them but not to each other. This will form a very large k star.

The nodes being created near the end are being attached to a much more random set of existing nodes as we have 1000 nodes with degree 30 or above. There are more connections between each other (and not to the original m nodes), so we will get smaller k-stars as neighbours are less likely to be independent of each other here. As more and more nodes are created, edges between nodes outside of the original m will become even more apparent. This explains the positive skew.

3 Question 3

Searching in graphs.

Please not that when talking about searching, it is always consistent with the definition of searching provided in the assignment. To ensure no 'cheating' when searching, I chose to randomise N(v) every time I encounter a vertex v.

Also, search time distributions are plotted for each algorithm discussed. Time is the search time. Frequency of a search time is the frequency of vertex pairs that average to this search time over a number of trials.

3.1 Searching in Random Graphs

Searching randomly. The most obvious (and simple) method for searching in a random graph is to just move about the graph randomly. I just moved to N(0) every single query to do so. Note, this works since I randomise N(v) when encountering v. Since the graph structure is entirely random, you would expect a normally distributed search time when averaging over numerous trials.

The rationale for this method is that you would expect a very consistent average search time. We have 100 vertices, each connected to each other with a fixed probability. So, each query is *completely* random, there is a 1/100 chance that a query is going to be the vertex you are looking for. Therefore, the expected number of queries is 100 with this method.

Since we only have local information, there is little information we can use to understand the structure of the random graph which would help with searching. I personally don't believe that it's possible to improve on simply searching randomly.

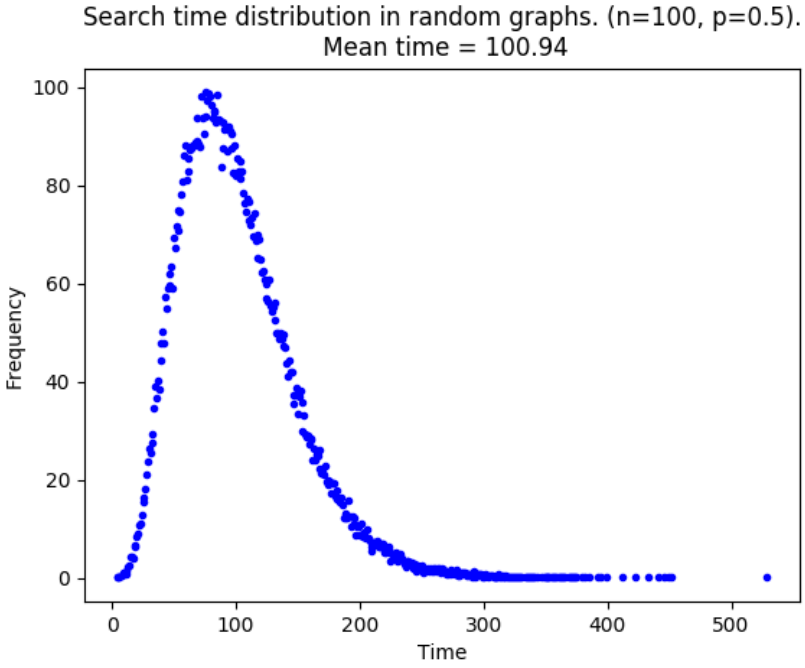

FIGURE 12: Searching a random graph ($n = 100, p = 0.5$) randomly.

Figure 12 shows the number of vertex pairs averaging to a range of search times over 5 trials, in 10 random graphs defined by $n = 100, p = 0.5$. We get a normal distribution with a positive skew, with the mean at 100 queries.

Since the queries are all made completely at random, the running times are fairly at random too with some extremely quick search times and some extremely long (500 queries). However, since we are averaging over 50 trials for each vertex pair, we get a normal distribution centering around the mean search time.

Exploring high-degree vertices more. Despite believing there are no better methods, I tried to find one.

My idea was to explore more neighbours of a vertex if it had high degree. For example, in our graph, the average degree of a neighbour is 50. So, if a vertex has degree higher than this, there is already a greater than 50% chance that the target vertex lies in this list. I decided to search through the full neighbour list if a vertex's degree was over 55.

Conversely, searching through this list would take more than 55 queries, and we may not even find our target vertex. In this case, if we then happen to encounter this vertex again, we have to make an additional 55+ queries with no result. Searching randomly doesn't suffer from this repeating query effect.

Therefore, this method showed even worse results than randomly searching. This had an average search time of 568 queries.

Interestingly, when searching through the whole list of neighbours for only vertices with degree less than 45, we got better results - namely 330 queries on average. Still, not nearly as good as searching randomly.

We cannot do better than randomly searching I strongly believe that we cannot do better than randomly searching random graphs, when using this definition for searching.

Every vertex is equally as likely to be connected to every other vertex, and so there is no notion of closeness to the target vertex, as is used in ring group graph searching to great effect (see below). We also cannot store a list of visited vertices so that we don't repeatedly move to the same vertex.

These facts, combined with the fact that we can only use local information at each vertex means that we cannot understand the overall structure of the graph, or use any heuristics to streamline our search and increase the chances of encountering the target vertex at any point.

3.2 Searching in Ring Group graphs

Searching randomly. Once again, the list of neighbours is shuffled randomly whenever I move to a vertex, to ensure that I can't 'cheat'.

I believe we can do better than searching randomly in the case of Ring Group graphs. Nonetheless, I investigated the performance of this to form a baseline for comparisons of other methods.

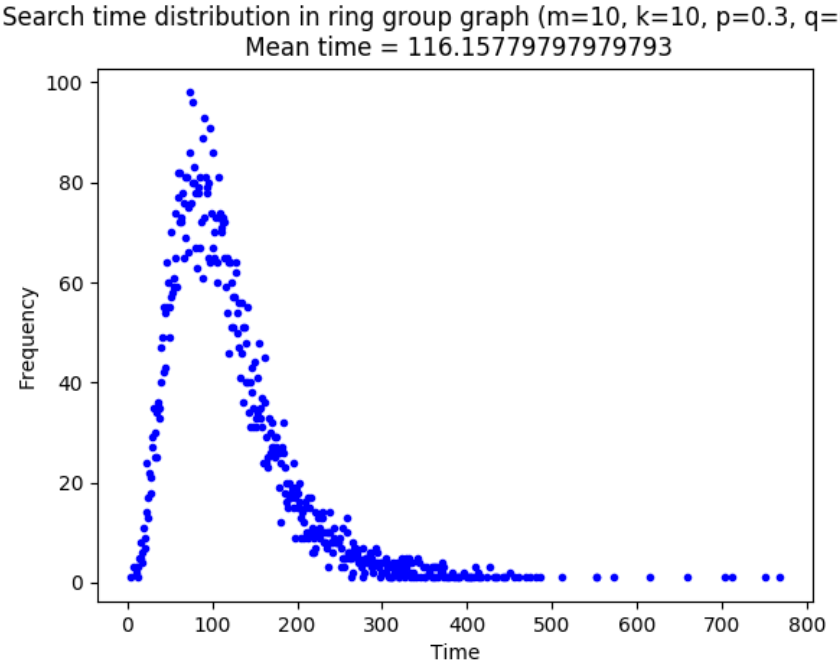

FIGURE 13: Searching a ring group graph ($m = 10, k = 10, p = 0.3, q = 0.05$) randomly.

The results I got were reasonable, with a mean number of 116 queries over 5 trials for each distinct pair of vertices in the graph. However, in ring group graphs with the use of group numbers, m and k, I thought of a better method.

'Homing in' on the target group. We have access to the group of our vertices. We can then use this group information to create a kind of 'homing in' effect.

Given the parameters of this graph ($m = 10, k = 10, p = 0.3, q = 0.05$), we have more p-edges than q-edges. From this, I decided a good method would be to get into the same (or adjacent) groups of the target vertex as soon as possible. Then, move around these groups until we find the target.

My algorithm is as follows:
The results from this are fantastic.

Algorithm 1 Searching Ring Group graphs

```
function RING GROUP SEARCH(source, target, m, k)
    Calculate target group using target vertex id and k
    Make source our current_vertex

    while current_vertex ≠ target do
        w ← Random Neighbour of Current Vertex

        if w is our target vertex then
            Move to w

        else if w is in adjacent group to target group then
            if We are not already in target group or adjacent to target group then
                Move to w

        else if w is in the target group then
            if We are not already in target group or adjacent to target group then
                Move to w

        else if w is closer to the target group than our current group then
            Move to w

        if No neighbours of current_vertex unqueried then
            Move to a random neighbour
```

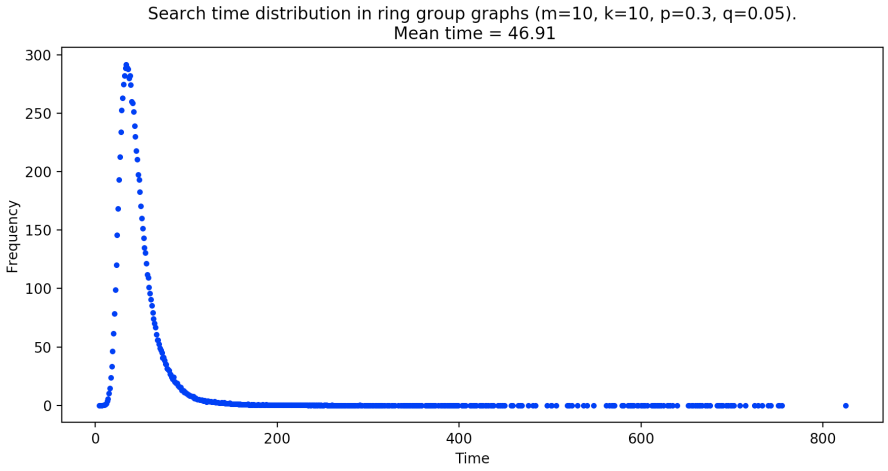

FIGURE 13: Searching a ring group graph ($m = 10, k = 10, p = 0.3, q = 0.05$) by 'homing in' on the target group.

As expected, we get much better results using this method than by searching randomly. We query - on average over 100 trials for each vertex - 46.91 vertices until we find the target vertex.

We get a positively skewed normal distribution of the search times, centered around our mean. Some searches take less time, and some take much longer!

Below, I detail some of the problems I had along the way, and solutions for them.

One problem I initially had was that I was spending a lot of time in the source vertex's group. This was because, I was only considering moving to vertices if they were in an adjacent group or the same group as the target vertex, but given our parameters, most edges are to adjacent groups or within the same group. This was where I introduced the notion of 'distance' to the target group, which I define as the number of groups around the ring from the current group to the target group. If I see a vertex that reduces this 'distance' to the target group, I move to the vertex and explore from there. This resulted in quickly moving around the ring towards the desired group, right from the start of searching.

I thought maybe it was a good idea to keep record of the closest vertex to the target, but only moving to it the end of vertex exploration if there are no neighbours in the target group or adjacent to the target group. However, this often meant searching through all of source's neighbours and vastly increasing search times. This provided the rationale for moving to 'closer' vertices to the target immediately once they are found.

Furthermore, I initially wrote the method to home in specifically on the target group and then stay in the target group to be near to the target vertex. This gave good running times, but not great. I realised, however, that the 2 adjacent groups are equally as useful for reaching the target vertex, as these vertices are also connected to the target vertex with probability p. So, I altered the algorithm to aim for these 3 groups as quickly as possible, and stay within these 3 groups once found. This gave much improved running times. I believe this is because during execution, we typically take p-edges around adjacent groups to reach our target group. Thus, we reach an adjacent group more quickly, meaning that we have an opportunity to find the target vertex more quickly.

References

1. Lecture notes/slides/exercises.
2. NetworkX package. (https://networkx.github.io/)
   Used algorithms for creating PA graphs (barabisi albert graphs with some modifications), diameter, maximum independent set (for calculating brilliance)