mdsw22

Network Analysis: Assignment Report

Q1: Investigate the degree distribution of Ring Group Graphs for $p + q = 0.5, p > q$

For every measurement in this question, I have run every test repeatedly over **5** repeated trials and taken an average. This is to remove any random error as the algorithms are non-deterministic. The graphs have also been normalised so that they can more easily be compared.

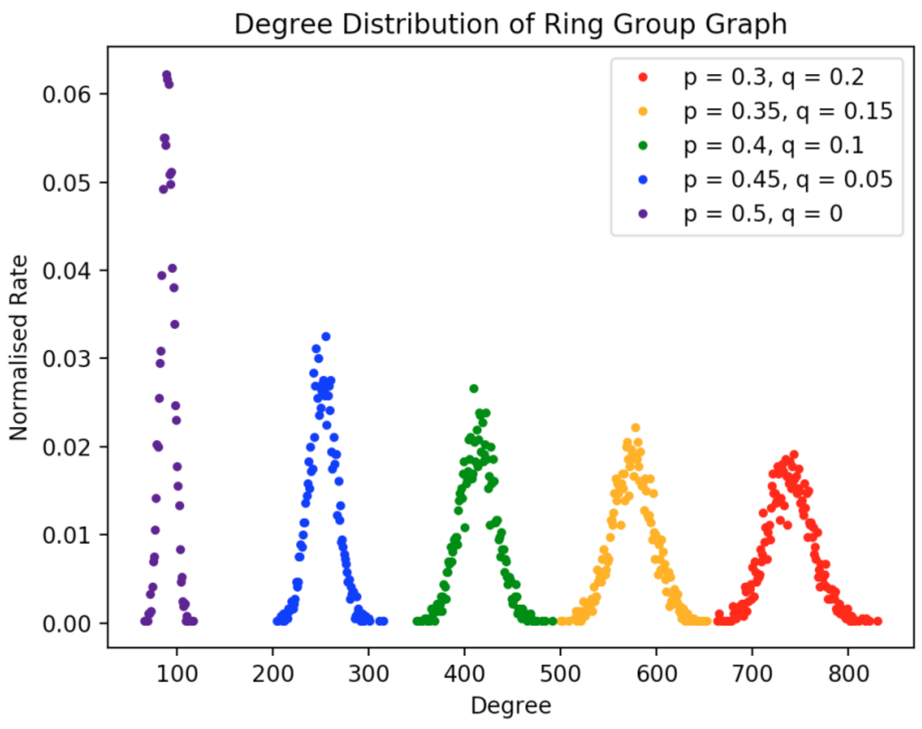
For this question, I have done three test cases:

i) $m = k$
ii) $m >> k$
iii) $m << k$

i) **m = 60, k = 60, Number of Nodes = 3600**

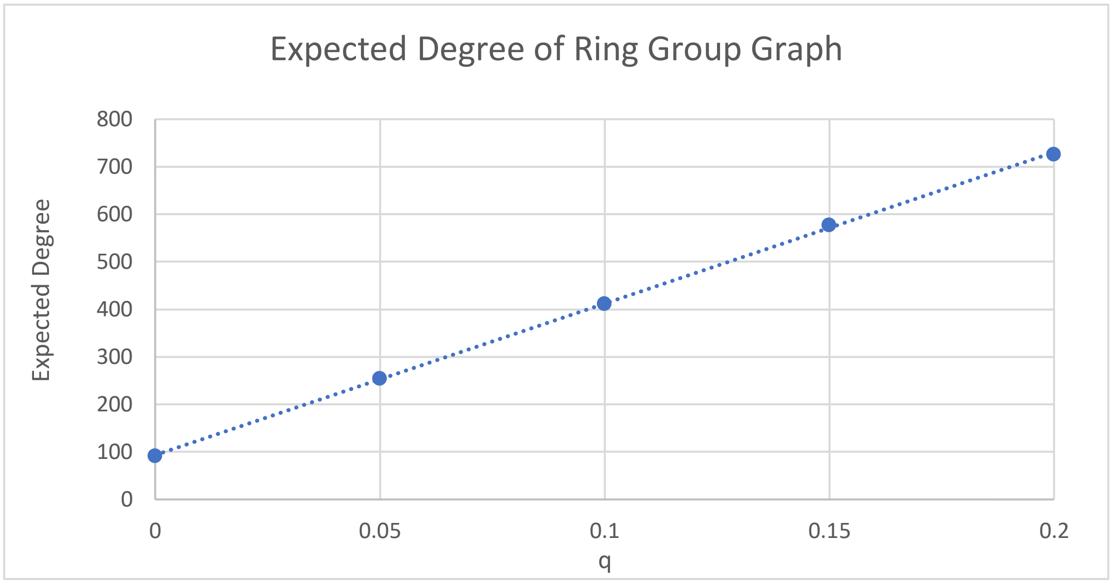
Graph #	p	q	Number of edges	Expected Degree	Degree Range	Max Normalised Rate
1	0.5	0	162,060	91	44	0.07
2	0.45	0.05	452,024	254	97	0.032
3	0.4	0.1	744,944	412	131	0.025
4	0.35	0.15	1,035,681	577	143	0.022
5	0.3	0.2	1,327,999	726	175	0.020

Graph 1 (m = 60, k = 60)



The results above show that, as the ring group graphs are essentially random graphs, the degree distribution generally follows a Gaussian distribution. As p decreases and q increases ($p + q = 0.5$), the number of edges increase and therefore the expected degree of the Ring Group Graph increases. As q increases and p decreases, the number of edges increases at a linear rate and as a result, the expected degree also increases at a linear rate, shown in Graph 2 below.

Graph 2

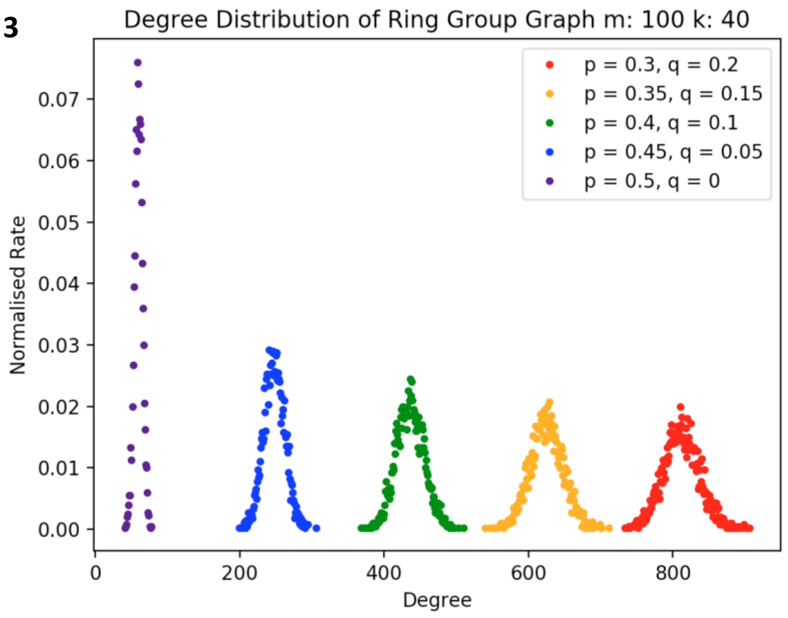


Moreover, as p decreases and q increases, the range of degrees increases and as such, the normalised rate of degrees decreases accordingly. This is shown clearly in Graph 1 as the peak of each plot decreases and the range of each plot increases.

ii) **m = 100, k = 40, Number of Nodes = 4000**

Graph #	p	q	Number of edges	Expected Degree	Degree Range	Max Normalised Rate
1	0.5	0	118,917	60	38	0.075
2	0.45	0.05	498, 893	249	100	0.030
3	0.4	0.1	871, 232	434	131	0.025
4	0.35	0.15	1,246,777	624	157	0.022
5	0.3	0.2	1,622,903	813	185	0.02

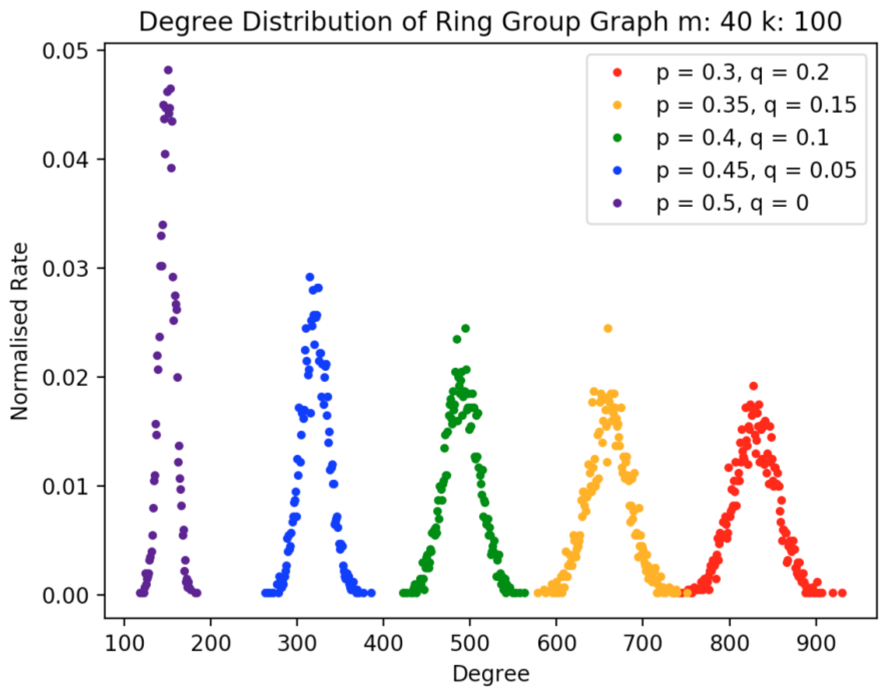
Graph 3



iii) m = 40, k = 100, Number of Nodes = 4000

Graph #	p	q	Number of edges	Expected Degree	Degree Range
1	0.5	0	298,893	151	65
2	0.45	0.05	639,335	371	106
3	0.4	0.1	979,383	494	138
4	0.35	0.15	1,318,715	653	161
5	0.3	0.2	1,661,115	834	198

Graph 4

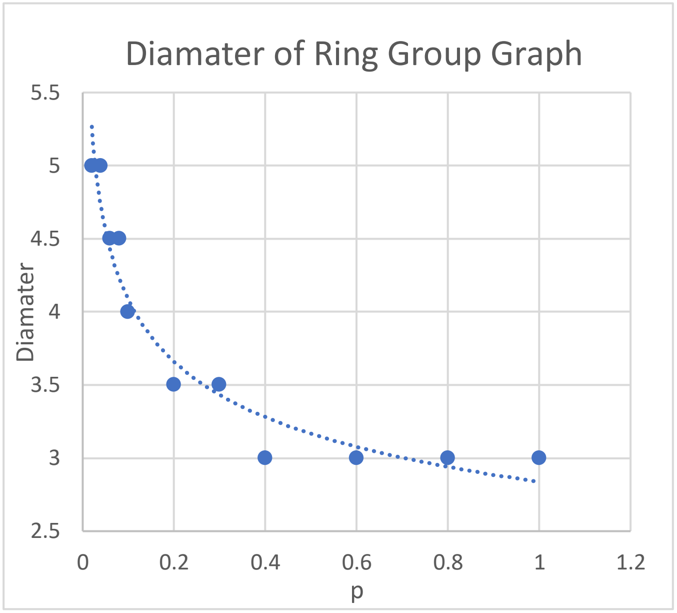


The results above, illustrated in Graph 3 and Graph 4, show that the structure does **not** change when m and k are altered.

Diameter (m = 30, k = 30, q = 0.01) Averaged over 10 repeated trials for each p. (Note, diameter is never non-integer however the values shown were averages and rounded to the nearest .5) for plotting.

P	Diameter
0.03	5
0.04	5
0.06	4.5
0.08	4.5
0.1	4
0.2	3.5
0.3	3.5
0.4	3
0.6	3
0.8	3
1.0	3

Graph 5

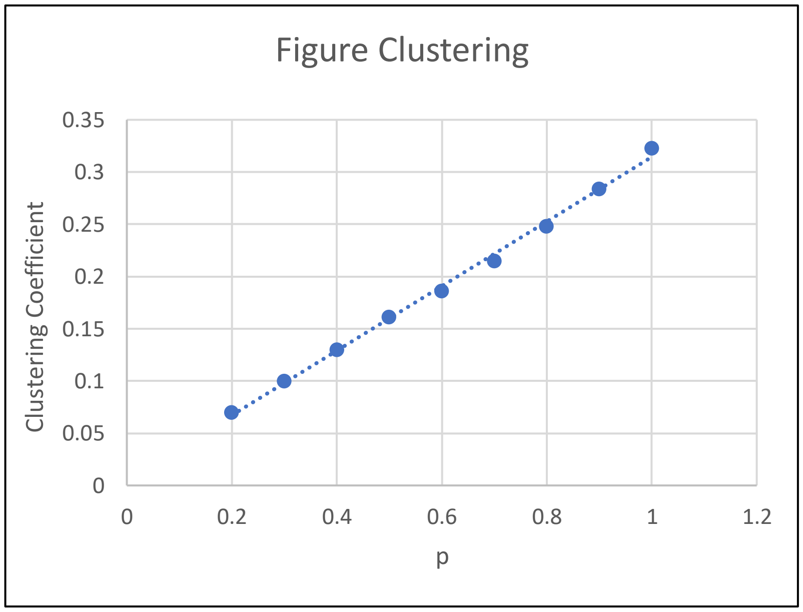


For a given fixed q, the dimeter of the graph decreases. Theoretically, when p is 0, the graph diameter would be infinity due to the fact that the graph is not fully connected. p and q were made large enough so that the created Ring Group Graph was fully connected however small enough so there was a visible change in diameter. The diameter decreases rapidly as p increases to 0.4 where it reaches its 'minimum' diameter (diameter does not go any lower as p goes to 1.0).

Clustering (m = 30 k = 30, q = 0.1)
10 repeated trails for each p

P	Clustering
0.2	0.070
0.3	0.101
0.4	0.130
0.5	0.161
0.6	0.186
0.7	0.215
0.8	0.248
0.9	0.284
1.0	0.323

Graph 6



For a given fixed q, the clustering of the graph increases linearly as p increases. As q is non-changing, the only parameter that is affecting the value of the clustering coefficient is p. As there are m groups of size k, the expected number of vertices a given vertex v is attached to is equal to $3kp$ (ignoring the constant effect of q). This is because a node is connected to nodes in its own group and its two adjacent groups with probability p.

If we take m = 3 and k = 10, the expected number of neighbours = $3kp$ = $3 * 10 * p$ = 3p.

p	Expected Number of Neighbours
0.1	3
0.2	6
0.3	9
0.4	12

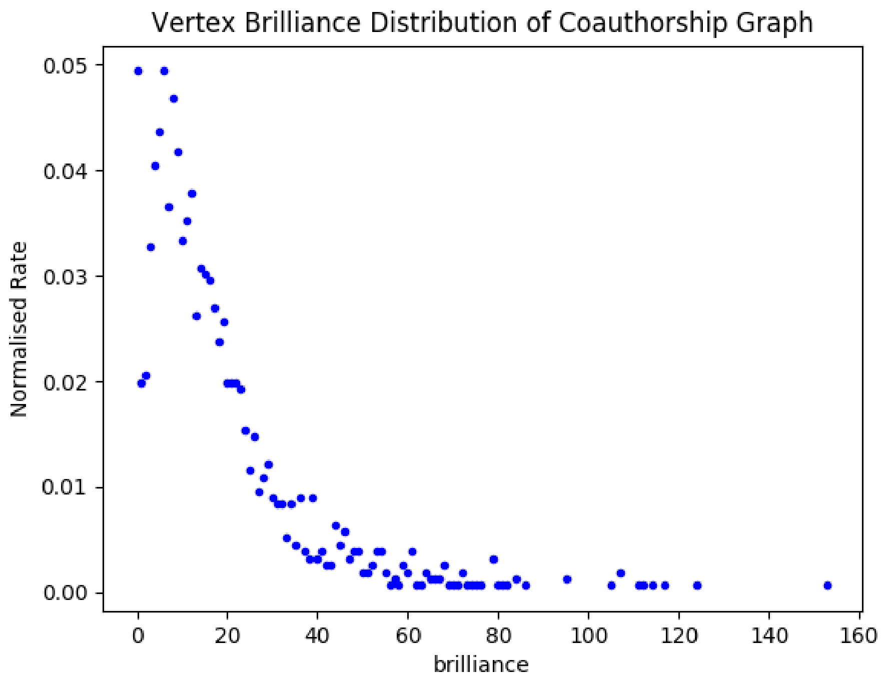
This shows that the expected number of neighbours and thus the average clustering coefficient increase linearly with an increase in p. This hypothesis is validated by our results above and illustrated in Graph 6.

Q2

For each of the plots in Question 2 I have normalised by dividing the brilliance distribution by the maximum possible brilliance.

"Coauthorship.txt" network graph

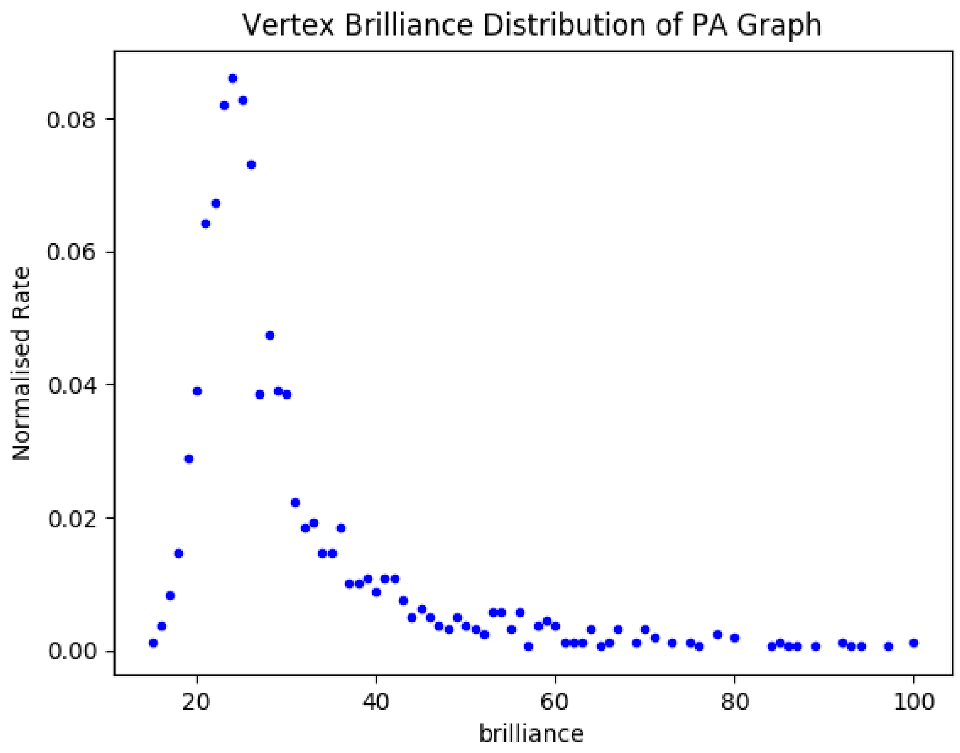
- 1559 nodes
- 40016 vertices.



This graph illustrates the fact that a large number of vertices in the network have a low vertex brilliance and as the brilliance increases, the proportion of nodes that obtain said brilliance decreases rapidly. This make sense when compared to real-world scientific co-authorship; a large number of authors work with a small number of other distinct authors (who do not work with each) and thus have a small brilliance. In reality, if two authors were to work together, they would be writing on a given topic and so are likely to collaborate with other authors on that same topic. As such the brilliance of most authors is likely to be low. On the other hand, there are a small number of authors with high vertex brilliance meaning that they are working with a large number of authors who are themselves not working with each other- **very rare**!

Preferential attachment network graph (n = 1559 and m = 27)

- 1559 nodes
- 41364 edges.

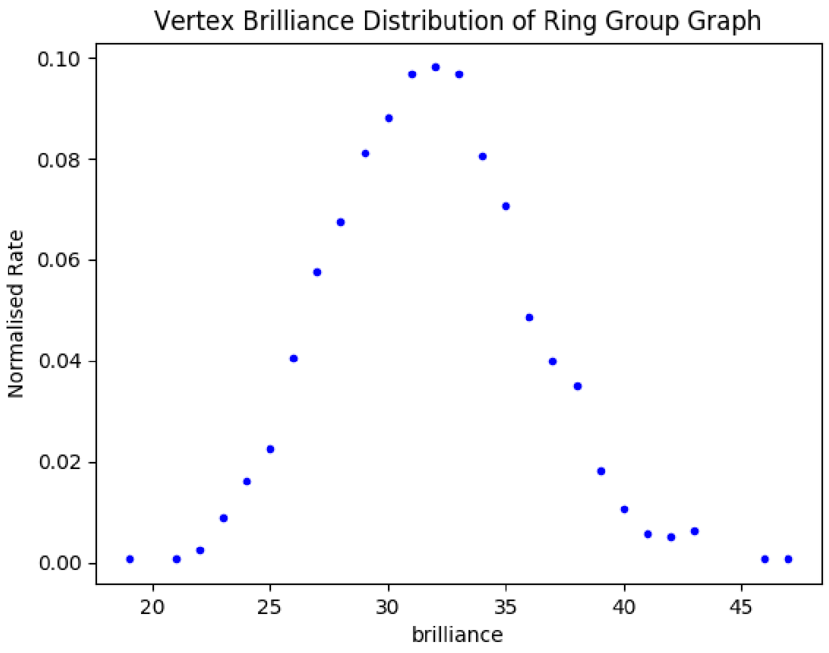


In preferential attachment graphs, edges are added from vertex v to existing vertex u in proportion to the degree of v. Each new node is connected to $\mathrm{m} > \mathrm{m}_0$ nodes with a probability that is proportional to the number of links that the existing nodes already have. When the other vertices are added, the first m nodes are more likely to have high brilliance because of the nature of preferential attachment. The other nodes roughly follow a Gaussian distribution which results in a fat-tailed distribution- very similar to the co-authorship graph.

Ring Group Graph

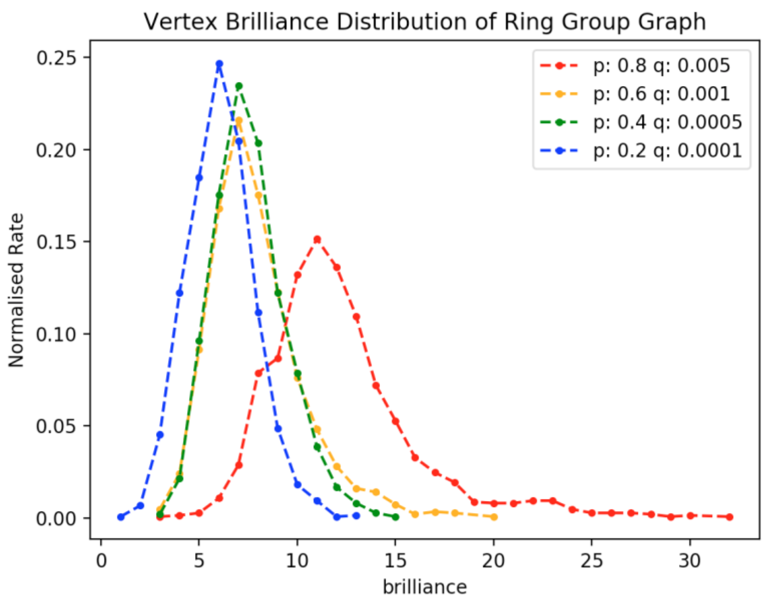
Test Case 1 (m= 156 k = 10, p = 0.034, q = 0.034) $(p \approx q)$

- 1560 nodes
- 40,352 edges
- Expected brilliance = 32.



This is an interesting test case as $p \approx q$ and so the graph is effectively a random graph as distinct nodes u and v are equally as likely to be attached to each other, irrespective of whether they are in the same, adjacent or non-adjacent groups. As a result, the vertex brilliance follows a Gaussian distribution.

Test Case 2: (m= 156 k = 10) $(p \gg q)$
This gives a well-defined Ring Group Graph Structure.



Note: Points have been joined via a dashed linen to emphasise the pattern in a clustered figure.

For Ring Group Graphs, they are essentially random graphs with additional grouping structure. Each pair of distinct vertices u and v are connected with probability p if they are in the same group or adjacent groups $|l_u - l_v| = 1 \; mod \; m$, and q if not. In terms of vertex brilliances, vertices tend to have a low brilliance, but the brilliance shits and becomes more Gaussian as the probability p decreases. This is to be expected as fewer neighbour will subsequently connected due to the properties of a Ring Group Graph as described above.

Q3- Search

Random Graph

For random graphs, as they are 'unsearchable', the search strategy doesn't make any difference as there is no network structure to base the search on. As such, the only consideration when designing a search method is to minimise the total number of queries made. There were three obvious search strategies:

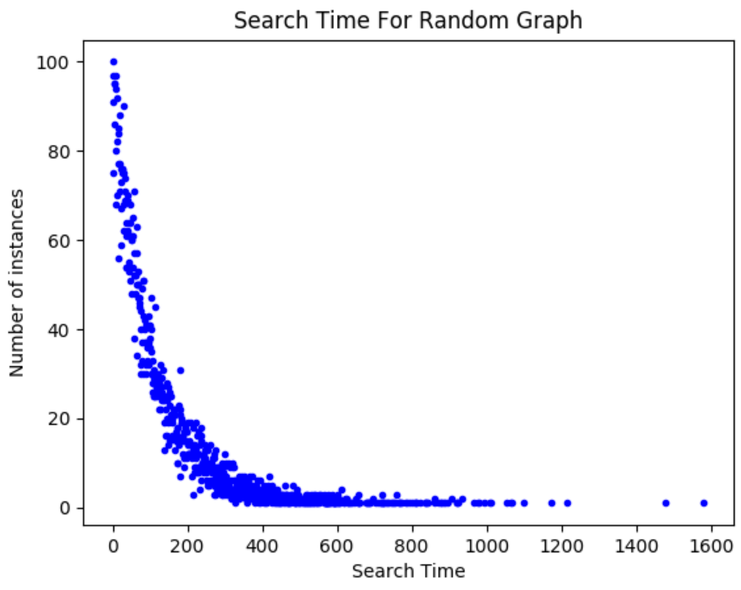
1. When the search algorithm moves to a new node u, it considers the array $N(u)$ (IDs of neighbours) and then randomly selects a vertex v to move to. This is repeated until it reaches the target vertex. The algorithm executes one query for every node it moves to so the number of queries is equal to the number of edges traversed in the search.
2. When the search algorithm moves to a new node u, it considers the array $N(u)$ (IDs of neighbours) and checks each id individually comparing it to the target vertex. If the target vertex t is not found as a neighbour, the search strategy randomly selects a neighbouring vertex v to move to. For every vertex v the search strategy moves to, it executes max n queries where n is equal to the degree of v.

The search strategies are similar in the sense that they both consider only neighbouring nodes however they differ in how they approach querying. The first strategy goes on the premise that the number of queries should be minimised at each step however it is naïve in the sense that it does not look at its neighbours (the target vertex may be connected to a given node u which the search strategy is at but it may completely ignore this and pick another neighbouring node v at random. On the other hand, the second search strategy follows the intuition that a given node u should use all the information available at each step of the algorithm (i.e. the IDs of its neighbours) at the expense of an increased search cost time through a greater number of queries per step. Despite these fundamental differences, due the random nature of the graphs, these search methods are very similar, illustrated in the figures below.

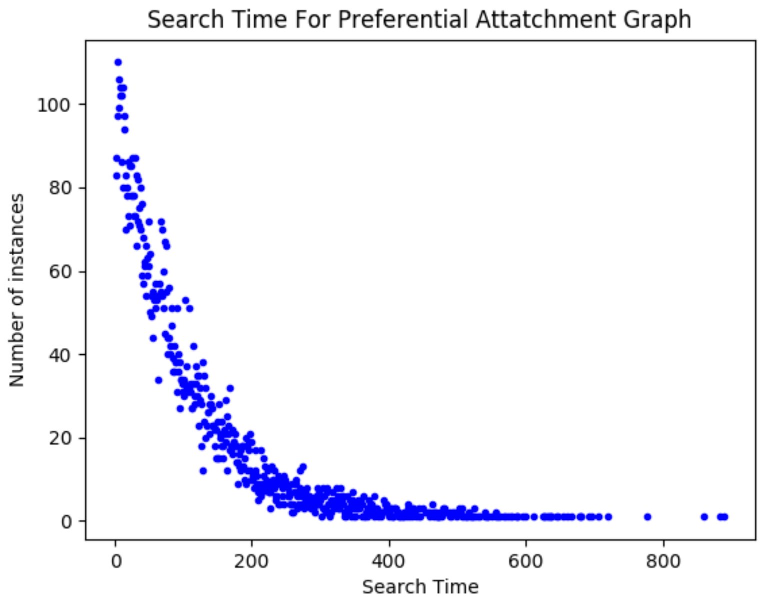
mdsw22

Results Search Strategy 1

Test Case 1: (n = 100, p = 0.1)

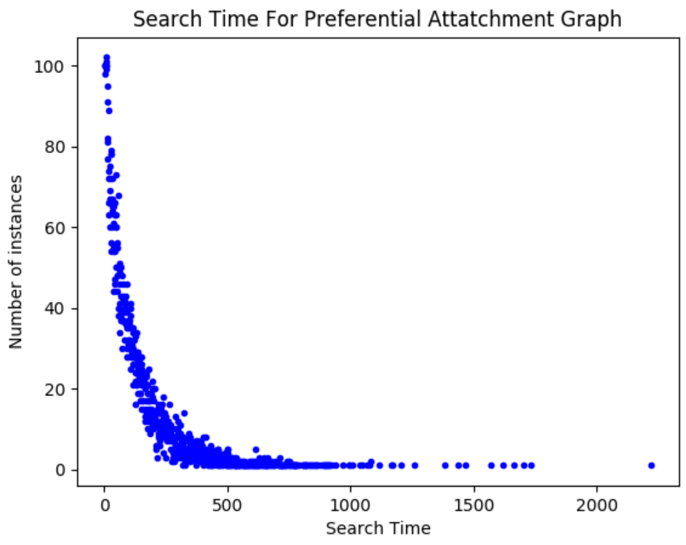


Test Case 2: (n =100, p = 0.9)

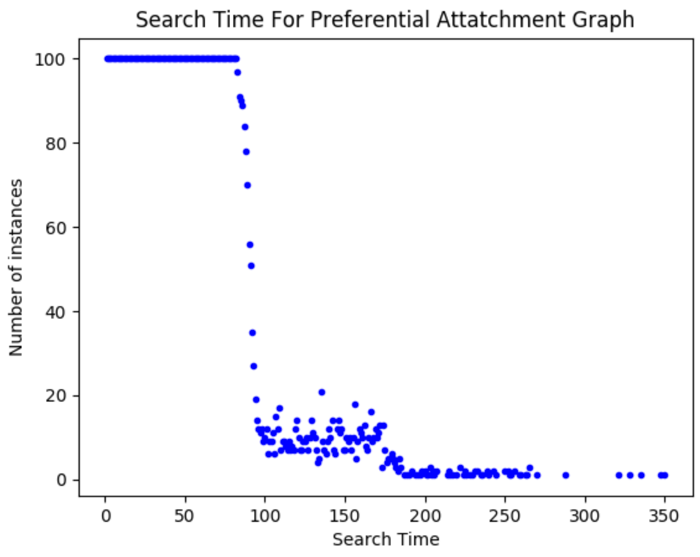


Results Search Strategy 2

Test Case 1: (n = 100, p = 0.1)



Test Case 2: (n = 100, p = 0.9)



For both search strategies, the number of instances achieving a given search time followed an exponential decay.
The results comparing the two different search strategies were very interesting. For a low probability (i.e. p = 0.1), the results were very similar. Search strategy 1 had an average search time of 120 queries whereas search strategy 2 has an average search time of 128 queries. Allowing for random variation, these are essentially equal. On the other hand, when considering a high probability (i.e. p = 0.9), when the graph is much denser, the results were much more interesting. Search strategy 1 had an average time of 97 queries whereas search strategy 2 had an average search time of 54 queries!

Search strategy 1 follows an exponential decay in terms of the number of instances that achieve a given search time for both values of p. This is expected as the algorithm randomly selects a neighbour to move to. When the value of p increased, the search strategy has a greater number of neighbours to choose from however as it is random, it makes no difference. On the contrary, a node u in search strategy 2 checks all its neighbouring nodes for the target vertex. When p is high (i.e. 0.9) there is a much greater chance that a given node will be connected to the target vertex.

The test case 2 for search strategy 2 illustrates this point, with a high proportion of nodes achieving a low search time before a sharp drop off. This is due to the fact that if the search strategy cannot reach the target vertex from the start vertex u, it must move randomly to a neighbouring vertex v and search all of $v's$ neighbours. This pattern is repeated again (illustrated in the diagram). This illustrates the fact that for search strategy 1, most searches find the target vertex in one step (traversal from node u to v) whereas less find the target vertex in two steps and even fewer find the target vertex in 3 steps. Within each step (shown by the levels in the diagram), the number of instances is more or less constant.

In summary, when p is low, the two algorithms are essentially the same. However, when p increase, search strategy 2 gains benefit from the increased degree of each node whereas search strategy 1 does not. Search strategy 2 follows more what a human would do when provided with this random graph, it exhibits curiosity (exploring neighbouring nodes before moving onto next iteration).

Ring Group Graph

For ring group graphs, they are essentially random graphs with additional grouping structure. As such, the only information we can use is the number of neighbours each vertex has and the group it is in. For this algorithm, I have **not** assumed $p > q$.

Each pair of distinct vertices u and v are connected with probability p if they are in the same group or adjacent groups $|l_u - l_v| = 1 \ mod \ m$, and q if not. As such, the search strategy is dependent on the values of p and q. More specifically, the search strategy is dependent on whether:

i) $p = q$
ii) $p > q$
iii) $p < q$

For each of these possible options, the search algorithm starts by finding out what group l_t the target vertex t is in.

i) $p = q$

Distinct vertices u and v are equally as likely to be attached to each other, irrespective of whether they are in the same, adjacent or non-adjacent groups. As a result, the search strategy searches randomly, similarly to the Random Graph search strategy as defined previously. When the search algorithm moves to a new node u, it considers the array $N(u)$ (ids of neighbours) and then randomly selects a vertex v to move to. The algorithm executes one query for every node it moves to so the number of queries is equal to the number of edges traversed in the search.

ii) $p > q$

A vertex u is more likely to be connected to vertex v if u and v are in the same ($l_u = l_v$) or adjacent groups $|l_u - l_v| = 1 \; mod \; m$ than if in non-adjacent groups. If the target vertex t is in group l_t, node u is more likely to be connected the target vertex t if they are in either in same group $l_u = l_t$ or in adjacent groups $|l_t - l_u| = 1 \; mod \; m$. As such, the search algorithm tries to move to a vertex u in the same or adjacent group to the target vertex t. If this is not possible, the search algorithm moves to a random neighbouring vertex of u.

i) $p < q$

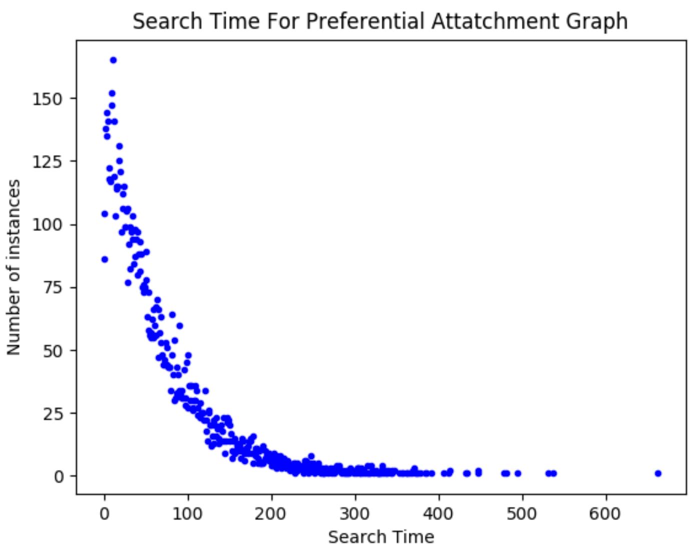
This is the inverse of (ii). A vertex u is more likely to be connected to vertex v if u and v are in non-adjacent groups (i.e. $|l_u - l_v| \neq 1 \; mod \; m$). If the target vertex t is in group l_t, node u is more likely to be connected the target vertex t if they are in non-adjacent groups $|l_t - l_u| \neq 1 \; mod \; m$. As such, the search algorithm tries to move to a vertex u which is in a non-adjacent group to t. If this is not possible, the search algorithm moves to a random neighbouring vertex of u.

Results

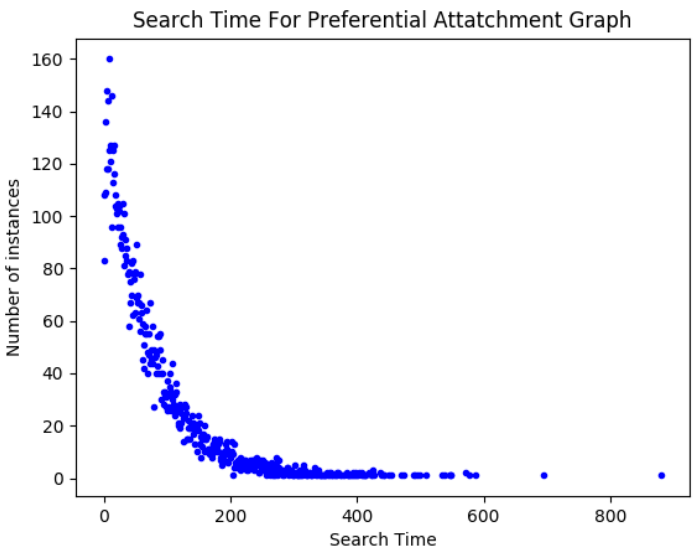
As $p = q$ is the same as the Random Graph question, I shall not present the results as it is redundant data. As $p > q$ is effectively the inverse ring group graph structure as $q > p$, the search time graphs show similar behaviour so we can treat them similarly. As a result, only the graphs for (ii) are shown.

mdsw22

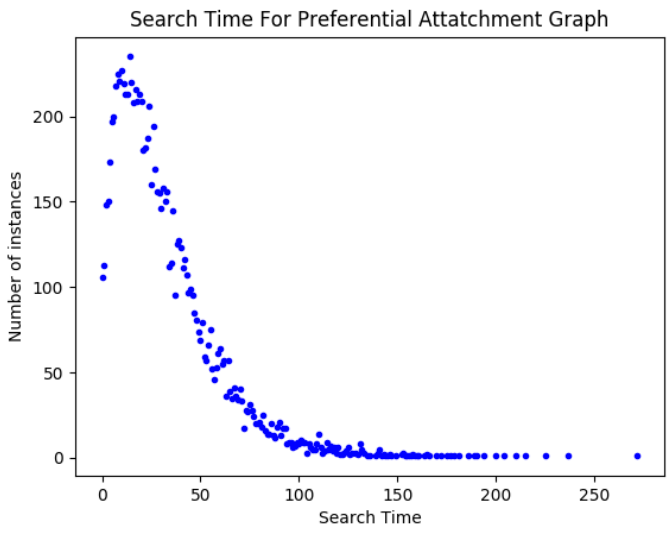
$m = 5,\ k = 20,\ p = 0.9,\ q = 0.1$



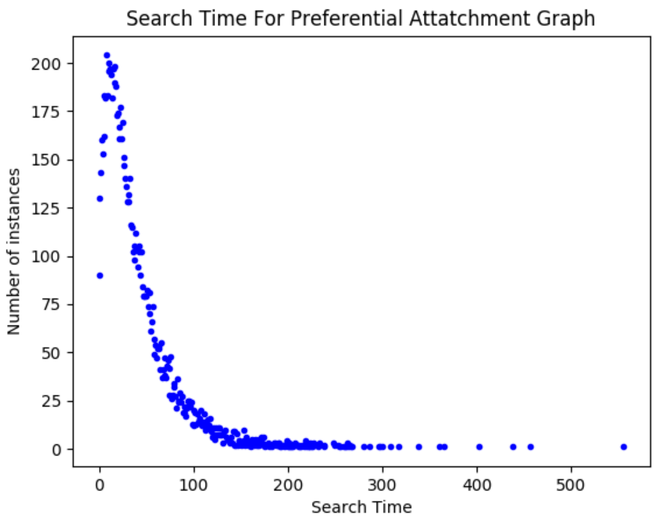
$m = 5,\ k = 20,\ p = 0.4,\ q = 0.1$



$m = 20,\ k = 5,\ p = 0.9,\ q = 0.1$



$m = 20,\ k = 5,\ p = 0.4,\ q = 0.1$



When $k > m$, as per the first set of graphs, the results follow an exponential decay for the number of instances that achieve a given search time. As p is decreased, the pattern stays the same however the graph shifts to the right as the average search time increases. This is expected due to the properties of Ring Group Graphs; as p is decreased, the likelihood of a given node u being connected to the target vertex t in its own group or adjacent group decreases. This results in an exponential decay in the number of instances that achieve a given search time.

On the other hand, when $m > k$, the number of groups is much greater than the number of vertices in each group. This results in a fat-left Gaussian distribution. In fact, both sets of graphs exhibit this fat-left Gaussian distribution however it is more prominent when $m > k$ due to the fact that each group (of which there are more) is composed of fewer vertices and so the search strategy is more likely to result in a number of instances which achieve a small search time.

PA Traversal

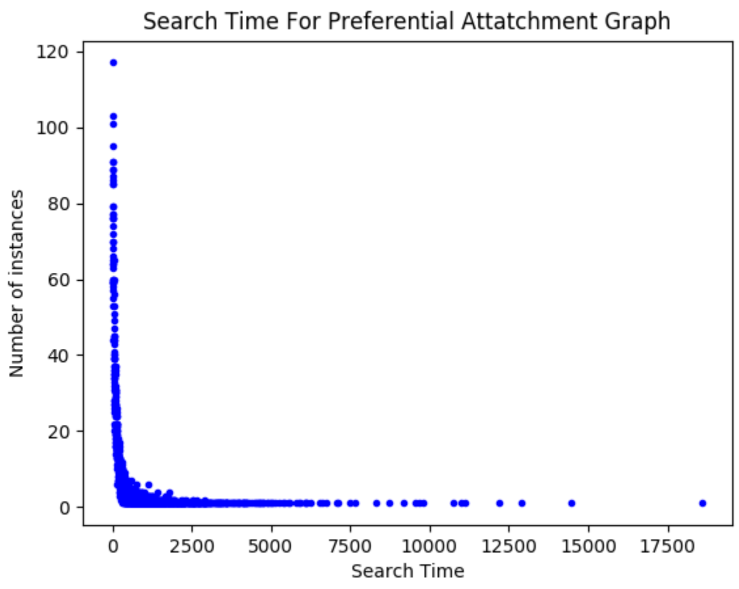
In preferential attachment graphs, edges are added from vertex v to existing vertex u in proportion to the degree of v. The network begins with an initial connected network of m_0 nodes. New nodes are added to the network one at a time. Each new node is connected to $m > m_0$ nodes with a probability that is proportional to the number of links that the existing nodes already have.

The search strategy starts by considering every neighbouring node v of the current vertex u and checks to see if it is the target vertex t. If not, the algorithm queries the ID of every neighbour from $N(v)$ and determine its degree. A nodes ID and its degree is stored in a dictionary of past queries at the current vertex v. The neighbouring vertex u that has the greatest chance of being connected to the target vertex t is the one with the greatest degree. Despite this, the search strategy cannot however just select the neighbour of greatest degree as this can get stuck in an infinite loop. As such, the algorithm introduces a random selection with a bias towards nodes of greater degree. This is done by creating a list of all the neighbours to a vertex v where each neighbour vertex u is added to the list multiple times equal to its degree. The search strategy then randomly selects a node from this list and the likelihood that a neighbouring vertex u is selected is proportional to its degree.

To test this algorithm, two test cases were conducted:

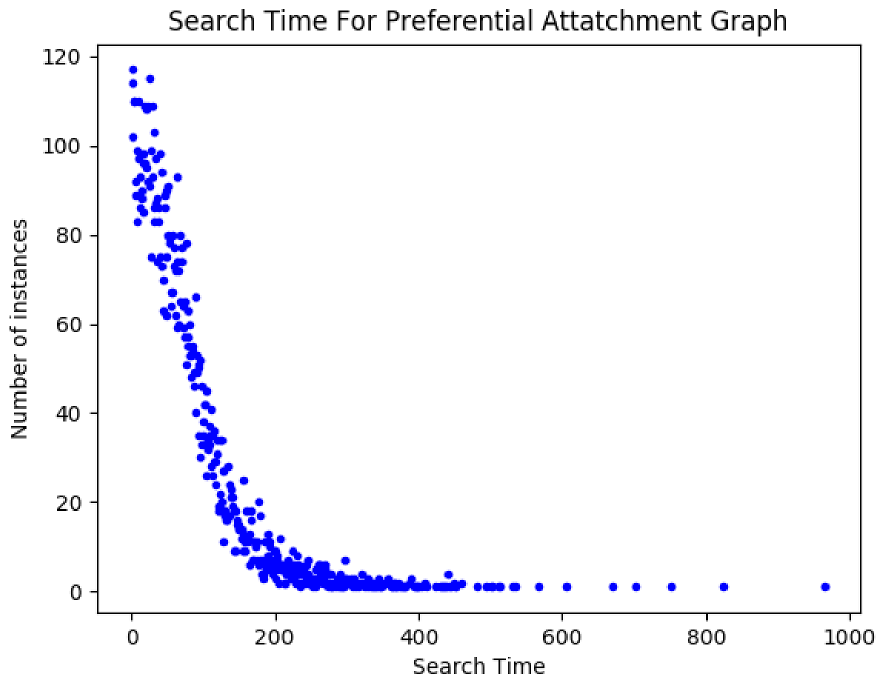
i) $n >> m$
ii) $n > m$ (but only by a little)

Test Case 1: $n = 100, m = 2$



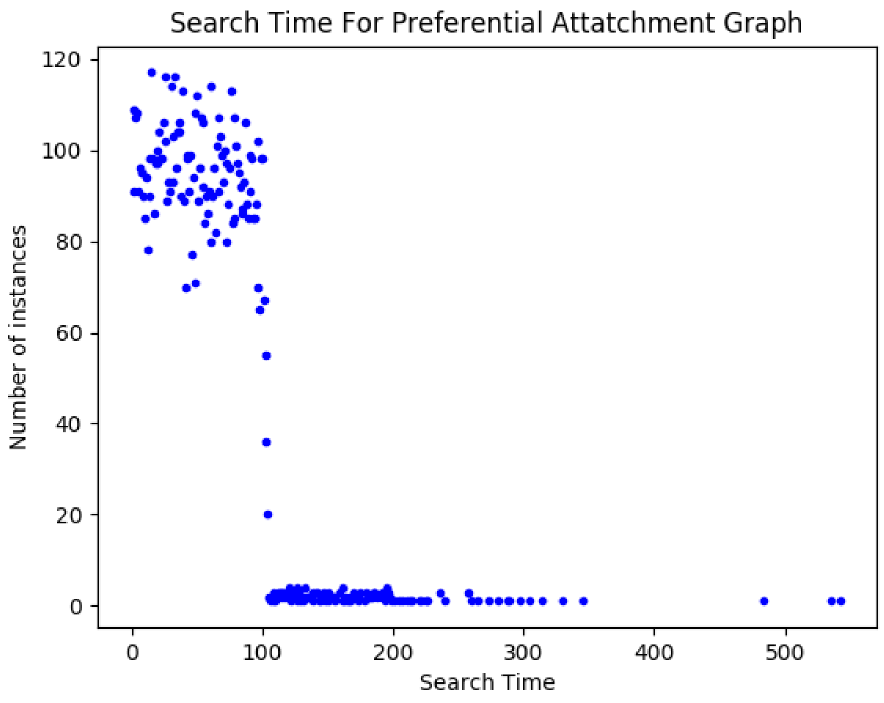
For the Test Case 1, where $\mathrm{n} >> \mathrm{m}$, the original m nodes end up with extremely high degrees as each additional node being added has greater affinity to be connected to those nodes. The degrees of the other $\mathrm{n}-\mathrm{m}$ nodes being added approximately follow a Gaussian distribution. In fact, the degree distribution resulting from the BA model is scale free, in particular, it is a power law of the form $\mathrm{P(k)} \sim \mathrm{k}^{-3}$.
For our search strategy, this means that in the first step, it is **very** likely that the algorithm will move to one of these original m vertices which are in turn very likely to be connected to the target vertex t. As such, this follows an extremely sharp exponential decay in terms of the number of instances that achieve a given search time. If the search strategy starts at one of the original m nodes, it is very likely that it will find the target vertex in a short search time.

Test Case 2: $(n = 100, m = 50)$



For the Test Case 2, where $m = \frac{n}{2}$, the search time follows a more gradual exponential decrease. This is due to the same logic as described above however is much less dramatic due to the fact that there are 50 original vertices and so when the other 50 vertices are added, there is a much greater of vertices that they can be potentially be connected to.

Test Case 3: $(n = 100, m = 95)$



For the Test Case 2, when m is much greater but still obviously less than n, the PA nature of the graph breaks down as the remaining 5 vertices are added with edges connected with little affinity to any of the original m vertcies. As the network begins with an initial connected network of m vertices, the final graph is almost fully connected, expect for the remaining $(n - m)$ vertices. For the search strategy, this means that it is extremely likely that a given starting vertex v is connected to the target vertex t, illustrated by the large number of instances with a search time less than 100 (i.e. one hop). There is then a sharp drop-off, illustrating the rare occurrence that the target vertex t is not found as a neighbour of the starting vertex v.