CCS - Network Analysis Assignment

Fatema Alkhanaizi - mmgw12

November 9, 2018

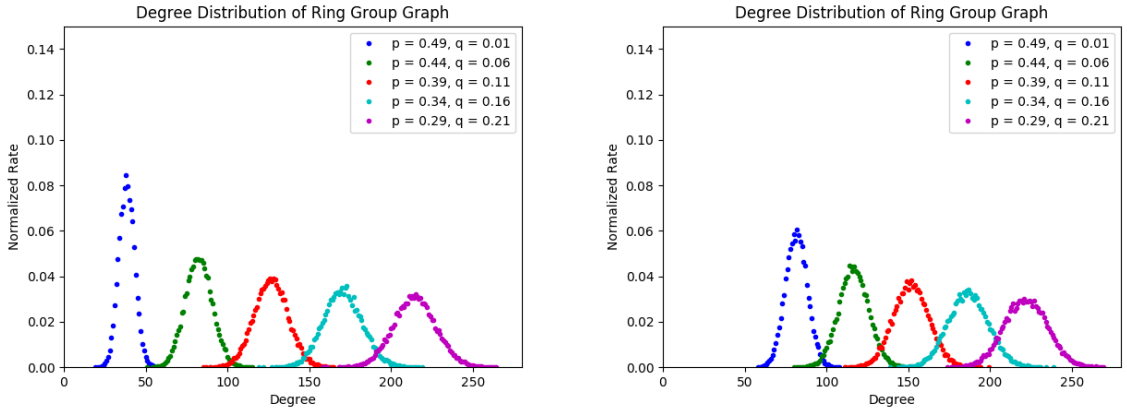
Question 1

Ring Group Graph implementation

The ring group graph was created by using **networkx** python3 package's graph structure. Networkx was also used to check the continuity of the graph instances used for the degree distribution (all unconnected graphs were skipped). In addition, it was used to calculate the diameter of the graph.

Ring Group Graph Degree Distribution when $p + q = 0.5$, $p > q$

The degree distrbution was normalized and averaged over 25 instances of the ring group graph. The following plots are for the case when $p + q = 0.5$, $p > q$:



(a) m = 50, k = 20 (b) m = 20, k = 50

Figure 1: Ring Group Graph when $p + q = 0.5$, $p > q$

By the Ring Group graph definition each vertex can be linked to 3 groups (two adjacent groups and its own group) and since each group has k members there is $p \times (3k - 1)$ edges the vertex can be connected to on average (the subtracted 1 is the edge to the vertex itself - no self edges are allowed). So, the average for the remaining edges is $q \times k \times (m - 3)$ (the vertices connected with p substracted from all vertices in the graph). So, the degree distribution plots can be explained by this equation:

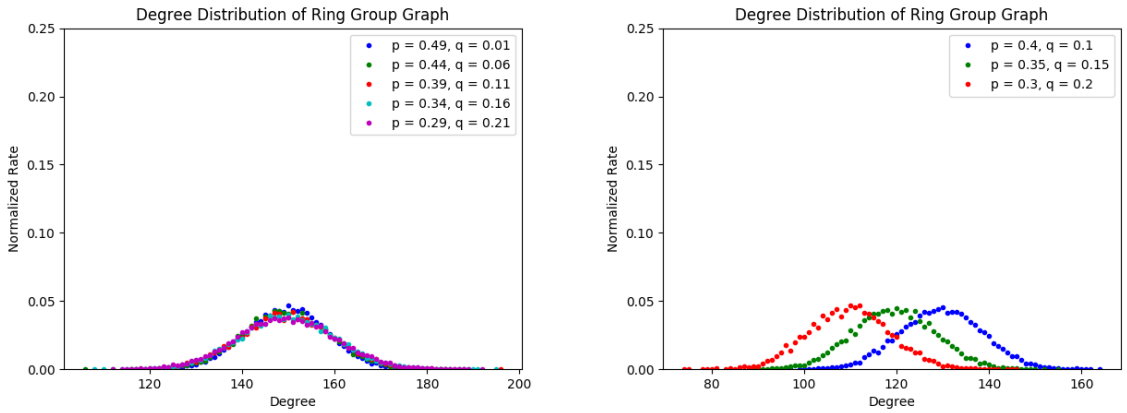
$$\mu_{degree} = p \times (3k - 1) + q \times k \times (m - 3) \tag{1}$$

The general shape of the degree distribution for a Ring Group Graph is bell shaped as shown in figure-1. As m, k, p and q varied the shape of the distribution remained constant; the mean and variance of the degree distribution however changed. A vertex average degree (mean) is

heavly influnced by the value of k which is evident in figure-1b and figure-1a; for the same number of nodes, p and q values, the distribution in figure-1b is shifted toward a greater degree distribution than in figure-1a however this effect decrease as p and q vary. We can make use of equation-1 to prove this as k is part of the computation for both p and q probailities where m only influences the part with q probability.

In addition as the value of p decreases in both figure-1a and 1b and q increases while maintaining the constraint mentioned previously, the degree distribution flattens out. The variance, the range of degrees increase more than the increase in the mean; m has more influence over the mean the greater q probability increases but as p decreases this increase is penalized (the increase steps are reduced as p decreases).

The following plots can be considered special cases which can be extrapolated from equation-1:



(a) m = 6, k = 100

(b) m = 4, k = 100

Figure 2: Special cases for Ring Group Graph when $p + q = 0.5$, $p > q$

For figure-2a, by equation-1 when m = 6 we get:

$$\mu_{degree} = p \times (3k - 1) + q \times 3k$$

so for this distribution regardless of the value of k, combinations of p and q where $p + q = 0.5$ and $p > q$ will result in almost the same average degree across all combinations for that k; hence the overlapped distributions in figure-2a.

For figure-2b, by equation-1 when m = 4 we get:

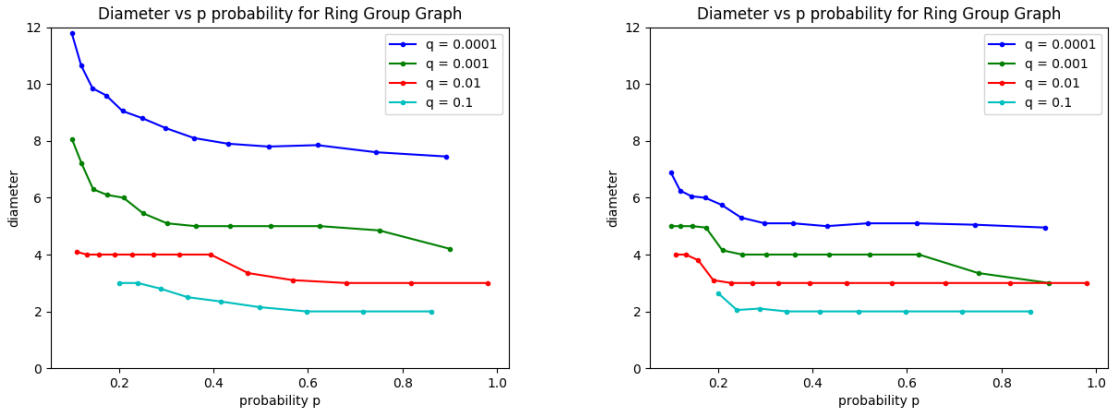
$$\mu_{degree} = p \times (3k - 1) + q \times k$$

In this case, the probability p has a greater influence over the mean (3 times more than the probability of q) so as p decrease the mean decreases as well. This explains why for the values p = 0.3 and q = 0.2, the distribution is on the right side of the distribution for when p = 0.4 and q = 0.1; Compared to the distributions in figure-1 where the degree distribution was always shifting to the right as p was decreasing and q increasing which can be contributed to the value of m.

Diameter of Ring Group Graph and p (for a fixed q, $p > q$)

To invesitgate the relationship between the diameter and the probability p, the diameter was caluculated and averaged over 20 instances of the ring group graph and run against multiple

fixed q values then tested against two m and k values (20 and 50):

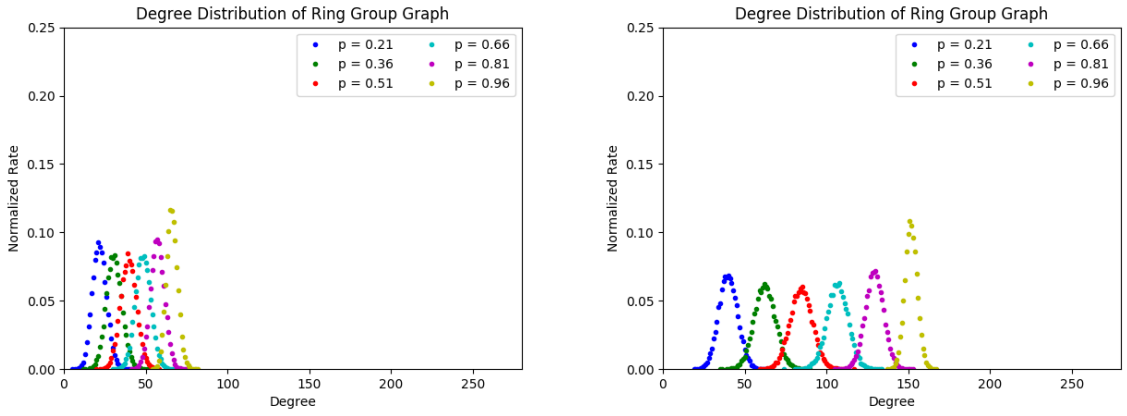


(a) m = 50, k = 20 (b) m = 20, k = 50

Figure 3: Diameter for Ring Group Graph when $p > q$ (q is fixed)

The picked values for m and k provided a good contrast on how the diameter is affected when m is greater than k and vice versa. As shown in figure-3, as p increases the diameter decreases and at some point it becomes constant. For various q values the diamter had a similar behaviour, the value of the diameter was generally higher as q decreased which is expected as the probability q essentially creates shortcuts between the groups and without those shortcuts the diameter depends on the value of m and p. The decrease in the diameter wasn't however constant or followed a certain pattern as p and q varied.

Figure-4 contains the degree distribution plots for when q = 0.01 to investigate how the degree distribution affects the diameter:



(a) m = 50, k = 20, q = 0.01 (b) m = 20, k = 50, q = 0.01

Figure 4: Ring Group Graph when $p > q$ (q is fixed)

As the degree distribution shifts to higher degrees, the diameter decreases (which is expected). In figure-4a, the degree distribution as p varied only shifted by a minor margin compared to figure-4b. The convergances of the diameter in figure-3a and figure-3b can be linked to the changes in the degree distribution as p varied. For figure-3b the diameter seemed to converge as p was between 0.2 and 0.3; figure-4b has the degree distribution for when p = 0.21 and p = 0.36 to be between 40 to 60. On the other hand, the diamter seems to reach a

converged value when p is close to 0.7; the degree distribution in figure-4a for when p = 0.66 and p = 0.81 had a mean range from 48 to 57. Based on those values, the diameter converged for q = 0.01 when the mean degree distribution was around 40 to 60 in this case.

Question 2

Distribution of Vertex Brilliance

Calculating Vertex Brilliance

The vertex brilliance was caluculated based on a similar approach to calculating the diameter; by using breadth-first search but by only doing breadth-first search at debth of 2 - checking the neighbours of the vertex's neighbours. Complete description of the algorithm used can be found the appendex-0.1. It was essentially an algorithm to get the maximum independent set of sub-graph that only contains v and it's neighbours. I opted to using my own algorithm instead of using networkx's implementation.

Creating Vertex Brilliance Distribution

For each generated graphs (PA graph and Ring Group graph), the vertex brilliance distribution was averaged and normalized over 100 instances. The distribution was created following the same code as the degree distribution.

Vertex Brilliance Results

Co-authorship Graph

The co-authorship graph was loaded by following code provided in the lectures; it was just basic file-reading and making sure that the edges were undirected. The following plots are the degree distribution (in blue) and the vertex brilliance distribution (in red) of the graph:

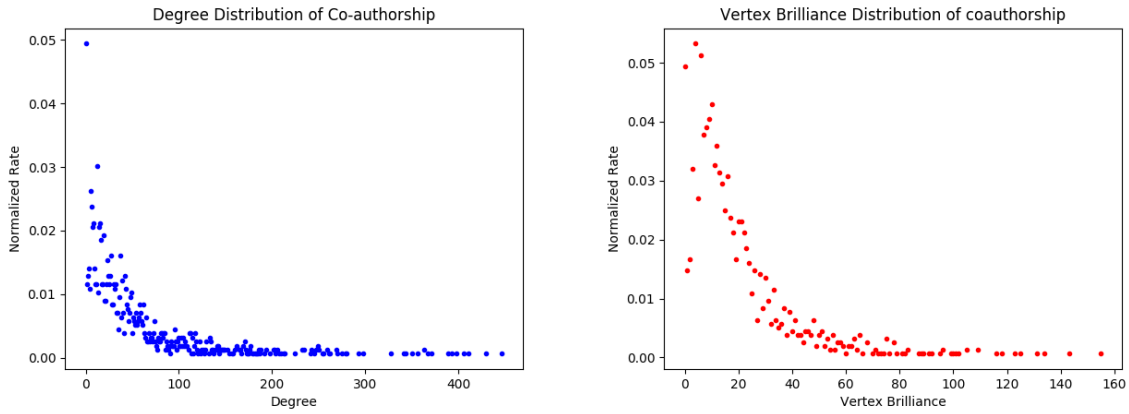


Figure 5: Loaded Co-authorship Graph including vertices with no edges

The general shape of both graphs is skewed to the right. With the degree distribution ranging from 0 to around 460, compared to the vertex brilliance ranging from 0 to less than 160. The vertex distrbution appears to be the same as the degree distribution but factorized by 0.3 in the x-axis.

PA Graph

As the implementation of the PA Graph that we used in the lectures was directed, networkx implementation was modified and matched to the PA Graph specifications provided in lecture:

- The first 1 to m vertices must be fully connected and each vertex repeated m times in a list for selecting the new edges for the remaining vertices
- For the remaining m + 1 to n vertices, a vertex was randomly picked from the previous list m times (to create an edge). For each iteration until n is reached, the new vertex was added m times in the selection list along with its neighbours.

The following plots are the degree distribution and vertex distribution of the PA graph:

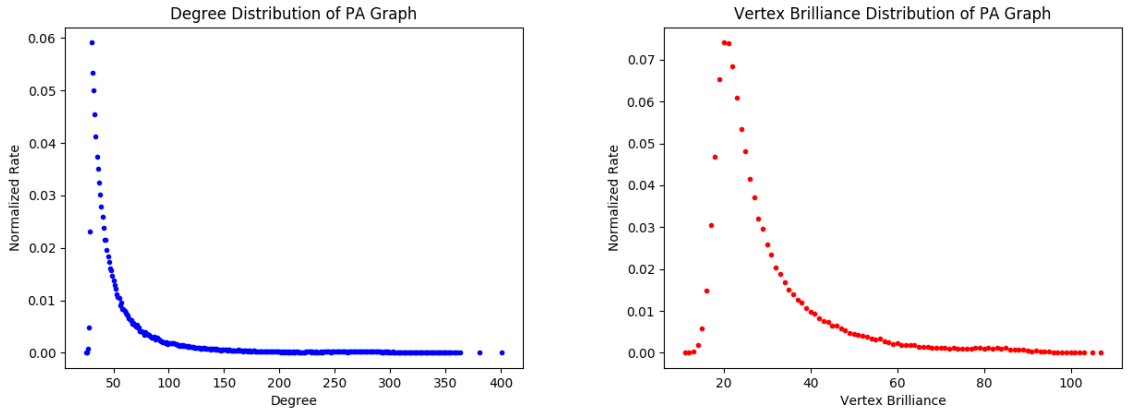


Figure 6: PA Graph parameters: m = 1559, m = 30

The paramaters for the PA graph were set so that they match the number of vertices and edges of the co-authorship graph (n = 1559 and p = 30). Both distributions had been averaged over 100 instances of the PA graph. Similar observation as the co-authorship graph, the resulted graphs for both degree and vertex brilliance distributions were skewed to the right with different ranges; the degree distribution ranged from around 10 to 400 while the vertex distribution ranged from around 10 to less than 120. The normalized rate over the number of instances (y-axis) for the distribution slightly increased compared to the degree distrbution. The co-authorship graph strongly resembled the PA graph based on both distributions.

Ring Group Graph

To get approximately the same number of edges and nodes as the co-authorship graph, the following parameters were used: m = 60, k = 26, p = 0.22, q = 0.03. There are other possible parameters but their distributions were similar to the distributions with the paramaters used here (in terms of shape) look at figure-7 and figure-8.

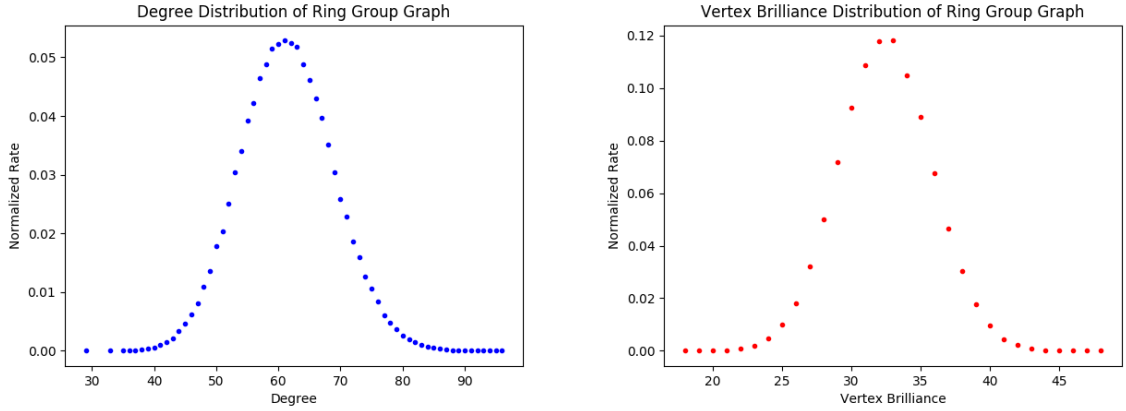


Figure 7: Ring Group Graph parameters: m = 60, k = 26, p = 0.22, q = 0.03

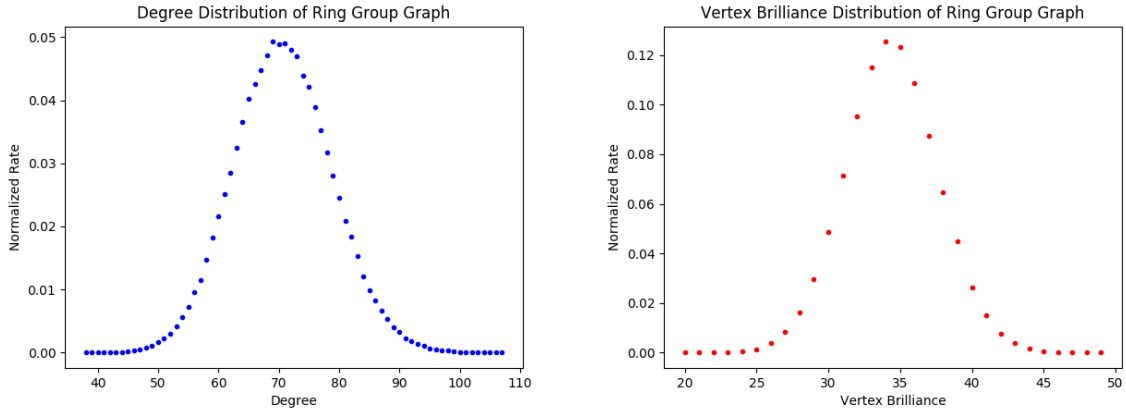


Figure 8: Ring Group Graph parameters: m = 120, k = 13, p = 0.26, q = 0.04

Both figures had bell shaped distributions (binomial distribution). The vertex brilliance distribution differed from the degree distribution by a factor of approximately 0.5 in the x-axis and normalized rate over the number of instances (y-axis)for the distribution approximately doubled compared to the degree distrbution.

Based on the previous plots, it can be concluded that the vertex distribution has the same shape as the degree distribution and the co-authorship graph closely resembles the PA graph.

Question 3

Search Random Graph

Search Algorithm

The search in a Random Graph was implemented as follow:

- Inputs:
  - The graph
  - The graph parameters - (n, p)
  - The start vertex id - (numerical id)

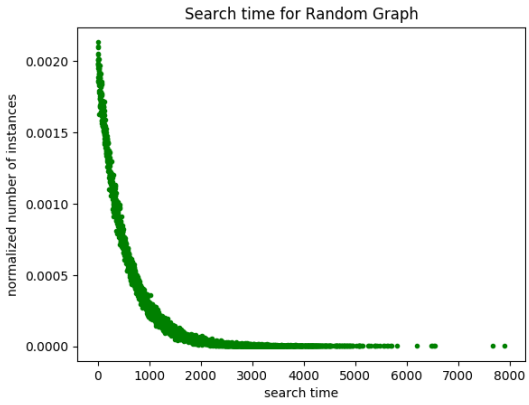
  - The target vertex id - (numerical id)

- Initialize a variable for search time and set it to 0
- Calculate the average number of neighbours based on the number of nodes and p
- check if p is greater than 0.5, if it is then set p to 1 - p
- start an infinite loop until the target vertex is found:
  - shuffle the current vertex's neighbours list (at the beginning it is the start vertex)
  - check if the number of neighbours is more than average if so then set the number of iterations up until the average
  - Iterate over the neighbours' list:
    * increment the search time by 1 for every iteration that happens
    * if the neighbour's id matches the target's id then stop, break out of the loop and return the search time
    * else move to the neighbour with a probability p (the probability of the random graph)
  - if no moves were made move to the last queried neighbour

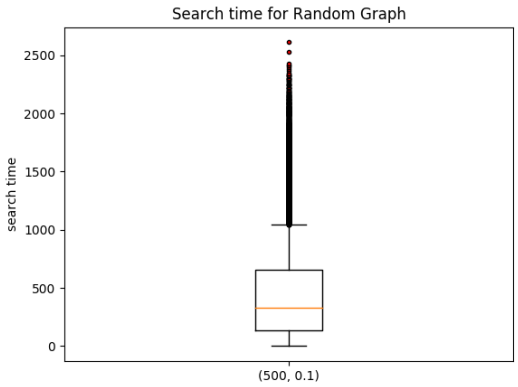
The algorithm above made use of the general knowledge of the graph i.e. total number of nodes (n) and the probability (p) that an edge exists between each pair of nodes in addition to the local knowledge i.e. vertex id and number of its neighbours to better find the target vertex (t). Every vertex had the same probability to be connected to t, hence the condition to move to a node based on the probability p. Either going until the average number of neighbours or going until all the neighbours did not result in a much better search time. This is evident from the search time distribution in figure-10a and figure-10b in the next section; going only until the average number of neighbours did not improve the search time by any significant margin even for bigger values of n. Still in the above algorithm the number of iterations is maxed at the average number of neighbours. By reseting p to a lower probability when p is more than 0.5, made it less likely to move to another vertex to find the target vertex as the high probability indicates that it is very likely that the target vertex is connected to the current vertex. Another note regarding the algorithm, as the neighbours were shuffled with every move, moving to the last neighbour every time no moves happened will be random in itself as well so a different vertex will be moved to in every iteration; of course based on the probability and how random the outcome of the shuffling.

Search Time Distribution

The search time distribution was created by searching 30,000 random pairs of vertices normalized and averaged over 10 random graph instances. To insure that the graph instance was connected, networkx package was used for both creating the graph using networkx graph structure and then checking if it's connected (all nodes have an edge between them). The plots below were created to investigate the search time distribution for a random graph and how it was affected by the total number of nodes (n) and probability p:



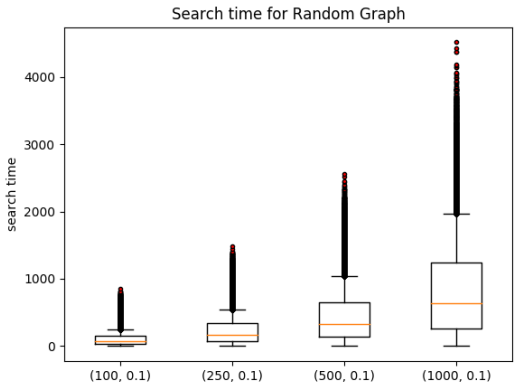
(a) Normalized search time distribution



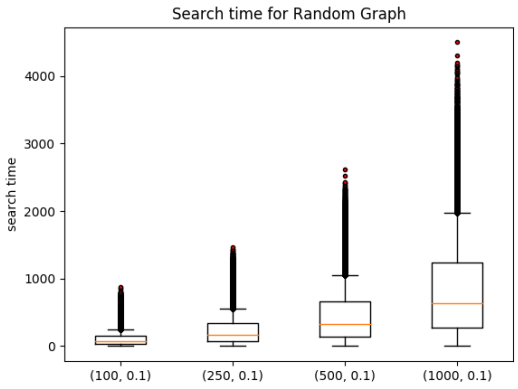
(b) Box-plot respresentation of the search time distribution

Figure 9: Search Time Distribution for Random Graph when n = 500 and p = 0.1

Figure-9a is an example of a search distribution for a random graph when n is 500 and p is 0.1. The box-plot will be used to represent the distribution as it is much simpler to interupt like figure-9b; some of the outlier values might not be included, those outliers have a low number of occurance so they were discarded in the box-plot and only the main distribution was included. In this case, the distrbution was skewed to the right with a lot of outlier values. More than 50% of the instances had a search time that was less than 500. The median search time was roughly around 300 for this graph instance.



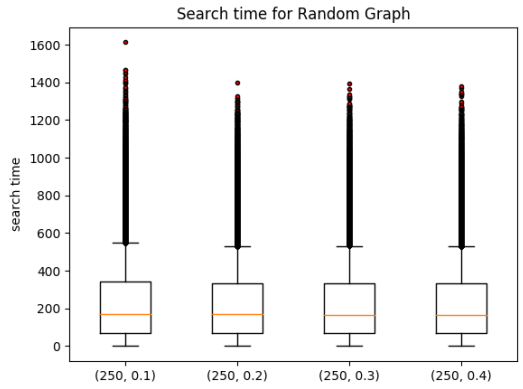
(a) Only iterate until average number of neighbours



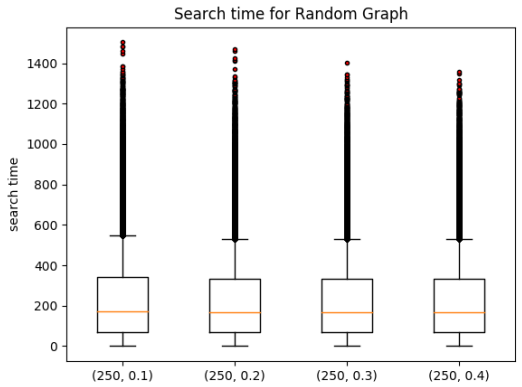
(b) iterate until the vertex's number of neighbours

Figure 10: Search Time Distribution for Random Graph when only n changes

Figure-10a and figure-10b had a constant probability p and an increasing n. As n increased the search time increased which was an expected outcome as more edges and vertices would be iterated over. The performance of the algorithm described in the previous section was the one used in figure-10a, it did not significantly improved the search time (the difference if there is any is barely noticable from the box plots in both figures 10a and 10b). The median search time across all graph instances was slightly more than half of n and the shape of the graph persisted as well.



(a) Only iterate until average number of neighbours



(b) iterate until the vertex's number of neighbours

Figure 11: Search Time Distribution for Random Graph when only n changes

Figure-11a and figure-11b had a constant n and an increasing p. As p increased the search time decreased but only very slightly.

Based on the search algorithm, the search time increased as n increased however it slightly decreased as p increased in a random graph. The shape of the distribution remained skewed to the right with a lot of outlier values throughout all the distributions in figures 10 and 11.

Search Ring Group Graph

Search Algorithm

By definition all vertices in a Ring Group graph are part of a group, and adjacent groups have higher probability (p) of having an edge between their vertices than with other groups which have a lower probability (q). So, the most basic strategy for searching this graph is by moving to the closest group or if possible to a group adjacent to the target vertex and look for the target in that group because of the higher probability. Unlike Random graph not all vertices are connected with the same probability and the group number is part of the vertex id so it helps narrow the search significantly. Based on this, the search in a Ring Group graph was implemented as follows:

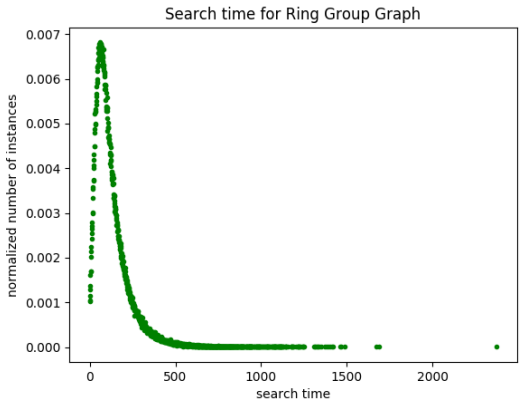
- Inputs:
  - The graph
  - The graph parameters - (m, k, p, q)
  - The start vertex id - (numerical id, group)
  - The target vertex id - (numerical id, group)
- Initialize a variable for search time and set it to 0
- Create a variable for the probability to move to a vertex that is closer to the target's group than the current vertex and set it to be equal to p; call it prob_close.
- check if p is greater than 0.5, if it is then set p to 1 - p
- start an infinite loop until the target vertex is found:
  - shuffle the current vertex's neighbours list (at the beginning it is the start vertex)

- Iterate over the neighbours list:
  - increment the search time by 1 for every iteration that happens
  - if the neighbour's id matches the target's id then stop, break out of the loop and return the search time
  - else if the neighbour's group is the same as the traget's group then move to that neighbour with a probability p (the probability for vertices in adjacent groups to have an edge), otherwise keep track of this neighbour in adjacent_neighbour variable
  - else if the neighbour's group is closer to the target than the current vertex's move to that neighbour with the probability that we set at the beginning of this algorithm (prob_close) otherwise keep track of this neighbour - closest_not_adjacent_v - the same way as the previous condition
  - else move to this neighbour with a probability of q (probability of an edge)
- if no moves has been made check if adjacent_neighbour has been assigned and move to it, otherwise check if closest_not_adjacent_v has been assigned and move to it. If neither values are assigned move to the last queried neighbour

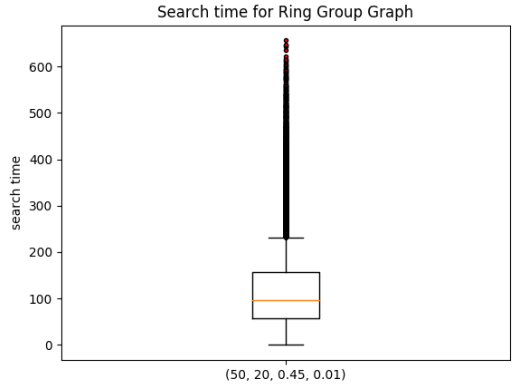
The assigned value for prob_close was determined through trial and error and proved to provide the best results (figure-13a and figure-13b). The condition for when p is more than 0.5 is similar to the one in the random graph, making it less likely to move to another vertex in the group to find the target vertex as the high probability indicates that it is very likely that the target vertex is connected to the current vertex. For adjacent_neighbour and closest_not_adjacent_v variables, only one neighbour is stored at a time so if another neighbour with the same condition is encountered that neighbour will be stored instead and as the neighbours list is shuffled with every move it is less likely this neighbour will be the same if the current vertex is returned to at some point.

Search Time Distribution

To insure that the graph instance was connected, networkx package was used for both creating the graph using networkx graph structure and then checking if it's connected (all nodes have an edge between them). The following plots are the search time against the number of instances that achieve that time evaluated for 30,000 randomly picked pairs of vertices normalized and averaged over 10 instances of the ring group graph and averaged again:



(a) Search time distribution normalized

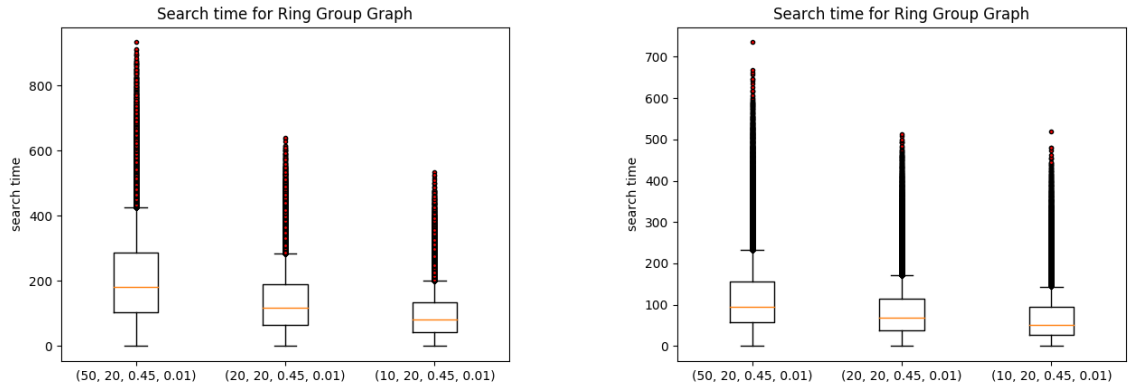


(b) Box-plpot respresentation of the search time distribution

Figure 12: Search Time Distribution for Ring Group Graph when m = 50, k = 20, p = 0.45 and q = 0.01

Figure-12a is an example of a search distribution for a ring group graph when the graph parameters are m = 50, k = 20, p = 0.45 and q = 0.01. Similarly to the search distribution for random graph, the box-plot will be used to represent the distribution as it is much simpler to interupt like figure-12b; some of the outlier values might not be included, those outliers have a low number of occurance so they were discarded in the box-plot and only the main distribution was included. In this case, the distrbution was slightly skewed to the right with a lot of outlier values. The shape of the distribution was more defined than the distribution for random graph and the median was closer to the middle of the inter-quartile range in the box-plot 12b. There were more nodes in this graph to search than the random graph and still the range of the search time values was much lower than the random graph searches. Around 50% of the instances had a search time that was around 100 and majority of the instances had a search time less than 300.

Before invesitigating the effect of paramater changes to the search in a ring group graph, the following plot demonstrates the benefit of introducing the variable prod_close to the algorithm:



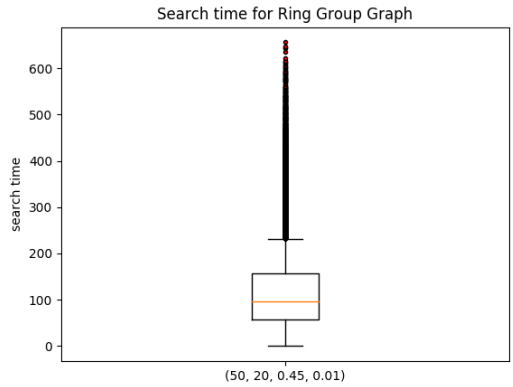
(a) without the probability to move to a closer neighbour

(b) with the probability to move to a closer neighbour set to p

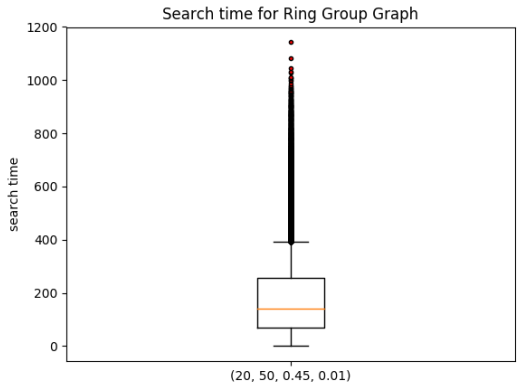
Figure 13: Ring Group Graph parameters: p = 0.45, p = 0.01, k = 20 and m is decreasing

As shown in figure-13b, the search time for the ring group graph with the specified paramaters was much lower with prob_close introduced and set to the value of probability p. Even as the value of m was decreasing the search time was still lower in figure-13b than figure-13a. The outlier values were less as well. This proves by making it more likely for the target to move closer to its adjacent group allows for faster search time (dependent on the value of p).

The remaining plots will look into the search time values as the paramater for the ring group graph:



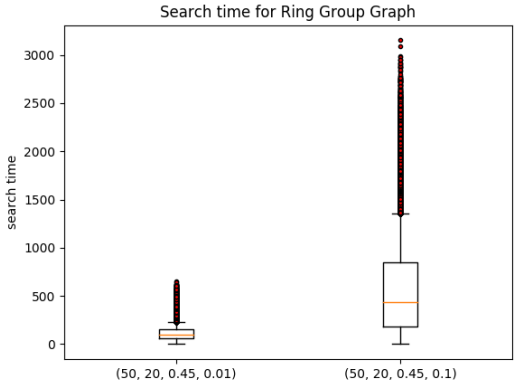
(a) Search Time distribution when m is greater than k



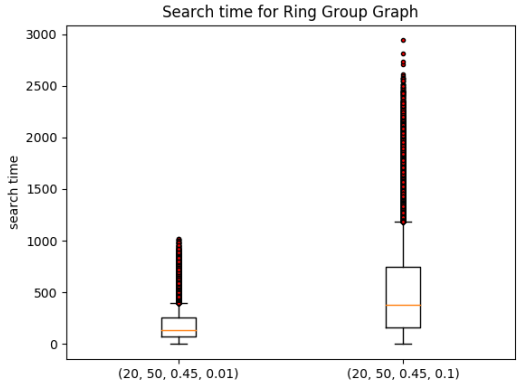
(b) Search Time distribution when k is greater than m

Figure 14: Ring Group Graph parameters: p = 0.45, q = 0.01

In this figure, the total number of the nodes is equal and the same p and q values are applied to m and k. As shown in figure-14a, the search time was much lower when m was greater than k. The shape of the distribution however was almost identical. Compared to figure-14a, the search time in figure-14b was almost twice as much; majority of instances had a search time less than 250 in 14a, in contrast in 14b, around 50% of instances had a search time between 150 and 400.



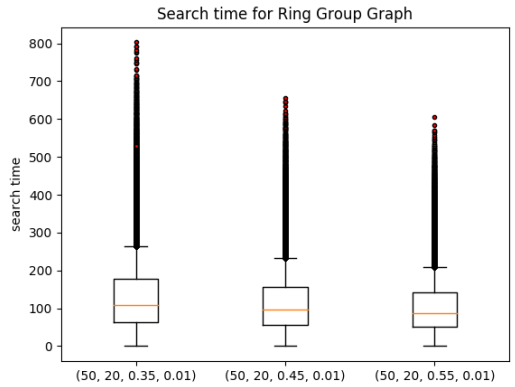
(a) Search Time distribution when m is greater than k and q is changing



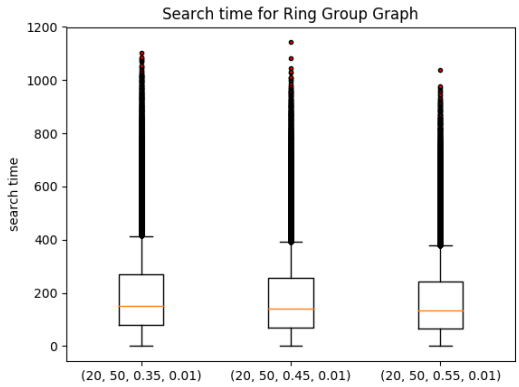
(b) Search Time distribution when k is greater than m and q is changing

Figure 15: Ring Group Graph parameters: p = 0.45

This figure extends the comparison made in figure-14. The value of q was increased to 0.1, this resulted in the search time for both pairs of m and k in 16b and 16a to significantly increase (more random edges has been introduced) The distribution is also more skewed to the right and the box-plot is similar to the plot for a random graph; the upper quartile covers way more search time values than the lower quartile and the median is slightly further down from the center of the box.



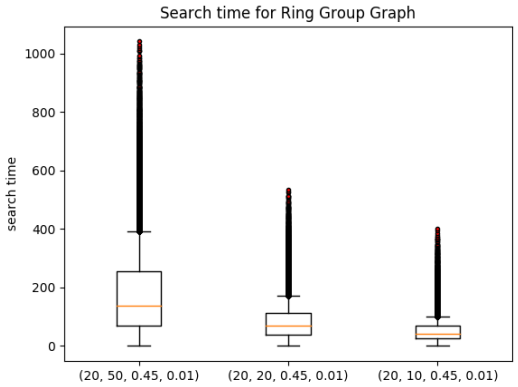
(a) Search Time Distribution when m is greater than k and changing p



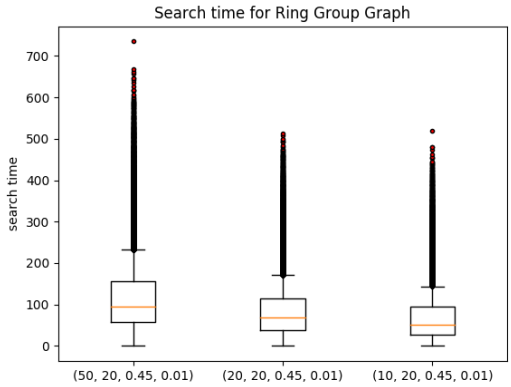
(b) Search Time Distribution when k is greater than m and changing p

Figure 16: Ring Group Graph parameters: q = 0.01

On the other hand, when p increaseed and q remained constant, the search time generally decreased as shown in figure-16. The decrease was more noticable when m was greater than k, figure-16a. This indicates that the search algorithm implemented for the ring group graph much favours high p values, low q values and m to be greater than k. This doesn't mean that this algorithm is less effective for the other cases but it is more optimized for the previous conditions.



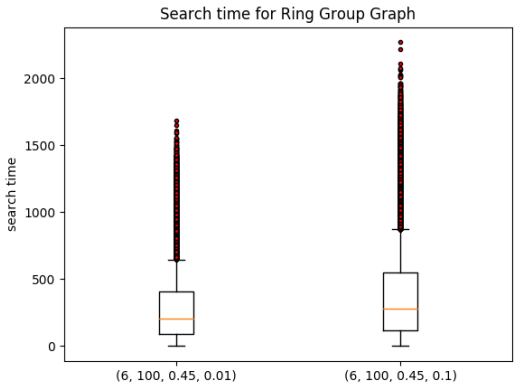
(a) Search Time Distribution with changing k and m = 20



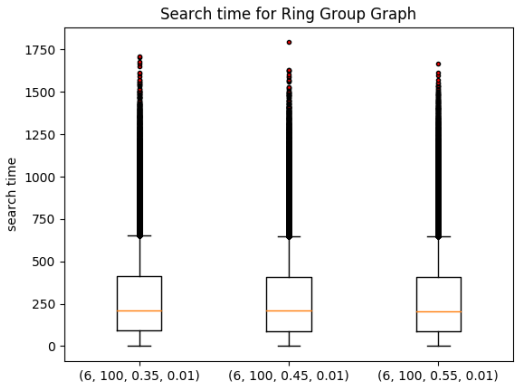
(b) Search Time Distribution with changing m and k = 20

Figure 17: Ring Group Graph parameters: p = 0.45, q = 0.01

Figure-17 further looks into the changes in the distribution while the value of k decreased as m, p and q remained constant, figure-17a, and the value of m decreased as k, p and q remained constant, figure-17b. Greater search time decrease is apparent as k decreased in 17a than as m decreased in 17b; the search time for when m and k were equal could be used as a reference point in this plot. In 13b when k decreased to 10 that resulted in a much greater difference than in 17b when m decreased to 10; as in the first figure m is greater than k and the other figure k is greater than m.



(a) Search Time Distribution with changing q



(b) Search Time Distribution with changing p

Figure 18: Ring Group Graph parameters: m = 6, k = 100

This last figure looks into the special case for the ring group graph when m = 6. k was always greater than m so as mentioned for the previous figure, the changes in the search time is less noticable when k is higher when p is increasing and the search time is generally much higher than the cases where m is greater than k.

Search in a ring group graph is much faster than in a random graph and the paramaters of the ring group graph greatly influences the search. The algorithm for searching in a ring group graph is optimized for paramaters where m is greater than k, p is a high probability and q is a low probability. The general distribution was skewed to the right with a lot of outlier values.

Appendix

0.1 Vertex Brilliance Algorithm

The algorithm simplified:

1. create an empty dictionary; this dictionary will hold the weight vertices for the k-star of the current vertex
2. populate the dictionary with v's neighbours and give each vertex in the dictionary a weight of 0 to begin with ¡ this weight will be used to determine which vertex to exclude to get the maximum k-star, the weight corresponds to how many v's neighbours the vertex is connected to
3. repeat until maximum k-star is found
    - for each vertex in the k-star dictionary go through its neighbours. If one of the vertex neighbour is a key in the k-star dictionary substract 1 from the weight of the vertex (I gave it negative value as just a notion that this is undesired vertex thus a negative affect but it can be calculated with the same manner by adding 1 but then the rest of the code will be modified so it get the maximum value from the k-star values instead of the minimum)
    - find the minimum weight (value) of the k-star dictionary
    - if the minimum value is 0 then stop as this indicates that non of the vertices in the dictionary are adjacent thus it is the maximum k-star set of vertices (ther can be more than one set but we only care about the length so the actual vertices of the maximum set don't have any actual value for us in this case) so stop the loop

- else find the vertex with the minumum weight and remove it from the dictionary (this vertex is the most influential as it connects to the most number of vertices and by removing it the rest of vertices will be less adjacent, only the first found vertix will be removed) and reset all the other vertices values to 0

4. return the length of the dictionary