CCS - Network Analysis Assignment

Fatema Alkhanaizi

November 8, 2018

Clearly document what you have done. For each question you should report on what you did and include, as needed, illustrative plots.

Question 1

Ring Group Graph implementation

The ring group graph was implemented based on the following steps:

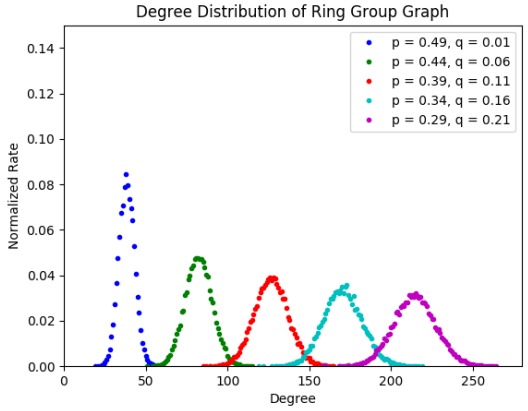
1. For each vertex from 0 to m * k (outer loop) where m is the number of groups and k is the number of members per group, the group label was calculated by dividing the current vertex id [0,m * k) over k, then truncate the value so only the integer value remains from the division
2. The vertex was then iterated over the other vertices (inner loop) starting from the current vertex id plus 1 until m * k.
3. At each iteration of the inner loop, the group was computed again for the inner loop's vertex and checked against the outer loop's group:
    - if the groups are the same or adjacent (absolute difference is one or absolute difference is m - 1, the group difference for the first and last group), then there is a p probability that an edge exists
    - all other cases, there is a q probability that an edge exists

networkx python3 package was used to create the graph and to check that the graph instances used for the degree distribution are fully connected (all unconnected graphs were skipped). It was also used to calculate the diameter of the graph (next section).

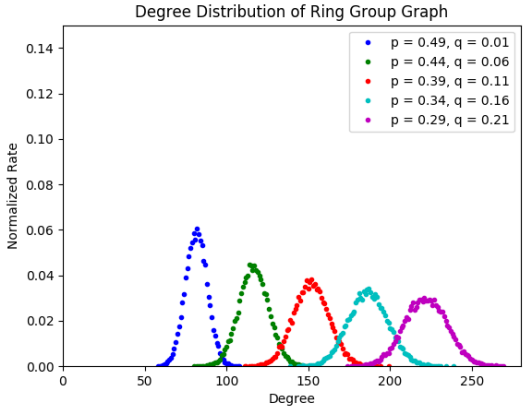
Ring Group Graph Degree Distribution when $p + q = 0.5$, $p > q$

- Investigate the degree distribution of Ring Group Graphs for p + q = 0.5, p ¿ q.
- Decide which values of m, k, p and q to investigate.
- You should report on how the structure changes as p and q vary and whether the same effects are found for different values of m and k.
- Use plots to illustrate your observations.

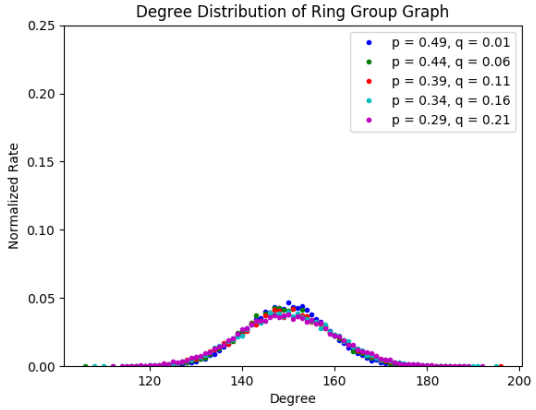
The degree distrbution was normalized and averaged over 25 instances of the ring group graph. The following figures are for the case when $p + q = 0.5$, $p > q$:



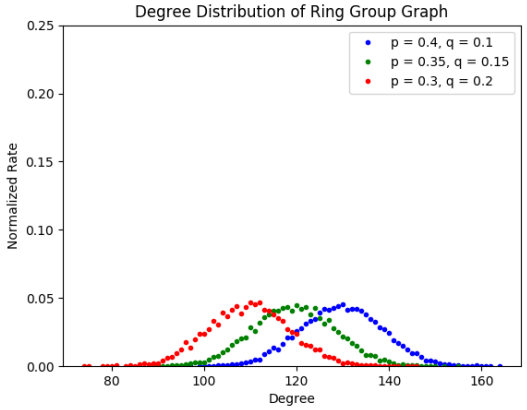
(a) m = 50, k = 20



(b) m = 20, k = 50



(c) m = 6, k = 100



(d) m = 4, k = 100

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 verrex can be conneced to on average with this condition (the subtracted 1 is the vertex itself - no self edges allowed). So, the average for the remaining edges is $q \times k \times (m - 3)$ (the vertices connected with p substracted from all the vertices). So, the degree distribution figures can be explained by these equations:

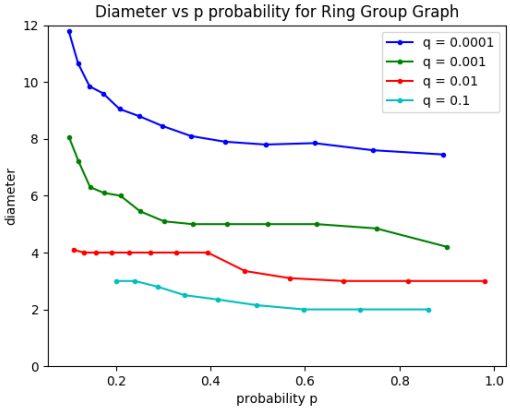
$$\mu_{degree} = p \times (3k - 1) + q \times k \times (m - 3)$$
$$\sigma_{degree} = \sqrt{(1 - p) \times p \times (3k - 1)} + \sqrt{(1 - q) \times q \times k \times (m - 3))}$$

The general shaped of the degree distribution for a Ring Group Graph is a bell shaped. A vertex average degree is heavly influed by the value of k which is evident in figure-**??** and figure-**??**; for the same number of nodes, p and q values, the distribution in figure-**??** is shifted toward greater degree distribution than in figure-**??**.

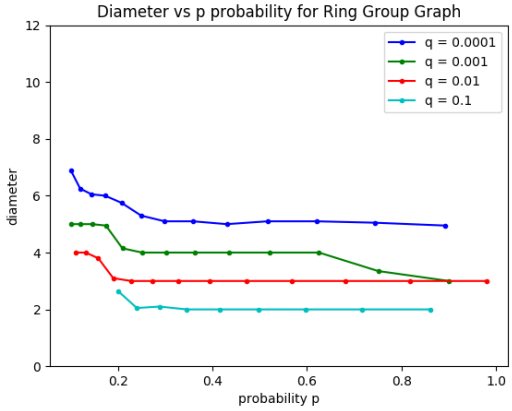
Diameter of Ring Group Graph and p (for a fixed q, $p > q$)

- Investigate the relationship between the diameter of Ring Group Graphs and p (for fixed q, p ¿ q).

The diameter was caluculated over 20 instances of the ring group graph and run against multiple fixed q values and tested against two m and k values (20 and 50):



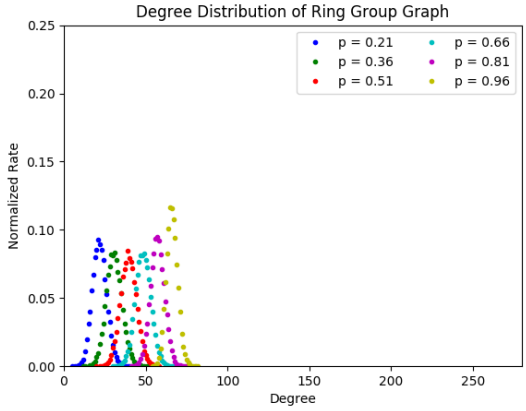
(a) m = 50, k = 20



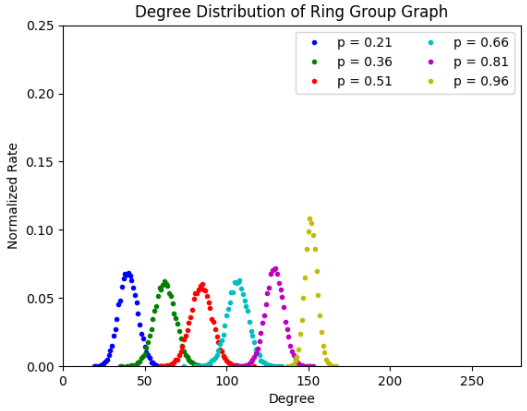
(b) m = 20, k = 50

Figure 2: Diameter for Ring Group Graph when $p > q$ (q is fixed)

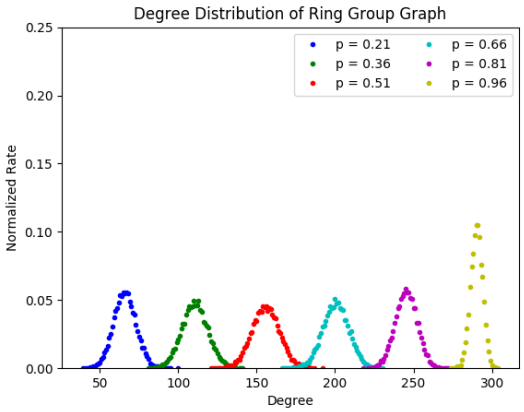
The picked values for m and k provided a good contrast on how the diameter is affected when m is greater than k and vice versa. As shown in figure-**??**, as p increases the diameter decreases and at some point it becomes constant. For various q values the diamter had a similar behaviour, the value of 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.



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



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



(c) m = 6, k = 100, q = 0.01

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

Question 2

Distribution of Vertex Brilliance

Calculating Vertex Brilliance

- Construct the undirected graph defined in coauthorship.txt. Ignore edge weights.
- For the graph from coauthorship.txt, investigate the distribution of vertex brilliance.
- For each of the following types of graph
  - PA Graphs
  - Ring Group Graphs

  create examples with approximately the same number of vertices and edges as the graph from coauthorship.txt and investigate the distribution of vertex brilliance. Comment on what you find.

The vertex brilliance was caluculated based on a similar approach to calculate the diameter by using breadth-first search but by only doing breadth-first search at debth of 2; by checking the neighbours of the vertex's neighbours. 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

This is 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 figures are the degree distribution (in blue) and the vertex brilliance distribution (in red) of the graph:

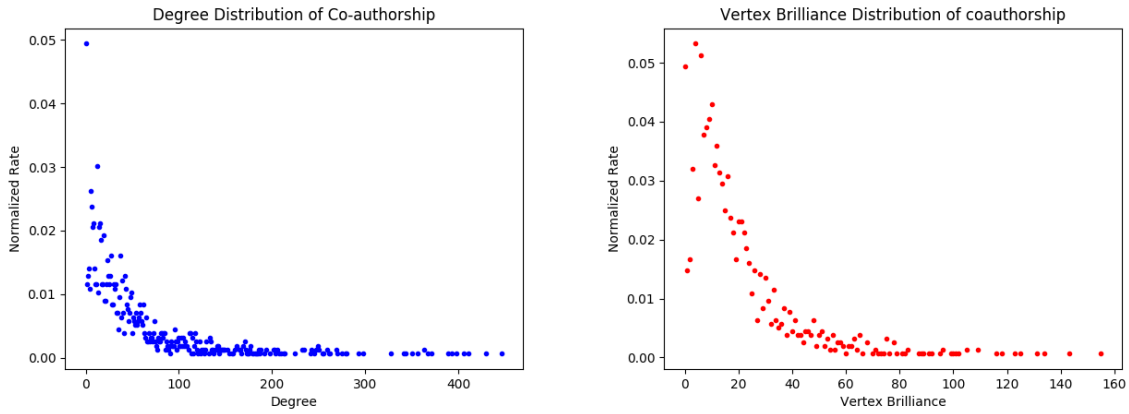


Figure 4: Loaded Co-authorship Graph including vertices with no edges

The general shape of both graphs is skewed to the right.

PA Graph

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

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

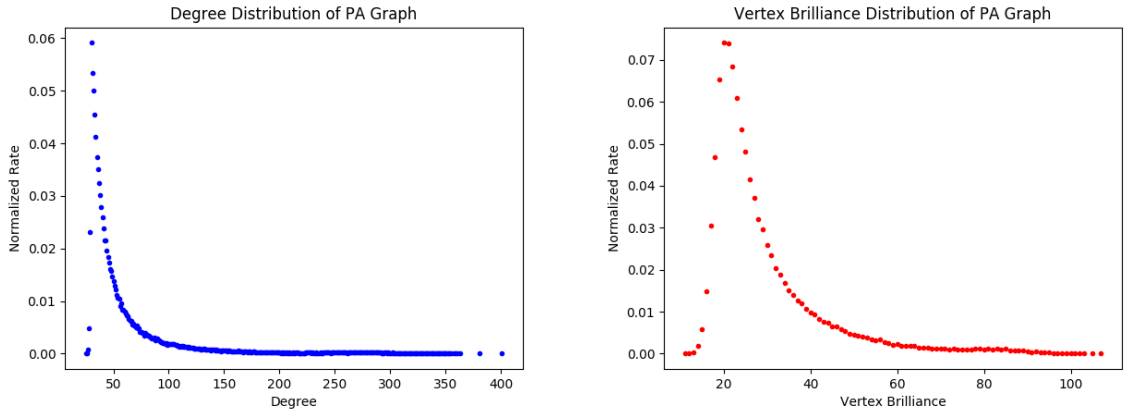


Figure 5: PA Graph parameters: m = 1559, m = 30

Both distributions has been averaged over 100 instances of the PA graph. The resulted graphs for both degree and vertex brilliance distributions are skewed to the right. The co-authorship graph strongly resembles the PA graph based on the 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 are similar to the distributions with the paramaters used here (in terms of shape).

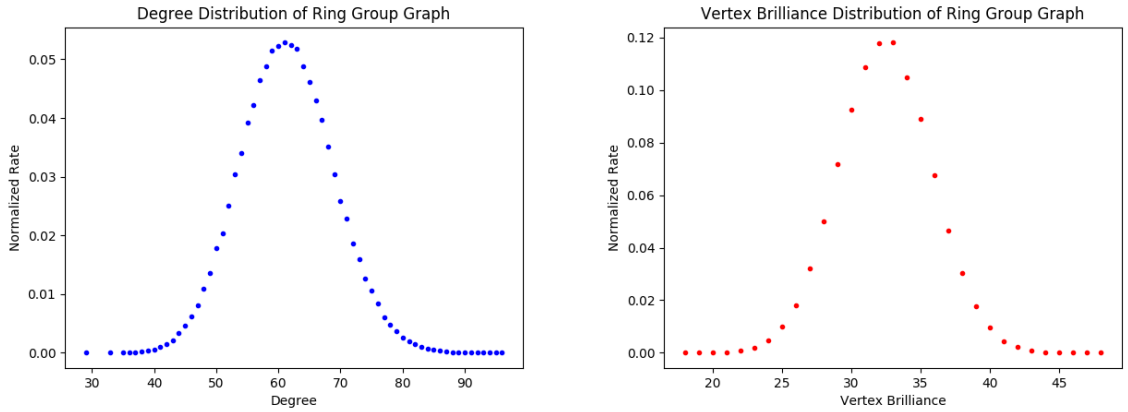


Figure 6: Ring Group Graph parameters: m = 60, k = 26, p = 0.22, q = 0.03

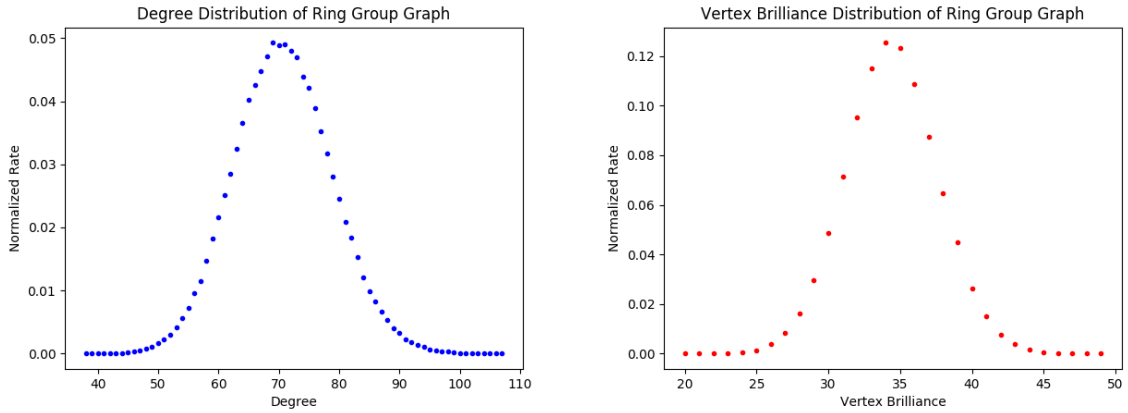


Figure 7: Ring Group Graph parameters: m = 120, k = 13, p = 0.26, q = 0.04

Both distributions are bell shaped (binomial distribution).

Based on the previous graphs, 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

- describe an algorithm for searching in the graphs. (I expect that you will have different algorithms for the different types of graphs.)
- You should explain why you believe your strategy might be effective and implement and test it on many instances. You can choose the parameters yourself as long as you are not perverse for example, the groups in the Ring Group Graph should not be of size 1 or n and it is acceptable that all your testing for a particular type of graph is on instances generated with the same parameters.
- As searching on graphs that are not connected can be impossible, you should choose parameters so that the graphs are very likely to be connected.

- Plot search time against the number of instances that achieve that time. Comment on your plots. It is acceptable to estimate search time by looking at only a sample set of pairs of vertices.
- Credit will be given for the effectiveness of the algorithm you design, and also, independently, for your explanation of the rationale behind the design.

Search Random Graph

Search Algorithm

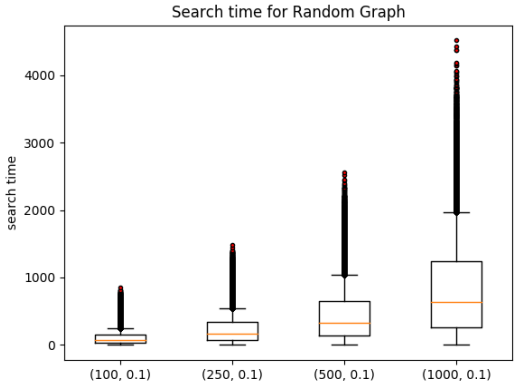
The search in a Random Graph was implemented as follow:

- 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

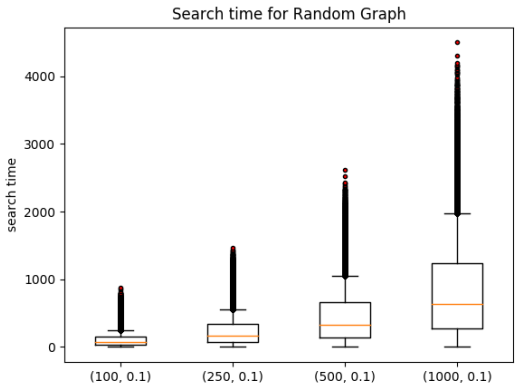
The algorithm above made use of the general knowledge of the graph i.e. total number of nodes (n) and the probability (p) that an edge exists between each pair of nodes in addition to the local knowledge i.e. vertex id and number of its neighbours to better find the target vertex (t). Every vertex had 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-**??** and figure-**??** 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) Only iterate until average number of neighbours



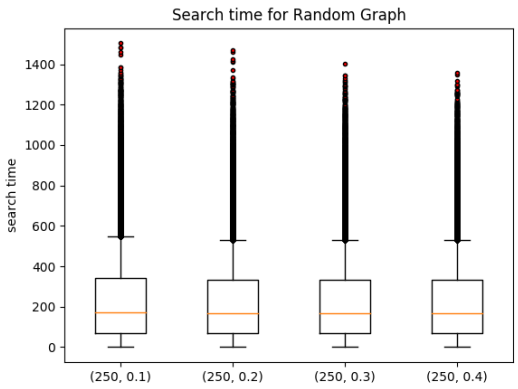
(b) iterate until the vertex's number of neighbours

Figure 8: Search Time Distribution for Random Graph when only n changes

Figure-**??** and figure-**??** 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-**??**, it did not significantly improved the search time (the difference if there is any is barely noticable from the box plots in both figures **??** and **??**)

box/box-random_graph-p-search_time_v1-av

(a) Only iterate until average number of neighbours



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

Figure 9: Search Time Distribution for Random Graph when only n changes

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

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 that the search in a Ring Group graph was implemented as follows:

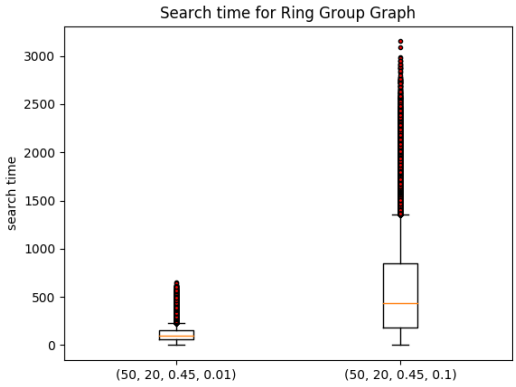
- Inputs:
  - The graph
  - The graph parameters - (m, k, p, q)
  - The start vertex id - (numerical id, group)
  - The target vertex id - (numerical id, group)
- Initialize a variable for search time and set it to 0
- Create a variable for the probability to move to a vertex that is closer to the target's 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; this check is to make 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
- 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 (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)
    - else if the neighbour's group is closer to the target than the current vertex 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, else move to the last queried neighbour

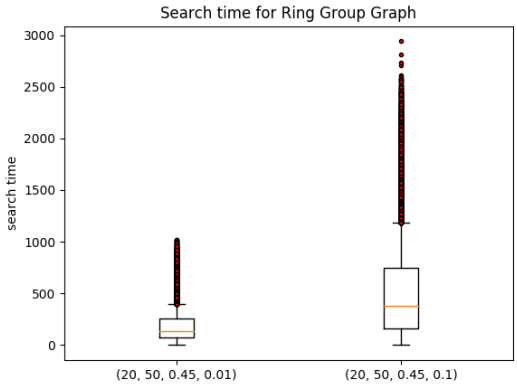
The assigned value for prob_close was determined through trial and error and proved to provide the best results (figure-**??** and figure-**??**).

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:

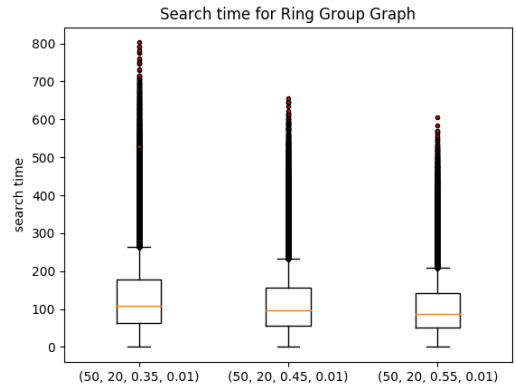


(a) without the probability to move to a closer neighbour nor reset p value when p is more than 0.5

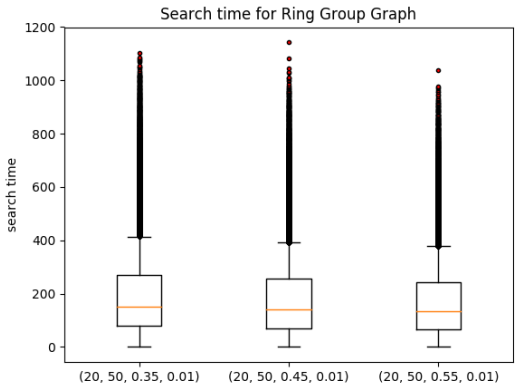


(b) with the probability to move to a closer neighbour set to p and reset p value when p is more than 0.5

Figure 10: Ring Group Graph parameters: m = 20, k = 50

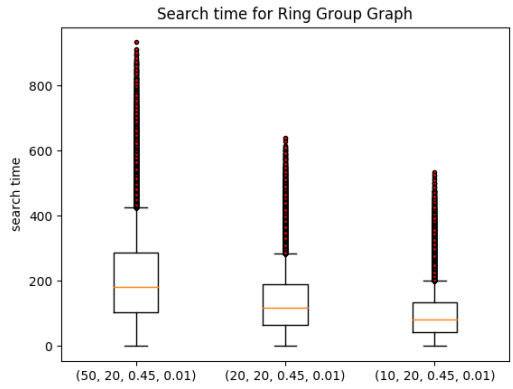


(a) Search Time Distribution with changing p

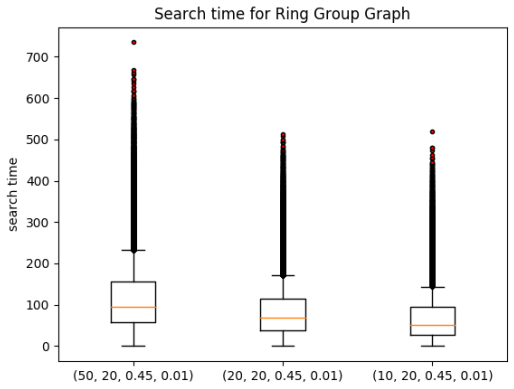


(b) Search Time Distribution with changing p

Figure 11: Ring Group Graph parameters: q = 0.05

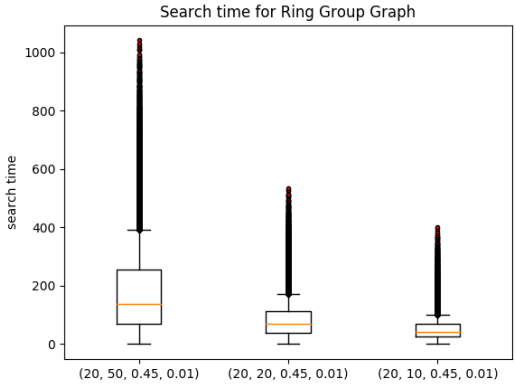


(a) without the probability to move to a closer neighbour

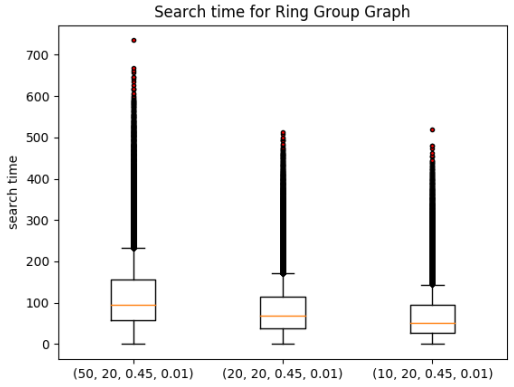


(b) with the probability to move to a closer neighbour set to p

Figure 12: Ring Group Graph parameters: m = 20, k = 50

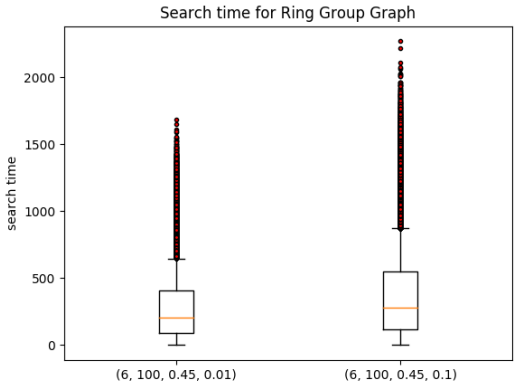


(a) Search Time Distribution with changing k

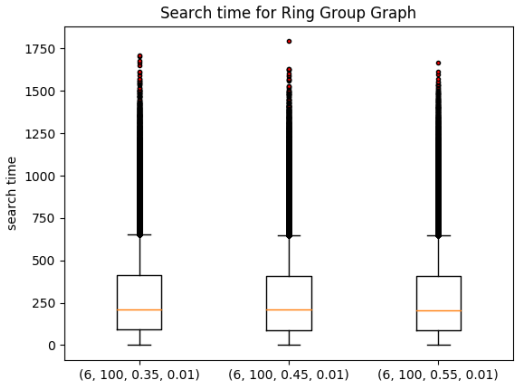


(b) Search Time Distribution with changing m

Figure 13: Ring Group Graph parameters: p = 0.45, q = 0.01



(a) Search Time Distribution with changing q



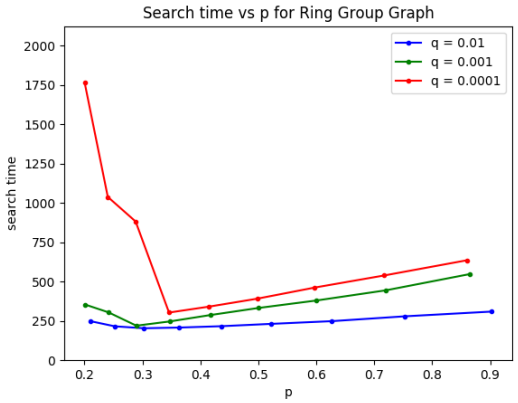
(b) Search Time Distribution with changing p

Figure 14: Ring Group Graph parameters: m = 6, k = 100

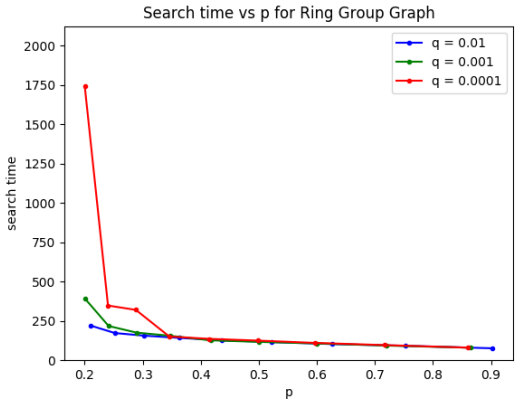
The paramaters picked for the ring group graph instances tested here were based on the results in

Figure-?? and Figure-??. The figures demonstrate the effictiveness of the algorithm by looking into relationship between the search time and the probability p (groups are more connected) tested over low q probabilities (less chance of shortcuts appearing the graph). The algorithm performs better with greater m values and greater p values.

Appendix

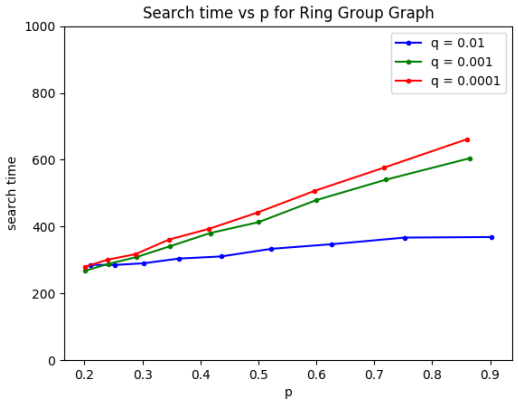


(a) without the probability to move to a closer neighbour nor reset p value when p is more than 0.5

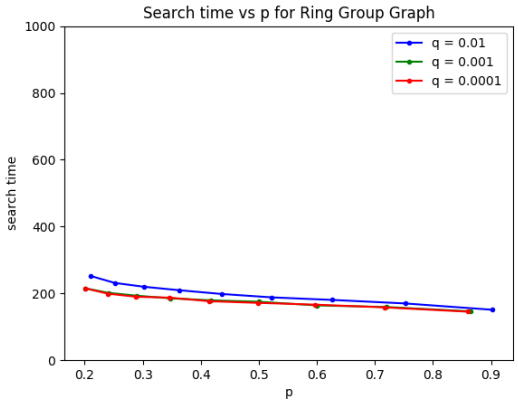


(b) with the probability to move to a closer neighbour set to p and reset p value when p is more than 0.5

Figure 15: Ring Group Graph parameters: m = 50, k = 20



(a) without the probability to move to a closer neighbour nor reset p value when p is more than 0.5



(b) with the probability to move to a closer neighbour set to p and reset p value when p is more than 0.5

Figure 16: Ring Group Graph parameters: m = 20, k = 50