Stockholms
universitet

Navigability of configuration model graphs

Alexander Käll

Kandidatuppsats i matematisk statistik
Bachelor Thesis in Mathematical Statistics

Kandidatuppsats 2023:8
Matematisk statistik
Juni 2023

www.math.su.se

Matematisk statistik
Matematiska institutionen
Stockholms universitet
106 91 Stockholm

Matematiska institutionen

Matematisk statistik
Stockholms universitet
Kandidatuppsats **2023:8**
http://www.math.su.se/matstat

Navigability of configuration model graphs

Alexander Käll*

June 2023

Abstract

This thesis explores the navigability of random graphs, in particular how an algorithm using only local information performs in configuration model random graphs with either power-law or constant degree distributions. The navigability of these graphs is measured by the average number of steps the algorithm takes and its failure rate. We find that the navigability of a graph seem to depend on average degree but also on how these degrees are distributed. The average number of steps the algorithm takes seems inversely proportional to the average degree.

*Postadress: Matematisk statistik, Stockholms universitet, 106 91, Sverige. E-post: alexanderkall42@gmail.com. Supervisor: Daniel Ahlberg, Johannes Heiny.

Abstract

This thesis explores the navigability of random graphs, in particular how an algorithm using only local information performs in configuration model random graphs with either power-law or constant degree distributions. The navigability of these graphs is measured by the average number of steps the algorithm takes and its failure rate. We find that the navigability of a graph seem to depend on average degree but also on how these degrees are distributed. The average number of steps the algorithm takes seems inversely proportional to the average degree.

Acknowledgements

I want to thank my supervisors Daniel Ahlberg and Johannes Heiny for helping me with everything from choosing a topic to reading drafts and giving feedback throughout the process of writing this thesis.

Contents

1 Introduction

In the late 1960s Stanley Milgram performed his famous experiment where participants were instructed to get a package to a final recipient by sending it a someone they knew on a first name basis who they thought had a better chance of getting the package to its destination [5]. He found that of the packages who made it the average number of steps was around six. This inspired the idea of six degrees of separation. Later Jon Kleinberg said that the takeaway from Milgram's experiment should not only by the short length of the paths the packages took but also the fact that they even made it [3]. Not only were the paths short but they were also easy to find. A network is said to be navigable if it has these two properties: short average path length and these paths being efficiently found by some algorithm using only local information. Here navigability will mean that a specific algorithm using only local information can find short paths between two vertices. The algorithm used in this thesis is one where two people (or two vertices on a graph) try and find a path to each other by going to their most popular friend, hoping they will have a better chance of finding a path and so on until either a path is found or some exit condition is met. This thesis explores the navigability of random graphs by simulating this algorithm and counting the number of steps it takes.

This thesis will attempt to answer the following questions: How well does a certain algorithm perform in some random graphs and what factors affect its performance?

2 Random graphs

A graph is a pair $G = (V, E)$ where V is a set of vertices and E is a set of paired vertices called edges. The degree of a vertex v is the number of edges v is a part of, denoted as $deg(v)$. Graphs can represent things like people connected by friendships or geographical locations connected by roads. Two vertices are connected if there exists a path, which is a sequence of edges, between them. A graph is connected if all its vertices are connected, or in other words the graph consists of one component where any vertex can be reached from any other by traveling along the edges. A graph can be used to model many things including social networks, road networks, the internet and much more. In the case of a social network the vertices could represent people and the edges friendships between them or for a road network vertices could represent intersections and edges roads.

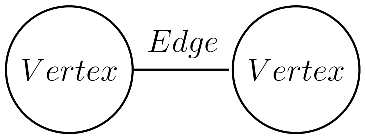
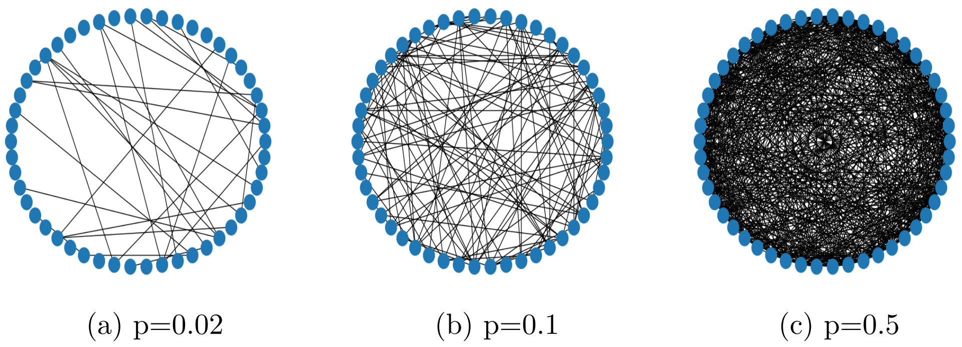


Figure 1: Example of a graph.

Graphs can be constructed in many different ways. A graph representing a road network can be created using actual road data but a graph can also be constructed using random processes. That's called a random graph. There are many different ways of generating random graphs and two of them will be covered here.

2.1 Erdös-Renyi model

The Erdös-Renyi model is a model for generating random graphs. A graph $G(N, p)$ is generated by first creating N vertices after which each possible edge is included with probability p, independent of all other edges.



(a) p=0.02 (b) p=0.1 (c) p=0.5

Figure 2: Erdös-Renyi model graphs with different edge probabilities, made with Python and the networkx library.

The degree of a vertex v is the number of edges where v is one of the vertices. The probability for a vertex having a certain degree is given by a binomial distribution [4], specifically

$$P(deg(v) = k) = \binom{N-1}{k} p^k (1-p)^{N-1-k}.$$

This makes the Erdös-Renyi model inflexible since its degree distribution cannot be any distribution but has to be a binomial distribution. This is where the configuration model becomes interesting since it allows you to choose any degree distribution.

2.2 Configuration model

The configuration model is another model for generating random graphs, but unlike the Erdös-Renyi model we start with a set of vertices $\{1, 2, 3, ..., N\}$ and a degree sequence $d = (d_1, d_2, ..., d_N)$. From this we can create stumps, which are edges not yet connected to their second vertex, corresponding to the degree of each vertex. Edges are then created by choosing two stumps uniformly at random and combining them to form an edge. This is done until there are no more stumps left.

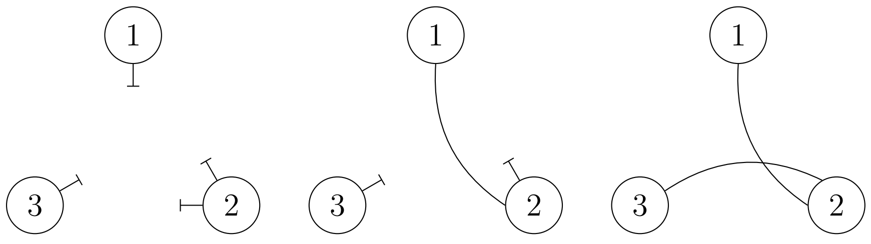


Figure 3: Generating a configuration model graph. Stubs are connected in pairs until none are left. In this case the graph is connected since all vertices can reach all other vertices.

What makes the configuration model unique is that the degree sequence can be sampled from any distribution, as long as the total degree is a multiple of 2. This condition is used to guarantee that all stumps become part of an edge. One drawback with this model is that it doesn't have to produce a simple graph, which is a graph without self-loops and multiple edges (more than one edge connecting two vertices).

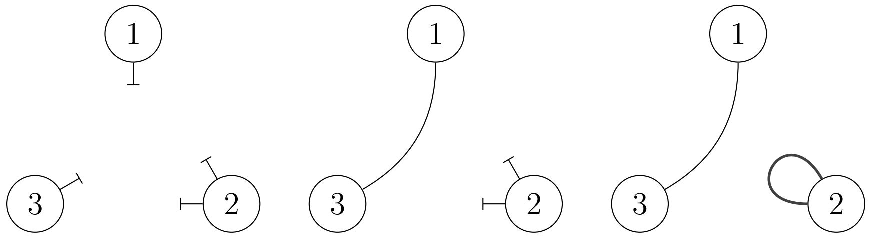


Figure 4: Generating a configuration model graph. Here 1 and 3 are paired which means 2 is left with a self-loop, creating a multigraph that is not connected.

Instead it creates a multigraph which allows for self-loops and multiple edges. These are less important for the simulations than that the graphs are connected since self-loops and multiple edges can simply be removed. This will mean that the graphs used in the simulation will not have their intended degree sequence exactly. The configuration model also does not have to produce graphs that are connected. This is of more concern since the algorithm that is used in the simulations makes no sense if a path between the starting vertices does not exist. A minimum degree of 2 and a maximum degree greater than 2 is therefore used to ensure that the graphs have a high probability of being connected [2]. Some more notation that will be useful later is the average degree $\langle k \rangle$ and the total degree $L = \sum_{n=1}^{N} d_n = \langle k \rangle N$. Since the configuration model can take on any degree distribution its properties very much depend on the degree distribution chosen.

3 Power-law distributions

Many real world networks follow a power-law distribution. One famous example is Zipf's law [6] which says that the frequency of a word is inversely proportional to its rank if ordered by frequency. Zipf's law has been observed in books in many languages. A power-law degree distribution means that

$$P(deg(v) = k) \propto k^{-\gamma}, k = 2, 3, 4, ...$$

This distribution has unbounded support but for a degree distribution it is practical to restrict the support to $[2, 3, 4, ..., N-1]$ where N is the number of vertices. The lower limit because it will produce connected graphs with high probability [2] and the upper limit because if a vertex has higher degree than other vertices in the graph it must have multiple edges, which together with self-loops will be removed for the simulations. The degree of a vertex is a whole number but since the sampling makes use of a continuous distribution we start with a continuous power law distribution. Consider a random variable X with density $f(x) = Cx^{-\gamma}$ defined for some interval $x \in [x_0, x_1]$ where C is a normalizing constant. Since $f(x)$ is a probability density function, integrating it over the support $[x_0, x_1]$ should yield 1

$$\int_{x_0}^{x_1} Cx^{-\gamma} dx = \frac{C}{1-\gamma}(x_1^{1-\gamma} - x_0^{1-\gamma}) = 1$$

which after rearranging gives

$$C = \frac{1-\gamma}{x_1^{1-\gamma} - x_0^{1-\gamma}}.$$

This gives us a proper probability density function. The expected value of $X \sim f(x)$ is

$$E[X] = \int_{x_0}^{x_1} xf(x)dx = \int_{x_0}^{x_1} Cx^{1-\gamma} = \frac{C}{2-\gamma}(x_1^{2-\gamma} - x_0^{2-\gamma}) \tag{1}$$

which is only well defined for $\gamma > 2$. The expected value is undefined for $\gamma = 2$ and for $\gamma < 2$ the expected value becomes less than or equal to 2 when $x_0 = 2, x_1 = 999$.

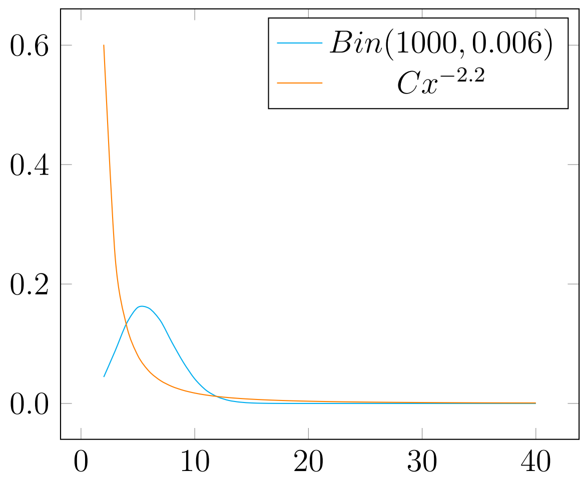


Figure 5: Comparison between a binomial distribution and a power-law distribution. This is meant to illustrate the difference between a binomial degree distribution and a power-law degree distribution with similar mean. Both distributions use $x_0 = 1$ and $x_1 = 1000$.

As figure 5 shows, a binomial degree sequence will likely be very different from a power-law degree sequence, even if they have similar expected degree.

3.1 Sampling

Most programming languages provide random sampling from a uniform distribution. This can be used in order to sample from any continuous distribution. Let $Y \sim Uniform(0,1), X \sim Cx^{-\gamma}$ and

$$T(y) = F_X^{-1}(y) = (y(x_1^{1-\gamma} - x_0^{1-\gamma}) + x_0^{1-\gamma})^{\frac{1}{1-\gamma}}. \tag{2}$$

Then it holds $T(Y) \overset{d}{=} X$. This is shown using inverse transform sampling which says that

$$F_X(x) = P(X \leq x) = P(T(Y) \leq x) = P(Y \leq T^{-1}(x)) = T^{-1}(x)$$

which together with

$$F_X(x) = \int_{x_0}^{x} Cx^{-\gamma} dx = \frac{C}{1-\gamma}(x^{1-\gamma} - x_0^{1-\gamma})$$

is enough to show (2). This method produces a continuous distribution which needs to be converted into a discrete distribution in order to be useful as a degree distribution. This is best done by rounding to the closest integer [1], or more formally

$$P(X = k) = \int_{k-0.5}^{k+0.5} f(x)dx$$

where X is the random variable being sampled from when generating a degree sequence d. You also need to choose values for the upper and lower bounds, x_0 and x_1. The lower bound can be as low as 1 but in order to reliably create connected graphs it is set to 2. The upper bound can be infinity but for practical purposes it is set to one less than the number of vertices.

4 Path finding

There are many algorithms that could be used when testing navigability but most interesting are ones using only local information. An algorithm using only local information means that it does not have an overview of the entire graph but instead sees only the vertices it visits and maybe their neighbors. Kleinberg [3] describes what he calls a decentralized algorithm which at each step only has limited information about the other vertices. The algorithm he describes also involve information about an underlying structure which could be thought of as geographical locations. The reason for using an algorithm like this is that they are not guaranteed to find the shortest path. The number of steps they take is very much dependant on the type of graph it is running on and therefore says a lot about the graph itself. This thesis uses an algorithm where two vertices try to find a path to each other by going to their respective neighbor with the most neighbors.

4.1 Algorithm

Step 1. Start with two vertices, i and j.

Step 2. Order the neighbors of i by number of neighbors.

Step 3. Go to i's neighbor with the highest number of neighbors if there is only one. Otherwise choose one with the highest number of neighbors uniformly at random and set i to be this new vertex.

Step 4. Check if i and j share an edge, stop if yes, continue if no.

Step 5. Order the neighbors of j by number of neighbors.

Step 6. Go to the j's neighbor with the highest number of neighbors if there is only one. Otherwise choose one with the highest number of neighbors uniformly at random and set j to be this new vertex.

Step 7. Check if i and j share an edge, stop if yes, continue if no.

Step 8. Repeat steps 2-7 until i and j share an edge or an exit condition is met.

The exit condition used is either that all neighbors of i or j have already been visited (the first exit condition) or that the number of steps reaches a preset maximum (the second exit condition).

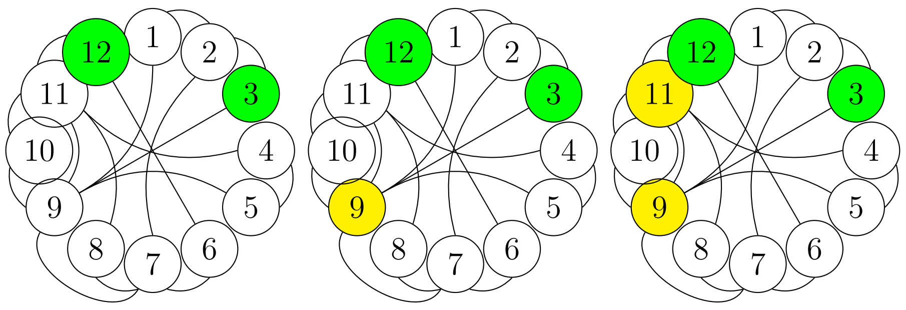


Figure 6: Example run of the algorithm

Figure 6 shows an example of the algorithm on a small graph. 12 and 3 are the chosen starting vertices and since they do not share an edge the algorithm keeps going. 3 goes to 9 since 9 has the most edges of 3:s neighbors. 12 and 9 does not share an edge so 12 goes to 11 which does share an edge with 9 and a path has been found, namely $12 \rightarrow 11 \rightarrow 9 \rightarrow 3$.

5 Constant degree graphs

How an algorithm like this might perform is is highly dependent on the degree sequence of the network. To gain some insight into this, let's first consider a simpler case where the degree sequence is a constant c. In this case the algorithm is identical to one where instead of choosing the neighbor with the most neighbors a random neighbor is chosen. Let S be the number of steps for the algorithm to complete. Then the expected number of steps is

$$E[S] = \sum_{n=1}^{\infty} nP(S = n)$$

where $P(S = n)$ is the probability that that the algorithm finishes in n steps. The probability that the algorithm completes in one step is the same as the probability that two vertices are connected which is

$$P(i \textit{ and } j \textit{ share an edge}) = \frac{d_i d_j}{L-1} \approx \frac{d_i d_j}{L} = \frac{c^2}{cN} = \frac{c}{N}.$$

This is true since vertices i and j have d_i and d_j stumps respectively which means that they can share an edge in $d_i d_j$ different ways and the probability of each of these edges being formed is $\frac{1}{L-1}$ since every stump can be combined with $L - 1$ other stumps. The probabilities of the algorithm taking more than one step can be approximated using the fact that after taking a step it is almost as if the algorithm is in the same position as when it started. For it to take two steps the first two vertices can't be connected but after the first step i and j should be connected. This means that

$$P(S = 2) \approx (1 - \frac{c}{N})\frac{c}{N}$$

and in general

$$P(S = s) \approx (1 - \frac{c}{N})^{s-1}\frac{c}{N}.$$

Since this assumes that going to a random neighbor is the same as going to any random vertex is the same these results are approximate. However, this can now be used to estimate the expected number of steps

$$E[S] \approx \sum_{n=1}^{\infty} nP(S = n) = \sum_{n=1}^{\infty} n(1 - \frac{c}{N})^{n-1}\frac{c}{N}$$

which can be calculated using

$$\sum_{n=1}^{\infty} nr^{n-1} = \frac{1}{(1-r)^2}$$

which yields

$$E[S] \approx \frac{N}{c}. \tag{3}$$

What this fails to take into account is, as previously mentioned, that which vertices are visited is dependent on the ones already visited. This should mean that the algorithm takes more steps than this would predict. For a network created using a power-law degree sequence the number of neighbors a vertex is expected to have increases with the number of steps the algorithm takes since it chooses the neighbor with the most neighbors. A similar calculation could be attempted for graphs with power-law degree sequences but is not in this paper.

6 Simulation

The simulations were made using python and the networkx library, a library made for creating and working with graphs in python. Configuration model graphs were created using the libraries built in configuration model generator. I implemented the algorithm described in chapter 4 in python and equation (2) which was used in order to sample from a power-law distribution. After each graph was created self-loops and multiple edges were removed.

6.1 Simulation 1

The first simulation was made by creating configuration model graphs with power-law degree sequences where $\gamma = [2.1, 2.2, ..., 3.5]$. These values where chosen since the range $2 < \gamma < 3$ is commonly studied [1] with some extra values at the end. Ten graphs with $N = 1000, x_0 = 2, x_1 = 999$ were created for each γ and the algorithm ran 100 times on each graph and the average path length was recorded. The two starting vertices were chosen uniformly at random. The first exit condition was used.

6.2 Simulation 2

The second simulation was made to compare different values for γ but where the average degree $\langle k \rangle$ is kept the same. This was achieved by varying x_0

since $\langle k \rangle$ depends on x_0. The values used for x_0 were 2,3,4 and 5. The corresponding value of γ is then calculated by choosing one of them, in this case $\gamma = 2.33$ was chosen for $x_0 = 2$ and the rest where calculated using (3) which gave $\gamma = 2.6$ for $x_0 = 3$, $\gamma = 3$ for $x_0 = 4$ and $\gamma = 3.67$ for $x_0 = 5$. The value $x_1 = 999$ was used for all γ. The algorithm was run on 10 graphs for each γ and 100 times on each graph. The two starting vertices were chosen uniformly at random. The first exit condition was used.

6.3 Simulation 3

The third simulation was made by creating configuration model graphs with constant degree sequences with $c = [3, 4, ..., 12]$. These values of c where chosen to make graphs with similar average degrees as the ones in the first simulation. Ten graphs with $N = 1000, x_0 = 2, x_1 = 999$ were created for each c and the algorithm ran 100 times on each graph. This was repeated for $N = 2000$. The two starting vertices were chosen uniformly at random. The second exit condition was used with a maximum number of steps of 2000.

6.4 Simulation 4

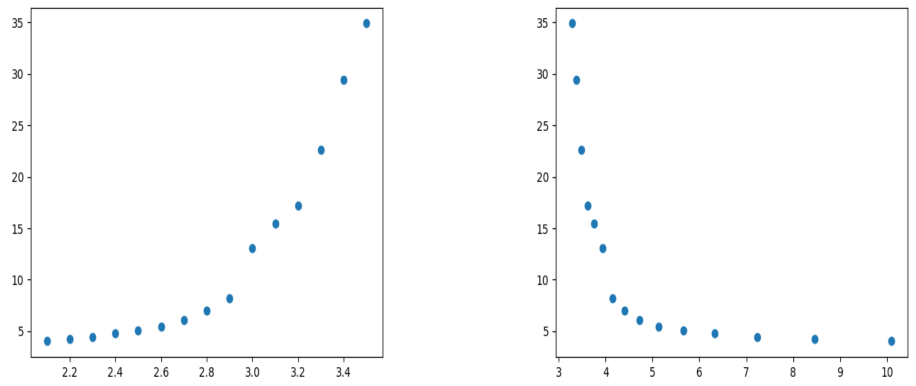
The fourth simulation was made the same way as the first ($\gamma = [2.1, 2.2, ..., 3.5]$, $N = 1000, x_0 = 2, x_1 = 999$) with the difference that the starting vertices were not chosen at random. Instead ten vertices with the highest degrees were found and the starting vertices were chosen uniformly at random from these ten. When the starting vertices are chosen uniformly at random from all vertices the ones with high degree will rarely be chosen so this simulation investigates if choosing starting vertices with high degree affects the algorithm's performance. The algorithm was run on 10 graphs for each γ and 100 times on each graph. The first exit condition was used.

6.5 Simulation 5

The fifth simulation was made in exactly the same way as the first but using $\gamma = [1.1, 1.2, 1.3, ..., 3.5]$.

7 Discussion and Results

7.1 Simulation 1



(a) Average number of steps vs power-law exponent

(b) Average number of steps vs average degree

Figure 7: Average number of steps the algorithm takes in power-law configuration model graphs with different values of γ. Figure 7a shows the average number of steps plotted against γ. Figure 7b shows the same simulation as figure 7a but the average number of steps is plotted against the average degree of the graphs for each γ.

The average number of steps the algorithm takes increases with γ as seen in figure 7a and decreases with average degree $\langle k \rangle$ as seen in figure 7b. Two main factors are thought to be causing this: lower values of γ lead to degree sequences which is more beneficial for finding shorter paths and lower values of γ produce graphs with higher expected average degree as seen in (1). Since different values of γ produce graphs with different expected average degrees it is difficult to say from the first simulation alone which if factor affects the average number of steps more. Intuitively it seems as though average degree has to affect the average number of steps the algorithm takes since it means that every vertex on average has more neighbors. In order to measure the effect γ has on the number of steps the algorithm takes without taking into account average degree the second simulation was performed. The failure rate, which is the percentage of time the algorithm reaches an exit condition before finding a path, starts reaching levels above 0% around $\gamma \geq 3$ seen in table 1.

7.2 Simulation 2

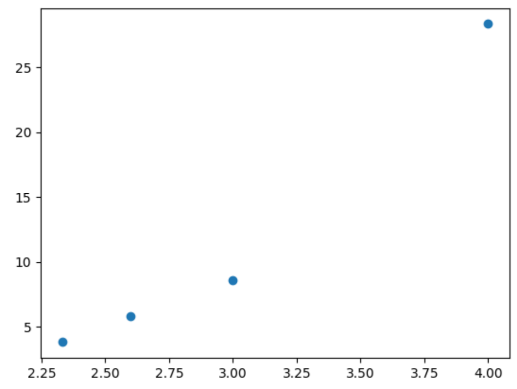
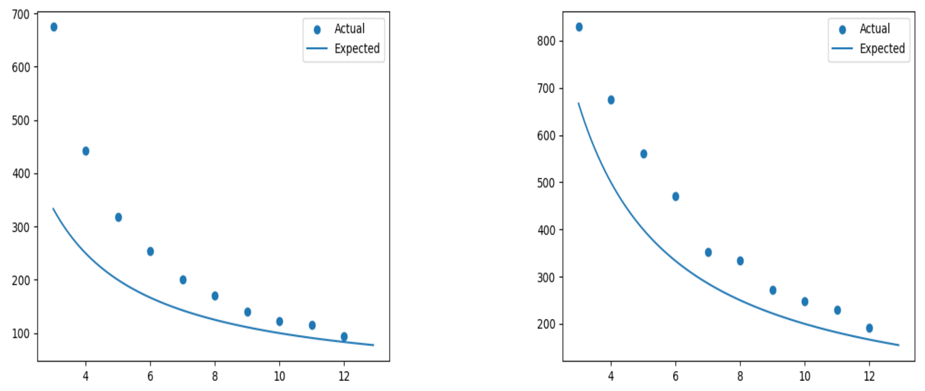


Figure 8: Average number of steps the algorithm takes in simulation 2 vs γ.

To measure the effect of γ on the number of steps the algorithm takes without average degree being a factor configuration model graphs were created where the different γ had different lower bounds x_0 so they would have the same expected average degree. Figure 8 shows that the average number of steps the algorithm takes increases with γ which means that the value of γ still affects the average number of steps the algorithm takes even when the average degree $\langle k \rangle$ is kept the same. This indicates that for a given average degree, this algorithm performs better when the degrees are divided less evenly among vertices than when most vertices have degrees closer to the average.

7.3 Simulation 3

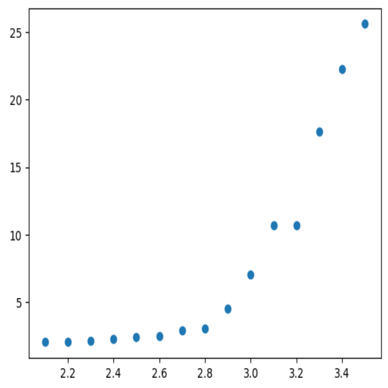


(a) Average number of steps vs degree constant c, $N = 1000$ (b) Average number of steps vs degree constant c, $N = 2000$

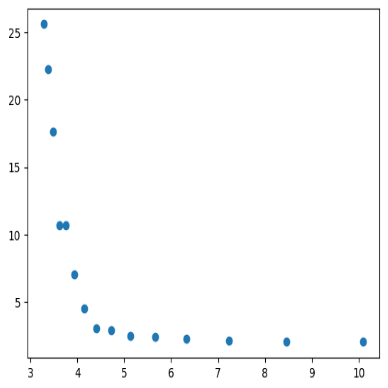
Figure 9: Average number of steps the algorithm takes, both actual simulated values and estimated expected values using equation (3). Figure 9a shows this for graphs with 1000 vertices and figure 9b for graphs with 2000 vertices.

To gain insight into the performance of the algorithm on power-law configuration model graphs it was simulated on configuration model graphs with constant degree. Power-law graphs become closer and closer to constant degree graphs the higher the value of γ since more degrees closer to the minimum degree become more likely and degrees further away become less likely. It is also the case that estimating the expected number of steps analytically is significantly easier for constant degree graphs and even tough they are very different from power-law graphs the relation between the number of steps the algorithm takes and the average degree $\langle k \rangle / c$ looks very similar. This indicates that the expected number of steps the algorithm takes in power-law graphs is proportional to $\frac{1}{\langle k \rangle}$. The percentage of time the algorithm reaches the used exit condition, or the failure rate, was rather high, especially for lower values c which can be seen in tables 3 and 4. This means that in a significant proportion of runs the algorithm would have taken more than 2000 steps if allowed to continue. The average number of steps shown in figure 9 should therefore be lower than if a higher maximum number of steps was used, at least for $c = 3$ and $c = 4$.

7.4 Simulation 4



(a) Average number of steps vs power-law exponent



(b) Average number of steps vs average degree

Figure 10: Average number of steps the algorithm takes in power-law configuration model graphs with different values of γ where the starting vertices where chosen from the vertices with the highest degree.

In the first three simulations the two starting vertices where chosen uniformly at random between all vertices. The fourth simulation explored the algorithms performance for more specific choices of starting vertices, more specifically when the two starting vertices where chosen uniformly at random among the ten vertices with the highest degree. When choosing them as done in the first three simulations vertices with high degree are less likely to be chosen since they are more rare. The average number of steps the algorithm takes increases with γ, shown in figure 10a, and decreases with $\langle k \rangle$, shown in figure 10b. These plots show similar relations as figures 7a and 7b but with a lower average number of steps for a given value of γ or $\langle k \rangle$. This is intuitive since two starting vertices with high degrees are more likely to share an edge than two starting vertices with low degrees. More interesting is that the relationship between average number of steps and $\gamma, \langle k \rangle$ is similar and that graphs with higher values of γ still need a large number of steps to find a path.

7.5 Simulation 5

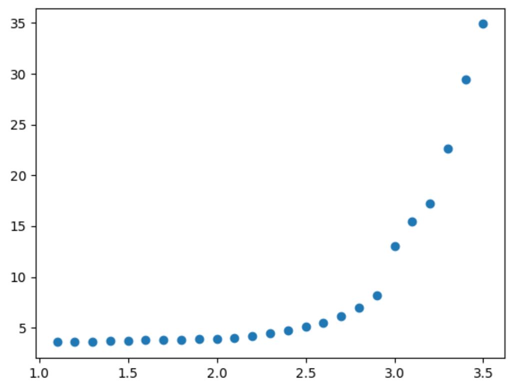


Figure 11: Average number of steps the algorithm takes in power-law configuration model graphs for different values of γ.

When analyzing power-law distributions γ is often between 2 and 3, but when simulating γ can be lower than 2. It cannot however equal 1 since that would cause a division by 0 error when sampling using equation (2) so values of $\gamma = [1.1, 1.2, ..., 3.5]$ where simulated to see if the average number if steps the algorithm takes for the lower levels of γ follows the same pattern as the higher values used in simulation 1. Figure 11 shows that this is indeed the case. The average number of steps seem to hit a lower bound at around $\gamma = 2$. This could be because high degrees are quite likely for these values of γ meaning that even if the starting vertices have a low degree, their neighbors have high degrees and are likely to share an edge. Since a minimum degree of $x_0 = 2$ was used the lower values of *gamma* will produce graphs with higher average degree with and no drawbacks in terms of something that could negatively affect the algorithms performance. This is difficult to do anything about because the reason $x_0 = 2$ was used is that it produces connected graphs.

8 Conclusions

Navigability of random graphs seems highly connected to average degree, at least for the graphs used in the simulations. More edges mean that the algorithm can find shorter paths. This is the case both for graphs with power-law degree distributions and with constant degree distributions. The navigability of a network also depends on the distribution of the degrees, even if the average degree is similar. This can be seen in figure 8 and when comparing figure 7b with figure 9a. The simulations done in this thesis used rather small graphs with $N = 1000$ or $N = 2000$. The performance of the algorithm could be very different for larger graphs, maybe it cannot find paths at all for large enough graphs or maybe it is just very slow in finding them. It would be interesting to see how the algorithm performs on graphs of different size when the support of the degree distribution is kept the same.

9 Appendix

In the following tables failure rate is calculated as the number times the algorithm reaches the exit condition used in that specific simulation divided by the total number of times the algorithm ran.

γ	2.1	2.2	2.3	2.4	2.5	2.6	2.7	2.8
Failure rate	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
γ	2.9	3.0	3.1	3.2	3.3	3.4	3.5	
Failure rate	0.0	0.021	0.069	0.149	0.113	0.202	0.247	

Table 1: Failure rate for each γ, from simulation 1

γ	2.33	2.6	3	3.67
Failure rate	0.0	0.0	0.0	0.0

Table 2: Failure rate for each γ, from simulation 2

c	3	4	5	6	7	8	9	10	11	12
Failure rate	0.145	0.023	0.004	0.001	0.0	0.0	0.0	0.0	0.0	0.0

Table 3: Failure rate for each c, from simulation 3, N=1000

c	3	4	5	6	7	8	9	10	11	12
Failure rate	0.359	0.141	0.044	0.020	0.013	0.001	0.002	0.0	0.0	0.0

Table 4: Failure rate for each c, from simulation 3, N=2000

γ	2.1	2.2	2.3	2.4	2.5	2.6	2.7	2.8
Failure rate	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
γ	2.9	3.0	3.1	3.2	3.3	3.4	3.5	
Failure rate	0.003	0.012	0.024	0.037	0.072	0.112	0.162	

Table 5: Failure rate for each γ, from simulation 4

γ	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8
Failure rate	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
γ	1.9	2.0	2.1	2.2	2.3	2.4	2.5	2.6
Failure rate	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
γ	2.7	2.8	2.9	3.0	3.1	3.2	3.3	3.4
Failure rate	0.0	0.003	0.019	0.032	0.039	0.108	0.172	
γ	3.5							
Failure rate	0.206							

Table 6: Failure rate for each γ, from simulation 5

References

[1] Aaron Clauset, Cosma Rohilla Shalizi, and Mark EJ Newman. Power-law distributions in empirical data. *SIAM review*, 51(4):661–703, 2009.

[2] Svante Janson and Malwina Luczak. A new approach to the giant component problem, 2007.

[3] Jon Kleinberg. Complex networks and decentralized search algorithms. In *Proceedings of the International Congress of Mathematicians (ICM)*, volume 3, pages 1019–1044. Citeseer, 2006.

[4] M. E. J. Newman, S. H. Strogatz, and D. J. Watts. Random graphs with arbitrary degree distributions and their applications. *Phys. Rev. E*, 64:026118, Jul 2001.

[5] Jeffrey Travers and Stanley Milgram. An experimental study of the small world problem. In *Social networks*, pages 179–197. Elsevier, 1977.

[6] George Kingsley Zipf. *Selected Studies of the Principle of Relative Frequency in Language.* Harvard University Press, Cambridge, MA and London, England, 1932.