Zachary Cain & Bryan Oswald

Sukumar Ghosh

P2P and Social Networks

December 9, 2015

Final Project Report

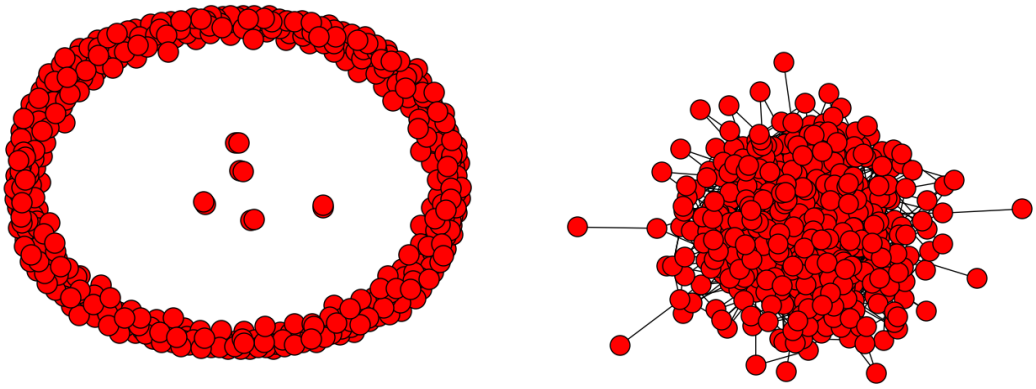
1. Introduction

Our project looked at Erdös-Rényi graphs with sizes ranging from 500 to 5000 nodes. Smaller than 500 nodes, we found that the graphs were not consistently connected, even when the probability p, used to generate the graph, is large. We examined some very large graphs, but above N = 5000 run-time became a major constraint. We looked at the structure of the graphs generated to see if it consisted of a single connected component, a giant component, or smaller trees. We also considered the diameter of the graphs when we found they were connected, comparing our results to the theoretical diameter distribution function. Next we wanted to take these graphs and have a population of target nodes ranging from 0.001 to 0.01. Then we ran three different search algorithms on each graph, recording results for each algorithm in terms of total nodes visited and time required before each of the algorithms found the target node. The three algorithms we decided to test were: gnutella-type flooding, random walker model and k-random walker model (with one-hop replication).

2. Our Approach and Results

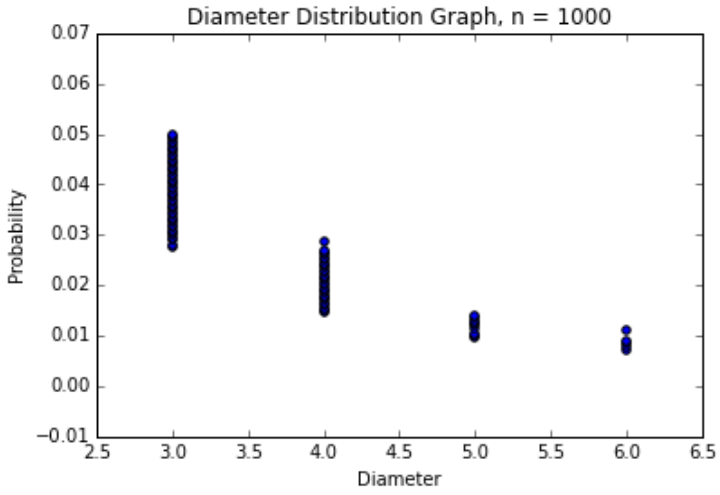
For this project we used networkx to generate our graphs and matplotlib to chart them. Before we dove deep into writing our code we wanted to see what Erdös-Rényi graphs looked like at different probabilities so we made some graphs with N = 500 and then looked at how they

changed over our probability range. When looking at graphs with p = 0.0001 the graph consisted primarily of single nodes with a few small connected components in the middle (Left).



Moving above p = ~0.001, we found the graph was usually a giant component with only a few unconnected nodes. The giant component size continued to grow until p = 0.012 where the graph was one single connected component (Right). This was what the graphs looked like for the majority of our results because connectivity increases based on N as well as p. The pictures above use our smallest value of N to display how the structure changes with p. These changes take place even more rapidly as N is increased. We used these charts to visualize how the diameter changed with different values of p, then compare charts with different values of N to see how the diameters changed at the same p values. Then we also wanted to compare these with theoretical results, which state that the diameter should be within the range: **MAX(log(N) / log(N*p)) - 4** to **MAX(log(n) / log(n*p)) +1** (N = number of nodes and p = probability). To measure this we generated 100 ER graphs, incrementing p each time. If the graph was connected, we add its diameter to the list, then plot the values. Running our code gave us the following results (Figure 1) and match up accordingly with the theoretical results (Table 1).

Figure 1 and Table 1



Probability	Expected Diameter	Our Diameter
0.04	-2 to 3	3
0.02	-1 to 4	4
0.009	0 to 5	5

As shown the diameters match up well with theoretical results, however the diameters always were on the higher end of the expected diameter (even at N = 5000). This is because we did not consider graphs which were not connected. Also the diameter is smaller at even probabilities with a graph with a higher N value. This is because there are more possible edges at this higher N value, as the **Number of Edges = N! / 2!(N-2)!** (Binomial coefficient). This can be seen in the figure to the left.

3. The Search Algorithms

The first algorithm we implemented followed the random walker model. This selects a node at random from the graph to use as a start point. It proceeds by selecting a random neighbor of the current node to visit next. If the current node has no neighbors the walker has not already visited, it terminates. At each node visited, the walker checks if the node has the object it is searching for. If the object was found then the nodes visited and elapsed time were returned. If the object could not be found then 0s were returned for both values.

For the k-random walker we first selected k random start node(s) (five was used for the value of k). If any of these nodes contain the object, return found. If not, mark each node as visited. Then check each node's neighbors to see if any of them contain the object (one-hop replication). If not, select a random, unvisited neighbor and repeat until a target node is found or there are no more possible steps. If the object was found we returned the number of nodes visited, the time it took to find, and the number of hops (by each walker) that were needed to find the target node. However if it failed to find the target node then zeros were returned for all three values.

The random walker results depend on the connectivity of the graph because it changes the paths available to each walker. Too few neighbors to choose from and a walker may run out of unvisited nodes on its path. We found that when $p > 0.01$, the graph was generally connected enough that the objects could consistently be found when at a high population density (0.07~0.01). However, if the population density was below 0.005, the random walker consistently failed to find a path to the object until $p > 0.02$. As population density approaches 0.001, the walker fails to find a path regardless of the p value. At lower density, the average case nodes visited is nearer the graph size, but at higher density most objects can be found in less than 100 steps for $N = 500$.

The k-random walker's results mirror the random walker. Random walker is essentially a depth-first search - you choose a single path and follow it to its end. K-random walkers does k of these in parallel, returning as soon as any of them finds the object. The largest difference is in the number of nodes visited, as well as k-random walker having a higher success rate at lower values of p. The largest number of nodes visited for k-walker is ~⅔ that of the individual walker.

Finally for the gnutella-type flooding we selected a random start node then check to see if this node is the target node. If not then we add all the neighbors into a list and remove the node we just looked at from that list. Then this process is repeated for the next node in the list until a target node is found or the time-to-live is met or there are no more possible next steps. If a target node is found then the nodes visited and the time are returned. However if a target node is not found, then both of these are returned as zeros.

The gnutella-type flooding results in terms of time when probability was 0.002 (N = 500) most results were under 0.01 seconds, with some outliers. Then when the probability was increased to 0.009 (N = 500) most results were under 0.0025 seconds, with some outliers. This was substantially faster time to find a target node! Then when looking at nodes visited with a population density of 0.003 compared to a population density of 0.01, the number of nodes visited before finding a target node was cut by around a third. When we implemented our flooding with no time-to-live, the flooding did much better at finding the target nodes at lower probability values, when compared to the random walker and k-random walker. As both the of these struggled to find the target node when the probability value dropped below 0.01. The flooding however did well until the probability value was below 0.005 (due to graph not being connected at this point). However it must be noted that the overhead of flooding without a ttl looking for a small number of target nodes is substantial.

4. Evaluations of the Search Algorithms

We found that the performance of each search algorithm depended heavily on both the structure of the graph being searched and the population density of objects placed in the graph. Flooding performs well when objects are plentiful, and a few queries will yield a result.

However, as N increases, the number of messages to send becomes overwhelming. Random walkers are good at finding sparse objects, but the graph must be connected enough that they do not run out of room to explore.

In general: as a graph becomes very densely connected, random walkers become more effective, and the overhead associated with flooding goes up. Then it depends on object density; if you can find an object within a couple hops then flooding is quick and simple, if you can't then the random walker's lower overhead for long paths becomes beneficial. Also it is important to note that one-hop replication helps the k-random walker in highly connected graphs, due to information replicated at each node. Determining which search to use requires global information of the network state, making judgment on this tradeoff difficult.

In addition, all of the work here assumes the network is static. In the real world that is not the case, and there are additional tradeoffs. In networks with high churn, neighbors may not be updated constantly. Flooding is slightly more resilient to churn (it will send to whoever listens) whereas random walkers performance can be affected substantially due to their advantages in sparsely connected networks; each edge has a higher significance to the overall network.

5. Conclusions and Further Work

The results for how the Erdös-Rényi graphs were connected was that at larger probabilities the graphs were a single connected component ($n= 500$, $p > 0.012$) and this was true at even lower probabilities, given a sufficiently large N. However at smaller probabilities ($N = 500$, $p < 0.005$) the graphs were mainly single nodes with a few small connected components. When N was increased, p would need to be much lower for this to remain true. We found that increasing N and decreasing p by the same factor kept graph structure fairly constant.

For the diameter the results were as expected when compared to theoretical results. It was interesting to see how small the diameter got after a very small increase in the probability at higher N values (N = 5000). This is because ER graphs include every node in the graph with an equal probability. Since for any node it is likely true that there are more nodes in the graph than in the node's neighborhood, a new edge is much more likely to be a bridge than a local connection.

Potential further work includes comparing search methods when used in different graph types, particularly with graphs following a power law degree distribution. It would also be interesting to simulate churn in the network and evaluate each algorithm's ability to respond. We primarily focused on the trade offs inherent in the algorithms based on changing graph structure. Further work could look more closely at performance in large networks, searching for objects at varying densities.