Social Networks: Structure and Performance

Over the last 2 decades, social networks have become an inescapable part of everyday modern life for almost everyone on this planet. They allow us to stay connected with our friends, family and anyone else in the world that we choose to connect with. With almost 8 billion people on this planet and many of them who have been building their personal social networks over the last ten years, how can we develop large social networks that are efficient and not memory intensive for their users. These factors are determined by the choices of data structures and the design of algorithms that are made whilst developing a social network. Certain structures may work efficiently with small networks, but perform poorly with larger networks, which is why every design choice is of extreme importance during development. In this report I will be investigating not only the behaviour of social networks, but also the effect of their structure on their overall performance. In order to achieve this, I will be executing simulations of a social network and altering the following variables: probability of liking a post, probability of following someone who posted, size of the network, number of posts (also who posted them) and choosing who is following/connected to who. It is expected that larger networks will perform worse with the majority of operations such as finding, inserting and deleting. Also the probabilities of liking and following are likely going to be unrealistic as humans decisions to like or follow someone in the real world are rather unpredictable.

The social network will be stored as a DSAGraph object, this is just a graph that contains two lists: a list of vertices and a list of edges. Each vertex in the list is a DSAGraphNode which contains a label and a reference to a Person object, therefore, each node represents a person in the social network. Each person needs to store a record of all of their posts, these will be stored in a linked list. Each edge must connect two nodes (people) and be either directed (one-way relationship) or undirected (two-way relationship, two people both follow each other). This is the basis of the social network, used for storage and also adding and removing connections and nodes. Once the graph is complete and tested, a class for reading in a network and events from a text file and a class for running the program and holding algorithms for the spread of data through the network.

This main class is SocialSim.java. The algorithm for spreading information must traverse the graph, but not necessarily the entire graph (every node). It must start at the person who made a post and then spread to their followers who then have an opportunity to ‘like’ the post based on the user input probability. This probability will be changed across multiple runs to assess the behaviour of the network. Once a post Is liked, the liker will have a chance to follow the original poster based on the user input for prob\_foll. The post will also be shared to the likers followers where they have the chance to like the post and follow the original poster. The aspects that we can investigate through this are: the time it takes to spread a post through a network depending on the size of the network and the amount of people following the person, the amount of people that a post is exposed to whilst using different probability values and network sizes. Also the overheads of certain data structures used in the program and how they affect memory usage whilst running (not actually present in the program) .

**Methodology**

The simulation will be run using a small network and a larger network which is almost 4 times the size of the first. This scale can be used to apply the results to networks of a much larger scale, otherwise the input files I use would be thousands of lines long and most likely very slow to execute, although this would be better for investigation purposes. With each sized network the program will be run with different probabilities of liking a post and of following the original poster. Firstly we will use 100% probability for both which will show the maximum spread possible with this network and use that to compare with other results. This of course is not realistic so the program will also be run with multiple other probabilities such as 20% and then finally with different follow and like probabilities with the different networks. As my program uses multiple linked lists for storage I predict that the larger the network is the slower the program will run as finding, inserting and deleting in a doubly-linked double-ended list is O(N) most of the time, meaning we will potentially have to iterate through the entire list (n elements) when performing these operations. This will occur when reading in the network from a file and inserting new edges and vertices into the graph as although items that are inserted at the end of the list takes (O(1)), when inserting an item we have to check if the item is already in the list which could be O(N). Also when reading in and running events (new posts) the propagation of these posts will take longer with larger networks and networks that have more following connections. These networks are also expected to have a greater spread (# of people the post reaches) than smaller networks, but with lower probabilities it is still possible for smaller networks to have similar spreads, just unlikely. Larger networks are also obviously expected to have larger memory usage, this information will be included in each timestep and will be based off of the memory usage of the graph.

The first network that I made to test the simulation consists of 26 vertices labelled a-z and every vertex has an edge connected to ‘a’ (everyone follows a, 25 edges). The second network consisted of 7 vertices labelled a-g and again every vertex followed ‘a’ (6 edges). The third network contained 7 vertices again labelled a-g, but every vertex followed every other vertex (each vertex had 6 followers, 42 edges total). The fourth one I made is the same as the second one, but instead of every vertex following ‘a’, every vertex follows the vertex before it. For example ‘b’ follows ‘a’, ‘c’ follows ‘b’ etc. This network allows me to compare the time it takes to spread a post with network 2. Using these 4 networks I can compare the differences in network creation time and the time it takes to spread information through the network to see the effect that more elements in the data structures, in the form of vertices and edges in a linked list, has on the performance of the overall network. Each of these files will be tested first with an event file containing “P:a:Hello”, causing the program to add and propagate one post made by vertex ‘a’ and the run times will be derived from this. For the sake of organisation I will refer to this as test 1. I will then test with the same inputs but the probabilities will be changed to 50% each for one test and 0% each for another. After this I will test again by adding a new vertex and a new edge to each of the networks using the event file. This was achieved by adding “A:a1” and “F:a1:b” to the event file and including the time it takes to perform both of these tasks in the timestep statistics (note: didn’t get to implement this test ).

**Results**

Note: times are in microseconds and “Reach” represents the number of people who saw the post.

|  |  |  |  |
| --- | --- | --- | --- |
| Network | Time to make graph | Time to spread post | Reach |
| 1 (26 nodes, 25 edges) | 8438.274 | 2869.852 | 25 |
| 2 (7 nodes, 6 edges) | 7076.586 | 1498.542 | 6 |
| 3 (7 nodes, 42 edges) | 9455.23 | 1362.93 | 25 |
| 4 (7 nodes, 6 edges) | 5655.47 | 3435.056 | 6 |

Test 1: ‘a’ has one post, probabilities are both 100%

Firstly looking at the time to create a graph, we can see that a graph with 26 nodes is faster (8438.274) than a graph with only 7 nodes (7076.586) and the third network takes the longest to make as it has a large number of edges. I think this is due to my algorithm for adding an edge to the graph having multiple validation checks such as checking if each node exists and if the new edge already exists. These both involve iterating through a linked list and checking each value to see if it matches, a task that takes O(N) time. If given more time I could have refined this process but it is interesting to see the effect that a few O(N) searches on a linked list have on performance time. Now looking at the spread we can see that the graph with more vertices took almost twice as long as the other two networks that have 7 vertices, which I think shows that the number of vertices has a direct correlation with the time taken to spread data in the network in my program. I believe this has to when the method validateFollow is called in SocialSim.propagate(), this method checks if a person is already following the person who made a post. This means in this case the program would have to check if each node is already following ‘a’, in this case every node is already following ‘a’ so this would take O(N) for every node as it must iterate through the list for every node (25 times). Network 4 took more than double the time of network 2 which is the same size, this is due to having to add edges between each node and the original poster ‘a’, because of the add edges algorithms that I have explained earlier. The reach in this test is not important as with this data it will always be n-1.

Test 2: ‘a’ has one post, probabilities are both 50%

|  |  |  |  |
| --- | --- | --- | --- |
| Network | Time to make graph | Time to spread post | Reach |
| 1 | 12864.19 | 3775.426 | 25 |
| 2 | 6684.692 | 1339.09 | 6 |
| 3 | 5292.744 | 1586.01 | 25 |
| 4 |  |  |  |

This test I can’t really determine much from as the probabilities would have caused each network to have a different amount of people liking and following which means some networks did more work than others

Test 3: ‘a’ has one post, probabilities are both 0%

|  |  |  |  |
| --- | --- | --- | --- |
| Network | Time to make graph | Time to spread post | Reach |
| 1 | 7665.97 | 535.584 | 25 |
| 2 | 6748 | 493.21 | 6 |
| 3 | 7756.152 | 560.89 | 25 |

Most of the time taken to spread a post in this test are about the same as the post spread only goes through one level out from the person who posted it due to the probability values. What is interesting is the difference in time when creating network 1 between test 2 and 3. Test 2 takes almost twice as long to create the exact same network. Although this difference is miniscule, on a larger scale this could potentially cause problems with performance of the social network.

Given all of these results and tests they have confirmed the number of elements stored in a data structure has a significant impact on finding, accessing, inserting and removing elements from that data structure. With a linear relationship with time of processing, the larger the number of elements the longer it will take to perform these actions. This impact significantly affects performance of a social network so it is extremely important to store data based on how you will be using that data. If you will be accessing that data many times perhaps it is best to store it in an array or if you don’t know how many elements to store maybe a dynamic data structure like a linked list will be more practical. In terms of behaviour of the social network, the probability values that were used in this program to simulate a person liking a post or following a user were fairly unrealistic and almost random in a way so I chose not to alter them too much in the testing. Regarding further investigation, I think it would be much more interesting to perform these tests on a network much larger in scale potentially having thousands of people and edges as this will be a much more accurate representation of reality and it will also be even easier to see the effect structure has on performance. My results matched my prediction in terms of larger networks will take longer to create and spread information throughout however, that is just common sense. What really did surprise me was the effect that inserting many edges had on the creation of the social network. Also I would have liked to explore memory usage in these tests a bit more to see how much the network uses and how much memory single operations use, but due to time constraints I didn’t implement it into the program.