Abstract

[Place holder abdtract]

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Using graphs as representation and algorithms that traverse graphs are common through many scientific domains[1]. We can use graphs to simulate how a disease would spread through a social network, we can predict how efficient viral marketing is and we can see how a new trend would spread through a community[2].To select the most influential nodes for a information diffusion through a social network is an NP-hard problem[2]. One of the common ways to select them, is by brute forcing through the network with algorithms like BFS. One of the solution is the greedy algorithm[3]. The greedy algorithm, proposed by[2] goes through all the nodes and computes the effect on the network that nodes had. The greedy algorithm takes the k top most influential nodes as the starter node. When a new seed node is to be added to the starter set S, the algorithm itterate over all the node in the network that is not in S and compute the spread.

The problem that we will explore with this report, is if there is a way to optimize the seed-selection of the most influential nodes in a graph. The report will focus on how to use hardware to achieve an optimization to this problem. One idea is to try to use the irregular memory access that is requested during a BFS search.

Chapter 2

Background

In this chapter, we will look at the fundamental concepts and background information needed to understand the the different infections model, problem with seed-selection for social networks, and solving Breadth-first search using matrix multiplication. This chapter will contain notations that we would use throughout the report and used in the field. One aspect we will focus focus on, is how we can perform graph algorithm such as breadth first search as matrix multiplication.

2.1 Network terminology and glossary

The fundamental unit in a network is *Vertex*(*pl.vertices*) sometimes called a node (computer science). For this report, vertex and node would be used interchangeably. The "bridge" or the line connecting two vertices is called a *Edge*, which serves as a connection between vertices as shown in Figure 2.1. Different network have different types of edge, some can be *Directed* or *undirected*. Directed edge is where a edge runs in only one direction (such as a one-way road between two points), and undirected if it runs in both directions. Directed edges, which are sometimes called arcs, can be thought of as sporting arrows indicating their orientation. A graph is directed if all of its edges are directed. An undirected graph can be represented by a directed one having two edges between each pair of connected vertices, one in each direction.

Each node have a value that is known as *Degree*. Degree for a node is the number of edges connected to a vertex. Note that the degree is not necessarily equal to the number of vertices adjacent to a vertex, since there may be more than one edge between any two vertices. In a few recent articles, the degree is referred to as the connectivity of a vertex, but we avoid this usage because the word connectivity already has another meaning in graph theory. A directed graph has both an in-degree and an out-degree for each vertex, which are the numbers of in-coming and out-going edges respectively. The *Component* to which a vertex belongs is that set of vertices that can be reached from it by

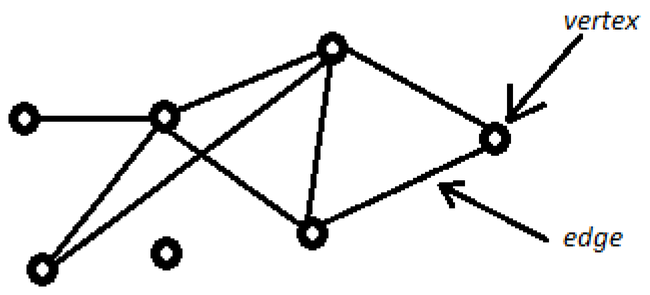


Figure 2.1: Simple network

paths running along edges of the graph. In a directed graph a vertex has both an in-component and an out-component, which are the sets of vertices from which the vertex can be reached and which can be reached from it.

A *Geodesicpath* is the shortest path through the network from one vertex to another. Note that there may be and often is more than one geodesic path between two vertices. The *Diameter* of a network, however, is the length (in number of edges) of the longest geodesic path between any two vertices. A few authors have also used this term to mean the average geodesic distance in a graph, although strictly the two quantities are quite distinct

2.2 Network

A *network* is a collection of *Vertices*, commonly known as nodes with *edges* connecting them together[4]. The edges serves as a connection, or a "bridge" between the nodes, while the nodes can represent something, or containing information. In the real world, multiple systems takes the form of networks around the world, examples are the internet, the World Wide Web, social media like Facebook, twitter etc. There are different types of network or *graphs*. These include from *social network*, *information networks*, *technological networks* ,and *biological network*. Each of them have a different properties, but we will focus more about social network.

A social network is a set of people connected to each other via some form of contact or interactions[4]. The nodes are people while the edges are the connections between peoples. The social network display information regarding connection, interaction or location of a set of people.It forms patterns regarding friendships, business interactions between companies and families history/ ancestral tree. The social network is often used in social science[4]. Some notable experiments are [**?**], which looks at the small world problem. The small world problem can be summarized as: "what is the probability that any two people, selected arbitrarily from a large population, such as that of the United States, will know each other?". [5]. This in itself is not that interesting, the [**?**] asked even tho person a and z does not know each other, do they have a set of individuals $\{\ b_1, b_2, ...b_n\}$ who are mutual friend or even a "chain" of such individual$(a - b - c - ... - y.z)$.

Some properties that networks often exhibits are the small world effect, transitivity or clustering, degree correlations, and community structure[4]. Different networks would have some of the specific properties and values. Social network would have a small world effect of around 6 degree, with high transitivity and have a community structure, while a information network, such as a citation network, would most likely have high transitivity since citation and publication have different publication date. Paper A refers to Paper B, which refers to paper C, but paper C can not possibly refer to papaer A since C is clearly published before paper A. As the Figure 2.2 shows, citation network is directed network.

2.2.1 The small world effect

The small world effect was first demonstrated by Stanley Milgram in the 1960s during his famous letter passing experiment[6]. The experiment was about passing letters from person to person to reach a designated target with only small steps. For the published case, the chain was around six [7]. This shows us that for most pair of vertices in a network can reach each other with a short path. A more precise wording is that "Networks are said to show the small world effect if the value of l scales logarithmic or slower with the network size for a fixed mean degree.". [4]. We have defined l to be the mean geodesic distance between vertex pairs in a network.

For the information diffusion problem, this kind of effect would result in that the diffusion through a network would need around 6 steps to have traveled through the entire network. Meaning that most node can reach each other through a relatively small step.

2.2.2 Transistivity/clustering

Transistivity, also sometimes called clustering is the properties that shows that there is a high number of triangles in the network. A triangle in a network is if node A is connected to node B and node C, and node B and C is also connected to each other, thus creating a triangle. In social network, we say that a friend of your friend is likely to be your friend too[4]. The transistivity is often used to show $networkdensity$.

For information diffusion and seed selection, this would mean that picking nodes that are geographically close/neighbor to each other, would have a smaller

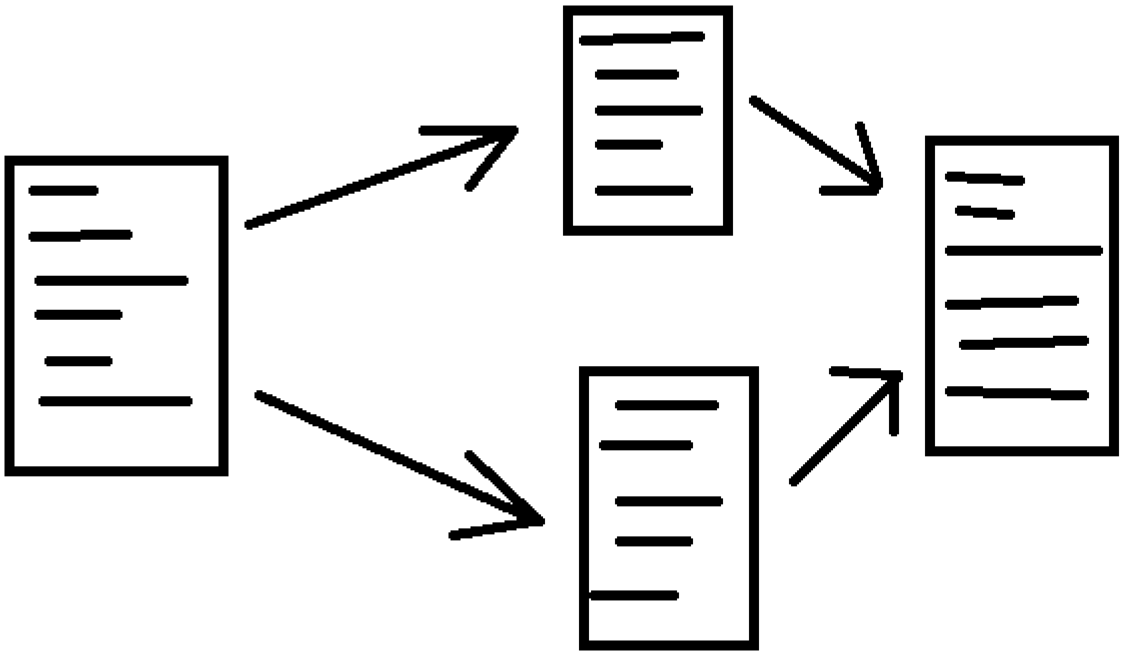


Figure 2.2: Citation network

impact/spread then picking two not neighboring vertices. By picking two nodes that are connected to each other, they would likely share a common neighbor thus having a limited reach.

2.2.3 Degree distribution

Degree distribution is the histogram of degrees of vertex. For random graphs, the distribution would most likely be a poiison distribution or binomial. For real world network, the distribution would often have a highly right skewed distribution.[INSER FIGURE OF HISTOGRAM] Resulting in that the distribution has a long right tail of values. The probability P_k is the probability that the new vertex v have the degree k. The shape have often as the figure 2.3

The degree distribution shows us how the degree to the graph is distributed. For social graph, the degree distribution is often in the shape of [INSERT FOIGURE of degree distribution histogram]. For the information diffusion, this distributon shows us how many high degree nodes there are in the network, those node would likely be high prioritized node and have a large impact.

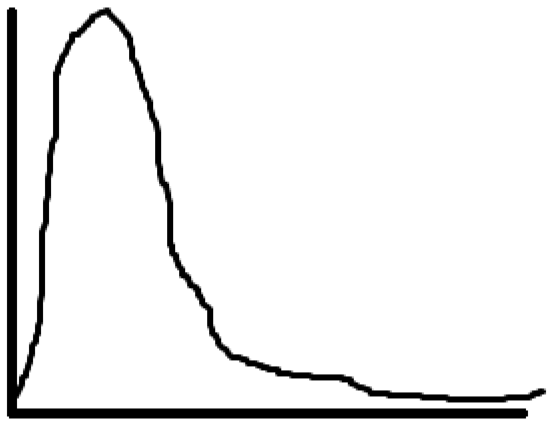


Figure 2.3: A example of a typical degree distribution

2.2.4 Degree correlations

As [REFER TO FIGURE OF DEGREE DISTRIBUTION] show, the network have a degree distribution with few high degree nodes and multiple low degree nodes. One interesting properties is the degree correlations. Degree correlations is how the high degree and low degree nodes connects to each other. One question is if high degree nodes tends to connect to other high degree nodes, or do they prefer to connect to low degree nodes. It turns out that both incidents is found in networks[**?**]. We can see that for all social network measured in [**?**], the social networks are assortative, meaning vertices have a selective linking, where high degree vertex connects to other high degree vertex[**?**]

For data diffusion, this kind of behavior would result in wasting a seed by picking majority of high degree node for starting seed, since most of them would be connected to each other. One solution is by mixing the selection, choose some percentage to be high degree, and some with lower degree.

2.2.5 Network resilience

For most network model, there is the need to remove nodes from the network. Removal of a node can have no effect on the network as a whole, or it can be devastating. Network resilience looks at how the network can ressist to such a removal. There are two different removal scheme, the random removal where nodes are randomly picked and removed, or the targeted removal where specific nodes are removed depending on the criteria.

The experiment mentioned in [**?**] by Albert *et.al* showed that for a subset of network representing the internet and the world wide web, targeted removal had a larger impact then random removal. The targeted removal removed the highest degree nodes from the network, and the random removal removed nodes randomly. The random removal had a minimal effect on the network, while the targeted removal had a much larger impact, the mean vertex-vertex distance increased. They proposed that the internet was highly resilient to random removal, while much more vulnerable to targeted removal.

There are other studies that proposed a different interpretation about the data found.

In an example of information diffusion, a targeted removal would result in massive change to the diffusion path. By removing high degree node would result in remove influential nodes and limit the spread of the data. Removing important node connection two different community would result in isolation and no path towards other community.

2.2.6 community structure

One properties that is often observed in a social network, is the community structure. The community structure is where a group of vertices having high density of edges with each other, while having low density of edges to other "community". We can see an example of the community structure clearly displayed from [**?**]. Where we can see the playground was divided into different community.

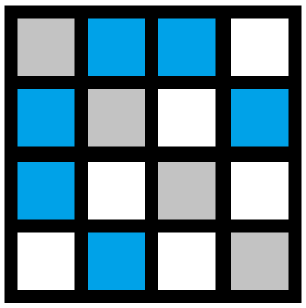
This type of community structure would have a large impact on how the algorithm would select a seed for information diffusion. If all the seed would be selected in one community, the probability of spreading over to other community would be smaller, them having a seed be in the other community.

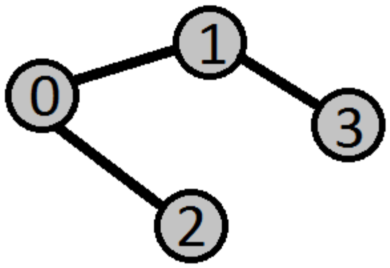
2.3 Matrix notations

By using a linear algebraic approach to solve graph algorithms often gives us a variety of benefit include easier implementation, higher performance and syntactic simplicity. [**?**]. There are multiple reasons to do graph algorithms as linear algebra. This can show potentially improvement, potentially optimization and be more visually clear how the algorithm is.

2.3.1 Semiring

A *semiring* is a set of elements with two binary operations. The two operations are often known as "addition" and "multiplication". By this definition, [SYMBOL FOR BOOLEAN] and [SYMBOL FOR INTEGER]. are both semirings. One way to perform graph algorithm, is apply matrix multiplication over such semirings. One of the common notation in writing the semiring, is $a+b$ and $a*b$. This can be confusion, so often there will be some special semiring operator.


(a) The adjacency matrix



(b) The graph corresponding to the adjacency matrix

Figure 2.4: Sparse matrix to graph

2.3.2 Sparse Matrix

A sparse matrix is an matrix with few non-zero elements in them. The social network can often be represented with a sparse matrix. An sparse matrix have its majority of element as zero, as shown in figINSERT FIGURE OF SPARCE MATRIX. We can represent a social graph in the form of a sparse matrix. In an adjacency matrix, a cell containing 1 means a connection is presence, while a 0 is lack no connection. So, as we can see in the Figure **??**, as we can see at coordinate $[1, 3] = 1$. This tells us that there is a edge between vertex 1 and vertex 3. The adjacency matrix is commonly used to represent an graph such as Figure 2.4a and Figure 2.4b.

2.4 Data diffusion

Data diffusion is looking at how information is propagated through a network or a graph. An example would be how a new Internet meme, a new trend or how a new disease is spread through a community. The process consist of a set of starter nodes that are ”infected”, during each time-step, there are a chance that the ”infected” node would ”infect” its neighbor. To start such a simulation, we would need to first pick out a set of initial ”starter node”. These k seed nodes is a set of node that in the initial time-step is infected. They will pass on the information/infection during each time-step and the information/infection will propagate through the network.

2.5 Basic Diffusion Models

When we talk about information propagation, we can look at how disease or technological innovations is spread through a social network. We can simulate those kind of behavior with different diffusion models. There are two basic diffusion models used to simulate the propagation of information through a network[2], the *linear threshold model* and the *independent cascade model*[2].

This process can be simulated by looking at a social network and how information propagates through the network. We look at each node in the graph as a person, they can be either active, or inactive. The activation of a node depends on which diffusion model we choose. A active node is "infected", while the inactive is the "healthy" ones. The activation of each nodes is dependent on which model we pick.

The linear threshold model uses a threshold θ_v between the interval [0,1], which represent the faction of $v's$ neighbors that need to be active to activate node v. The Linear Threshold Model activates the current node v when the weight $b_{v,w} > \theta$ from its neighbor w outweighs the θ_v. This is more in line with the situation were each person have a chance to adopt to a new trend if exposed to the trend enough time by his close friends. An example would be a product promoted on social network like twitter and Facebook. The user will adopt the new trend if he is exposed to it from enough friends or idols. As show in Figure 2.5, the threshold for vertex v to be activated is θ_v, a percentage of the total neighbour, in this instance, three.

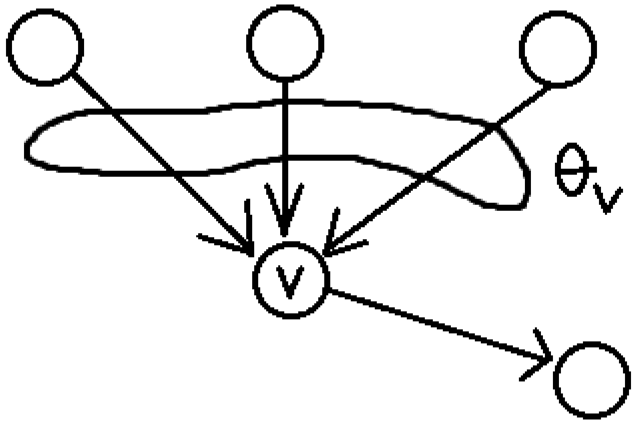


Figure 2.5: Linear Threshold model

The independent cascade model changes states with a probability $p_{v,w}$, where v is current node, and w is it's neighbor. During each propagation, the node v have a $p_{v,w}$ chance to change state if the neighbor w changed state. If during time-step t a node v changed state, its neighbor w would have a $p_{v,w}$ chance to change state in the next time step. An example here would be spread of a disease. The current node$v$ have a chance(θ_v) to activate its neighbor. As shown in Figure 2.6, there is an probability of p_v to infect the neighbors to vertex v, in this figure, the probability is the same, bit that is not necessary true. the probability can be independent, each vertex can have different probability to activate neighbor, or we can have a global probability.

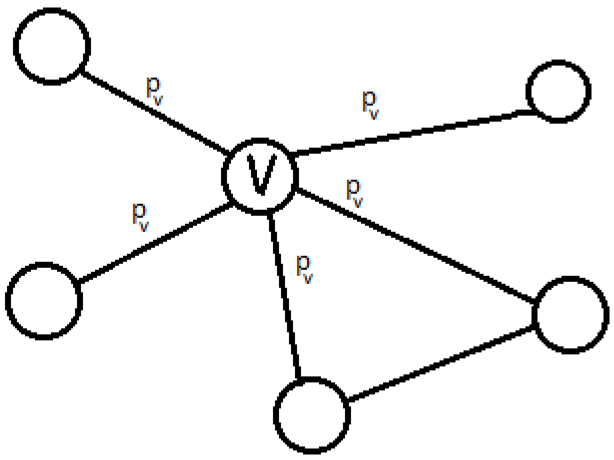


Figure 2.6: Independent cascade model

We can see the similarity between the breadth first search and information diffusion. Breath first search, as mentioned before, starts at the root node and add the root nodes children in a queue, then tales the first child node in the queue and adds all its children to the queue. Each new child node is added to the back of the queue, while finished node is placed in a $finished_queue$. The sequences of nodes tested is the same as during an information diffusion, where the diffusion neighbor is placed in the $boarder_queue$ and having a probability to be infected. If the node is infected, then that nodes neighbor would be added to the $boarder_queue$, while if the node is not infected, the neighbor to the node would be safe. The information diffusion can therefore be looked at as an breadth first search, with an probability to add the child to the queue.

2.6 Breadth First Search as a matrix multiplication.

By looking at the connectivity matrix, we can see that a connectivity matrix is a graph represented in a matrix format. The BFS can be achieved by looking at the BFS operations as a matrix multiplication [8]. If we look at the graph as a connectivity matrix, we can apply matrix multiplication to generate the breadth first search.

Breadth first search is an tree traversal algorithm. BFS start at the root node r, or any arbitrary node in a n tree, and stores all its child node in an *queue*. The algorithm then takes the first node from the queue, v_1 and stores all the child node to v_1 in the back of the queue, this process continuous until the queue is empty and all the nodes have been iterated over. From the pseudocode we can see:

Algorithm 1 Breadth First Search

```
1:  $dist[\forall v \in V] = -1; currentQ, nextQ = \emptyset$
2:  $step = 0; dist[root] = step$
3:  ENQUEUE(nextQ,root)
4:  while $nextQ \neq \emptyset$ do
5:      $currentQ = newxtQ; nextQ = \emptyset$
6:      $step = step + 1$
7:      while $currentQ \neq \emptyset$ do
8:          $u$ = DEQUEUE(currentQ)
9:          for $v \in Adj[u]$ do
10:             if $dist[v] == -1$ then
11:                 $dist[v] = step$
12:                 ENQUEUE(nextQ, v)
    return dist
```

We can also look at how an breadth first search can be executed by applying sparse matrix multiplication. We can see that the adjacency matrix is an sparse matrix. We can use sparse matrix multiplication to do graph algorithm, one example is the BFS.

2.7 BFS to data diffusion

We can draw an distinctive line between the breadth first search and data diffusion. The breadth first search adds all the child node from the parent node to a queue, then change the current node to the first node in the queue and repeats until the entire graph is iterated over, or the queue is empty. This is in theory the same as data diffusion, where information is passed to the connected vertex. The new "infected node" can then pass on the information along to the other nodes that are connected to the new node and so on. This is in practice, the same as breadth first search, minus the searching part.

We can in then draw the conclusion that Independent cascade model, is a modified version of the breadth first search, where each child note have a specific percent to be infected. The iteration is the same as a breadth first approach, but the result of the addition of node is dependent on a random "coin-toss".

2.8 greedy algorithm

The greedy algorithm was proposed by Kempe [3]. The algorithm is a greedy algorithm where it finds the starter set of nodes S by iterating through all the vertex in V and calculate the total amount of spread. The spread is saved and the k most influential nodes would be choose.

Algorithm 2 Greedy Algorithm

```
1: Start with $A = \emptyset$
2: while $|A| \leq l$ do
3:     For each node $x$, use repeated sampling to approximate $\sigma(A \cup x)$ to
       within $(1 \pm \varepsilon)$ with probability $1\delta$
4:     Add the node with largest estimate for $\sigma(A \cup x)$ to A.
5: Output the set $A$ of nodes.
```

Another popular algorithm is the degree algorithm[3]. Unlike the greedy algorithm, the degree sort all the node according to their degree distribution. The algorithm picks the top k nodes according to the degree distribution. This approach does not take the degree correlation into account, by picking only high degree node, there would be multiple overlapping activated node.

Algorithm 3 Degree Algorithm

```
1: Start with $A = \emptyset$
2: while $|A| \leq l$ do
3:     For each node $x$, use repeated sampling to approximate $\sigma(A \cup x)$ to
       within $(1 \pm \varepsilon)$ with probability $1\delta$
4:     Add the node with largest degree for $\sigma(A \cup x)$ to A.
5: Output the set $A$ of nodes.
```

Random algorithm The last one is the random algorithm. The random algorithm just pick a random seed node. This approach is the simplest to implement and easiest. The downside is that this is random and there are no strategical choosing of seed node.

2.9 Cache oblivious model

A cache oblivious model ignores the cache size and line and designs the algorithm to be cross platform and optimized. An algorithm is *cacheaware* if it contains parameter that can be tuned to optimize the cache complexity for the cache size[**?**]. Such algorithm have the disadvantage where a adaptation is needed for a new architecture[**?**]. The cache oblivious model is designed for two level of memory, and assumes that such an optimization is optimized for multiple levels as well. There are multiple algorithm designed that shows that such a model actually improves the algorithm. One of such is an cache oblivious sparse matrix multiplication. Sparse matrix multiplication is notorious to be inefficient use of the cache system, it forces the user to jump through data in main memory[**?**]. The sparse matrix approach proposed [**?**] focus on reordering the matrix and row in an cache oblivious manner. There are results that shows that a cache oblivious approach have given improvement to computation time for some matrices during sparse matrix multiplication, but for cache-friendly structure, this is shown to be a minor improvement and even small loss.

Chapter 3

Related works

related work, there are multiple people working on this kind of problem. One of the more well known is Kemnp, where he tries to solve how to maximize the spread.

Other solution include the hybrid FPGA-CPU architecture, which reduces the computation time for a BFS over a small world graph by XX%[1]

As we can see, there are many different research around this subjects, even tho there aren't many that looks at how to accelerate seed selection and the Independent cascade model in hardware, which we will look more at in the end of this report.

—————-am looking into the different related works from the hybrid BFS paper from Yaman.

Chapter 4

Method

For this report, we created a python simulation to simulate the data diffusion and the seed selection. We utilized the Pycx libraries to create the GUI, and applied the R-mat generator to create the different network. Our diffusion simulation finds *k* seed node, and for each k, runs the simulation 50 times. Each run runs until there are no more vertex to be activated. Since each node can only try to activate their neighbore once, regardless of the result, that node can't try to activate the same neighbors again. For our simulation, when no more nodes can be tested, the simulation stops.

4.1 PyCx

Pycx an libraries that help python to generate an GUI[**?**]. The pycx have a clear structure, initialize, observe and update. The initialize part, the graph is generated, the starter seed is found and the position to the graph is generated. The observe part is where python generate graphic for our simulation. for each step, the observe is called to generate a new frame. the update section i called every step, for our program, the diffusion is calculated as each step we can see how the data is diffused. The simulation allow the information and data to be displayed

4.2 R-mat

One problem during graph analyzation and calculation is finding suitable graphs to analyses. Generate graphs with desired properties is not easy to do. One solution proposed by Chakrabartiy *etal* is to use the "recursive matrix" or R-mat model. The R-mat model generates graph with only a few parameters, the generated graph will naturally have the small world properties and follows the laws of normal graphs, and have a quick generation speed[9]. The R-mat models goal is to generate graphs that matches the degree distribution, exhibits a " community " structure and have a small diameter and matches other criteria.[9].

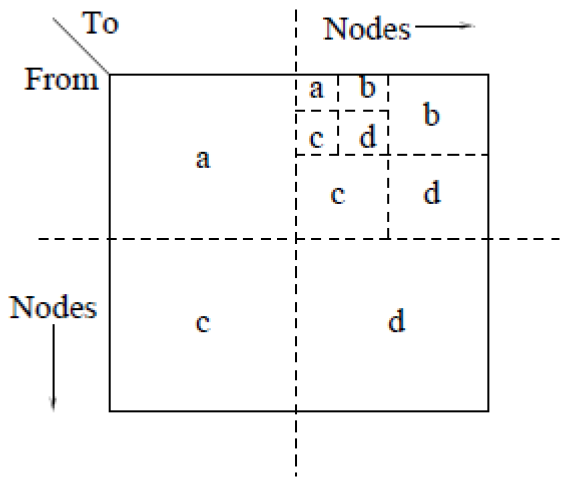


Figure 4.1: The R-mat model

The R-mat generator generates social network with the community structure. The different probability for the four partition is :$A = 0.57$,$B = 0.19$,$C = 0.19$,$D = 1 - A - B - C = 0.05$. These different probability was used by GRAPH500[CITATION NEEDED]. The R-mat generated three different adjacency matrices of different size.

The algorithm to generate such a recursive matrix is as follow: The idea is to partition the adjacency matrix into four equally sized part branded A,B,C,D, like shown in Figure4.2. The adjacency matrix starts by having all element set to 0. Each new edge is "dropped" onto the adjacency matrix. Which section the edge would be placed in, is chosen randomly. Each section have a probability of a, b, c, d, and $a + b + c + d = 1$. After a section is chosen, the partition that was chosen is partitioned again. This continuous until the chosen section is a 1x1 square and the edge is dropped there.

From the algorithm, we can see that the R-mat generator is capable to generate graphs with total numbers of node $V = 2^x$. Since the algorithm partitioned the matrix into four part. This is approach would only generate a directed graph. To generate undirected graph, $b = c$ and the adjacency matrix must make a "copy flip" on the diagonal elements, like Figure 4.2.

4.3 Adjacency matrices to graphs

For this report, three different sized adjacency matrix was created. One of the restriction to the R-mat generator is that the size of the adjacency matrix have to be 2^n. This resulted in that our adjacency matrix was originally of the

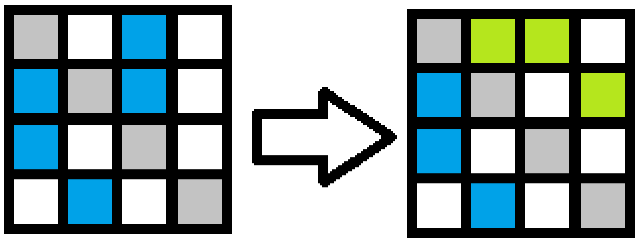


Figure 4.2: How the adjacency matrix is flipped on the diagonal

size, $128 \times 128, 512 \times 512 and 1024 \times 1024$. The resulted graphs had multiple singletons and unconnected nodes, for the sake of the simulation, those nodes were discarded and resulted in a graph with 75 nodes and 307 edges, 287 nodes with 2415 edges and 617 nodes and 8374 edges. This was not surprising consider that for a larger graph, those singletons would most likely result in the outskirt and small community.

4.4 The algorithm

The simulation creates an social network by reading from the adjacency matrix. We implemented 4 different algorithm. The greedy algorithm, the degree algorithm, random algorithm, and a independent greedy algorithm. The algorithm implemented, finds the k vertices to be the starter nodes. k is [1,2, ... 20]. This is to be able to see how the size of the starting nodes affect the coverage. For each k, the simulation applies the diffusion for 50 times. This is to find the mean coverage to remove the randomness

The greedy algorithms finds the most influential nodes in relation with the other previous picked seeds. The algorithm starts by finding the most influential nodes s_1 from the entire graph G, in this instance, k=1. The most influential node is found by just choosing a node and see how much spread it will result in, the value is stored with the node, and after the entire G is iterated over, the node with the highest value is chosen. Then the algorithm stores the node in S and applies data diffusion and stores the effect this runs had. The next run, k=2 and the greedy algorithm finds the most influential node s_2 where $s_2 \neq s_1$ and $s_2 + s_1 = maxCoverage$. This is repeated until k=20. The new run, the seed selection will keep the previous selected seed and during the finding maximum coverage phase, the previous chosen seed will have impact on the run.

The degree algorithm chooses the vertex with the highest degree. Unlike the greedy algorithm. The degree algorithm just finds the vertices with the highest degree. The algorithm chooses $s_1, s_2\dot{2}_k$ from G that have the highest degree. One of the problem would be that higher degree nodes would often be connected to each other, the community structure that was mentioned in previous section. The degree histogram shows us that there are very few high degree node, while having more low degree nodes.

The random algorithm picks random vertices as the starter nodes. This is the simplest algorithm, where each runs, a new random node is added to the set S. Then the diffusion is applied.

One experimental algorithm we tested was the independent greedy algorithm, where the algorithm choose the largest impact node as a seed node. This algorithm is like the greedy algorithm, with one major difference, the algorithm chooses a new seed node for each run. The algorithm picks the k seed nodes each independently to the previous picked node. This results in that the chosen nodes would often result in

Chapter 5

Result

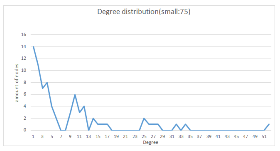
The result from the multiple runs we have gotten is as shown in the different figures. Figure 5.2 shows the spread and coverage from the simulation of the small graph. As we can see, the four different algorithm had little difference. The y-akses shows us the average coverage after 50 runs, and the x-akses gives us how many seed node was selected. We can see from figure 5.2, figure 5.3 and figure **??** that the resulted coverage was around 0.3-0.45. The smaller graph would result in a smaller spread, while larger graph results in large spread. Figure 5.2 have a more linear increase as the seed nodes increased. Figure 5.3 and Figure **??** on the other hand, had an massive increase in coverage after the addition of the second seed node, then the increase slowed down drastically.

The network that we ran our simulation on, resulted in a degree distribution as Figure 5.1a, Figure 5.1b, and Figure 5.1c. The degree distribution is similar to the 2.3. We can see that there are one vertex with high degree, while the majority of the nodes have a small degree.

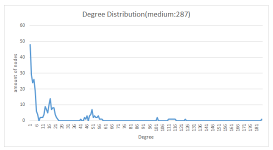
We can see that the greedy algorithm performed better then the other algorithm for the large and medium graph, except the random algorithm for the large graph, where it was best.

For each of the

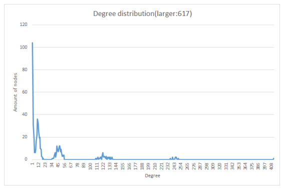
Our result as a table:



(a) Degree distribution of the small graph.



(b) Degree distribution of the medium graph.



(c) Degree distribution of the large graph.

Figure 5.1: The degree distribution of the graph that was used for the simulation

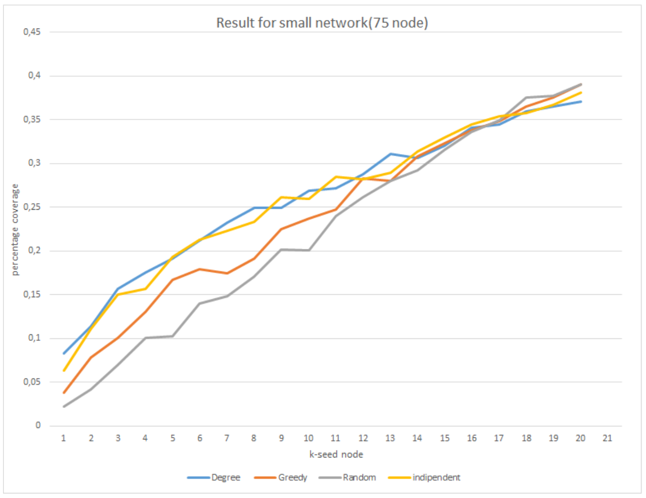


Figure 5.2: The result from the small graph

k	Degree	Greedy	Random	Indipendent
1	0.0832	0.0381333333333	0.0224	0.634666666667
2	0.114133333333	0.0778666666667	0.0413333333333	0.110933333333
3	0.157066666667	0.100533333333	0.0696	0.149866666667
4	0.175733333333	0.130666666667	0.100266666667	0.156266666667
5	0.190933333333	0.166666666667	0.102933333333	0.193066666667
6	0.212	0.178933333333	0.139466666667	0.212266666667
7	0.232533333333	0.174666666667	0.148533333333	0.2232
8	0.2488	0.1912	0.1704	0.233333333333
9	0.249066666667	0.225066666667	0.201333333333	0.261066666667
10	0.2688	0.237333333333	0.200266666667	0.2592
11	0.271466666667	0.246933333333	0.239466666667	0.285066666667
12	0.287733333333	0.2832	0.261066666667	0.281866666667
13	0.310933333333	0.280266666667	0.279733333333	0.2896
14	0.306133333333	0.308	0.292533333333	0.313866666667
15	0.320266666667	0.322666666667	0.3152	0.329066666667
16	0.3408	0.338133333333	0.336	0.3448
17	0.344533333333	0.348266666667	0.3496	0.354133333333
18	0.359733333333	0.348266666667	0.375466666667	0.3576
19	0.3648	0.375733333333	0.377333333333	0.366933333333
20	0.370933333333	0.389866666667	0.3904	0.381333333333

Table 5.1: Result from small graph

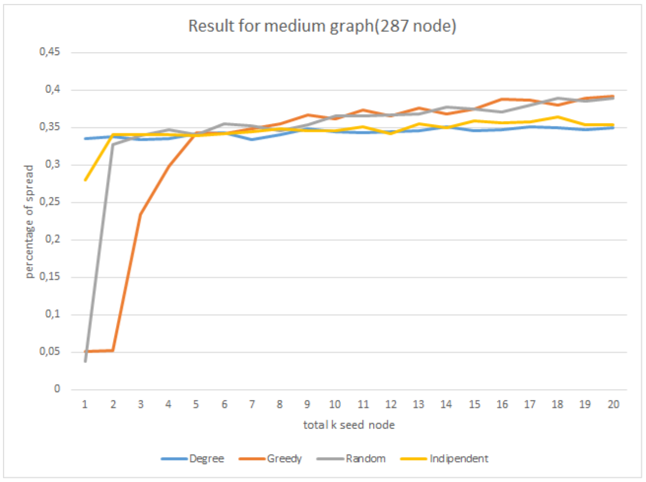


Figure 5.3: The result from the medium graph

k	Degree	Greedy	Random	Indipendent
1	0.334773519164	0.0506620209059	0.0384668989547	0.279790940767
2	0.337421602787	0.0529616724739	0.327456445993	0.340069686411
3	0.334216027875	0.23393728223	0.339442508711	0.340975609756
4	0.335400696864	0.298048780488	0.34668989547	0.341324041812
5	0.341742160279	0.342926829268	0.340557491289	0.339930313589
6	0.34362369338	0.341672473868	0.354773519164	0.341463414634
7	0.334076655052	0.348571428571	0.352055749129	0.345156794425
8	0.341324041812	0.355191637631	0.346550522648	0.348780487805
9	0.347944250871	0.366968641115	0.353867595819	0.345644599303
10	0.344390243902	0.362369337979	0.365156794425	0.345365853659
11	0.343554006969	0.373449477352	0.365783972125	0.350871080139
12	0.34425087108	0.366271777003	0.36668989547	0.341881533101
13	0.346132404181	0.375609756098	0.368432055749	0.354912891986
14	0.351358885017	0.368710801394	0.377979094077	0.349337979094
15	0.345714285714	0.374285714286	0.37456445993	0.359442508711
16	0.347804878049	0.38850174216	0.371567944251	0.355818815331
17	0.351010452962	0.386620209059	0.379721254355	0.357909407666
18	0.350383275261	0.380766550523	0.389128919861	0.364320557491
19	0.347386759582	0.388989547038	0.385714285714	0.354285714286
20	0.349268292683	0.391986062718	0.389477351916	0.353658536585

Table 5.2: Result from medium graph

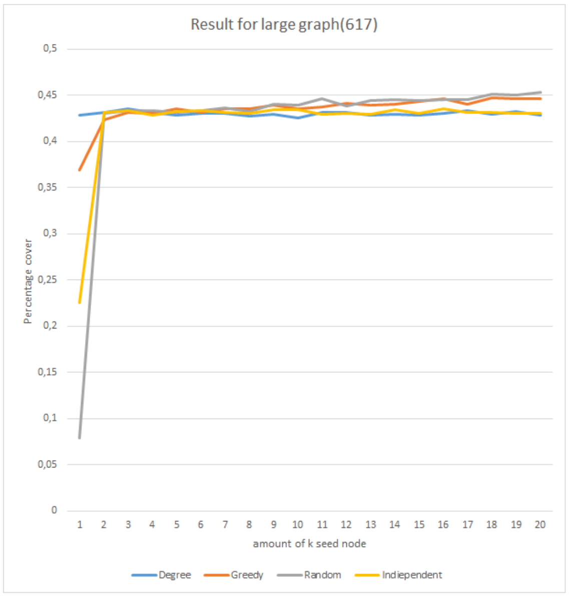


Figure 5.4: The result from the large graph

k	Degree	Greedy	Random	Indipendent
1	0.428363047002	0.368881685575	0.0788330632091	0.225705024311
2	0.431993517018	0.423857374392	0.430696920583	0.431928687196
3	0.435364667747	0.431831442464	0.433873581848	0.433387358185
4	0.431150729335	0.430470016207	0.433614262561	0.428849270665
5	0.42829821718	0.435818476499	0.431863857374	0.432544570502
6	0.430275526742	0.432025931929	0.433614262561	0.433776337115
7	0.430210696921	0.435559157212	0.436369529984	0.431377633712
8	0.427487844408	0.435170178282	0.432414910859	0.430923824959
9	0.429951377634	0.439643435981	0.440713128039	0.434424635332
10	0.425380875203	0.435170178282	0.439189627229	0.435072933549
11	0.431280388979	0.437471636953	0.446612641815	0.429692058347
12	0.431993517018	0.441847649919	0.438995137763	0.431118314425
13	0.428200972447	0.439384116694	0.444278768233	0.429627228525
14	0.429529983793	0.440842787682	0.445316045381	0.434586709887
15	0.428719611021	0.443824959481	0.444829821718	0.431118314425
16	0.431118314425	0.446126418152	0.4456726094	0.43510534846
17	0.433938411669	0.440097244733	0.445510534846	0.43209076175
18	0.42982171799	0.447066450567	0.451280388979	0.431345218801
19	0.432350081037	0.446677471637	0.450437601297	0.431021069692
20	0.428784440843	0.446839546191	0.453452188006	0.430632090762

Table 5.3: result from large graph

Chapter 6

Discussion

The result was somewhat surprising, the greedy algorithm that I thought was gonna exel in this simulation was not a huge improvement over the other algorithm. The random algorithm performed better then i expected and the degree algorithm was in some stages better then the greedy algorithm. The independent coverage algorithm performed better or equally good as most of the other algorithm. Toward the end where we choose twenty seed nodes, most of the algorithm performed somewhat the same.

These results can be explained by looking at our simulated social networks. The networks we ran the simulation over was a small graph compared to real world graphs, by choosing at most 20 seed nodes, a large percentage of the graph would be covered. Since the graphs was all connected in a network, the small world effect would result in that every vertex would be subjected to activation. Our global percentage was 0.05.

One paper showed how implementing a palatalized BFS algorithm on a distributed system reduced the communication times by a factor of 3.5, compared to a common vertex based approach[10]. In scientific application, ,graph based computation is pervasive and often data-intensive, this especially for distributed systems.

Our simulation runs until there are no more nodes too test. This can explain some of the result we got. Since the simulation runs until no more nodes can be activated, the coverage after 50 runs would most likely activate most of the node. This can result in no distinctive better algorithm.

Chapter 7

Future work

One interesting aspect that does not have huge previous work on is a hardware approach to speed up the seed selection and the information diffusion. We have seen different algorithms to select the most appropriate seed, the problem is that for larger graph, the seed selection algorithm is very taxing for the system. For this report, the graphs generated is still small, yet the largest of the used graph proved to be quite slow for the system. It would be interesting to see how an CPU-FPGA heterogeneous platform like [1] would improve the information diffusion.

Another interesting branch to explore would be to see how a paralyzed independent cascade model would measure against the non-parallel version. There are studies exploring the paralyzed BFS[10]

For this report, we created the simulation by storing arrays and "pointer chasing", as mentioned in **??**, by applying sparse matrix-vector multiplication, we could have explored other potentially beneficial optimization.

The cache oblivious sparse matrix-vector multiplication is one aspect we have yet to touch upon. It would be interesting to further investigate how much of an improvement this would potentially give compare to a standard sparse matrix multiplication.

————————still lacking a lot of information, would add more detail of how each point mentioned can be expanded.

Chapter 8

Conclution

As we have seen here the

Bibliography

[1] Magnus Jahre Yaman Umuroglu, Donn Morrison. Hybrid breadth-first search on a single-chip fpga-cpuheterogeneous platform. 2015.

[2] Éva Tardos David Kampe, Jon Klein. Maximizing the spread of influence through a social network. 2003.

[3] Éva Tardos David Kampe, Jon Klein. Maximizing the spread of influence through a social network. 2003.

[4] M. E. J. Newman. *The structure and function of complex networks*. 2003.

[5] Stanley Milgram Jeffrey Travers. An experimental study of the small world problem. 1969.

[6] Stanley Milgram. The small-world problem. 1967.

[7] Stanley Milgram Fjeffrey Travers. An experimental study of the small world problem. 1969.

[8] John Gilbert Jeremy Kepner. Graph algorithms in the language of linear algebra. 2011.

[9] Christos Faloutsosx Deepayan Chakrabartiy, Yiping Zhanz. R-mat: A recursive model for graph mining. 2004.

[10] Kamesh Madduri Aydib Buluc. Parallel breadth-first search on distributed memory systems. 2011.