Bachelor thesis
Computer Science

Radboud University

Shortest Path Estimation for small-world networks using a taxonomy

Author:
Marvin Barron
S4048555

First supervisor/assessor:
prof. dr., T.M. Heskes
t.heskes@science.ru.nl

Second assessor:
dr, P. Van Bommel
P.vanBommel@cs.ru.nl

August 19, 2014

Abstract

Can a hierarchy of semantic information of a social graph be used to effectively estimated shortest paths within that graph? The need to compute the shortest paths in social graphs has increased with the rise of social media and the internet before that. This paper attempts to find a way to lower computation cost of the all pairs shortest path problem within social networks. Showing that a hierarchical structure of information on the social graph provides a good basis for path estimation.

Contents

1 Introduction **2**

2 Preliminaries **4**
2.1 Overview 4
2.2 Small-world networks 5
2.3 The All Pairs Shortest Path (APSP) problem 6
2.4 Taxonomy tree 7
2.5 The data set 8

3 Research **11**
3.1 Research 11
3.2 The algorithm 12
3.2.1 Breadth first search in a small-world 12
3.2.2 Taxonomy 13
3.2.3 Memory problem 13
3.2.4 Pseudo code 14
3.3 Results 20
3.3.1 Category subset 20
3.3.2 Dynamic category distance computation 21
3.4 Possible solutions 22
3.4.1 Extended BFS 22
3.4.2 Extended selection process 22

4 Related Work **23**
4.1 Social networks 23
4.2 Shortest path in social networks 23

5 Conclusions **25**

Chapter 1

Introduction

How fast do rumors or diseases spread around the world. How long does it take to send packets from one PC to another across the Internet? Similar questions have been asked for decades and since the 1960s have been correlated with the term small-world. It all depends on the connectivity in the network involved [3]. If it only takes a few steps to be able to reach any node in a network of people a rumor will spread much faster.

To estimate how fast a rumor or a killer-disease will spread, you would have to know something about the average distance between all of the people worldwide. If we look at Facebook friendships, this can be calculated using simple breadth-first search in $O(|V| * (|V| * |E|))$ time, with $|V|$ the number of vertices and $|E|$ the number of edges in the graph. For each node all shortest paths from that node can be obtained by traversing through the whole graph. Visiting $|V|$ vertices across $|E|$ edges. For very large networks this could take a long time and even using faster algorithms for networks with millions of nodes there is still much room to improve on. Is there a way to do it faster? The solution might lie in attempting to find the path using real world knowledge. When looking for a path from one person to another, using Facebook friendships for instance, we could first attempt to find a path to the right continent, then city and so on. This should reduce the search space drastically. The real world knowledge mentioned is in the form of a taxonomy, a hierarchy of information about the graph we are searching in.

In this paper we will be attempting to use this technique to estimate shortest paths between Wikipedia articles, using a taxonomy of the categories. This thesis will look into:

- What is a small-world network?
- Why is the All Pairs Shortest Path (APSP) problem important when classifying a network as a small-world?
- How can a taxonomy of Wikipedia categories be used to estimate the shortest path between Wikipedia articles?
- How does shortest path estimation, using a taxonomy, compare to conventional shortest path computation in terms of efficiency and time?

Chapter 2

Preliminaries

2.1 Overview

The main question posed in this paper is whether or not a taxonomy will decrease the time needed to find APSP. Instead if finding the shortest path between two nodes by using breadth-first search, i.e. try all possible paths until the goal is reached, you could always try to guess what the shortest path is in one go. Obviously this could lead to disastrous mistakes when guessing randomly, so an educated guess is needed. This is where the taxonomy comes in. Below in section 2.4 a detailed description is given of what a taxonomy is and more importantly how it will be used to compute APSP. In short it is explained best by the example given in the introduction. "When looking for a path from one person to another, using Facebook friendships, we could first attempt to find a path to the right continent, then city and so on." It makes sense that if I would want to know what the shortest path from me to someone in China is, I will most likely find that path through other people in China. Once "in China" I will take a path that brings me to the right city. For this shortest path problem you could say I used a geographical taxonomy as shown by the figure below.

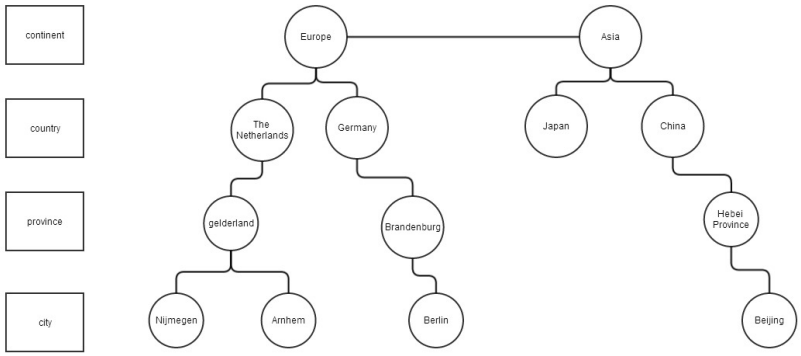

Figure 2.1: Taxonomy example.

Figure 2.1 shows a hierarchy between cities which are geographically in certain provinces, countries and on certain continents. This is a simple example of a part of the taxonomy that could be used to find the shortest path from me to someone in Beijing. It should show for cities and provinces how far, measured in km, it is to Beijing. If we would have a worldwide list of these distances to Beijing from each city or province, when attempting to find the shortest path via friends of friends I could always make sure I am moving closer geographically. Presuming that I am most likely to find an acquaintance of my Chinese goal the closer geographically I get.

The question is, by only attempting to move closer to my goal by choosing a path through nodes that are closer to my goal in the taxonomy, will the chosen path be a "good guess" in terms of the estimated distance and time needed?.

2.2 Small-world networks

Why small-world networks? To define a small-world we must first take a look into social studies. In the late 70's Pool and Kochen published an article in the first issue of Social Networks. The meeting of two strangers who later find out they have a common acquaintance, would lead them to reflect on how "small" the world really is. Pool and Kochen wanted to show that "people are not only linked to their immediate friends and family, but they are embedded in a larger structure" [12].

What type of small-world networks are we using and how is it different from any other network? The term was first used to describe social networks, i.e. networks of people. These networks seem to have a very high inter connectivity, but this could not be proven [12]. Due to the millions of nodes and high level of randomness in connectivity between these nodes, logistically it is nearly impossible in the real world. Recently the term has

been used to describe large networks like Facebook and Wikipedia who have the same properties as the real world social networks [12]. Because these networks are stored on databases it is now possible to research into the characteristics of small-world networks. In social networks, like Facebook for instance, this is used to create a socially sensitive search [8]. When a user is searching for other people, first results will be the people closest within the network to that user. In this method the shortest path distance is used as a simple ranking function [8].

Why use a small-world network for this paper? Because of the small-world properties, the node selection process might have even greater advantages than a conventional shortest path algorithm compared to searching in non small-world networks [Section 3.2.1].

A small-world network is characterized by the two following properties: the local neighborhood is preserved and the diameter of the network, quantified by the average shortest distance between two vertices, increases logarithmically with the number of vertices [5]. The last property gives the small-world network its name, because any two vertices in the network are connected with just a few links. The diameter of a network is equal to the maximum of the shortest path lengths of all pairs of nodes in the graph [15]. Research shows that the largest Wikipedias, i.e. the languages with the most articles. are indeed small-world networks [16]. However the paper states that this property might not hold for all Wikipedias. The reason our research prefers a small-world network is simple. Besides the fact that the small-world network phenomenon arose when Milgram wondered how many acquaintances were needed to connect any two people in the world [10]. To answer the problem stated by Milgram you would have to know APSP or at least compute the shortest distance between a single pair of people. We already established that for a lot of social networks knowing APSP is beneficial [8], but besides that a small-world has one defining characteristic that comes in handy when computing APSP. Because of the low average shortest path distance in a small-world we hope to keep the computation time for our shortest path estimation to a minimum.

2.3 The All Pairs Shortest Path (APSP) problem

Attempts to find the cost of the lowest cost path from v to w. Given a directed graph $G = (V, E)$, where each edge (v, w) has a non-negative cost, for all pairs of vertices (v, w). The problem above can be solved by several algorithms, for instance Floyd's algorithm and Johnson's algorithm [2] [9]. But how efficient are these algorithms when trying to find APSP for a graph containing several millions of nodes? Worst case most algorithms are able to compute the APSP somewhere around $O(n^2)$, with n the number of nodes. So for extremely large graphs it could take a very long time. Potamias

points out that for some large networks(4 M nodes and 50 M edges) a BFS traversal would need roughly a minute in on a standard desktop computer [8]. This is an estimation using the BFS routine of the C++ boost library. Note that in all cases C++ will run faster than JAVA, which is used in this research. This makes it clear that the APSP problem for this type of network is impossible to compute using BFS, at least within a large network.

So might there be a way to improve on this for very large networks, i.e. networks with millions of nodes? In some cases there might yet be a simple intuitive way to do so! In this paper we will confine ourselves to using small-world networks. Two examples of small-world networks are Facebook and Wikipedia. So how would we go about finding APSP in these networks faster than $O(n^2)$? By attempting to use real world information perhaps we can find the shortest path between nodes much faster. All Facebook users are linked to each other through friendship relations. When looking for a path from one random user to another we could first attempt to find a path to the right content, then city and so on. This should reduce the search space drastically, by using a so called taxonomy.

2.4 Taxonomy tree

When attempting to find the shortest path between two nodes we will use a node selection process that uses a taxonomy tree. This tree is a hierarchy of all the categories of the articles in our graph. Like the Facebook example we will assume that the shortest path is through nodes that are "closer" to each other in regards to their categories. To do this we would first have to know the distance between each category. This means our solution for the tedious computation of the APSP problem contains another APSP computation. But in this case we will be using a much smaller graph in terms of vertices and edges [Section 3.2.2]. The shortest paths within the taxonomy then gives us insight on how to estimate the shortest path in the "larger" network. How the taxonomy is obtained is explained in the next chapter. Why can the taxonomy be used to estimate the shortest path?

In figure 2.2 the relationship between the Wikipedia articles and categories is shown. On the left is the Wikipedia Category graph (WCG) with each node C_I a category in the WCG and nodes A_I articles in the article graph. The categories can be seen as semantic tags for the articles and have a clear hierarchy. Research has shown that the categories are good semantic tags for the articles [15] [13]. In an empirical study on how to search social graphs several strategies where tested on there ability to find shortest paths within social networks. In this study email traffic within a large organization was monitored. The research showed that when attempting to send an message to the right person, with only the mailing information of their direct contacts. The research showed that attempting to direct the mail

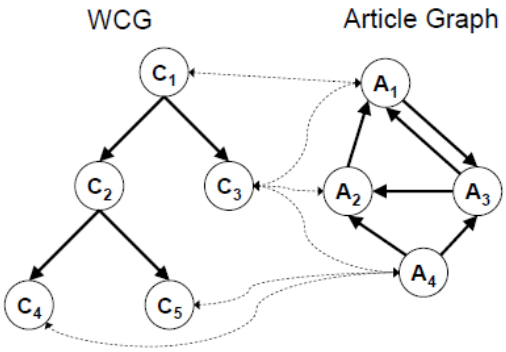

Figure 2.2: Relations between Wiki articles graph and category graph [15].

traffic through people who are closer to the goal within the hierarchy of the organization was a good tactic [4]. These papers support our claim that the taxonomic structure of the Wikipedia category graph could be used to estimate the shortest paths between Wikipedia articles.

2.5 The data set

In order to attempt to find an efficient way of solving the APSP problem we need a data set large enough to show that conventional techniques are slower [Chapter 3]. Wikipedia seemed to be the best option because of the fact that the this large network is easily downloaded off the Internet. Wikipedia consists of several different projects, i.e. different languages, with several graphs such as the article and category graphs. The smaller projects however consist of less articles than categories which makes computing APSP for the category graph redundant, because the APSP for the article graph could always be computed faster. In this case using the categories for a node selection process will not improve in any way.
First the APSP problem will attempted to be solved for to the Simple English Wikipedia articles. Because of the fact that Simple Wikipedia "only" has about 160.000 articles, it is easier to work with than English Wikipedia. However, for testing reasons even this is too much. Because of the fact that the solution to the APSP problem is a $N \times N$ matrix, with N = number of nodes, the amount of memory needed is very high [Section 3.2.2].

Wikipedia can be downloaded through several dumps consisting of one table, some of these dumps then have to be combined in order to retrieve the full article and category graphs. Throughout the Wikipedia dumps a lot of data is duplicated because of the fact that each table is linked through a few primary keys. Each dump can be downloaded as a large SQL file that should be able to reconstruct the original tables. However, due to the size

of the dumps the only way to access the data inside is to split the file and edit each chunk as a text file, due to the fact that database management tools are not able to import large SQL databases. This is achieved with the binary split operation in Linux. The tables are structured and used as followed:

Field	Type	Null	Key	Default
cl_from	int(10) unsigned	NO	PRI	0
cl_to	varbinary(255)	NO	PRI	
cl_sortkey	varbinary(230)	NO		
cl_sortkey_prefix	varbinary(255)	NO		
cl_timestamp	timestamp	NO		CURRENT_TIMES
cl_collation	varbinary(32)	NO	MUL	
cl_type	enum('page','subcat','file')	NO		page

Table 2.1: Category database table.

This database table is used to obtain the id of an article, the categories the article link to and the name of the article. The first three fields are used for this data in that order. In essence only the first three fields are necessary for our algorithm. In short the file is parsed by using a regular expression to obtain one row. Then a check is done to see if the last obtained row has the same id. If so that means that only the category needs to be added to the node created earlier. Otherwise all categories found in the last rows are added to the previous node and a new node is created.

Field	Type	Null	Key	Default
cl_from	int(10) unsigned	NO	PRI	0
cl_namespace	int(11)	NO	PRI	0
cl_title	varbinary(255)	NO	PRI	

Table 2.2: Pagelink database table.

A row in this table contains the id of a certain page, the namespace of that page and title of a page it links to. The namespace indicates if the page is an article, category or one of the other types of Wikipedia pages. Just as the previous table the data here is extracted with a regular expression resulting in a list of tuples containing (Id, Name). So the page with id = cl_from links to a page with name = cl_title.

The next two tables are use to construct the taxonomy of all the English Wikipedia categories, these tables were not provided by Wikipedia but by an independent source. The taxonomy tree was constructed in 2010, since then the category graph has changed however [14]. Because there is no further documentation to how the taxonomy was derived we will presume it is correct. The two tables used to construct the taxonomy tree are:

Field	Type
Id	int(11)
name	varchar(255)

Table 2.3: Taxonomy Category database table.

This table contains every category (id, name) used in the taxonomy which was created in 2010. Because of that fact there might be newer categories today, the taxonomy does not match up completely. Even worse the categories used in Simple English Wikipedia only match about 66% of the English Wikipedia. However this is due to the fact that the English Wikipedia category graph is larger then the Simple English Wikipedia category graph, explaining why most of the English Wikipedia taxonomy is not used when matching with Simple English Wikipedia. Unfortunately there was no trivial way to subtract the Simple English Wikipedia category graph.

Field	Type
from	int(10)
to	int(11)

Table 2.4: Taxonomy link database table.

The last table is a simple list of tuples of ids of categories that link to each other in the taxonomy tree.

Chapter 3

Research

3.1 Research

To classify a network as a small-world one would have to calculate the diameter of the network, i.e. compute the average distance between all pairs of nodes. All small-world networks have such a high connectivity that this average distance is surprisingly low. The table below shows some statistics for a few small-world networks [7].

	network	type	n	m	z	d
social	film actors	undirected	449 913	25 516 482	113.43	3.48
	company directors	undirected	7 673	55 392	14.44	4.60
	math coauthorship	undirected	253 339	496 489	3.92	7.57
technological	Internet	undirected	10 697	31 992	5.98	3.31
	train routes	undirected	587	19 603	66.79	2.16

Table 3.1: Basic statistics for a number of published networks. The properties measured are: type of graph, directed or undirected; total number of vertices n; total number of edges m; mean degree z; mean vertex–vertex distance d.

Currently most shortest path algorithms do not attempt to get around using every node to calculate the distance between all node pairs. Which causes a problem in very large networks [8]. A path estimation algorithm, specifically one using a taxonomy, could be used to drastically lower the time needed. This would avoid searching through all elements of the original network. This solution is very close to the intuitive way a human would do it. For instance if you have to use Facebook friends lists to find your way from one random user to another. It makes sense to try and find your way to the right content, then the right country and so on.

Because this, depending on the categorization used, potentially drastically lowers the search space it is much easier the find all shortest paths between nodes. Or at least find a close estimation of the shortest paths. Simply because the number of vertices and edges in the taxonomy is nothing compared to the original network. Computing the shortest distances in your taxonomy leaves you with an easy way to decide which nodes to expand when searching in the original network.

3.2 The algorithm

This thesis is centered around the all pairs shortest path problem, specifically with the estimation of the shortest path by using a taxonomy. Up until now the only explanation on why this would work was the intuitive approach used in the introduction and preliminary chapters. A deeper look into the usage of a taxonomy in comparison to a breadth-first search is given below.

3.2.1 Breadth first search in a small-world

As seen in the preliminaries a small-world network is a network with a high connectivity between nodes. For Simple English Wikipedia each node has an average of 45 links, for Facebook this is 338 [1]. The high number of edges is in line with what you would expect from a small-world and is exactly why the path estimation in small-world networks could be much faster than breadth first search (BFS).

Why would a BFS suffice in this complicated graph? Because all edges are weighted 1, the APSP problem for Wikipedia or Facebook is just a question of counting nodes in a potential shortest path. The BFS algorithm need only check which nodes have been visited. In the image below a graph is illustrated where each node has N children, with N the average amount of children in the network. When attempting to find a path from node A to node G, first the algorithm expands the N nodes at distance 1 from A and adds them to a queue. If the goal is not reached all children of the first N nodes are expanded and added. This is done until the goal is reached at which point the algorithm can stop when the depth of the search exceeds the length of the shortest path, due to the fact that no shorter path will be found.

Because of the fact that nodes in a small-world network have a relatively high amount of edges the longer the average shortest distance between nodes within that network the more you could benefit from an accurate path estimation. In the graph above the shortest distance from A to G is three steps. With breadth-first search this would mean worst case $N \times N \times N$ nodes are visited, with N the average number of edges for each node. This happens when G is located at depth 3 on the far right of the graph. When using node selection, if the correct node is chosen with a single attempt, this

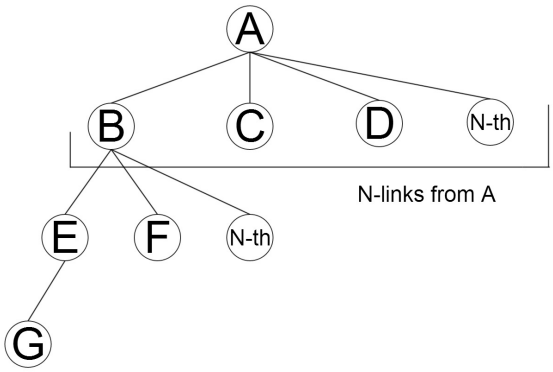

Figure 3.1: BFS example.

could be reduced to $N + N + N$ nodes visited in the worst case. Because at each level one node is expanded and has an average amount of N nodes. The question is how good of a selection process can the taxonomy provide us?

3.2.2 Taxonomy

In order to have an accurate node selection a taxonomy tree of the Wikipedia categories is used [14]. Using figure 3.1 as a reference, when attempting to find the shortest path from node A to node G. We start of by selecting one of A's children, c in this example, to expand. This is done by looking at the average distance between C's categories and G's categories. Which ideally would be obtained from a distance matrix created at the beginning of the computation. However for the APSP problem of the Wikipedia categories a matrix of $M \times M$ is needed, with M the number of categories. This would mean that 474168×474168 elements would have to be stored. In JAVA, the language used during the research, the matrix would be represented by a two dimensional array which commonly uses between 16 bits and 32 bits to store integer values. The memory usage comes from the fact that primitive integers use 16 bits and the array uses headers to store the elements. If we assume that the headers are not needed to store all of the categories, we would need $(474168 \times 474168) \times 16$ bits = 418.78GB of data. The servers at the university have 256GB at most, which make creating the category APSP matrix impossible.

3.2.3 Memory problem

The large data sets used cause several memory problems when attempting to pre-compute APSP for the category graph. Since it is not possible to create a APSP table for the categories, when attempting to estimate the shortest

path between articles we must dynamical calculate the average distance between the categories. Similar research on other large networks have come to the same conclusion regarding the pre-computation of APSP in even larger networks [8]. Unfortunately for our relatively smaller category graph the same holds. This greatly increases the time needed to find which article should be expanded to next. For the Simple English Wikipedia problems only occur when attempting to make an APSP matrix for the categories. But the English Wikipedia articles graph is even greater than that of the Simple Wikipedia articles graph and this increases problems when handling the SQL tables.

When testing the time efficiency of the proposed node estimation algorithm the Simple English Wikipedia will not suffice. Even though the Simple English Wikipedia articles graph has more vertices and edges than the Category graph, the algorithm has to dynamical calculate shortest distances for categories. So often even that the time needed to calculate a path is extremely high. Most articles in the Simple English Wikipedia have around 3 categories. Finding the average category distance from one article to the goal then results in 9 shortest path computations in the category graph. Each category of the article being evaluated is matched with all of the goal categories. If an APSP table for the categories could be constructed this lengthy computation would be done only once. The inability to use a APSP table for the categories, significantly lowers the speed of the algorithm.

Graph	$\|V\|$	$\|E\|$
Simple English Wiki articles	163,103	6.942.262
Simple English Wiki categories	474,168	995,860
English Wiki articles	4,577,364	78,300,000
English Wiki categories	474,168	995,860

Table 3.2: Wikipedia graph statistics [17].

3.2.4 Pseudo code

In order to find APSP for the Simple English Wikipedia articles we would first have to find a path between a pair of articles. The algorithm used to estimate the distance between nodes is described in pseudo code below.

```
 1: procedure ExpandNode(Node Start, Node Goal)
 2:     Start.SetPointer("Done");
 3:     while !GoalFound do
 4:         if GoalFound then
 5:             GetPath(Goal);
 6:         end if
 7:         list.clear;
 8:         list.add(Start);
 9:         Start = FillQueue(Start, Goal);
10:     end while
11: end procedure
```

Annotation to ExpandNode(Node Start, Node Goal)

2: Set a pointer for the goal which the GetPath method will look for;
8: The queue contains all the children of the current node after fillQueue();
9: Start is set to the current node after each node selection;

This method loops until the goal node is reached. In essence it sets the variable Start to the next node that the algorithm will expand. Then a queue will be filled with al the children the node points to. This queue will be used to determine which node, out of all nodes seen, has a combination of categories that is closest to that of the goal.

```
1: procedure FILLQUEUE(Node Start, Node Goal)
2:     Node Current = Start;
3:     Current.visited = true;
4:     for Each ChildNode N in Current.Childs do
5:         Node Child = N
6:         if !Child.visited then
7:             if (Child == Goal) then
8:                 GoalFound = true;
9:                 Goal.Setpointer(Current);
10:                Break;
11:            end if
12:            Child.DistanceFromGoal = GetAvgCategoryDistance(Child,
    Goal)
13:            OrderdNodes.insertChild                ▷ queue sorted by
    DistanceFromGoal
14:            Child.SetPointer(Current)
15:        end if
16:    end for
17:    Node next = FindNext(OrderdNodes);
18:    return next
19: end procedure
```

Annotation to Fillqueue(Node Start, Node Goal)

9/14: The path from Node Start to Node Goal will be retraced through the pointers. When a child is visited the first time the pointer is set to the current node. After the goal is reached the pointers are followed until "done" is reached;
13: Each child is entered in a list ordered by the value of DistanceFromGoal. The nodes with an average category distance closest to the goal first;
17: Node next = the first node in the OrderdNodes queue that has not been visited.

All of the child nodes of the current node are reviewed and checked to see if there is a match with the goal node. If the child has not been visited yet it is added to the queue. This queue is sorted by the DistanceFromGoal, this has to be computed using the GetAvgCategoryDistance(Child, Goal) method. Finally a pointer is set in order to remember the path in which the child was visited. Note that these pointers are used to avoid having to create a matrix containing this information.

```
1: procedure GetAvgCategoryDistance(Node Child, Node Goal)
2:     double sum = 0;
3:     double total = -1;
4:     for Each Goalcategory G in Goal.Categories do
5:         for Each Childcategory C in Child.Categories do
6:             double temp = GetCategoryDistance(C , G);
7:             if  G != C then
8:                 sum = sum + temp;
9:             else
10:                sum = 0;
11:                break;
12:            end if
13:        end for
14:        total = total + (sum / Child.NumbOfChildcategories) ;
15:    end for
16:    return total Goal.NumbOfGoalcategories;
17: end procedure
```

To find the average category distance between two nodes, the distance between all pairs of categories is computed using the GetCategoryDistance method. For each goal category the distance to all of the child category is added and an average value is calculated by dividing the amount by the number of child categories. The average distance of one goal category to all the child categories is obtained. This value is set to zero however if the goal category matches one of the child categories. This is done in order to lower the average category distance between nodes in the case of a category match. Finally this sum is divided by the number of goal categories. This is done to make sure that a node with less categories will not have a greater chance of being selected. The table below explains.

Goal categories	Child Categories	sum	total
A	B	1	0
	C	3	0
	D	6	6
Returned avg category distance	2		
A	D	3	3
B		2	5
C		1	6
Returned avg category distance	2		

```
1: procedure GetCategoryDistance(Category C, Category G)
2:     int [ ] left , int [ ] right;                    ▷ filled with −1
3:     q.add(C)
4:     while !GoalFound && ! q.isEmpty do
5:         for Each ParentCategory p ϵ q.first.Parents do
6:             SetParentLeft(p, q, left);
7:         end for
8:         q.remove(first);
9:     end while
10:    q.clear();
11:    if !GoalFound1 then
12:        q.add(G);
13:        while !GoalFound2 do
14:            for Each ParentCategory r ϵ q.first.Parents do
15:                SetParentRight(r, q, right);
16:            end for
17:            q.remove(first);
18:        end while
19:    end if
20:    GetPath(left, right, GoalFound, GoalFound2)
21: end procedure
```

Annotation to GetCategoryDistance

To find the distance between two categories the taxonomic structure is used. The parent relationship is used to find the distance between categories. The path is split into two sections, left and right. First the childCategory is added to a list. After each pass of the for loop in lines 5-7 all of the parents of the categories currently in the list are added. The SetParent method adds the parents to the list and sets a pointer in order to remember the shortest path in the taxonomy tree. This is done until the top of the tree is reached. Then for the goal category the same is done until a category is reached that was visited on the left side. Finally the GetPath method finds the distance between the categories using the pointers in the left/right arrays. This methods works similarly to the GetPath method.

```
1: procedure SetParentLeft(Category P, List l, int[ ] left)
2:     if P == Goal then
3:         GoalFound1 = true ;
4:         P.visitedBFS = true ;
5:         left[indexOf(P)] = indexOf(l.first);
6:     end if
7:     if !P.visitedBFS then
8:         l.add(P) ;
9:         P.visitedBFS = true ;
10:        left[indexOf(P)] = indexOf(l.first) ;
11:    end if
12: end procedure
```

Annotation to SetParentLeft

In lines 5 and 11 a pointer is set at the position of the indexOf(category). The indexOf method gets the index of the given category in a list of all categories. The value is set to the index of the corresponding Child. The GetPath method retraces the pointer array to a -1 value and counts the number of steps. This is done to avoid using an matrix which stores the parent-child relationship, which again is not possible due to lack of memory. This method could however return an incorrect distance if implemented as above as illustrated below. The problem occurs due to the fact that the Wikipedia category graph could contain cycles [15]. In a research done in 2006 six only few cycles where found, however since then the category graph has surely grown an which could increase this amount.

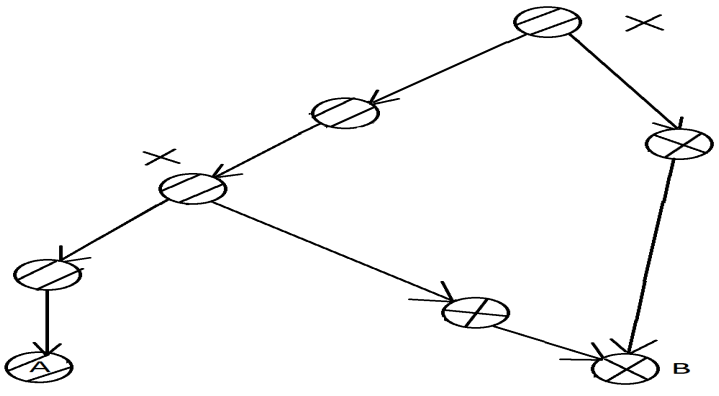

Figure 3.2: GetPath method example.

In figure 3.2 an example is given to show how the distance between Node A and Node B is found. The nodes visited by the left part of the algorithm are striped, the nodes visited by the right part of the algorithm are crossed. Node B has two parents which each have a parent that is visited by the left side of the algorithm. However the distance only on path will result in the shortest distance. To make sure the shortest path is a check is done making sure the depth of the Breadth First search is ultimately equal to the length of the shortest path. This check is not displayed in the GetCategoryDistance method.

3.3 Results

Several attempts where made to get around the problems faced due to the high memory usage. The first being the use of a subset of the data. The category taxonomy was used to subtract a subset of categories. The articles used for a shortest path estimation were located within this subset.

3.3.1 Category subset

A subset was created using a tool which can subtract a subset of categories from the taxonomy. For the subset the root chosen was the category "Computer Science" and a subset of depth six was created. This resulted in the largest possible subset the tool could generate from that root. The language chosen for the categories was English and not Simple English due to the fact that the taxonomy used is that of English Wikipedia categories. These categories however only match the Simple English categories about 75%. Unfortunately the Simple English Wikipedia does not have a taxonomy tree that is easily obtained. An article has at least one category that is in the category subset.

Subset	Size	percentage of total
Categories	12,125	2,56%
Articles	3,010	1,85%

Table 3.4: Category subset statistics [6].

The subset was used to attempt to estimate the shortest path between two nodes. A table was created to store the shortest paths between the subset categories. This however created more problems. Because a taxonomy based subset of categories is used, all categories in the subset are relatively close to each other. Each node in the subset could only be compared by categories that were located within the subset. Because of this the node selection process was not precise enough. Every node had an average category

distance to the goal that was very low. Most nodes only had 1 category within the subset. This resulted in the algorithm expanding almost every node it encountered, because it seemed as though it was close to the goal. Mostly this resulted in expanding all nodes of categories that where close to that of the goal without ever reaching a goal category.

This resulted in the conclusion that all, or at least more, categories are needed to compare nodes. When a taxonomic subset is used the distance between categories is limited to the depth of the subset. The distance between the categories outside the subset are needed to widen the range of the average distance. When all categories are used each node will have a more unique combination of categories and thus a low average category distance would also be more unique. However we already established that creating an APSP table for the categories was not possible. If JAVA byte integers are used it is likely that all distances can be stored within the range of -128 to 127. Assuming that all distances are smaller than 256 steps. At best this means that an APSP table could be made using 8 bits for each pair. This would mean that the table could be created using 209GB of data. The algorithm however would not be able to copy that table during computation. Besides that the time needed to construct this table was estimated at 30+ days using Johnson's algorithm, after short testing. To long to attempt in the remaining time to complete this paper.

3.3.2 Dynamic category distance computation

Because a static computation of category distances was not an option the choice was made to compute the distance between categories dynamical using a BFS [Section 3.2.4]. We found that due to the fact that the category distance has to be computed so often the time needed to select the next node to expand is very high. Efficiency was improved by saving all category distances of category pairs with one goal category. Because the pairs (GoalCategory, OtherCategory) might return often. These pairs would then only have to be computed once. Also the use of more efficient data structures in the JAVA code greatly improved efficiency. Still an average computation would take over a 100 hours to expand to roughly 18% of the nodes. Because of this not enough data could be collected to analyses the algorithms efficiency in terms of the estimated shortest path distance compared to the actual distance. But the data collected did show that the algorithm could often easily find a path to the right category. The problem was finding a path to the goal from there, because so many articles would be evaluated in the process.

Finally an attempt was made to once again use a category subset to reduce the time needed to estimate a shortest path. Instead of using the taxonomy to find a category subset, 50% of the categories where chosen at random and nodes where selected with the same criteria as with the first

subset. This time the category distances where computed dynamical. This however often resulted in the algorithm reaching a dead end with nodes that did not contain children with any category in the subset.Still the time needed to compute shortest paths between categories had not decreased either. Because these distances needed to be computed in the full graph.

3.4 Possible solutions

3.4.1 Extended BFS

Besides obtaining more memory and computing the category APSP statically, which still would be a lengthy computation. There might be ways to work around the problems faced. Throughout the research done it seemed clear that it is not hard to find a path to nodes which have a low average category distance. After that however finding the goal takes significantly longer. To reduce the number of category breadth first searches the algorithm could be adapted to first find a path to nodes that have a certain minimum average category distance and then switch to a BFS starting from the current node. Perhaps the BFS will have improved as it always start from a category which is closely related to the goal.

3.4.2 Extended selection process

Another solution might be to once again first move to a node with a low average category distance. Then instead of further using category distances select nodes by only expanding to nodes that share one or more of the goal categories. Selecting nodes with the most matches first. Assuming that the goal node is most likely to be linked to by a node in the same category. Less nodes have to be visited then with a BFS. Furthermore this experiment fully relies on the effectiveness of the categorization of articles. Compared to the extended BFS proposal.

Chapter 4

Related Work

4.1 Social networks

Social networks have been a target for many empirical studies for a long time, however recently there has been an increase in studies on small-world networks due to the applications in social search [8]. The question of how to find the shortest path between nodes in a small-world network is tackled in different ways [8] [4] [11], but mostly the problems faced are the same. In the first empirical studies by Milgram for instances participants had to send a letter to an individual either directly, if they knew the individual, or indirectly by sending it to someone who might know where to send the letter [11] [10]. With these so called letter chains an estimation of the shortest distance between the participating population could be made. However this could only be done in a small subset of the population. The network of people was just to big. A problem that in modern small-world research, including our research, still occurs.

4.2 Shortest path in social networks

Several other papers propose a way of efficiently searching for shortest distances in a small-world network. The landmark selection process proposed by Potamias has found a way to lower run time computations significantly [8]. This is done by computing the needed landmarks beforehand and doing it only once. Our solution to this problem, based on the idea that the category hierarchy can be used to obtained information pre-run time, simply can not be used in the same way. We can assume the the information provided by the categories is useful in estimating the shortest path [15] [13] and that using a hierarchy proved to be a good strategy in simulations [4]. However because of the fact that this information can not be stored because the APSP of the category graph can not be stored [8], it currently does not decrease the time needed to compute shortest paths in a large network

enough to prove that our assumptions are correct.

Chapter 5

Conclusions

The inability to store the APSP of the Wikipedia category graph has made it impossible to show that the Wikipedia categories can be used to efficiently estimate the shortest distances between Simple English Wikipedia articles. However this thesis has shown that the category graph is likely to contain information that can used for a node selection process. Research has shown that the inability to store the APSP table of the Wikipedia category graph, has caused the proposed solution to significantly lose efficiency when searching in the Simple English Wikipedia articles graph. Because the Wikipedia category graph $>$ Simple English Wikipedia articles graph in terms of the number of vertices and edges.

This however does not mean that the same will hold for the English Wikipedia articles, which in turn is larger then the Wikipedia category graph in terms of the number of vertices and edges. This perhaps could negate the time lost having to dynamically computing the shortest distances between categories. In order to fully show that an Wikipedia category taxonomy could be used to estimate the shortest paths between Wikipedia articles either the APSP table for the category needs to be created, i.e. more memory needs to be obtained, or the hypotheses should be tested within the English Wikipedia.

Bibliography

[1] Smith A. 6 new facts about facebook. "http://www.pewresearch.org/fact-tank/2014/02/03/6-new-facts-about-facebook/", 2014.

[2] Johnson D. B. Efficient algorithms for shortest paths in sparse networks. *Journal of the ACM (JACM) 24.1*, pages 1–13, 1977.

[3] Watts J. and Strogatz S. Collective dynamics of 'small-world'networks. *nature 393.6684*, pages 440–442, 1998.

[4] Adamic L. and Adar E. How to search a social network. *Social Networks 27.3*, pages 187–203, 2005.

[5] Amaral L. A. N. et al. Classes of small-world networks. *Proceedings of the National Academy of Sciencs 97.21*, pages 11149–11152, 2000.

[6] Manske M. Catscan. "http://tools.wmflabs.org/catscan2/catscan2.php". Accessed: 14-05-2014.

[7] Newman M. E. J. The structure and function of complex networks. *SIAM Review*, 45:167–256, 2003.

[8] Potamias M. et al. Fast shortest path distance estimation in large networks. *Proceedings of the 18th ACM conference on Information and knowledge management*, 2009.

[9] Floyd R. W. Algorithm 97: shortest path. *Communications of the ACM 5.6*, page 345, 1962.

[10] Milgram S. The small world problem. *Psychology today 2.1*, pages 60–67, 1967.

[11] Schnettler S. A small world on feet of clay? a comparison of empirical small-world studies against best-practice criteria. *Social Networks 31.3*, pages 179–189, 2009.

[12] Schnettler S. A structured overview of 50 years of small-world research. *Social Networks 31.3*, pages 165–178, 2009.

[13] Ponzetto S. P. and Strube M. Deriving a large scale taxonomy from wikipedia. *AAAI. Vol. 7.*, 2007.

[14] sourceforge.net. "entire" wikipedia category taxonomy. "http://http://wikicategory.sourceforge.net/", 2010. Accessed: 17-04-2014.

[15] Zesch T. and Gurevych I. Analysis of the wikipedia category graph for nlp applications. *Proceedings of the textGraphs-2 Workshop (NAACL HLT 2007)*, 2007.

[16] Zlatić V. et al. Wikipedias: Collaborative web-based encyclopedias as complex networks. *Physical Review E 74.1*, 2006.

[17] Wikipedia. Wikipedia statistics english. "http://stats.wikimedia.org/EN/TablesWikipediaEN.htm/", 2014.