Finding Important Road Intersections using a Neighbourhood Approximation Function

Social Network Analysis for Computer Scientists — Course paper

https://github.com/mattiejas/SNACS-road-intersections

Matthias Aarnoutse
m.f.a.aarnoutse@umail.leidenuniv.nl
Leiden University
Leiden, Netherlands

Niels Witte
n.witte@umail.leidenuniv.nl
Leiden University
Leiden, Netherlands

KEYWORDS

neighourhood approximation, hop exponent, large graphs, social network analysis, network science, ANF

ACM Reference Format:
Matthias Aarnoutse and Niels Witte. 2021. Finding Important Road Intersections using a Neighbourhood Approximation Function: Social Network Analysis for Computer Scientists — Course paper https://github.com/mattiejas/SNACS-road-intersections. In *Proceedings of Social Network Analysis for Computer Scientists 2021 (SNACS '21).* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The question *"What are the most important nodes in a graph?"* is becoming an increasingly important question over the past 20 years with applications ranging from indexing the web to finding actors of organised crime. This question can be answered using a tool called *Approximate Neighbourhood Function* (ANF) [12] which aims to approximate the size of the neighbourhood for any given node whilst scaling incredibly well for larger graphs.

In this paper we further prove the capability of ANF by employing it on a data set of road networks of countries in Europe. Our main goal is to compare the road network of The Netherlands to the rest of Europe. We do this by answering the following questions:

- What are the countries with the highest connectivity in Europe and how does the Netherlands compare against them?
- What impact do road closures have in the Netherlands compared to the rest of Europe?
- How does population density relate to road connectivity in the Netherlands?

The remainder of this paper is organised as follows; Section 2 discusses previously done related work, Section 3 presents the basic definitions used for this paper, Section 4 describes our approach to solving the problem, Section 5 describes our data set used to verify our solution and Sections 6, 7 describe our experiments and our results, respectively, we then conclude our paper with Section 8

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
SNACS '21, Master CS, Fall 2021, Leiden, the Netherlands
© 2021 Copyright held by the owner/author(s).
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 RELATED WORK

In order to answer our research question we would need to calculate an individual neighbourhood function $IN(u, h)$ and the neighbourhood function $N(h)$. These functions are defined in Section 3. The problem with calculating the neighbourhood function of a graph is that it requires a lot of memory. For every node the individual neighbourhood needs to be computed. This could be implemented by using a simple breadth-first search to calculate $IN(u, h)$, do this for every every node in V, sum the result, and you get $N(h)$.

Up until the introduction of ANF [12] the only approximation technique appeared to be *RI approximation* [4] claimed by [Palmer et al.], the authors of ANF. This RI approximation technique makes use of breadth-first search which makes it ill-suited for larger graphs due to the random-like access to the edge-list.

ANF seeks to solve this problem and introduces an approximation to the neighbourhood function without random-like memory access, instead it follows the order of the edge list.

This led to the introduction of HADI [8] which is a MapReduce-based distributed implementation of ANF. Around the same time HyperANF [3] showed up, which is an improvement upon ANF. Where ANF uses $O(n \log n)$ memory, HyperANF scales almost linearly with $O(n \log \log n)$.

As the ANF paper released in 2002, the implementation found on the internet was written in C and extremely complex. The authors of HyperANF instead compared their solution with an implementation by the tool `snap`. They state that it only works on small networks due to the high memory footprint of `snap` [3]. Which ultimately defeats the purpose as ANF was created in order to use on larger graphs. To see what it would be like to run ANF on a larger network we would like to implement a modern day solution using Rust [9] ourselves. One of the motivations in the original ANF paper was the question *"Which set of street closures would least affect traffic?"*, we used this as our inspiration for this paper where we investigate the road network of Europe.

3 PRELIMINARIES

Let $G = (V, E)$ be our graph with n vertices (V) and m edges (E). These symbols summarised in Table 1 are used throughout the paper.

The individual neighbourhood function as shown in Equation 1 calculates the number of nodes for u at a distance h or less, where $dist(u, v)$ represents the number of edges in the shortest path of u to v.

Name	Description
n	Number of vertices
m	Number of edges
V	Vertex set $\{0, 1, \ldots, n-1\}$
E	Edge set $\{(u, v) : u, b \in V\}$
d	Diameter of the graph

Table 1: Commonly used symbols

$$IN(u, h) = |\{v : v \in V, dist(u, v) \leq h\}| \quad (1)$$

The previous individual neighbourhood function can then be extended to calculate the entire neighbourhood for each node in V.

$$N(h) = \sum_{u \in V} IN(u, h) \quad (2)$$

While $N(h)$ can be used to compare different graphs, we are more interested in the individual neighbourhood function. We define a road intersection as important when it connects to a lot of other different roads. This is exactly what the individual neighbourhood function represents, the importance of a node.

ANF-0. Traversing a graph in a breadth-first manner causes access to the edge list in a random order, which leads to problems when the entire graph no longer fits in memory. [Palmer et al.] propose a method which iteratively builds the neighbourhoods for every node x at the same time. This can be achieved by repeatedly iterating over the edge list and tracking the set of nodes currently in the neighbourhood x and adding nodes to it over time.

Algorithm 1 Basics of ANF algorithm, please note that $\mathcal{M}(x, h-1)$ represents the neighbourhood of x from the previous, already computed, distance step.

```
M(x, 0) = {x} for all x ∈ V
for each distance h do
    for each node x ∈ V do
        M(x, h) = M(x, h − 1)
    end for
    for each edge (x, y) do
        M(x, h) = M(x, h) ∪ M(y, h − 1)
    end for
end for
```

The basic ANF algorithm as presented in Figure 1 demonstrates the idea that the neighbourhood of x with distance h contains the nodes from the neighbourhood of y with distance $h-1$ given $(x, y) \in E$. We can track the neighbouring sets by using a bit mask $M(x, h)$ of length n. This however, still requires $O(n^2)$ memory space in order to track the sets. [Palmer et al.] therefore propose to use a probabilistic method [7] in order to track the size of a neighbourhood.

The goal is to compress the amount of space required to store the number of nodes which are part of the neighbourhood, this is were the probabilistic counting comes in. Instead of giving every node its own potential spot in a set, several nodes or 25% in fact, are assigned to bit 1 and 12.5% assigned to bit 2. This can be generalised to a string of length $\log n + r$ with every bit having a chance of $1/2^{i+1}$ of being set to 1. In order to add a node to the neighbouring set a bitwise-OR operation is performed on the masks. This results in a new approximation for the set including the added nodes.

When evaluating the value of the approximated bit masks we check for the first 0 in the bit mask. The position of this first 0 determines the approximated size of the neighbouring set. If, for example the first bit is found to be not set, we can say that that approximately 4 or less nodes have been visited, and thus the size of the neighbouring set is likely to be 4 or less. If however the second or third bit are set to 0 we can say that the size is between 4-8 and 8-16 respectively. [Palmer et al.] calculate the size as 2^b where b is the index of the first bit that was not set (0). In order to make the numbers more stable we can execute the algorithm multiple times in parallel and take the mean of the results.

Improvements. There are two variations on ANF-0; the algorithm ANF itself and ANF-C. The former is an improvement for bigger data sets as they are dominated by the I/O costs. The latter improvement is compression on the bit masks which would have the most performance gain on smaller data sets.

The ANF-0 algorithm reads the edge list in order, but accessing the previous set of bit masks is effectively random order [12]. Recall that in the ANF-0 algorithm $M(x, h)$ is written to with a bitwise-OR including the value of $M(y, h-1)$. Keeping performance up would mean that both the previous and the current set should be in-memory. As this probably is not the case, memory swaps will happen which kills the performance. In order to solve this issue, one pass of bucket sort is used to partition the edges. Having partitioned the edges results in predictable I/O, which in turn allow for prefetching buckets.

The second improvement (ANF-C) is applying leading ones compression on the bit strings. As the bit strings are initialised with the probability $1/2^{i+1}$ most of the bit string accumulate a few leading ones, such as:

$$1111111110xxxxxx$$

Additionally, as we run k parallel approximations, with each many leading ones, we can apply *bit shuffling*. Interleaving bit strings such as

$$11010, 11100 \Rightarrow 1111011000$$

results in more leading ones, which makes the compression more effective. The number of leading ones is being tracked as a counter, which can be prepended to the bit string. [Palmer et al.] claim that the compression of leading ones result in an improvement up to 25% from the ANF algorithm.

4 APPROACH

We plan on answering our research questions by using a similar strategy as proposed by [Palmer et al.] which uses the *Hop Exponent* of a given node, $\mathcal{H}_x$. This exponent is defined as the gradient of the line in log-log space given by the neighbourhood function over distance $h \rightarrow d$ for any given node x. This solution is only

Rank	Country	Nodes	Edges
1	France	64953	190004
11	Netherlands	3705	10870
36	Liechtenstein	14	36

Table 2: Network sizes of largest (France), the smallest (Liechtenstein) and Netherlands

viable because of the power-law distribution found in the sizes of neighbourhoods for many real world networks [6].

The resulting *hop exponents* can be seen as a way of measuring the connectivity of a given node in our graph which is what we are going to use as a metric to locate the most important nodes in any given country. We will then aggregate this data for every European country in order to get a ranking of the most robust and connected countries. We finally cross-reference the data specific to The Netherlands with a map of density population in an attempt to find relations between the population density and the connectivity within a given area.

5 DATA

Our primary source of data is Open Street Map (OSM) [10] which is a massive database containing all sorts of cartographic data, including networks of roads. Country specific road data is downloaded and converted from a binary format into `XML` using `osmium-tool` [1]. The `XML` containing the nodes and edges is converted using `OsmToRoadGraph` [2] into a `networkx` graph in `JSON`-format. It then outputs an edgelist which will be used for our ANF-0 implementation. `osmium-tool` also allows for filtering of specific road segments of which we have selected *motorway*, *trunk* and *primary*. We also select the links between these roads by adding *motorway_link*, *trunk_link* and *primary_link*. This increases the chance of the graph being connected. We further narrow this down by selecting the giant component from every graph.

In order to represent a bend in a road, OSM introduces extra nodes with a degree of 2. These clutter up the dataset and introduce noise into the neighbourhood function as a single road could now exist out of thousands of nodes, even though it does not have many connections, it just bends very frequently. In order to solve this problem we simplify the graphs by taking all nodes of degree 2 introduce a new edge which shortcuts this node and then delete the original node. These steps result in the main road network for all European countries. In these unweighted and undirected graphs an edge represents a road segment with nodes representing the merging of at least 3 road segments. Every exit on a motorway would be a node, for example.

Table 2 presents several networks representing European countries. Our largest network is the one of France, this makes sense since it is also the largest country in Europe (Ukraine and Russia are not part of the dataset). Our smallest network is the network of Liechtenstein. The Netherlands has a respectable network size, landing on the 11th position.

The urban data regarding population density was retrieved from Eurostat[5] which is the statistical office of the European Union. They provide access to their database containing a ton of urban data. We specifically aggregate the population density statistic for the *NUTS0* level. We use the most recent values available for every country and thus we have to use the 2018 value for the United Kingdom. Table 3 presents the top and bottom 3 regarding density. We see that Malta has a significantly higher population density that any other entry on the list being 3 times as much as the 2nd entry; Netherlands, which is our country of interest. The lower ranks seem to be dominated by the Scandinavian countries, with Sweden, Finland, Norway and Iceland being at the bottom of the list. We suspect that this is due to the climate in the northern parts of these countries, as these areas are known to have an extremely harsh climate.

Rank	Country	Density km^2
1	Malta	1595.1
2	Netherlands	507.3
3	Belgium	377.3
36	Finland	18.2
37	Norway	17.3
38	Iceland	3.6

Table 3: Top and bottom 3 countries regarding population density in people per square *km*

6 EXPERIMENTS

Since ANF has such a focus on performance and memory efficiency we were able to run the experiments on our desktop computers equipped with 5th generation Ryzen 7 CPUs and 32GB of RAM. The original implementation of ANF is available on the original authors website [11]. All though this code is still in good shape and fairly well documented, it is a mess of bitwise operators. This makes it very hard to interpret and maintain. We therefore implemented a part of the ANF algorithm in a more modern language called Rust[9]. Rust is a low-level programming language without the overhead of garbage collection available in high level languages such as Python and C#. In contrast to C and C++ it has memory safety which allows the developer to write performant code, without the memory leaks other low levels languages can have. All our experiments where run with $k = 128$ approximations and $r = 7$ extra bits per node as values higher than that show diminishing returns [12]. We report the results for a maximum distance of $h = 20$

Determining connectivity. In this experiment we will be using our own implementations of ANF to approximate the average neighbourhood size over distance, which results in the hop exponent. We will use the country based hop exponent to measure how connected a graph is, with higher hop exponents exponents meaning that a road network of a country is well connected.

Determining impact of road closures. During this experiment we will be analysing the individual hop exponent of nodes for all countries in Europe with the goal of extracting the most important segments of road. We will then proceed to close the top $z\%$ roads and measure the influence on the country based hop exponent. This effectively results in a metric where we can determine the impact

of these road closures on the connectivity within the countries. We compare these metrics and rank European countries

Finding a correlation between population density and connectivity. In the final experiment we analyse the correlation between population density and road connectivity. This is due to the intuition that the Netherlands has an extremely high population density compared to other European countries, and in order to have a road network be able to handle such densities it has to be incredibly dense itself.

Rank	Country	Hop Exponent
1	Hungary	1.778460
2	Austria	1.718059
3	Great-Britain	1.706288
4	Luxembourg	1.696478
5	Croatia	1.685870
15	Netherlands	1.445797
35	Finland	0.541242
36	Liechtenstein	0.313347

Table 4: Countries ranked by hop exponent

7 RESULTS AND DISCUSSION

Table 4 answers the question regarding connectivity. It shows that Hungary is the most connected country according to the hop exponent. We also see that the Netherlands resides somewhere in the middle, at rank 15. Liechtenstein has the lowest rank in the list, which is caused by the size of the neighbourhood stagnating before the selected distance of $h = 20$. This in turn leads to a significantly lower hop exponent.

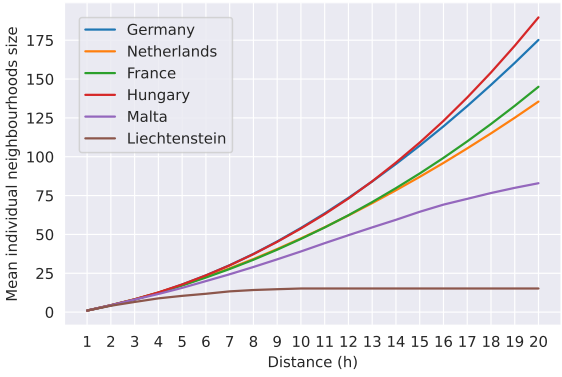

Figure 1: Mean individual neighbourhood sizes per distance

Figure 1 presents the average growth of the neighbourhood for a given country while increasing distance h. Here we picked Germany because it is often praised for its excellent highways, The Netherlands because it is our main country of interest, France because it was our largest network, Hungary because it is the country with the highest hop exponent, Malta because it is the country with the highest population density and lastly Liechtenstein because it was the country with the lowest hop exponent. In this graph we see the series of Malta and Liechtenstein plateau near $h = 17$ and $h = 8$ respectively. This is due to the neighbourhood function reaching the size of the entire network, and thus no longer increases.

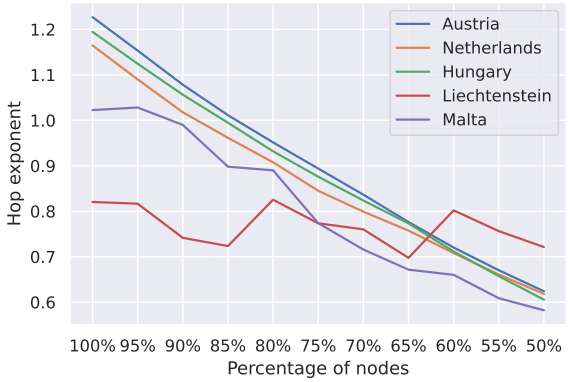

Figure 2: Hop exponent on a subset of nodes

Figure 2 shows the results of our experiment regarding road closures. We see that as the number of total nodes available, which are obtained by removing the top-z% of nodes by hop exponent, decreases, the hop exponent of the global network also decreases. We included Hungary and Austria as they are the rank 1 and 2 on the hop exponent list, Netherlands because it is our topic of interest and Liechtenstein and Malta for their small network sizes and high population density respectively. In the larger networks we see a steady almost linear decline in hop exponent as more nodes are removed where as in the smaller networks we see strange patterns in the line. We suspect that this is due to the poor connectivity that these countries already have, and thus removing only a couple of nodes (15% for Malta) we get disconnected graphs which result in inconsistent results regarding the hop exponent. The Netherlands seems to follow a slightly different trend that Austria and Hungary here, flattening at a slightly faster speed than its competitors.

Figure 3 presents the hop exponent relative to a countries density where every dot represents a single country. From this graph we can clearly see that there is no correlation between the population density and the hop exponent. The one outlier with a population density of 1600 is Malta. The dots with low hop exponents are Liechtenstein, Finland and Iceland.

Lastly, we also evaluated the performance of our Rust implementation. Figure 4 shows the runtime of our implementation of ANF as opposed to the runtime in milliseconds. As the original authors from ANF state, the runtime is linear to the amount of edges.

In this experiment we run ANF on every graph in our data set. The largest network shown in this figure with close to 200K edges (France) only takes approximately 17 seconds to compute the neighbourhood of every individual node for distance of 20.

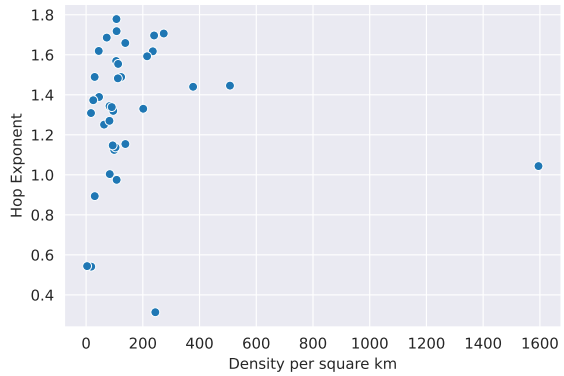

Figure 3: hop exponent vs. density

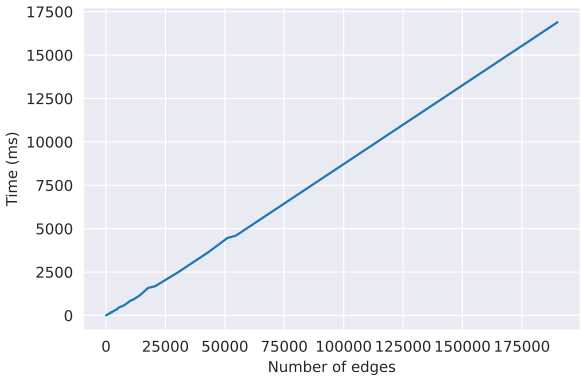

Figure 4: Runtime of the Rust implementation

8 CONCLUSION

In this paper we implemented our own version of ANF [12] and applied to answer multiple questions regarding the structure of road networks for countries in Europe. We used data from OpenStreetMaps [10] and Eurostat [5] and analysed every countries highways. Our results show that Hungary, Austria and Great Britain are the most connected countries when it comes to roads, and also disprove our original hypothesis that connectivity is correlated to population density. We test the resilience of the road networks by removing the top $z\%$ nodes by hop exponent and show that the hop exponent of most countries with large road networks are linearly scaling with the percentage of removed nodes. Once enough nodes have been removed the network falls apart, which occurred in the networks of Malta and Liechtenstein. We finally show that our Rust implementation scales linearly in time with the amount of edges that a graph has.

Doing these experiments made us realise that the Netherlands, although often praised for its incredible infrastructure, is not incredibly connected compared to other countries. We suspect that the praise is caused by more than just the layout of major roads. City planning and different distinct road types could for example be major factors into what could be considered "excellent" infrastructure.

Although ANF is a very old algorithm (2002) we still think that even in modern times it still has its place as it gives access to a statistic, the hop exponent which compared to a degree based measure keeps a more global structure into account.

REFERENCES

[1] [n.d.]. Osmium Tool. https://osmcode.org/osmium-tool/index.html
[2] Andreas. 2018. OsmToRoadGraph. https://github.com/AndGem/OsmToRoadGraph
[3] Paolo Boldi, Marco Rosa, and Sebastiano Vigna. 2011. HyperANF: Approximating the Neighbourhood Function of Very Large Graphs on a Budget. arXiv:1011.5599 [cs.DS]
[4] Edith Cohen. 1997. Size-estimation framework with applications to transitive closure and reachability. *J. Comput. System Sci.* 55, 3 (1997), 441–453.
[5] Eurostat. 2019. Eurostat database; Population density by NUTS 3 region. http://appsso.eurostat.ec.europa.eu/nui/show.do?dataset=demo_r_d3dens&lang=en
[6] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. 1999. On Power-Law Relationships of the Internet Topology. *SIGCOMM Comput. Commun. Rev.* 29, 4 (Aug. 1999), 251–262. https://doi.org/10.1145/316194.316229
[7] Philippe Flajolet and G. Nigel Martin. 1985. Probabilistic Counting Algorithms for Data Base Applications. *J. Comput. Syst. Sci.* 31 (1985), 182–209.
[8] U. Kang, Charalampos E. Tsourakakis, Ana Paula Appel, Christos Faloutsos, and Jure Leskovec. 2011. HADI: Mining Radii of Large Graphs. *ACM Trans. Knowl. Discov. Data* 5, 2, Article 8 (Feb. 2011), 24 pages. https://doi.org/10.1145/1921632.1921634
[9] Nicholas D Matsakis and Felix S Klock II. 2014. The rust language. In *ACM SIGAda Ada Letters*, Vol. 34. ACM, 103–104.
[10] OpenStreetMap contributors. 2017. Planet dump retrieved from https://planet.osm.org . https://www.openstreetmap.org.
[11] Christopher R. Palmer. 2001. ANF v1.0. https://www.cs.cmu.edu/~crpalmer/pubs.html/anf-1.0.tar.gz
[12] Christopher R. Palmer, Phillip B. Gibbons, and Christos Faloutsos. 2002. ANF: A Fast and Scalable Tool for Data Mining in Massive Graphs. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Edmonton, Alberta, Canada) *(KDD '02)*. Association for Computing Machinery, New York, NY, USA, 81–90. https://doi.org/10.1145/775047.775059