Hierarchical Clustering and Visualization of Wikipedia

INTERACTIVE DEMO:
http://slothbigdatademo-env-vvjkvbvppp.elasticbeanstalk.com/

github:
https://github.com/mrotmensch/Sloth/tree/master/Modeling

The Sloth Collective

Justin Mao-Jones
Peter Li
Maya Rotmensch

Table of Contents

1 Introduction

Jimmy Wales and Larry Sanger founded Wikipedia in January 2001 with one goal – offer a free-access, free-content Internet encyclopedia that is editable by the same people who access it. From this simple idea, Wikipedia has grown to more than 35 million articles in nearly 250 different languages[1]. With 500 million unique visitors each month and 18 billion total page views, Wikipedia ranks as the Internet's sixth most popular site[2].

Like all encyclopedia, Wikipedia is "a reference work that contains information on all branches of knowledge"[3]. Unlike traditional paper versions however, Wikipedia exists wholly in the digital realm. This difference in media makes Wikipedia an entirely different type of reference. Since the format is digital and editable by almost everyone, articles can be added and updated with much greater speed. New findings can be added as soon as they are discovered. More importantly, however, Wikipedia's digital format allows for hyperlinks that connect one article to another. As a result, encyclopedia entries are no longer separate and unconnected. Instead, the links between pages creates a rich and complicated graph structure.

For our project, we examine the Wikipedia graph and its community structure. We propose a hierarchical community model built using a stochastic block model. Further, we develop an interactive visualization scheme to display our hierarchical model of Wikipedia. As a result of our work, users can explore the Wikipedia graph in an organized and intuitive manner. Since our model builds communities based on connections, topics are naturally placed into context. Users can gain insight into a particular subject by visiting neighbors. This allows users to discover new, previously unknown concepts and unlocks a previously underutilized component of Wikipedia.

2 Data Processing and MapReduce Analysis

2.1 Data Makeup

To uncover the underlying community structure of wikipedia, we first needed to obtain and process data pertaining to all internal wikipedia links. We chose to use a "data dump" of the English Wikipedia published by the Wikimedia Foundation this March[4].

Our project focused on the analysis of two files: a file containing all internal page link information in wikipedia ("Pagelinks", 33GB), and a file containing extensive information on each individual page which used to complement and decode the the Pagelinks file ("Decoder", 3.5GB). The Pagelinks file was made up of comma separated tuples where each tuple was of the form:

[1] http://stats.wikimedia.org/EN/TablesWikipediaZZ.htm
[2] http://www.alexa.com/siteinfo/wikipedia.org
[3] Merriam-Webster Dictionary
[4] WikiFoundation data dump for the month of March (2015): http://dumps.wikimedia.org/enwiki/20150304/

(source_page_id, destination_page_namespace, destination_page_title, source_page_namespace). There were a few thousand tuples in each row of the file.

The Decoder file also consisted of comma separated tuples with a few thousand tuples making up a row. The decoder file was of the form:
(page_id, page_namespace, page_title, page_restrictions, page_counter, page_is_redirect, page_is_new, page_random, page_touched, page_links_updated(boolean), page_latest, page_len, page_content_model).

We processed both files in MapReduce, parsing relevant information from each separately, filtering unwanted information and joining both processed tables to create tuples of the form: (Source_id, Dest_id). this was the form that was needed as input for the Nested Stochastic Block Model used in a later section.

2.2 Process

Parsing the raw files turned out to be the trickiest part of the MapReduce programs. Since Wikipedia allows users to input a wide variety of characters in page titles (including multiple commas and parentheses in the title name), finding the exact out-of-the-box program or regular expression to correctly parse the each tuple was near impossible . Moreover, due to the sheer size of the data, many regular expressions that parsed smaller sample files correctly would often fail on unsampled varieties of the full file. After a fair amount of trial and error, we finally settled on using an out-of-the-box CSV parser that was robust to many of the title-formatting outliers and supplemented it with our own code to correct incorrectly parsed outliers. For example, the CSV parser often parsed tuples with extremely complicated titles involving multiple punctuation marks, commas and parentheses into many more “sub-tuples” than expected (for instance the tuple may be broken up to 6 pieces instead of 4). Therefore before emitting the parsed tuples in the mapper we implemented a check that ensures that the format of the tuple meets our expectation (each tuple begins and ends with a parentheses). In cases where the tuples were broken down into too fine of a grain, we “stitched” together the divided title sections before emitting the tuple. Even though this program worked much better than any of our other attempts, our program still had trouble dealing with some of the more exotic titles (See the Error Analysis section for details and evaluation).
Altogether, in order to process the data, we created 4 MapReduce programs:

MapReduce Program	Setup Characteristics	Time

Program to clean and parse the Pagelinks file	Run 1: (2 nodes,1 mappers, 1 reducer) Run 2: - 4 nodes - 550 mappers - 6 reducers	Run 1: 18 hours Run 2: ~ in progress [5]
Program to parse and clean the Decoder file	- 2 nodes - 58 mappers - 1 reducer,	48 minutes
Program to join the parsed Pagelinks and Decoder files	- 2 nodes - 550 mappers - 1 reducer,	8 hours and 46 minutes
Program to create id to title decoder for articles only (involved mostly filtering and formatting the output into an easily readable format for future programs)	- 2 nodes - 58 mappers - 1 reducer	41 minutes

2.3 Error analysis

As mentioned above the parsing of the Pagelinks file proved to be particularly challenging, and even after our program ran without failing we wanted to both make sure that the output was correct and measure how many tuples were mis-parsed as a benchmark for performance.
We determined early on that the program containing the CSV parser significantly outperformed any configuration of regular expressions we used by randomly sampling 1000 lines of text from the Pagelinks file and running them through both programs. As the CSV parser program parsed all samples of the file perfectly while the regex programs routinely mis-parsed 1-2 lines out of the sample, we felt confident running only the former program on the entire 33GB file. After running our chosen program on the entire file we then wrote a (sequential) script that cycled through each tuple, determining whether it was parsed correctly by verifying the tuple format and saving any errors into a separate file for further examination. Analysis of the result has shown that only 2,117 tuples out of 892,250,147 were incorrectly parsed. As the mis-parsed portion represented only 0.00024% of the processed data we felt comfortable using the output in future steps of the program. We decided that the output of our MapReduce programs was sufficient given that,

[5] At the time of the first run we were unable to upload the large file in its unzipped form, and we did not realize until later that the zipped format prevented hadoop from dividing the file and assigning it to multiple mappers. In an attempt we solve this problem we've broken up the file into 15 shards and at the time of writing this report are re running the analysis. Initial results successful processing of the data and we predict that the program terminate in substantially less time.

because the number of links so high, a negligible fraction of missing links would not significantly change the structure of the graph.

Examining the file containing all mis parsed tuples, we determine that most of the time the program failed to correctly parse tuples that used non-alphanumeric characters. For example, titles involving complex algebra notation such as "(3Î²,4Î±,16Î²)-',2" or characters not common to the english language such as "Î±-dodecyl-Ï‰-hydroxy" were among the mis-parsed tuples. It is important to note that because of the structure of the file, where there exist multiple tuples in the same row, failure on a single tuple might cause a cascading effect where following tuples also be parsed incorrectly. This is a possible explanation for why even though such unique characters in titles are uncommon over 2,000 tuples are mis-parsed.

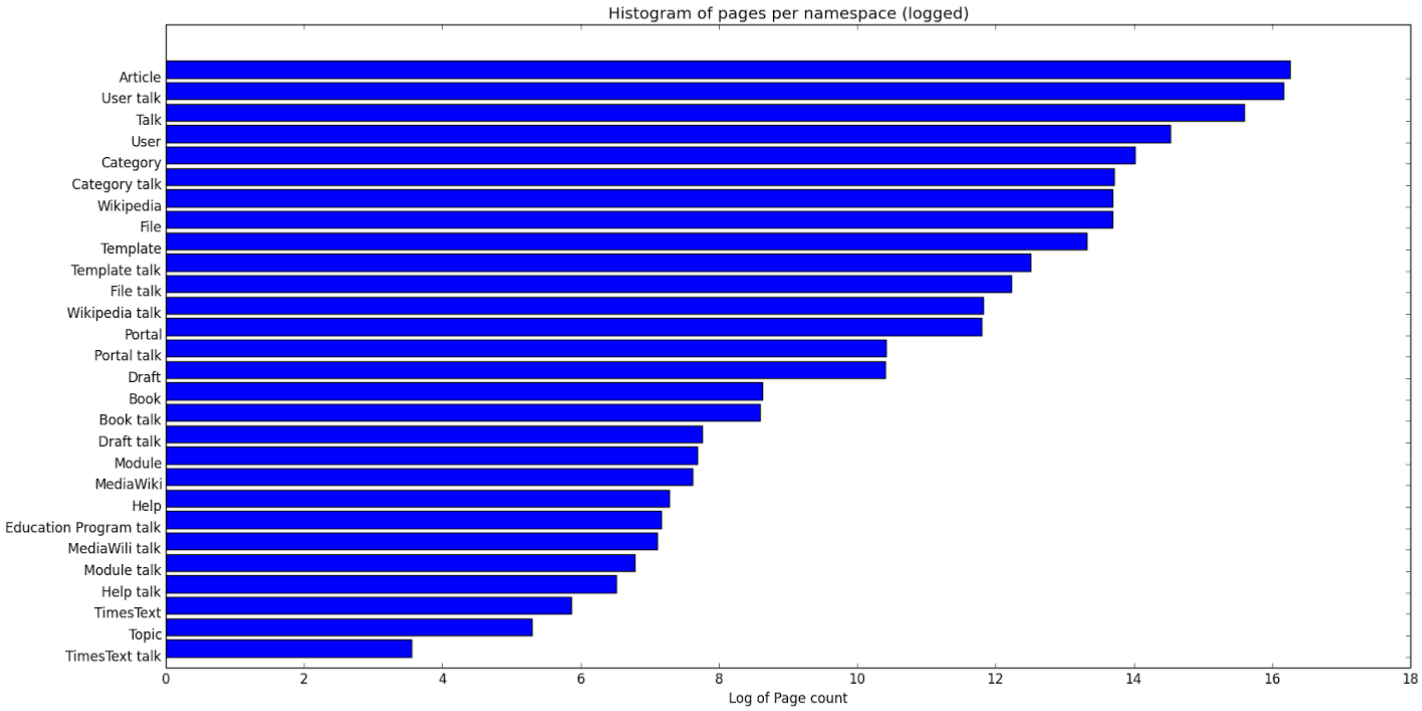
2.4 Data Statistics with MapReduce

To supplement our Data Processing MapReduce programs we also created 5 more MapReduce programs that calculate key statistics about the data:

MapReduce Program	Setup Characteristics	Time
Program to calculate a histogram of pages per namespace	- 2 nodes - 57 mappers - 1 reducer	3 minutes
Program to calculate top 3 pages with most incoming links (from any namespace)	- 4 nodes - 479 mappers - 1 reducer	1 hour and 11 minutes
Program to calculate top 3 articles with most incoming links	- 4 nodes - 479 mappers - 1 reducer	51 minutes
Program to calculate top 3 pages with most outgoing links (from any namespace)	- 4 nodes - 479 mappers - 1 reducer	1 hour and 9 minutes
Program to calculate top 3 articles with most outgoing links	- 4 nodes - 479 mappers - 1 reducer	51 minutes

As a starting point we first visualize a distribution of pages per different wikipedia namespaces in the figure below, where namespaces are wikipedia's way of dividing the information contained within its pages into content (such as articles and categories) and meta data and unpublished

content (such as user information and discussion boards)[6]. For the bulk of our analysis we focus on article-to-article links and therefore filter out pages and links relating to all over namespaces.



In addition to the distribution of namespace we've also calculated:

<u>The top 3 most linked-to articles are:</u>

1) Geographic_coordinate_system (with 914,589 links)
2) United_States (with 538,108 links)
3) International_Standard_Book_Number (with 500,157 links)

<u>The top 3 most linked-to pages are (not necessarily in the article namespace):</u>

1) Sandbox (with 5,129,187 links)
2) WikiProject_Biography (with 5,035,955 links)
3) Introduction (with 4,733,135 links)

It is interesting to note that the most linked-to pages are not articles, and that even the most linked-to articles contain an order of magnitude less links than the top pages. Moreover, all three of the most linked-to pages involve pages concerned with the inner workings of Wikipedia. For example, "Sandbox" is a page spanning multiple namespaces that describes a "Wikipedia feature where anyone can experiment with editing Wikipedia pages" [7]. Likewise "WikiProject_Biography"

[6] http://en.wikipedia.org/wiki/Wikipedia:Namespace
[7] http://en.wikipedia.org/wiki/Sandbox

and “Introduction” are pages that exist across many namespaces and relate Wikipedia mission statement.

Similarly the top 3 articles with the most outgoing links are:

1) New_General_Catalogue (with 7,868 outgoing links)
2) List_of_NGC_objects (with 7,847 outgoing links)
3) List_of_places_in_Afghanistan (with 7,179 outgoing links)

The top 3 pages with the most outgoing links are:

1) Ebraminio/sandbox (with 45,184 outgoing links) - connected to the “user” namespace “containing user pages and other pages created by individual users for their own personal use. “
2) WikiProject_England/WatchAll (with 38,763 outgoing links) - connected with the “Wikipedia” namespace containing pages such as: information, policy, essays, processes, discussion, etc.
3) ChrisGualtieri/Backlog/1 (with 33,969 outgoing links) - connected to the “user” namespace

2.5 Improvements

In order to make our programs as fast and as memory efficient as possible we added the following optimizations to our code. First, whenever possible we always performed any needed filtering on the mapper side rather than on the reducer side. For example, for some of our analysis we were only interested in article to article links, therefore we filter based on namespace in the mapper instead of the reducer. Another example of mapper filtering is apparent in our visualization of the namespace distribution. When creating the visualization we emitted tuple where the key was the page namespace. However, in some of the tuples, the namespace information is missing. Therefore instead of “discovering” that the namespace information was missing in the reducer, we performed a check in the mapper and only emitted tuples with full information for analysis. Furthermore, we used python optimized packages such as CSV and packages with C underpinning such as cString (instead of String) to improve performance. Lastly, while we didn’t suggest a set number of mappers, choosing to rely on the internal hadoop optimization, we did in some of our program increase the number of core nodes to shorten the runtime.

3 Modeling and Computation

Knowledge can be represented hierarchically, and thus we expect that there exists hierarchical orderings in the links between Wikipedia articles. Intuitively, there should be subsets of articles

that form natural communities, such as the articles representing the different variations of jazz. There should also be communities of communities. For example, each genre of music might form its own subcommunity, and the collection of these subcommunities would form an overall community, which in turn could be a subcommunity to an overall 'art' community. Our goal is to find a representation of this hierarchy of communities, and thus, we pursued hierarchical community finding approaches.

Choosing the right approach is a combination of evaluating method concepts, algorithmic complexity, and tool availability. We researched known hierarchical community finding algorithms, concentrating our focus on three specific community finding algorithms: the Girvan-Newman algorithm[8], Louvain Modularity[9], and Stochastic Block Modeling[10]. Given the computational complexity of these methods and the size of the Wikipedia link graph, it was important that we choose the right tool to implement the chosen method. Thus, we explored available tools for implementing these methods. Descriptions and our assessments of these methods and corresponding tools are discussed below.

3.1 Models

2.0.1 Girvan-Newman

The Girvan-Newman algorithm makes use of betweenness centrality to detect communities within a graph. The betweenness of a particular node is determined by counting the number of times it appears in the shortest path between all pairs of nodes in the graph. Thus, it is a measure of how central a node is. The Girvan-Newman algorithm detects communities by progressively removing nodes from the graph in order of decreasing centrality. Communities are identified when distinctly separate clusters appear after the removal of a node. Communities are separated into subcommunities, which are separated further, thus constructing a hierarchical ordering of communities within a graph. Note that this process can also be performed with edges instead of nodes.

A primary issue with the Girvan Newman algorithm is that the determination of betweenness for all nodes in the entire graph requires O(ne) calculations. Using graph-tool[11], which appears to be the fastest open-source graph analytics package we were able to find, a graph size of approximately 14,000 nodes and 1.5 million edges took approximately 30 minutes to calculate on a Dell Westmere 2.67GHz with 1 core. Extrapolating this to the Wikipedia link graph, which contains 11.5 million nodes and 360 million edges, the calculation would take approximately 11 years to calculate. This time could potentially reduced down by a factor of 4 with parallelization,

[8] Girvan M. and Newman M.E.J. 2002. Physical Sciences - Applied Mathematics. 99 (12) 7821-7826; http://www.pnas.org/content/99/12/7821
[9] Vincent B.D. and Guillaume J.L. and Lambiotte R. and Lefebvre E. 2008. Journal of Statistical Mechanics: Theory and Experiment. Issue 10. http://stacks.iop.org/1742-5468/2008/i=10/a=P10008
[10] Peixoto, T. 2014. Phys. Rev. X 4, 011047. http://arxiv.org/pdf/1310.4377v6.pdf
[11] https://graph-tool.skewed.de/

but this would still not be practical. Betweenness can be approximated through random sampling, but we were unable to find this approximation method combined with the Girvan-Newman method in the packages that we surveyed.

2.0.2 Louvain Modularity

The Louvain method attempts to find communities within a graph by assigning each node to a community in such a way that the modularity of the graph is maximized. The modularity of a network is defined as the fraction of edges that do not cross community boundaries minus the expected such fraction of edges if the edges were distributed at random. A graph with higher modularity is more densely connected within communities relative to the connectedness between communities. The Louvain method is a greedy optimization approach, and works as follows:

1. First assigns each node to its own community.
2. Move each node to the community of one of its neighbors that results in the greatest increase in modularity of the graph. If no such move exists, leave the node in its current community.
3. Collapse each of the resultant communities into a single node, and repeat from step 2 until no increase in modularity is possible.

The Louvain Modularity method has O(NlogN) algorithmic complexity. Thus it should be feasible to compute in a reasonable amount of time. This conclusion is discussed further below.

2.0.3 Stochastic Block Modeling[12]

Stochastic block modeling finds communities by maximizing the likelihood of a parametric model, where the parameters of the model are the probabilities that any node in one one community is connected to a node in any other community, including the source community. A nested hierarchy of communities can be formed iteratively by inferring communities on the original graph, then collapsing each community into a single node and weighting edges between these new nodes according to the number of edges between the communities, and then repeating the stochastic block modeling process. This iterates until the entire graph has collapsed into a single node.

Stochastic Block Modeling has the benefit of having O(Nlog^2N) algorithmic complexity. Thus it should be feasible to compute in a reasonable amount of time. This conclusion is discussed further below.

[12] Introductory tutorial: http://tuvalu.santafe.edu/~aaronc/courses/5352/fall2013/csci5352_2013_L17.pdf

3.2 Finding the Right Tool

We quickly learned that choosing the right method was also a function of what tools were available. It turns out that the options are limited. We surveyed several open source packages. We focused on packages implemented in Python, which is the language that our team is most comfortable with. The following table is a summary of the survey.

Package Name	Availability of Relevant Methods
NetworkX[13]	None
igraph[14]	None
GraphX[15]	None
Snap[16]	Girvan-Newman (betweenness cannot be approximated in this package)
GraphLab[17]	None
Sotera[18]	Louvain Modularity
graph-tool[19]	Stochastic Block Modeling

We were able to find the Girvan-Newman algorithm in the Snap package. However, the betweenness calculations were calculated in full and not approximated, which rendered that function inviable for our project. A small test of the Girvan-Newman Snap algorithm on a randomly generated graph of 100 nodes and 1,000 edges took 18 seconds, but a slightly larger graph of 1,000 nodes and 10,000 edges took over four hours.

The Louvain modularity method was difficult to find in standard graph analytics packages. It is available in a distribution provided by Sotera Defense Solutions. We found that there was a bug in the Louvain Modularity calculations on small graphs. We were able to work around the bug, but when deployed to Hadoop clusters on the NYU Dumbo cluster and an Amazon Web Services cluster, the distribution performed incredibly slowly on very small examples. It is possible that we did not implement this package correctly, but we decided early on to abandon this approach, primarily to due to the viability of the stochastic block modeling option.

[13] https://networkx.github.io/
[14] http://igraph.org/redirect.html
[15] https://spark.apache.org/graphx/
[16] https://snap.stanford.edu/snap/description.html
[17] https://dato.com/products/create/open_source.html
[18] http://sotera.github.io/distributed-graph-analytics/
[19] https://graph-tool.skewed.de/

We ultimately chose stochastic block modeling. It is suitable to our problem, has reasonable algorithmic complexity, and is available in graph-tool, which is advertised as a fast, efficient graph analytics package implemented in C++ that parallelizes many of its algorithms. In performance comparisons, it has been shown to be faster than igraph and orders of magnitude faster than NetworkX. Unfortunately, we ran into unresolved problems with the parallelized versions of the stochastic block models in our installation of graph-tool on the NYU Mercer[20] cluster and eventually resorted to single core processing for many of our tasks, though overall this was not a project-ending hurdle.

3.3 Generating Hierarchies

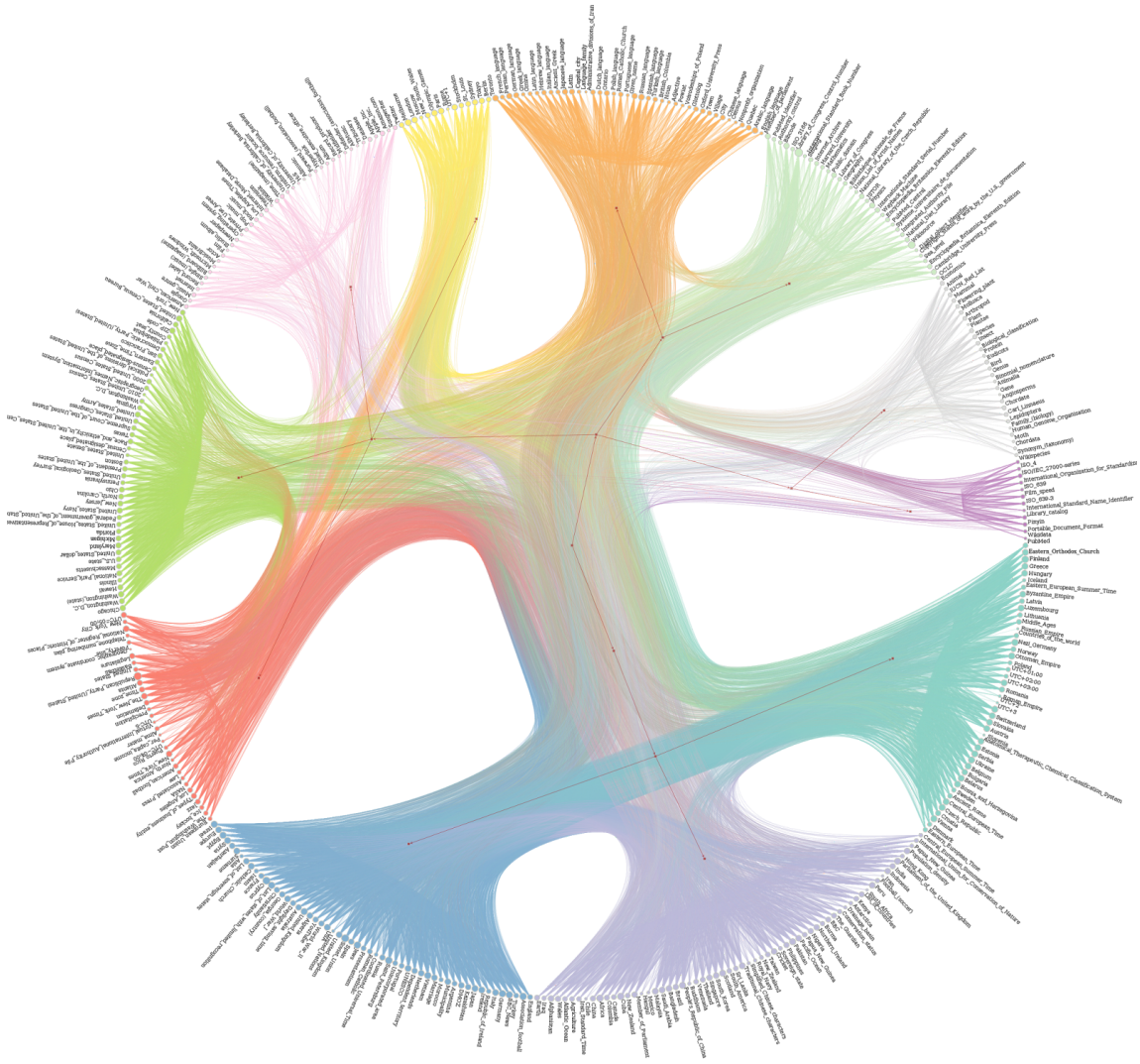
Even with O(Nlog^2N) complexity, the stochastic block modeling calculations take a considerably long time to run and require a large amount of memory. In order to generate results for the project pipeline, we began by reducing the size of the graph by filtering out "unimportant" nodes. We used PageRank[21], a centrality measure used by Google to rank the importance of each page on the internet, to determine an importance score for each wikipedia article. We then set a threshold. Nodes below the threshold were filtered out. We generated several reduced graphs this way.

PageRank Threshold	**# Nodes**	**# Edges**	**Block Modeling (single core)**
5e-5	408	16,499	4 seconds
1e-5	5,106	484,857	184 seconds
5e-6	13,676	1,516,816	1,329 seconds (~22 min)
1e-6	117,153	16,322,339	19,180 seconds (~5 hrs)
5e-7	263,621	36,626,586	78,787 seconds (~22 hrs)
1e-7	1,506,661	197,025,405	*estimated 100 hrs*
Full graph	11,567,005	358,585,207	*estimated 1000 hrs*

[20] https://wikis.nyu.edu/display/NYUHPC/Clusters

[21] Page L., Brin S., Motwani R., Winograd T. 1999. The PageRank Citation Ranking: Bringing Order to the Web. Stanford InfoLab 1999-66. http://ilpubs.stanford.edu:8090/422/

A graph size of 117k nodes required over 5 hours to calculate. We estimated that it would take at least 1000 hours to calculate the nested block model for the entire graph on a single core. Empirical tests on small graphs showed that this computation could be reduced by potentially up to 20% with parallel computations, which would make it 800 hours. Unfortunately, this would still be infeasible for the project. The largest nested block model we were able to generate was for the 5e-7 PageRank threshold, which included 263k nodes, or approximately 2% of the entire graph. The figure below is a plot of the hierarchical groupings of the smallest (408 nodes) graph.



As mentioned previously, we ran into difficulty with the parallelized versions of graph-tool stochastic block modeling. When running on multiple cores, the program would terminate on an error before finishing the block model. Thus we resorted to single core processing for these operations. The table above lists the running times for each size of graph. The running times are roughly equivalent to the O(Nlog^2N) complexity, though it should be noted that the block

models were run on the NYU Mercer cluster and each run could have been performed on a different machine. At the present moment, the largest models are still running.

We also discovered that stochastic block modeling in graph-tool requires a large amount of memory. Empirical tests led us to conclude that 250GB, which is available on the second largest machine in terms of memory size on Mercer, was insufficient to process the full graph. There is only one machine on the NYU Mercer cluster that can handle more than 250GB of memory, and it took a couple of weeks before this machine was finally available to process our nested block model. Due to these wait times, it was very important for us to make sure that our code would run properly once the machine finally became available.

3.4 Latent Node Labeling

Each node in the hierarchy tree of the stochastic block model is latent, meaning that it is a hidden state that is inferred. We can call these latent nodes.

Our next task towards visualizing the Wikipedia links is to label these latent nodes. Each latent node represents a community, or communities of communities, of Wikipedia articles. Thus, the latent node label should describe some core concept common to all of these articles. We decided to use centrality measures to determine these labels. The intuition is that the more central a node is, the more relevance it has across the community, and thus centrality would be a proxy for the importance of a node within a particular community.

Betweenness centrality[22] is an intuitive choice for the centrality approach, since it measures how many shortest paths flow through each node. Betweenness was used on the smaller graphs. We used PageRank on larger graphs, since it is has a much faster calculation time. On the full graph, PageRank can be calculated in under 15 minutes.

A top-down approach and a bottom-up approach were used to append labels to latent nodes via centrality score.

3.5 Top-Down Labeling

In top-down labeling, we label the root of the hierarchy tree with name of the article with the highest centrality score. The article is then tracked and removed from consideration for subsequent labeling of latent nodes. Each child of the root is another subtree. Within each subtree, the centrality scores are re-calculated and the root of the subtree is labelled with the name of the article with the highest score, which is then tracked and removed from future label considerations. This process is performed for all latent nodes using a breadth first search.

[22] Leskovec J., Rajaram A., Ullman J.D. 2014. Mining of Massive Datasets. Cambridge University Press: New York. Ch. 10. http://infolab.stanford.edu/~ullman/mmds/ch10.pdf

The table below shows the calculation times required to process the top-down labeling for both PageRank and Betweenness. Given the amount of time required to sufficiently research the methods and tools, install graph-tool, learn and code in graph-tool, debug parallel computation issues, and 150 hours computation time, we were unable to generate the PageRank top-down labelings for the two largest graph sizes. Betweenness computations were infeasible for the two largest graph sizes.

PageRank Threshold	**# Nodes**	**# Edges**	**PageRank Labelling** O(N+M)	**Betweenness Labelling** O(NE)
5e-5	408	16,499	0.3 seconds	0.6 seconds
1e-5	5,106	484,857	9.2 seconds	161.7 seconds
5e-6	13,676	1,516,816	67 seconds	1,782 seconds
1e-6	117,153	16,322,339	2,770 seconds	*est. 27 hrs*
5e-7	263,621	36,626,586	14,787 seconds	*est. 151 hrs*
1e-7	1,506,661	197,025,405	*est. 57 hrs*	*est. 221 days*
Full graph	11,567,005	358,585,207	*est. 140 hrs*	*est. 9.6 yrs*

3.6 Bottom-Up Labeling

In bottom-up labeling, a recursive breadth first search is used to find all of the bottom layer communities within the hierarchy tree. Once the bottom layer community is found, the centrality scores are calculated for this community, and the latent node is labelled with the concatenation of the names of the articles with the three highest centrality scores. This is performed for all communities on the bottom layer. At the next layer up, the process is performed again, where a name is chosen from within each child subtree, and the top three names are concatenated together to become the name of the current latent node. This continues to the top of the tree. The number of names for concatenation was defaulted to three, but can be adjusted as deemed fit.

The intuition behind bottom-up labeling is that the name of single article cannot possibly encapsulate the nuances in the structure of a community. A small conglomeration of names should provide a more expressive description of the community.

3.7 Hierarchy Tree Pruning

The hierarchy tree generated by the stochastic block modeling may sometimes result in latent nodes with only a single branch. In other words, a latent node has a single child, and the set of articles encapsulated by the latent node subtree is exactly the same set as that represented by its child node subtree. In this case the latent node and its child are collapsed into a single node.

3.8 Conversion to JSON format

Once the hierarchy tree is computed and labelled, the tree must be converted into JSON format for visualization. This requires a simple breadth first search to construct a nested dictionary, which is then converted to JSON in the following format:


```
{
	"children": [
	{
		"children": [
		{"id": 11567185,"name": "Biological_classification"},
		{"id": 11567186,"name": "Authority_control"}
				],
		"id": 11567191,
		"name": "United_States"
	},

	{
		"children": [
		{"id": 11567187,"name": "International_Standard_Name_Identifier"}
				],
		"id": 11567192,
		"name": "International_Standard_Book_Number"
	}
	],

	"id": 11567193,
	"name": "Geographic_coordinate_system"
}
```


Example Data Representation - JavaScript Object Notation(JSON)

3.9 Construction of JSON Tree Tendrils

Even the reduced graphs can still be quite large for a visualization engine. Our goal is to make the graphs visualizable for a typical user. A JSON tree of 100k nodes could potentially consume a relatively large amount of memory on a single laptop. The table below shows file sizes of each JSON tree. Note that the JSON file size does not reflect the amount of memory required to run the visualization.

PageRank Threshold	**# Nodes**	**JSON file size**
5e-5	408	69KB
1e-5	5,106	1.3MB
5e-6	13,676	3.6MB
1e-6	117,153	43MB
5e-7	263,621	98MB
1e-7	1,506,661	*estimated 500MB*
Full graph	11,567,005	*estimated 4GB*

Since a typical user will likely not want to traverse the entire graph, we decided that it would make sense to divide the tree into much smaller chunks. A typical user will traverse the tree one node at a time, and so will only ever want to have a view into subtrees that is perhaps one or two levels deep. Thus we constructed tree “tendrils”.

The tendrils of a node is a tree consisting of that nodes immediate children and grandchildren, and thus only 2 levels deep, though this could be adjusted if so desired. Each tendril is stored in a separate file tagged with the unique id of the root node of the tendril. The root of the entire tree is tagged as “root”. When a user clicks on a child node, that that ID is searched for in the tendrils list, and the relevant file is loaded. This method is allows for the user to be able to traverse a very large tree without having to load the entire tree into memory.

The tendrils are constructed by performing a recursive breadth first search. A python dictionary whose key is the id of each node in the graph and value is a JSON format of the tendril. These key-value pairs are then saved to files.

4 Data Visualization

Data visualization is not merely the display of data. Instead, an effective visualization should reveal the data. In his seminal book, *The Visual Display of Quantitative Information*, Edward Tufte defines a set of criteria for "excellence in statistical graphics". Specifically, graphical displays should:

- show the data
- make large data sets coherent
- encourage the eye to compare different pieces of data
- reveal the data at several levels of detail, from a broad overview to the fine structure
- serve a reasonably clear purpose: description, exploration, tabulation or decoration

For the Wikipedia data, we face an immediate problem. Miller's Law[23] from cognitive psychology tells us the average number of elements a human can hold in working memory is 7±2 elements. With more than 35 million pages and 892 million connections, any direct visualization of the Wikipedia graph will overload the user with data.

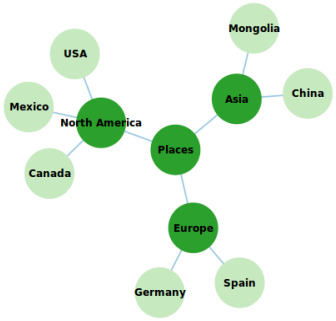
Our solution to this problem is to first build a multi-level model of the data. Using a stochastic block model, we group the data into a collection of nested communities. This give us a hierarchal representation of the data that organizes the data at different scales and allows us to make sense of the connection between Wikipedia pages. For visualization, we work with the hierarchal representation of the original data. Because the data is hierarchical, we can collapse lower nodes in the tree and only show information that is relevant for the current resolution. This is analogous to the display of information in online maps. Street names are not shown when looking at the entire country. This information only becomes relevant when you zoom in.

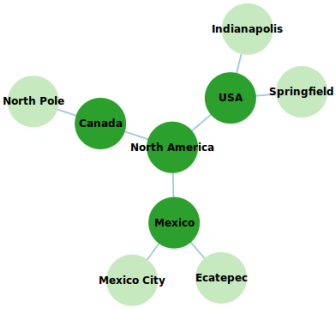
4.1 Visualization Approach

To visualize the hierarchical representation of the data, we explored three approaches: force directed layouts, zoomable circle packing, and Reinhold-Tilford trees. Each of these visualizations has the feature that they allow for zoomable interaction with the data. This allows us to interactively visualize the entire Wikipedia graph without information overload.

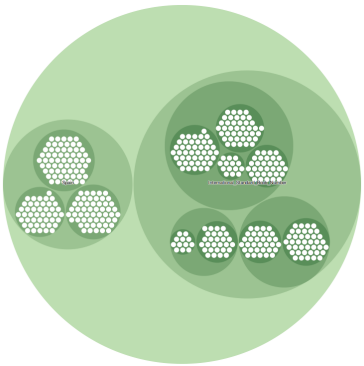
[23] Miller, G. A. (1956). "The magical number seven, plus or minus two: Some limits on our capacity for processing information". Psychological Review 63 (2): 81–97.

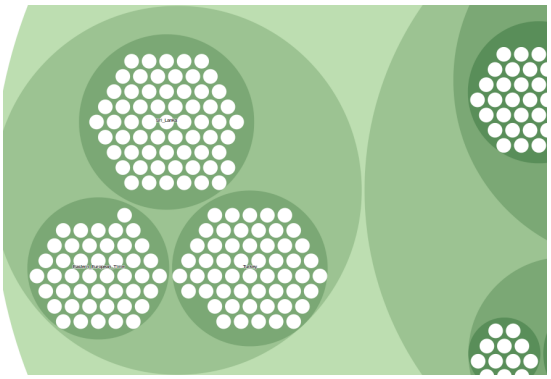
3.1.1 Force Directed Layout



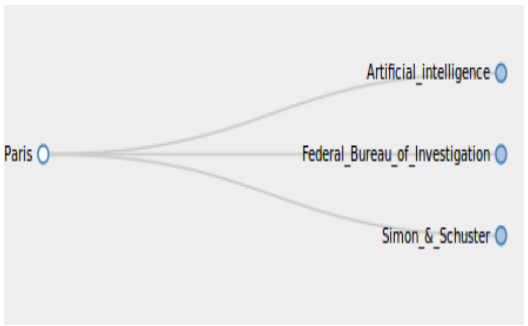


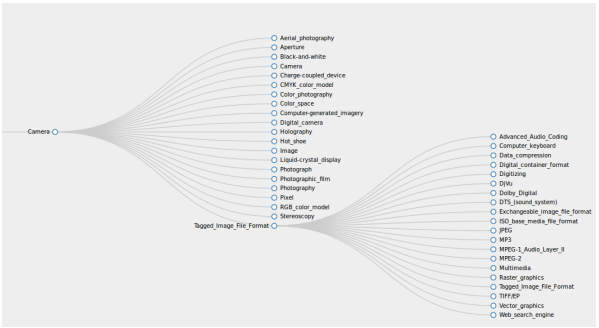
3.1.2 Zoomable Circle Packing





3.1.3 Reinhold-Tilford Trees





4.2 Implementation

For the final deployment, we chose to display the visualization in the web browser. This allows for a high degree of interactivity with relatively few software requirements. The individual

visualizations were created with D3.js. The web demonstration was written in Python using the Flask framework and deployed using Amazon Elastic Beanstalk. Implementation details are below.

3.2.1 Visualization

D3 provides a powerful framework for interactive data visualization. D3 allows you to bind arbitrary data to a Document Object Model (DOM) and perform complex calculations. For our demonstration, the underlying data is stored in a json file that was generated during the modeling step. Through Flask, this data gets mapped to the */data* URL. Each visualization is stored in a separate html file that contains the D3 code and references to */data*. When the Flask application is executed, each visualization html is rendered and assigned a URL. Code for this portion of the project can be found in the visualization portion of our team github[24].

One important consideration for displaying visualizations is performance. Since calculations take place client side, not all visualizations are created equal. In particular, we found that zoomable circle packing slows down considerably as the size of the data grows. This is mostly due to the fact that the circle packing visualization initializes with all the nodes shown. Directed force layout and Reinhold-Tilford Trees do not face this problem since they start in a collapsed state.

3.2.2 Deployment

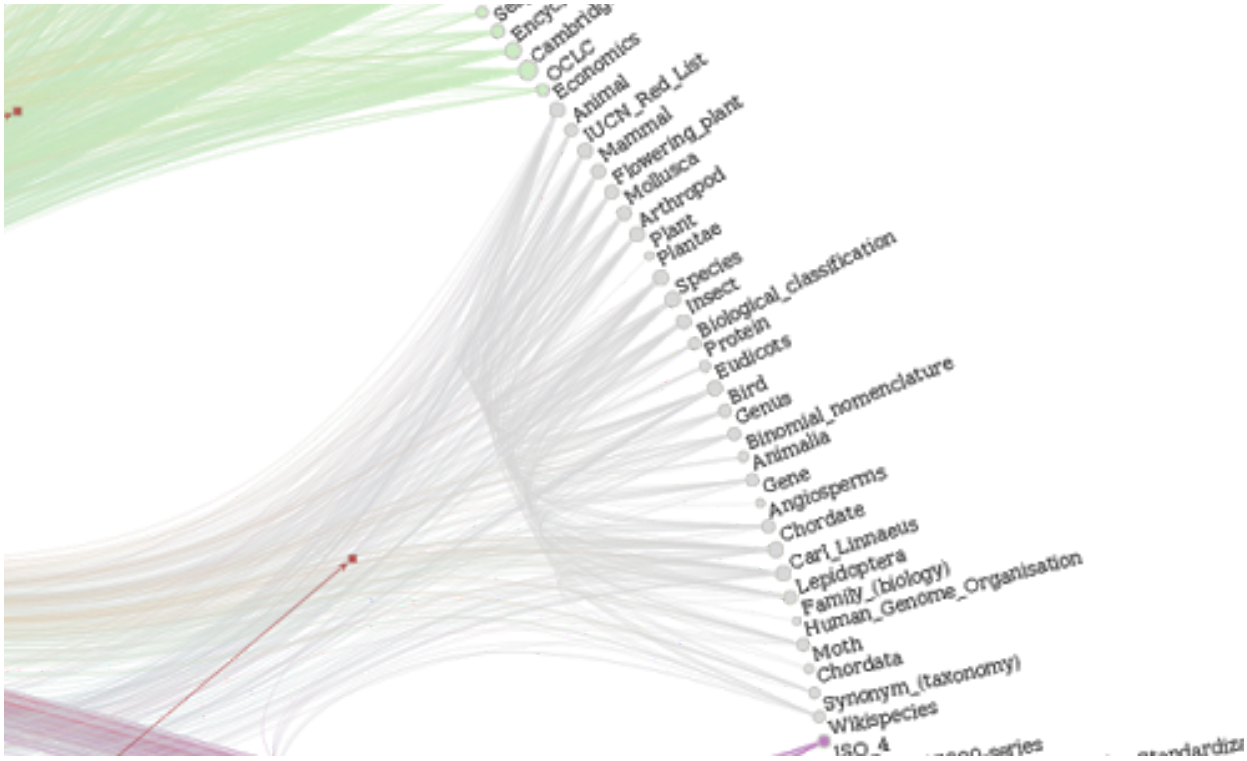
Since the visualization was developed in the Flask framework, deployment was considerably painless. Elastic Beanstalk on AWS automatically launches an environment and creates and configures the AWS resources needed to run code. To deploy our demo, we requested a web server running 64 bit Linux and Python 2.7. With a server instance initialized, the web demonstration was created by pushing the local Flask folder.

5 Evaluation

5.1 Evaluation of the community detection results

Our construction of latent hierarchies is an unsupervised machine learning task, and thus inherently difficult to evaluate. This is somewhat intentional. One aim of our project is to generate a reasonable representation that would uncover interesting relationships that would intrigue a typical user of the visualization tools. For example, consider the figure below, which is one of the communities generated on the smallest reduced graph of 408 nodes.

[24] https://github.com/mrotmensch/Sloth/tree/master/Visualization/WebDemo

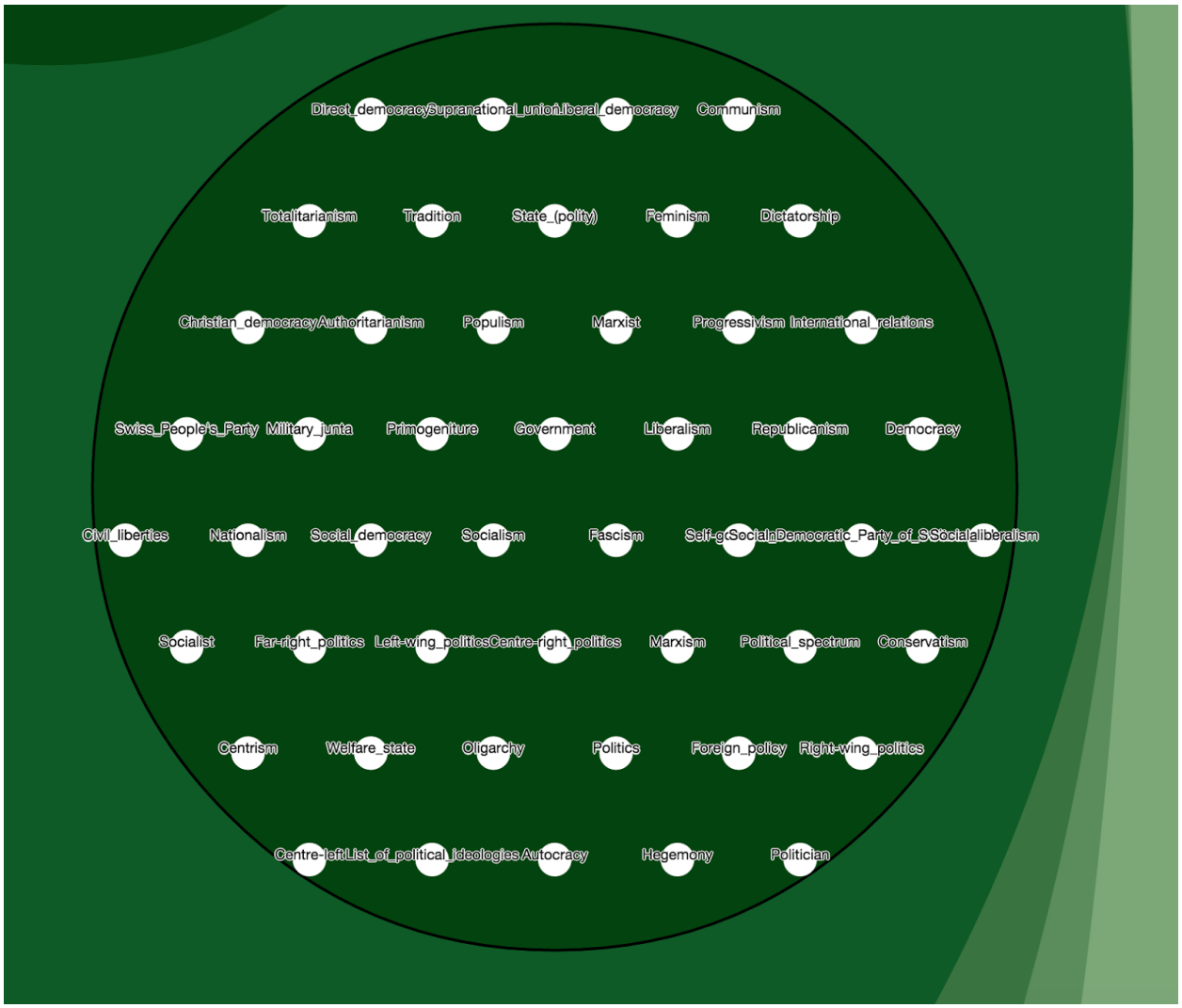


This is clearly a community of biology-related articles. The name "Carl Linnaeus" stands out as a distinctly unique article title. It turns out that Linnaeus laid the foundations for biological naming conventions. This was an interesting finding to the authors of this report. Having this community structure is valuable, because it can be used to provide a map of wikipedia generated purely by links and article titles. This organic generation of communities can in turn lead us to discover interesting and nontrivial connections that are not previously know to the user, such as in the case of Carl Linnaeus. The visualization tool leverages this structure to enable a user to zoom in and out of the map, just like one would do when trying to find things on Google Maps.

It would, perhaps, be possible to have human evaluators rate the quality of each community, but this would be extremely time intensive. It would also be possible to use the human-created categorizations of each article in Wikipedia to evaluate the results of the stochastic block models. However, we felt that this would not be in line with the vision for the project. Our aim is to uncover hierarchical representations within the links. A link between articles is the result of a human who deemed it important enough to place a link between one article and another. Often times, these links cross category boundaries, and that is exactly the kind of structure we would like to uncover with this project. Judging the quality of the stochastic block model results using wikipedia categorizations would result in a biased judgment.

5.2 Evaluation of Latent Node Labeling

After uncovering the underlying hierarchical structure of the wikipedia graph, the next logical step would be to try and name the latent nodes identified by our model. As mentioned in section 3.4 , since each latent node represents a community or communities of communities of Wikipedia articles, we want the assigned labels to describe the core concept common to all of these articles. Unfortunately, choosing the correct label from just the articles names and corresponding link is a very difficult task to achieve. As mentioned above, we've experimented with both top-down (section 3.5) and bottom-up (section 3.6) labeling as well as with a myriad of centrality metrics in the hopes of identifying a promising measure. However, none of the centrality measures used provided with consistently meaningful labeling of the latent nodes. In the figure below we observe a typical case of mislabeled latent nodes:



In the presented figure the community above was named as "Democracy" using the betweenness centrality measure. As we can see, while many of the concepts, such as "Politics", "Government", "Marxism" and "Oligarchy", are related to "Democracy", they are by no means subsets of "Democracy". In this latent node the core concept of the community related more closely to

different forms of governance rather than democracy. Therefore, even though "Democracy" might be central to the discussion of varying government regimes, it is not a general enough concept to encompass the entire meaning of the community. Moreover, the farther away we are from the article granularity level, the less descriptive the names become. For example, one of the most general latent nodes in the largest graph we visualized (available here) is named "Simon and Schuster" after a successful publishing company. This is puzzling when learning that the latent node contains subcommunities pertaining to politics, economics, law and many more. Our model, when faced with collapsing these seemingly disparate subjects together, chose the one article that was most central to all the subjects as the title : a publishing company that presumably has published many books pertaining to subjects in Law, Government, Economics and so forth.

While our attempts have shown that using centrality measures to name the latent nodes is a suboptimal solution, determining an effective way to name the latent nodes still remains an open question. One possible explanation for this, is that the higher up the hierarchy tree we are, the less chance there is of finding a single existing article with a title that encapsulates the common concept of a community. An additional explanation could be that there simply isn't enough information embedded in the graph to establish a meaningful naming system for latent nodes. One possible solution would be to include the actual text of the articles assigned to a node and use natural language processing to determine the proper label.

6 Future Work

There are three major tasks that are the next logical steps from our project. First, stochastic block model should be generated for the entire link graph. This simply requires hardware and time.

Second, as mentioned previously, latent node labeling is a difficult task. We showed that centrality measures are not suitable for this task. We believe that future work could focus on using topic modeling to find the most important or central topics within the content of communities of articles.

7 Team Member Contributions

Maya Rotmensch

- Creating MapReduce programs to clean, parse, filter and join the data
- Creating MapReduce programs to glean key statistics from the data.
- Final Report
- Final Presentation

Peter Li

- Visualization
- Web Demo
- Final Report
- Final Presentation

Justin Mao-Jones

- Modeling (research of methods and tools, block model computation, latent node labeling, output to JSON)
- Graph plotting
- Final report
- Final presentation