TECHNICAL UNIVERSITY OF DENMARK

FINAL PROJECT

02807 Computational Tools for Data Science

Jules BELVEZE, s182291

Renaud JESTER, s1822690

Lasse KRISTENSEN, s123505

December 5, 2018

Contents

1 Introduction

1.1 Project description

Related to the course Computational Tools for Data Science, the final project aims at demonstrating the ability to use relevant computational tools on the *Wikipedia* database. Indeed, the English *Wikipedia* database represents 5 757 658 articles composed of texts, numbers and links. This project will try to use tools from the course and others tools using *Wikipedia* pages to answer a problem.

The idea of our project is to list all *Wikipedia* pages related to a specific one, so that it is possible to give a reader suggestions of related pages, it should not only suggest pages in the same category, but also related articles from outside the category. Not only will this problem be interesting to compute, but this is also something that can be very useful for users (as shown in the picture below).

Figure 1: Photo editing of how our program can be used by the *Wikipedia*'s visitors. The red square represents the suggestions we could obtain thanks to our program on the page *England*

.

1.2 Problem description and choice of tools

The first goal is to end up with a graph based on the links between pages. From the first page, it is possible to get all the other pages mentioned inside. Then, the process iteratively goes through the found pages. The graph will be used to pinpoint the relations between each pages. The idea is now to give suggestions to the user. For this purpose, *PageRank* algorithm can be used since it computes the "popularity" of a page.

However, if only *PageRank* is used, the suggestions might be vague *Wikipedia* articles not directly related to the input, for instance "America" is a link found in many articles, however it might not be relevant to the input article. To get round this issue, we will also use clustering algorithms to gather interlinked pages.

To present our work we will describe linearly the tools we have used and the way we have implemented them. First, we will explain how we made the preprocessing of the data and created a graph out of it. Then, we will see how we selected the suggested pages by using *ForceAtlas2*, *PageRank* and clustering algorithms. We will eventually show the results of the program using an example.

2 Preprocessing and creation of a graph

As we will need to study the links between two pages, we have decided to use a network. This will be created with *Wikipedia* pages as nodes and edges will highlight whether a page mentions another (as an example see Figure 2). This section of the report will mainly focus on the definition of the network and how to construct it.

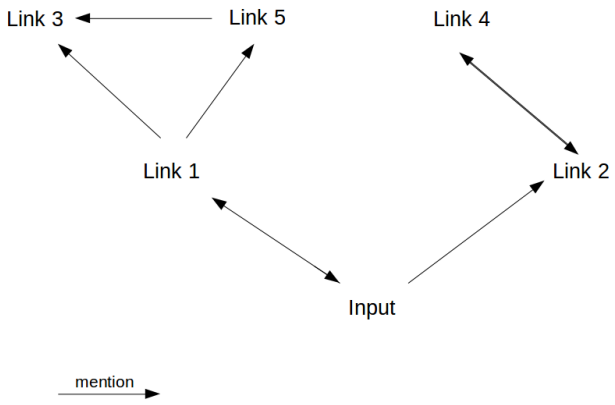

Figure 2: Example of network we could get from a *Wikipedia* page

2.1 Graph generation

Two possibilities are practicable: whether we download all the *Wikipedia* database or we can work with its API. We choose the last option since the input will naturally lead to other links and the purpose of the project is not to go through all the pages. The full database takes up 62GB of space which will prevent a many potential users from using it.

In order to get all the pages quoted by a given one we request the *Wikipedia* API to get the content of that article as a *JSON* file. Then, since all articles are surrounded by two pairs of square brackets, we use a regular expression to capture them.
In order not to go through already visited pages, and thus to save time, we have used a hashing table that will keep a record of all seen pages. Another important parameter to define is the stopping criterion, so that we will not browse the whole data set. Actually, it is a bit more difficult to define because our final network should be big enough to be relevant and not oversized because the pages found will be off subject.
The depth of the graph is then: $\begin{cases} 0 & if\ it\ is\ input\ node \\ i & if\ it\ is\ mentionned\ by\ a\ node\ at\ depth\ i-1 \end{cases}$

Concerning the graph, we have used the *Networkx* library which was relevant for our use because we can distinguish the link between the different pages and visualize the graph. The network is a directed graph were: a node is a *Wikipedia* page and an edge from A to B exists if A mentions the article B.

We found that a depth of 2 is particularly appropriate since it gathers at least 10000 links, a depth of 1 can give satisfactory results aswell, if a lower computation time is wanted. Also, for our network, we will need to go through all the links mentioned in the pages of the last depth without adding new nodes. It is important to have all the relation between the pages. The input might be mentioned by an article of the last layer.

2.2 Implementation

To implement what we have described in the previous subsection, we have created a *FIFO (First In, First Out)* [4] class which is basically a list where the "oldest" element is the first to be treated, like a queue. This *FIFO* will contain all the links and makes sure that the program go through all of them in the order of the depth of the graph.

Then, as presented in the previous subsection, we created a hashing table class to know if we have already visited a link or not. This is a bit vector where bit $j = 0$ if we have not met any page where the hashed url is j, 1 otherwise.

Finally, we have created a depth controller class to keep track of where we are during the process and to tell the program to stop when the stopping criterion is reached.

3 Clustering and Page Rank

In order to get relevant suggestions of pages for the user we have decided to use two tools: clustering and the *PageRank* algorithm. With those tools we will be able to display relevant suggestions to the user. We do not use the existing Wikipedia categories as these are too specific within its topic, we want our results to suggest articles across different topics

To cluster, we first generate a force-directed graph using the *ForceAtlas2* algorithm which is a force-directed layout for directed graph. Then, we use the algorithm *K-means* as described in the course using the euclidean distance. As presented in figure 3, some nodes are very close to one another. The difficulty is then to tune the parameters. It could lead to very different results.

3.1 ForceAtlas2 algorithm

As mentioned it is possible to translate or represent a graphs into euclidean space which can be used as a base for clustering. There are different algorithms that can perform this translation. We chose the *ForceAtlas2*[1] algorithm for a directed graph because it can deal with a big number of nodes.
The idea behind this algorithm is to create a physical system in order to represent spatially the nodes. It will lead to a layout where nodes will repulse each others while edges will attract their nodes. Thus, nodes within masses will be close from one another, whereas isolated nodes will be far from the main core. The main drawback using that algorithm is that it varies, depending on the initial state.
The algorithm is available from the ForceAtlas2 library in Python, which we used.

3.2 Clustering algorithms

The idea is to find categories independent from the existing *Wikipedia* categories in our nodes so that the suggestions are leading to different subjects related to the input in some way. A cluster should reveal at what degree are the pages in the network linked to each other. That is why the first idea was to use the geodesic distance of a graph: the distance between two vertices in a graph is the number of edges in a shortest path [3]. However, this approach is limited since there is only a finite number of distances. In our case, the maximal distance would be five which is good to distinguish the noise but very difficult to cluster the core of the network. That is why we use the *ForceAtlas2*.

Initially we used the *DBSCAN* algorithm on the euclidean coordinates of the nodes, but found that the *K-means* algorithm gave a better result. The *DBSCAN* returned a large cluster and some smaller cluster along with a lot of noise (Figure 3) depending on the parameters, which resulted in too many pages was considered and thereby suggestions will either be to similar, suggestions from the large cluster, or too irrelevant, the small clusters.

The *K-means* algorithm on the other hand, gave some smaller clusters more equal in size (Figure 3), meaning that the considered articles would be more diverse in topic, resulting in better results. By using the *K-means* algorithm instead of the *DBSCAN* algorithm, we also avoids the issue of tuning the input parameter for the *DBSCAN* algorithm to the current data.

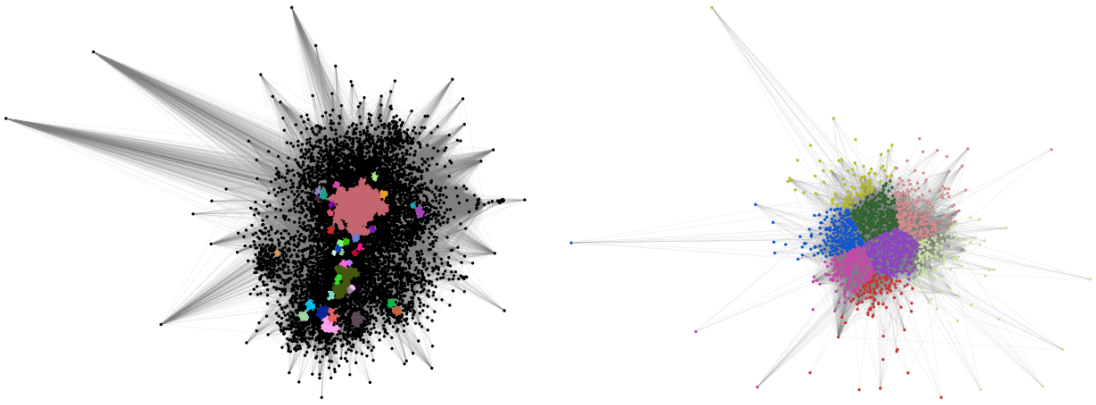

Figure 3: Clustering of a force-directed network using DBSCAN (left) and K-means (right). The input is `https://en.wikipedia.org/wiki/Asbel_Kiprop` and the program has stopped at depth two. Parameters of DBSCAN: $\epsilon = 40$ and $minPts = 20$

3.3 PageRank algorithm

We have chosen to use *Google's PageRank* algorithm because it roughly estimate how "popular" a page is. Here is a quick overview of this algorithm.
We first build a square matrix A of size $n * n$, where n is the number of nodes, so that $a_{i,j} = p$ if there is a link from page j to page i, where p is the probability to go from page j to page i. Then we compute the product of A and $\begin{bmatrix} 1/n \\ ... \\ 1/n \end{bmatrix}_{n,1}$ to end up with a vector reflecting the importance of each node. In fact, the greater the i^{th} value of that product is, the more popular this page is [2].

The final step is to return the pages with the highest page rank within clusters. By doing so we will get relevant pages because they are mentioned inside a lot of articles but in different categories (we can guess that pages inside a cluster might be about the same topic).

As described the idea behind the *PageRank* algorithm is not really hard to understand. To implement it we have made the choice to create a class called *PageRank* allowing us to compute the vector page rank giving a graph. The idea is to construct a adjacency matrix with $a_{i,j} = 1$ if there is an edge from node j to node i and then to divide all elements by the sum of the row.

4 Example

Here is an example of what result our project can lead to. We have used the *Wikipedia* page of "Data Science" as input. In order to compare the suitability of our results we have run in depth one and two using the *K-Means* algorithm for clustering.
The graphs obtained can be seen in figure 4. It took 50s to construct the depth one graph and proceed to all the computations whereas it took more than one hour for the depth two. This is one aspect to improve because no user is going to wait that time to obtain suggestions.

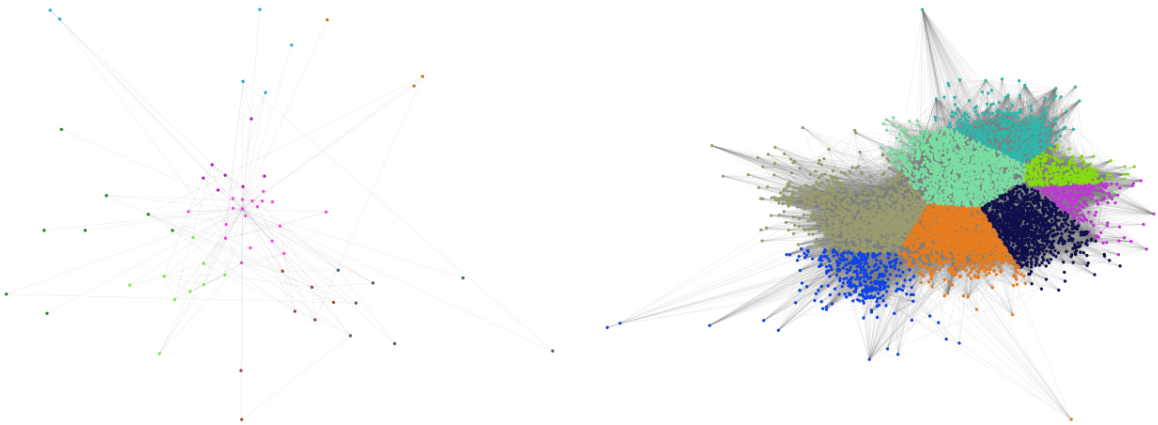

Figure 4: Graphs for the following input `https://en.wikipedia.org/wiki/Data_science`, left one is using depth one and the right one is using depth two.

As we can see in Figure 4 we may need more nodes to end up with a relevant result by running the program in depth one. On the right side we can clearly distinguish the different clusters. We are now going to analyze if the suggestions are really different according to the depth. The figure 5 displays us the suggested pages and their related *PageRank*. In both outputs the results are somehow related to "Data Science". If we focus on the outputs of depth two, it surprisingly comes out that the page with the highest *PageRank* is "New York University" and not "mathematics" as we could have been expected.

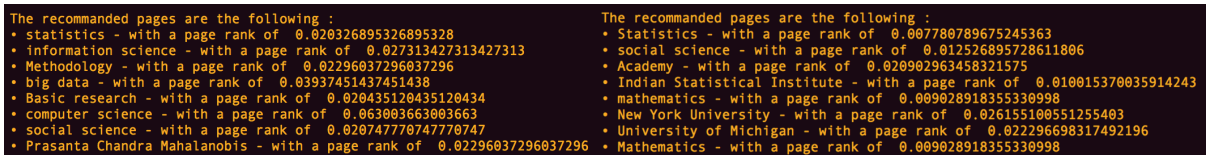

Figure 5: Outputs `https://en.wikipedia.org/wiki/Data_science` as input, using *K-Mean* algorithm to cluster. The left one is using the program in depth one and the right one depth two.

5 Conclusion and Future works

In this project we have developed a program which takes a article as input and display a list of suggested pages. To do so, we have first gone through the mentioned pages in order to identify the links between pages. Then we have partitioned those pages into clusters, using one algorithm (either *DBSCAN* or *K-Mean*). Finally we have get the most quoted pages using the *PageRank* algorithm.

However, the program has a relatively large computation time, this might restrain users from using it on a regular basis. The program uses the *Wikipedia* API and is collecting data from the internet. The running time is then dependant on the quality of the internet connection. Moreover, due to a relatively large amount of pages the drawing of the graph is also time-consuming.

Bibliography

[1] MATHIEU JACOMY, SEBASTIEN HEYMANN, T. V., AND BASTIAN, M. Forceatlas2, a continuous graph layout algorithm for handy network visualization.

[2] STANFORD. The pagerank computation. https://nlp.stanford.edu/IR-book/html/htmledition/the-pagerank-computation-1.html, 2008.

[3] WIKIPEDIA. Distance (graph theory). https://en.wikipedia.org/wiki/Distance(graphtheory), 2018.

[4] WIKIPEDIA. Fifo (computing and electronics). https://en.wikipedia.org/wiki/FIFO_(computing_and_electronics), 2018.