A Case Study on Working With Wikipedia Data and an Application of the Wikipedia Game

Austin Schmidt   
University of New Orleans  
Department of Computer ScienceLouisiana, United States  
sbaustin@uno.edu

Abstract – In this paper, we discuss the key ideas of working with Wikipedia’s public data dumps. The public data is diverse and can be found as raw XML data or in SQL tables. In some detail, we examine both types of data in this paper – including downloading and parsing. Using this data, we aim to play the “Wikipedia Game” which can be represented as a shortest path graph problem from a starting article to a specified ending article. The naïve breadth first search approach of finding shortest paths does not work at this scale, so a bidirectional search is used instead. We find speedy results using this method and consider what this tells us about our data. In conclusion, we will have shown different ways to work with and consider Wikipedia as a graph problem thus gaining a better understanding of the data as a whole.

Keywords—Wikipedia Game, Breadth First Search, Jupyter Notebook

# Introduction

Wikipedia is one of the world’s largest open source repositories of information. Due to Wikipedia being entirely community driven, it is a natural step that they provide massive data dumps to their users. These data dumps provide two purposes. First, they work as snapshots of the Wikipedia data to be archived by users across the world. Second, the data is available for anyone who might want to run some analysis or gather data for some specific work. This data includes raw XML and SQL tables. In this paper, we describe some of the different ways that the information can be gathered and worked with.

There is an inherent challenge to working with such a large dataset, so special attention must be given to all aspects of the process, which will be described in some detail. We include how to efficiently find and download the data dumps, how to process the raw XML and SQL tables, and how to determine which data source would be most useful for a given project. We then go on to describe a special application of our data to the “Wikipedia Game” which is popular among school children. We give a more detailed formulation of this game in section II, but the basic idea is to travel from one Wiki article to another using only the built-in hyperlinks that connect every Wiki article.

Games are one of the greatest gateways into understanding a problem or dataset. Many games contain one or two simple rules that need to be considered along with a pre-defined goal. In this way, we can begin working with a small subset of techniques and information to solve our small game. Later, the experience gained from these simple problems can be used and expanded into overarching theories that may be more impactful. We aim to provide a strong jumping point into working with Wikipedia’s data.

This work was completed using Jupyter Notebook files and Python as an education resource for anyone interesting in working with a very large dataset. Jupyter’s cell-based programming style allows the reader to more closely follow along with the information being presented as only one block of code is run at a time. The intent is to provide enough context for the reader that they may avoid common pitfalls and make more interesting progress more quickly. In this way we make our contribution.

# The Wikipedia Game

The Wikipedia Game is very simple and has very few rules. First, out of all the articles contained in Wikipedia, two are selected randomly. One article A is designated as the start position and the second article C is labeled as the end position. We aim to traverse along Wikipedia’s articles from A to C using only the outgoing hyperlinks for each article. An example solution path could be [A, B, C] where B is an outgoing link of A and C is an outgoing link from B. In this way we show there is a directed link from A to C. If playing with another player, the goal is to find the shortest possible path in the smallest amount of time.

This game is based of the theory of 6-degrees of separation which states that every person on the planet is 6 or fewer social connections (or handshakes) away from any person. This property has shown to hold true in other social networks and graphs. The hypothesis is that the same property will hold true on average for a graph generated by the collection of articles on Wikipedia.

# Data Collection

In this section we will be considering two different datasets provided by Wikipedia – XML and SQL. We will describe some techniques used to download the data as well as possible applications for each dataset. As we will see, both have their own positives and negatives.

## XML Dataset

The XML data that Wikipedia collects is very rich and varied. To gain all the data there is to offer, it is possible to download large 60-80 gigabyte sized bzip2 zipped files where bzip2 is an even more efficient compression technique designed for massive datasets. This large datafile contains every article on Wikipedia including article titles, article text, templates, media descriptions, and meta-pages. For our purposes, we avoid the large one file dump because Wikipedia also offers smaller partitions of their data – around 50 partitions instead of one large file. Individual partitions are useful because they provide a natural checkpoint state. Should our download fail, for example, we would only need to continue from the last properly downloaded partition instead of the entire dataset. This is also helpful when parsing through the dataset later.

Dealing with 50 partition comes with its own problems. It is cumbersome to download 1-5 gigabyte files one at a time by hand. Instead of downloading each file, we need to programmatically download all the partitions. For this purpose, we can use the BeautifulSoup Python library. Using this tool, it is possible to download the HTML page that includes all the downloadable links to the data and query for each link by their HTML tags. Once these links have been generated, it is simple to download all items. More details, examples, and resources on how to complete this task are outlined further in the provided Jypter Notebook files. The data generated for this portion is the public data dump from Wikipedia on December 20th, 2019.

Before downloading these large files, it is also appropriate to discuss that there are two main types of XML datasets. There are the pages and articles without any organization in each partition which are labeled pages-articles. An example would look like:

wikidatawiki-20200120-pages-articles.xml-p1p235321.bz2

The second type of XML file is something called multistream. We did not use multistream in any of our examples, but it is an organizational technique that has to do with the bzip2 compression. Bzip2 supports a parallel version which is tagged with multistream. This is a powerful property but can make the initial leap into working with Wikipedia data more difficult. Multistream files use a separate index file which needs to be downloaded as well. An example would look like:

wikidatawiki-20200120-pages-articles-multistream.xml.bz2

wikidatawiki-20200120-pages-articles-multistream-index.txt.bz2

The reason this distinction is given is because the two different files can be easily confused. A lot of time can be wasted downloading improper files so, special care should be given to make sure any downloaded files are needed.

After the proper partitions have been downloaded, they may be examined. The problem is that even though we have individual partitions, each file is still too large to easily open in main memory. Instead, we develop techniques for parsing through a file without ever unzipping the large files. This is a nice attribute of using the bzip2 compression format. There are a few main things we can look for while parsing a partition such as article titles, wiki links, categories, and infoboxes.

|  |  |
| --- | --- |
| **Article Titles** | The title of some specific article. |
| **Wiki Links** | Hyperlinks that link the current article to some other related article. |
| **Categories** | User created descriptive tags. There can be many of these tags and they can be either broad terms or very specific. |
| **Infoboxes** | Template boxes that are included for many types of articles. For example, every article describing an animal has an associated template for an infobox. Every animal would have the same infobox template. |

Fig. 1 – Examples of what to look for when parsing.

Let us focus on what this large amount of information could be used for. While we can find all out links for each article in this way, it becomes obvious that this technique will not work for our Wikipedia Game after an example application. Consider parsing through just one partition with the goal of finding every book included based on the book infobox. To find the 982 books contained in just one partition took almost 30 minutes. An obvious improvement would be to run multiple searches on a large distributed system. In this way, large datasets can be compiled based on some criteria.

To wrap up, one of the most useful things that can be done with the raw XML data is the compilation of some large dataset. If some specific data is needed, such as a collection of animals or books, that is not readily available elsewhere, it can be generated from Wikipedia. This process is nice because there is a variety of ways that information can be searched for in this way. The data lends itself very nicely to finding communities from categories or data aggregation from the infoboxes. The downside is that the data is very difficult and time consuming to work with in this raw form.

## SQL Dataset

Wikipedia offers SQL tables that are more specific than the previous XML data. For example, one SQL table is a list of all redirects from one page to another. A redirect is defined as a query, say “UK”, that brings the user to another page, i.e. “The United Kingdom”. These tables hold less information, but it is more organized and easier to parse through or query. For the purposes of playing the Wikipedia game, a publicly available script was run using Google’s Compute Engine technology to create three useful tables, which takes about eight hours. We lose a lot of context in article text and category information, but we gain an efficient way to represent Wikipedia as a graph. The data generated is from the public data dump from Wikipedia on January 20th, 2020.

The tables generated are for the purpose of playing the Wikipedia game properly and are labeled Pages, Query Links, and Redirects.

1. Pages

The pages table has the columns id, title, and is redirect. We can query based off only the title of an article. However, searching the database by a string takes significantly longer than searching by the id. The difference is between three full seconds and half a second. Should we need to load the whole table into memory, it takes about 40 seconds. In total, there are 14,973,896 articles included in our table.

1. Query Links

The query links table handles the incoming and outgoing link relationships we build our graph upon. The table contains the columns id, outgoing links count, incoming links count, outgoing links, and incoming links. Loading the entire table into memory takes a full 35 minutes to achieve although the total number of entries is less than half of the Pages table. This has to do with the fact that any given article is connected to an average of 80 other articles. These connections are represented in outgoing links and incoming links as long text strings with separators between ids. The max number of outgoing links is 10,000 belonging to the “Index of Singapore-related articles” means there are some articles that link directly to many articles like a hub. Other articles draw in a lot of importance from other articles by having a large amount of incoming links. The max found contained 1,116,378 links and belongs to the article “International Standard Book Number”. In our data, there does exist articles which have no outgoing links or no incoming links that can cause impossible solutions for pathfinding. The population of articles with no incoming links is small with 313,101 members (about 6% of our non-redirect articles) and the population with no outgoing links is only 932. There does not exist an article that is completely disconnected from the graph i.e. no incoming links and no outgoing links. That is not to say there are no disconnected communities – more analysis would be required to find out.

1. Redirects

Redirects table holds only two columns, the source id and target id. This table takes only 24 seconds to load into memory and has a size of 8,975,917. Interestingly, almost nine million of our articles are links to a different article entirely.

There is much insight to be gained just by examining the links between articles – even without any article context. Once the dataset has been understood, the problem of finding shortest paths between two articles becomes clearer. Once again, a Jupyter Notebook file has been generated to easily provide these metrics as a starting point to continue investigation.

# Application of Breadth First Search

Here we consider two techniques for finding shortest paths on graphs. The first, Naïve Breadth First Search (BFS), is a very poor performer but is great benchmark to help illustrate how important efficiency is when dealing with larger datasets. The second algorithm we use is a more efficient BFS algorithm called bidirectional BFS. We go into more detail, but this is a much more efficient approach that will allow us to examine the connectivity of sub-partitions of the graph. Both algorithms produce a list of shortest paths from a start goal to some end goal. We can turn individual lists into graphs as referenced in Fig. 2, or we can collect many of these lists and compute some metrics on efficiency and how connected our graph is.

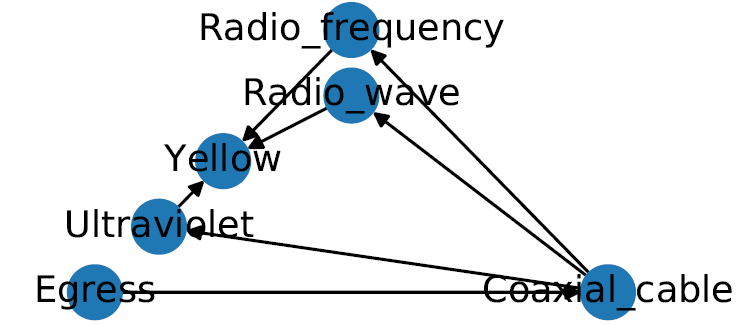


Fig. 2 – Shortest paths going from ‘Egress’ to ‘Yellow’ 3-degrees of separation

An example output to generate the above Fig. 2 would look like: [[11271498, 46380, 31990, 34368], [11271498, 46380, 42852, 34368], [11271498, 46380, 98132, 34368]]. Each sub-list is used, and the ids are transformed into the title for readability. Graphs are saved as PDF files to easily zoom in or out of a graph with many paths found.

## Naïve Breadth First Search

First, we consider our baseline Naïve BFS. This implementation was designed without stopping after finding the first path. We let the algorithm run k times where k is the number of times we pop a new neighbor from our queue. The choice for this comes from a potential extension of the problem – longest paths. Wikipedia contains self-connecting paths which means paths of infinite length can be generated if explored articles are not considered. As discussed in section II, the number of average hyperlinks between two articles is around 80 and some get much larger than the average. This leaves us with a branching factor around O(bd) where b is the branching factor and d is the distance from our start to the end. Due to this, the explorable path space explodes rapidly in such a way that no path is feasibly found after only 2-degrees of separation and even that feat takes a few minutes to run. The efficiency of the Naïve algorithm must be improved, or this task will be impossible. One way for improvement is the Bidirectional BFS discussed next.

## Bidirectional Breadth First Search

To understand how bidirectional BFS (BBFS) corrects efficiency problems, it will be quickly defined. BBFS works by conducting two BFS searched simultaneously. While one search begins at the ‘Start’ and travels forwards through outward links, the other search begins at the ‘End’ and travels backwards through the incoming links of each article. At each explore step, new neighbors are added to the queue from the search that currently has fewer neighbors. The reasoning is that the search space will be lowered by always searching the direction with the smallest space. As the search continues, both directions check to see if they have any same articles in their personal search space. At this point, a clear line from the start to end can is found. Once that has been determined, the backward path can be reversed and then concatenated with the forward path for our answer.

This method is much more efficient because we are doing two searches instead of one. Total complexity to run is O(bd/2+bd/2) and that simplifies to only O(bd/2) which is much better than the naïve approach.

Although this approach is very efficient and will allow for good analysis of the connectivity of the graph, there is a problem to be addressed. The Wikipedia game often has the stipulation that a player cannot go ‘back’ or return to an article after getting into a bad place. Our BBFS has the full context from the start and end positions, so the spirit of the game is ruined slightly.

Due to time constraints, instead of creating an implementation of BBFS, a version by Jacob Wenger has been used to get good results. Two collections of results from collections of articles have been established. The first is a list of random 350 articles and every permutation of the data was explored. For 350 articles we get around 100,000 different queries to make and this takes around 13 hours to complete. The permutation is taken to get the full context of every starting and ending path from the 350 randomly chosen articles. The other collection used contains 1000 different words, but every word is only used as a beginning once and as an end once. This allows for more articles to be used where 1000 articles = 1000 query pairs, but less context is gained overall.

# Results and Evaluation

As discussed in Section IV, we completed and compiled data from two query lists. The results are saved as .csv documents so that they can be analyzed thoroughly. Specifically, the combination of ids, start title, end title, paths found, shortest path length, and the time taken are saved for analysis.

1. Permutation Dataset

We examine some of the metrics generated from the first dataset with 100,000 query results. There is a reason that permutations of the data were chosen over combinations. The path from A to B is not necessarily the same length as the path from B to A. Due to this property permutations are examined as order matters. The mean distance of shortest paths is 4.95 with a standard deviation of about 1.45. Fig. 3 below displays a histogram of the path lengths for each permutation. Notice there is a large amount of weight towards paths of size 0. In this case, a path size of 0 means that no path was able to be determined – we discussed why this may be the case in section III part B. Otherwise, the extremes lie at a minimum of 3 and a maximum of 9. There are only 74 articles that have the maximum 9-degree value. The distribution is Gaussian in shape.

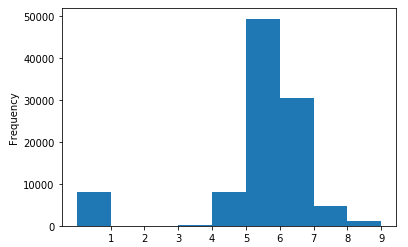


Fig 3 – Histogram for permutation dataset

The average time taken for any given query is 0.46 seconds and the max time generated is 17.88 seconds. There is no interesting reason for such a long compute time, and this is likely due to a hardware issue. Otherwise, the range of time is between 0 and 7 seconds with a high bias towards 0 seconds. The total time taken to generate the shortest paths for each permutation of 350 random articles was approximately 13 hours.

For any given pair of start and end articles, they can produce any number of shortest paths. The mean number of paths found is 121.28 although the max number is 10984. There are exactly 7976 occurrences of no path being which implies that the randomly chosen 350 articles are very nicely connected.

While the results from the permutation dataset is very insightful, there is only context gained for 350 words. In order to see if similar metrics are found in other subsets of the Wikipedia dataset, the random combination dataset is generated to poll larger numbers of articles in less time.

1. Random Combination Dataset

For the random combination dataset, 1000 words were randomly chosen and put into a list. The first item is the start position of a pair and its neighbor is the end, the end article is then given a pair where it is the start and its neighbor is the new end. In this way, 1000 random items generate 1000 pairs that are very disconnected. In this way, more articles can be added into the analysis without adding huge computation times, although context is lost.

In this case the mean length of the shortest paths is 5.126. The max of length 9 is found only once, but there are 74 paths with a length of 0. Fig. 4 below displays the histogram for the spread of shortest paths. The number of paths found has a mean value of 129 and the max number is 2915. There are very similar results to the previous dataset.

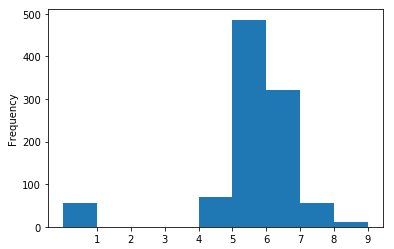


Fig. 4 – Histogram for random combination dataset

The average time taken to complete finding the paths is 0.6 seconds with the longest time being 18.94 seconds. Again, there is not an explainable reason for this long time. The total time taken to generate this data is 10 minutes. The scalability of this approach allows for a much broader net of available articles.

##### VI. Conclusion & Future Work

In this paper, techniques have been discussed to download and work with both SQL and XML data for the publicly available Wikipedia data dumps. Jupyter Notebook files have been created as examples to help others begin academically working with Wikipedia. To get a better idea of how the data is connected, an in-depth dive into the Wikipedia Game is conducted. Two versions of a BFS algorithm are considered for the purpose of analysis on many combinations of queries. In both collected subsets of articles for pathfinding, similar results are found.

For future work, there is a wide collection of possibilities with this dataset. Any project that might need a large aggregated dataset will benefit from the techniques described here. Specifically, with the Wikipedia game, it would be interesting to take this a step further and design an AI system to traverse based on some heuristic value so it more appropriately represents how a human player would look for the best path. There are many types of interesting graph problems that could work on this dataset. If the SQL data was expanded slightly, a community clustering analysis of Wikipedia would be a great project to continue with. Alternatively, more tutorial style academic journals for students new in big data research might be appreciated. Further research can even be done by taking larger and more subsets for the Wikipedia game problem as it currently stands.