Wikipedia Degrees of Separation

Andre Duarte, Nick Levitt, Shivakanth Thudi, Juan Pablo Oberhauser

October 10, 2016

Contents

Project Description

The goal of this project is to create a measure a relationship in terms of degrees of separation between certain terms in Wikipedia. The degrees of separation between wikipedia search terms would be calculated

by counting the number of links that would need to be clicked, starting with a link contained in the first article, until the second search term is reached. This calculation would be a proxy for "how closely related" those two search terms are. There are two main options that the user can select. The first is to build a path from one wikipedia article to another, starting with a selected term. The second is to find a path between two terms that the user selects. Each part of the project presents its own algorithmic, data storage, and data retrieval challenges that will be further discussed. The first option is called **"Build a Path"** and the second is called **"Find a Path"**.

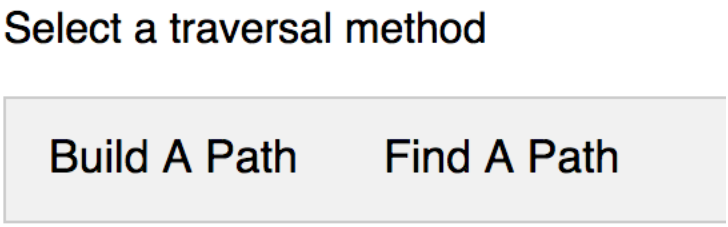

Figure 1: Landing Page with 2 traversal method options

Executive Summary

Data Acquisition

Overview of Available Options

For this project we require Wikipedia data. More specifically, we require all of the wikipedia search terms or article titles, and a list of all of the link contained in each article. The wikipedia English language dump is 15 GB compressed, so we have to be creative in our solution to acquire either a subset of articles or a different methodology to get this data. We propose several solutions, their tests, and drawbacks.

1. The English language static **html dump:** "A copy of all pages from all Wikipedia wikis, in HTML form." (14.3 GB as a .7z file)
2. **Pywikibot:** Python library that provides functionality to automate work on Wikipedia sites.

3. **SQL interconnectedness files**, provided by Wikipedia. Files contain metadata on which articles contain which links.

4. **CURL the urls** for the article that is chosen on the fly, get the list of links from a parsed HTML, and provide a choice of new articles to explore and build a navigation path.

5. Build a fully connected tree of from a **subset of articles**, limiting search options to terms included in the subset, and navigate the tree of parents(terms) and children(links in the term article) to find the shortest path.

Drawbacks and Advantages of Acquisition Options

- The English language static **html dump:** "A copy of all pages from all Wikipedia wikis, in HTML form." (14.3 GB as a .7z file)

This option had to be quickly discarded because even though the dump was successfully stored in our local machines, it would have been impossible to host on pythonanywhere.com. pythonanywhere.com limits the amount of storage on disk that is available to the user as well as the 'CPU seconds'. 'CPU seconds' would not necessarily be the limiting factor for our project since these are only used up when the CPU is actually busy. Our limiting factor was more on the storage limitations.

- **Pywikibot:** Python library that provides functionality to automate work on Wikipedia sites.

Using a python library that provides functionality to work with wikipedia pages seemed very reasonable since it seems like many people around the web are using it to update article content and give maintenance to content.

Pywikibot allows about 5,000 article downloads without being stopped. Even though this seems like a viable solution, we need to make sure that the subset of articles is very interconnected so that there can be a shortest path solution between terms. Making sure that the subset was that interconnected proved to be a difficult task and this option had to be foregone.

- **SQL interconnectedness files**, provided by Wikipedia. Files contain metadata on which articles contain which links.

This ides proved doable with a subset of data from the Philippines. We were able to download the files and search through the content in the file. We used a shortest path algorithm and were able to always get the

shortest path between two terms. When trying to switch to the full English interconnectedness files, storage again became a limiting factor. The English language full dump is 40GB, so this again proved impossible.

- **CURL the urls** for the article that is chosen on the fly, get the list of links from a parsed HTML, and provide a choice of new articles to explore and build a navigation path.

This was the viable solution for the data part of the 'Build a Path' part of the project.

- Build a fully connected tree of from a **subset of articles**, limiting search options to terms included in the subset, and navigate the tree of parents(terms) and children(links in the term article) to find the shortest path.

The two methods that are used in the final implementation of this project are CURLing articles on the fly for the 'Build a Path' part, and getting a subset of 400 fully-connected articles to build a traversable tree for the 'Find a Path' part of the project.

CURL Solution for 'Build a Path'

For the 'Build a Path' section, we use urllib2 functions, specifically urlopen(). This allows us to open a **network object** for reading. To get the URL, we get the user-input term and make a URL to send a request to. To ensure functionality, we use error handling in the URLs in the following manner.

```
def getLinksFromSearchString(searchString):
  searchURL = template_wikiURL % searchString
  URL = mainWikiURL % searchURL
  try:
      return getLinksFromURL(URL)
  except:
      return ["%s is not a valid search term" % searchString]
```

Solution for List of Links (1st step for acquiring data)

Given the difficulty of dealing with a data dump and the difficulty of using the python bot to extract information, we will be implementing a script that does a request to a given URL and brings back the html, to be parsed using BeautifulSoup.

After the retrieval of the text, we look at all of the pieces of html containing a “class: mw-content-ltr”

For example:

```
'"<div class="mw-content-ltr" dir="ltr" id="mw-content-text" lang="en">
<div class="hatnote" role="note">
This article is about artificial satellites. For natural satellites, also known as moons, see <a
Natural satellite</a>"'
```

Once we obtain a link object, we look for the ‘a’ anchor to get the actual part of the text which contains the ‘href’ , we store that link and its name in a python list to be utilized when the html is rendered. At this stage of the search for a list of links in the article body, it is necessary to check for a valid link. To do so, we use string properties to check whether the URL string starts with ‘wiki/’ and check for invalid characters.

Linked List

As a first solution to store the data of which links are in the article of the selected search term, we store article names and links in a linked list. We propose the linked list solution since it provides with a way to begin with a node (which is the first search term as input from the user) and a pointer, which will reference the next node on the list. The next node on the list will then contain the name of the article that was clicked from that first list, and so on.

This data structure provides enough flexibility in that insertion at the final index is fast. Given that no additions are expected to be made to an index in the middle of the list (given its chronological nature), there is no need to insert or update pointers within the list. Another useful characteristic of the linked list is that to perform a search, the program checks at each node starting from the first node and returns the first instance of where a match is found.

This linked list still provides challenges in the sense that in order to visualize the user’s steps through the links, there needs to be a method to check for previously clicked-on links.

Solution for Degrees of Separation and ‘Find a Path’

As opposed to the ‘Build a Path’ section of the project, the ‘Find a Path’ needs to find the shortest path between two given terms that are user-input. This provides a challenging data acquisition project since to

find a path from Article A to Article B, we need to ensure there is in a fact a way to get there using only links in each article.

The proposed solution for this problem is the following:

1. Get a subset of 400 articles (Parent)
2. For each article, get 5 links contained in the article (Children), making sure that each Child links back to the Parent.
3. Create traversable tree with 400 Parent nodes, each of which has 5 children that link back to the parent.

The limit of 400 articles makes sure that we have a very interconnected set of articles. Unlike the 'Build a Path' part which gets data on the fly, the 'Find a Path' section has all of the data pre-stored on disk and a tree of nodes that will traverse and find the nearest path between two terms each time the user inputs two terms. The tree is stored in a txt file.

Data types for Storing hierarchical data

To be able to find relationships between the terms that the user chooses (i.e find the shortest path between two terms), we need a very specific data structure. To iterate a tree-like data structure, python provides list functionality.

A simple example is:

```
def walk(node):
    """ iterate tree in pre-order depth-first search order """
    yield node
    for child in node.children:
        for n in walk(child):
            yield n
```

Figure 2: Tree-like iteration with lists

Journey through shortest path

As a visual bonus for the user, we have included a visual journey through the shortest path between two articles. Harnessing Selenium, we can create a visual tour that mimics that shortest path and actually opens a browser window showing each click that it would take to get from Article A to Article B.

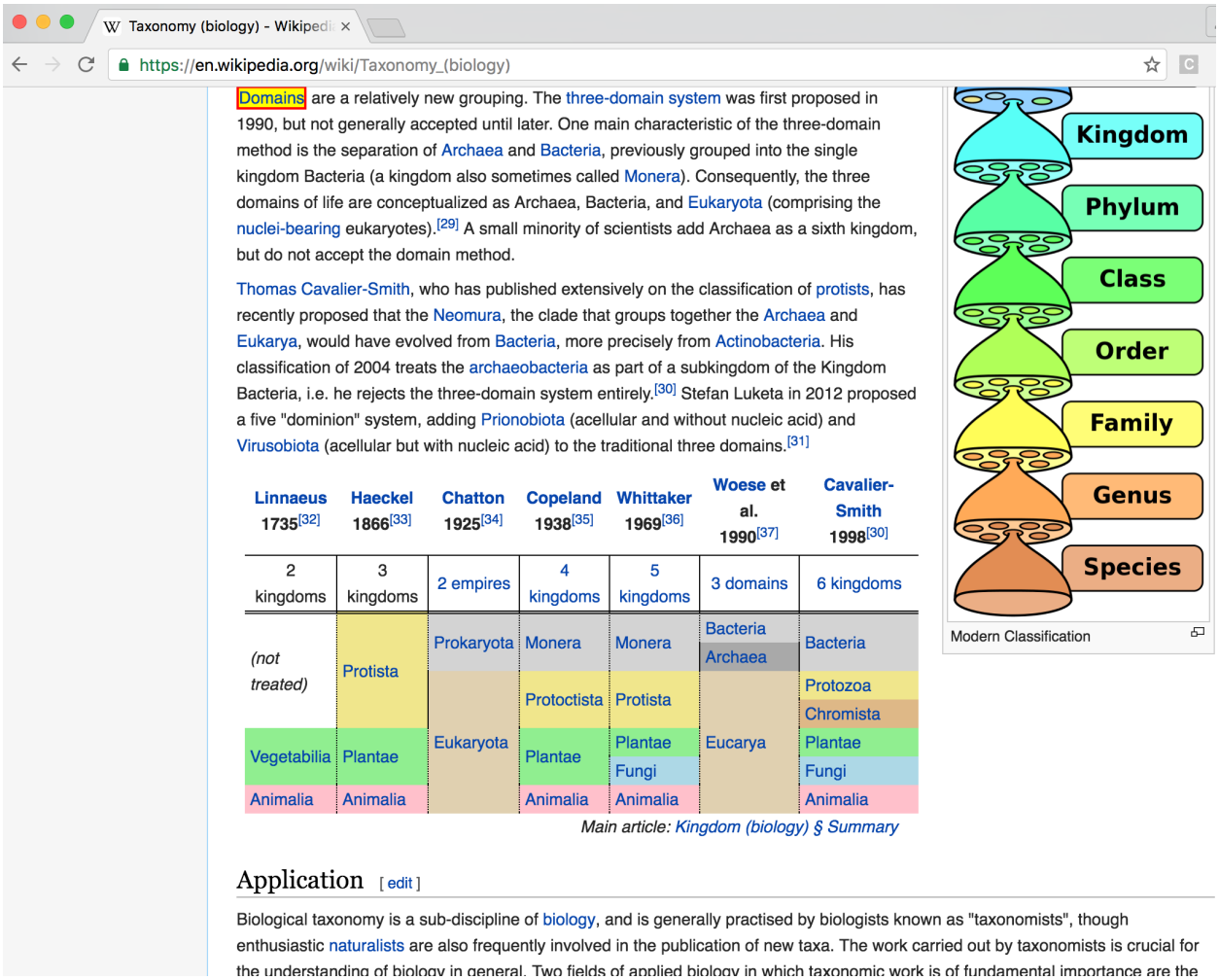

Figure 3: Journey Step Example

In the example above, the journey has taken us to the 'Taxonomy_biology' page and is now highlighting what the next step will be, which in this case is 'Domains'. this specific examples is a step in the journey to get to the 'Number 1' wikipedia page from the 'Almhult Municipality page'.

Since the final project is being hosted on pythonanywhere.com, we need error handling around this 'Take the Journey' part because it would not be possible to open a driver with Selenium on this platform for each user. For this reason, we disable the take the journey part, and instead print a visual representation of what the shortest path would look like.

Other tools used

Flask, Selenium, and BeautifulSoup

To get the list of links, we use BeautifulSoup to parse the html returned from our request to Wikipedia.

To provide the "journey" functionality, Selenium python bindings were utilized. These bindings provide API access to web-drivers or web browsers.

Flask is used to provide all of the functional bindings to turn functions into html. What Flask does is map a url to a python function. This allows us to set up event methods that are mapped to a particular URL.

Bibliography

http://stackoverflow.com/questions/3009935/looking-for-a-good-python-tree-data-structure https://docs.python.org/2/library/urllib.html https://help.pythonanywhere.com/pages/WhatAreCPUSeconds https://docs.python.org/3/tutorial/datastructures.html