Oxford Internet Institute, University of Oxford

Assignment Cover Sheet

Candidate Number	1037074
Assignment	Data Analytics at Scale
Term	Michaelmas Term 2018
Title/Question	Modeling Human Web Navigation by Calculating Shortest Paths in the Wikipedia Graph
Word Count	4,608

Modeling Human Web Navigation by Calculating Shortest Paths in the Wikipedia Graph

1037074

1 Introduction

Frequent users of the web know that a single search engine query is often insufficient for finding the information they need. Instead, they often find this information through a combination of search queries and navigation between hyperlinked webpages, either out of necessity or preference (White and Morris 2007). A lack of comprehensive hyperlinks can frustrate users, leading them to abandon their queries prematurely (Scaria et al. 2014: 1).

1.1 Related work:

A number of researchers have attempted to model web users' search behavior, with the end goal of improving online search experiences. West and Leskovec (2012: 619) identified strategies searchers used when navigating in the information space of Wikipedia. They found that human navigation paths were efficient but differed qualitatively from the shortest paths in that humans tended to follow trails of conceptually related pages. They measured conceptual relatedness with term frequency-inverse document frequency (TF-IDF) cosine similarity. TF-IDF is a weighting scheme for measuring the relative importance of terms in a set of documents, and an individual document can be represented as a vector of the TF-IDF weights of its terms. These vectors are then compared using their cosine similarity, which is a measure of the similarity of two nonzero vectors.

Scaria et al. (2014: 5) investigated why users give up on navigation in Wikipedia. They quantified the difficulty of navigation paths using features such as the shortest path length from the source in the Wikipedia graph, the sum of the TF-IDF between each pair of articles along the shortest path to the target, and the number of links pointing toward the target (the target's in-degree). They used a similar feature set to predict the probability of success or failure of a given game, or "mission."

Ageev et al. (2011) defined a more general model for search success on the internet and used this model to distinguish between successful and unsuccessful searchers. They examined user search behaviors including the time between clicks, the number of pages visited before a user reached their goal, and their query length (Ageev et al. 2011: 348).

1.2 Research questions and hypotheses:

These approaches have been effective in predicting user search success. Ageev et al. (2011: 352) reported an accuracy of up to 0.55, and Scaria et al. (2014: 4) reported an area under the curve (AUC) between .802 and .958 for predicting whether users would abandon a search task before completion. While these models offer predictive power, it is unclear if they reflect underlying cognitive realities, as they depend on information that is not available to users when they are navigating in Wikipedia or searching on the wider web. For example, calculating TF-IDF similarity between pages depends on knowledge of the entire content of both pages, including the one that the user has not yet visited. Determining the shortest path length from the current page to the target page, as in Scaria et al. (2014: 5) requires global knowledge of the Wikipedia network structure. In the case of Ageev et al. (2011: 348), the researchers used information that was theoretically available to their subjects but did not leverage the textual content of the pages when predicting search success.

This study takes steps towards addressing this gap by adding measures of semantic relatedness between content that human players have access to as weights on the Wikipedia graph. It then asks whether the paths that successful players follow are more similar to the unweighted or the semantically weighted shortest paths. This question is operationalized by considering the correlation between the lengths of the unweighted and semantically weighted shortest paths to the human path lengths. Due to the computational intensiveness of answering this research question, this paper will focus on the methodology necessary to do so, particularly on optimizing the computation of the aforementioned shortest paths.

2 Data:

These research questions will be explored in the Wikispeedia dataset. West et al. (2009) developed the Wikispeedia game and dataset to create a semantic distance measure based on human information network navigation. Players were given a source article and had to reach a target article using only the links in Wikipedia. For example, a player could be assigned the source article "Seychelles" and would have to reach the target article "Great Lakes." An actual human path for this article consisted of navigating through the following pages: Seychelles, Fishing, North America, Canada, and Great Lakes (West et al. 2009: 1599). Users were allowed to back click without penalty.

The Wikispeedia dataset consists of the 4,604 articles (vertices) used in the game, 119,882 links (edges) between them, 51,318 finished human navigation paths, and 24,875 unfinished human paths. The authors also included a matrix of the shortest path distance by number of clicks between each article. West et al. (2009) did not collect demographic data on subjects or describe how they were recruited, though the game is freely available through West's website (https://www.cs.mcgill.ca/ rwest/wikispeedia/). The dataset also includes information on the duration of each game in seconds and an optional difficulty rating for finished games.

Although the Wikispeedia game is more constrained than a "wild web" navigation task in which subjects can use any internet resource at their disposal, the resulting data is optimal for this research because it offers a known target, source, and information network. Therefore, it is possible to determine whether players reached their targets, which can only be approximated in datasets such as the Wikipedia click stream dataset (Wulczyn and Taraborelli 2015). Ageev et al. (2011: 346) used a question-answering game to study human search behavior online in which players could use any of a number of commercial search engines, including Google, Yahoo, and Bing. This approach, while potentially more generalizable to sources beyond Wikipedia, does not allow for analysis of search behavior within an information network with a clearly defined structure.

3 Methodology:

The semantically weighted and unweighted shortest paths were calculated using three approaches, which I will describe in detail in this section. First, I computed the shortest paths between individual pairs of source and target articles using Dijkstra's algorithm—a widely used method for finding the shortest paths between vertices in a graph—with a single processor. I then parallelized this computation by sending batches of these path calculations to be completed by multiple processors. Finally, I used a variant of the Floyd-Warshall algorithm, a dynamic programming algorithm that calculates the shortest paths between all pairs of vertices in a graph, to determine the shortest paths between source and target vertices.

3.1 Preprocessing:

Before computing the shortest path lengths, I replicated West et al.'s (2009) Wikipedia graph structure and weighted its edges using a measure of semantic similarity between the page titles of each article. Using text files of article names and the links between them provided by West et al., I implemented their version of Wikipedia as a directed graph using Networkx, a Python package for network analysis (Hagberg et al. 2008). Articles were represented as vertices and links as directed edges.

Semantic distances were calculated using WordNet, a lexical database that describes semantic relationships between words, such as synonymy and antonymy (Fellbaum 1998). WordNet was used as a proxy for human semantic knowledge because it was hand-coded by human annotators and because its tree structure was inspired by psycholinguistic experiments on lexical semantic processing (Miller et al. 1993: 7). Multiple semantic similarity and distance measures that use WordNet have been proposed (Pedersen et al. 2004); I used the Leacock-Chodorow similarity measure because it has been shown to be the most closely correlated with human semantic similarity judgments (Budanitsky and Hirst 2001: 5). The Leacock-Chodorow measure computes semantic similarity using the length of the shortest paths between each pair of inputs in WordNet. Because Leacock-Chodorow is a similarity measure rather than a distance measure, I used its reciprocal when adding edge weights so that smaller values would correspond to shorter paths between nodes and vice versa. The Natural Language Toolkit (NLTK) (Bird et al. 2009) was used to access WordNet and calculate the Leacock-Chodorow distance between each pair of vertices.

To ensure that semantic distances could be calculated for all edges, I filtered the Wikipedia graph. Vertices (article titles) which were not present in WordNet were excluded, as were human paths that passed through those vertices. To avoid unnecessarily excluding edges, I used WordNet's Morphy morphological analyzer to determine if a word's base form was included in WordNet even if its inflected form was not. For example, the word "dogs" is not present in WordNet, but Morphy would return "dog," which is. Paths in which users pressed the back button were also excluded, as there is little research to indicate how these back-clicks should be incorporated into semantic distance calculations. This filtering resulted in a final graph with 2,344 vertices and 61,066 edges and a total of 17,129 finished human navigation paths.

3.2 Description of computational approaches:

3.2.1 Dijkstra's algorithm with a single processor:

My first approach used Dijkstra's algorithm to calculate the shortest weighted and unweighted paths between pairs of source and target vertices. In a general sense, Dijkstra's algorithm works by repeatedly selecting unexplored vertices, examining all of their neighbors, and considering whether they can be reached faster than previously hypothesized by passing through the current vertex.

More specifically, Dijkstra's algorithm assumes a weighted, directed graph $G = (V, E)$, where V and E are vertices and edges. It also assumes that all edge weights are non-negative, that is, for all edge weights w between vertices u and v, $w(u, v) \geq 0$ for all edges $(u, v) \epsilon E$. Pseudocode for this algorithm is shown below (Cormen et al. 2009: 658): In the initialization steps in lines 2 through 6, $v.d$ is the upper bound on the weight of a shortest path from the source vertex s to vertex v, and $v.\pi$ represents the predecessor vertex in the shortest path. In the "relaxation" steps shown in lines 13 through 16, we check whether we can update the shortest path to v that we've found thus far by going through vertex u. If so, we update d and π (Cormen et al. 2009: 648). This pseudocode produces the shortest path from the source vertex to all other vertices in the graph, from which the path, the source, and a given target vertex can be extracted.

Algorithm 1 Dijkstra's algorithm

```
 1: procedure DIJKSTRA($G, w, s$)
 2:     for each vertex $v \epsilon G.V$ do                    ▷ Begin initialization
 3:         $v.d = \infty$
 4:         $v.\pi =$NIL
 5:     end for
 6:     $s.d = 0$                                            ▷ End initialization
 7:     $S = \emptyset$
 8:     $Q = G.V$
 9:     while $Q \neq \emptyset$ do
10:         $u =$ EXTRACT-MIN($Q$)
11:         $S = S \cup \{u\}$
12:         for each vertex $v \epsilon G.Adj[u]$ do
13:             if $v.d > u.d + w(u, v)$ then              ▷ Begin relaxation of vertex
14:                 $v.d = u.d + w(u, v)$
15:                 $v.\pi = u$
16:             end if                                       ▷ End relaxation of vertex
17:         end for
18:     end while
19: end procedure
```

When the queue Q is implemented with a Fibonacci heap, Dijkstra's algorithm can achieve a worst-case runtime of $O(VlogV + E)$.

3.2.2 Dijkstra's algorithm with multiprocessing:

I attempted to achieve a speedup over the single-process Dijkstra's algorithm approach by parallelizing the computation of the shortest paths. Parallelization refers to the act of sending processes or computations out to multiple processors or threads to be carried out simultaneously. This computation is embarrassingly parallel, as determining the shortest path from one source to one target does not depend on any of the other path computations, at least when using the implementation of Dijkstra's algorithm in question. It is also an example of data parallelism in which data is split across multiple processors but the same code is run on each dataset, as opposed to task parallelism, in which multiple tasks are run in parallel on the same dataset (Singh et al. 2013: 3).

Parallelization was implemented using the Python multiprocessing module. I used the pool/map approach, in which the pool operation spawns a group of processes, and the map operation applies the parallelized computation to every element in an iterable and returns a list (Singh et al. 2013: 4). Because map can only accept one iterable argument, I had to create a list of tuples of a given data frame row and the graph in which the shortest paths were being calculated before using map. I then unpacked these values in the function that map called. Singh et al. (2013: 9) showed that the pool/map approach performs best when the number of processes is equal to the number of physical cores on the machine, which was four in the case of my personal laptop. However, I set the number of processes using the multiprocessing method cpu_count(), so this number could change depending on the machine on which the script was being run.

3.2.3 The Floyd-Warshall algorithm:

The Floyd-Warshall algorithm is a dynamic programming approach that computes the shortest path distances between all pairs of vertices in a graph. As a dynamic programming algorithm, it allows us to improve on the performance of a naive recursive algorithm by breaking the problem of finding all shortest path distances into smaller subproblems, finding optimal solutions for these subproblems, and using them to solve the original problem (Cormen et al. 2009: 694). The Floyd-Warshall algorithm uses a bottom-up approach, in which we start with the smaller subproblems, save their solutions in a table, and build up from these solutions.

Formally, we begin by by initializing a table of weights W with our known weights and set unknown weights to ∞. Specifically, entry w_{ij} is 0 if $i == j$, the weight of the edge between i and j if $i \neq j$ and both are in set ϵ, and ∞ if we don't yet know an edge's weight, as seen below:

$$w_{ij} = \begin{cases} 0 & i = j \\ \text{the weight of the directed edge(i,j)} & i \neq j \text{ and } (i,j)\epsilon E \\ \infty & i \neq j \text{ and } (i,j) \notin E \end{cases} \tag{1}$$

Next, consider $d_i^{(k)}j$, the weight of a shortest path from i to j using vertices only using vertices from the set 1. . . k as intermediate points. We then define our problem recursively (Cormen et al. 2009: 695):

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & k = 0 \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) & k \geq 1 \end{cases} \tag{2}$$

With this definition and our initialization, the complete pseudocode for the Floyd-Warshall algorithm is shown below:

Algorithm 2 The Floyd-Warshall algorithm

```
 1: procedure FLOYD-WARSHALL($W$)
 2:     $n = W.rows$                          ▷ the matrix of weights, as initialized above
 3:     $D^{(0)} = W$
 4:     for k = 1 . . . n do
 5:         let $D^{(k)} = (d_{ij}^{(k)})$ be a new n x n matrix
 6:         for i = 1 . . . n do
 7:             for j = 1 . . . n do
 8:                 $d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$      ▷ the recursive case
 9:             end for
10:         end for
11:     end for
12:     return $D^{(n)}$                      ▷ the final matrix of shortest-path distances
13: end procedure
```

Because of the three nested for loops, the Floyd-Warshall algorithm runs in $O(V^3)$ time, where V is the number of vertices in the graph. This algorithm can also be extended to return a matrix of predecessors, from which the shortest path between each pair of vertices can be reconstructed.

3.3 Comparison of computational approaches:

In this section, I compare the three computational approaches in terms of their speed, memory usage, and programming difficulty and readability. I argue that the Floyd-Warshall algorithm performs the best in terms of speed and memory usage but that it is less readable than the Dijkstra-based approaches.

3.3.1 Speed:

The three approaches were profiled for speed using the Python time_profile and line_profiler modules. Each approach was run 10 times to produce a mean and standard deviation rather than a point estimate; these statistics are shown below. Because the single-process Dijkstra approach did not finish running in a reasonable amount of time on the full dataset, the statistics below were computed on a smaller subgraph of vertices with an in-degree greater than 100 to ensure comparability between approaches. The resulting subgraph had 6,398 vertices and 229 edges.

For both of the Dijkstra-based approaches, line profiling indicated that the large majority of the time was spent on the call to Dijkstra's algorithm, as expected (80.0 percent for the single-processor approach and 90.7 percent of the time for the multiprocessing approach). For the Floyd-Warshall algorithm, only 50.4 percent of the time was spent actually calculating the shortest paths, while the remainder was used for the overhead needed to convert the graph from the NetworkX format to the SciPy format.

As seen in the table below, the Floyd-Warshall algorithm was the fastest for this dataset, followed by the parallelized Dijkstra approach and then the single-process Dijkstra implementation.

Table 1: Time profiling results

Approach	Single-process Dijkstra	Parallelized Dijkstra	Floyd-Warshall
Mean (ms)	6.6E5	2.2E3	205
Standard deviation (ms)	43.4	116	6.99

A possible explanation for Floyd-Warshall's speed advantage over that of both Dijkstra-based approaches could be that it is from the SciPy package rather than NetworkX. SciPy is written primarily in Fortan and C, whereas NetworkX is written in pure Python (Hagberg et al. 2008: 14). Fortran and C code can run faster than Python code because the first two languages are statically typed, as opposed to Python, which is dynamically typed. In statically typed languages, variable types are declared when the variable is created, and type checking is done at compile time, rather than runtime, as in a dynamically typed language such as Python. For instance, in C, an integer variable d could be declared and set to a value of 3 by writing either:

```
int d;

d = 3;
```

or

```
int d = 3;
```

In Python, we could simply write

```
d = 3
```

without declaring the variable's type. The extra time needed to infer that d is an integer can, depending on the specific Python implementation, slow Python down at runtime relative to C or Fortran.

However, it is somewhat surprising that the Floyd-Warshall algorithm delivered the performance improvement that it did given that it should only outperform repeated calls to Dijkstra's algorithm on a dense graph (Johnson 1977: 4). The Wikipedia graph used in these analyses had a density of only .0057. Graph density is defined below, where $|E|$ is the number of edges, and $|V|$ is the number of vertices (Coleman and Moré 1983: 203)

$$\frac{|E|}{|V|(|V|-1)}$$

Because the Floyd-Warshall algorithm only solves the all-pairs shortest paths problem, it also computes some unnecessary shortest paths, given that humans did not complete every path between possible pairs of vertices in the Wikispeedia dataset. On the other hand, Dijkstra's algorithm can be called on only the necessary source-target pairs and can be modified to compute only a single source-target path rather than a path from the source to every other vertex, as in this specific NetworkX implementation.

Between the two implementations of Dijkstra's algorithm, the parallelized version showed a substantial speedup, as expected. The size of the parallelized approach's advantage—approximately two orders of magnitude—was considerable. If the parallelized approach were run on an external server or a machine with more CPUs, this speedup could be even greater.

3.3.2 Memory usage:

Each approach was profiled for memory usage using the Python memit() function from the memory-profiler module. Each approach was run 10 times, and results are shown below. The "peak memory usage" row represents the total memory usage of the system when the function was running, whereas the "increment" row represents the change in memory usage from immediately before the function was run to the time when it was running. It is therefore a somewhat better indicator of how much memory the function used. Measurements are shown in mebibytes (MiB):

Table 2: Memory profiling results

Approach	Single-process Dijkstra	Parallelized Dijkstra	Floyd-Warshall
Peak memory usage (MiB)	656.38	767.77	699.60
Increment (MiB)	30.86	53.62	4.47

The smaller memory requirements of the SciPy implementation of the Floyd-Warshall algorithm is somewhat unsurprising. The space complexity of the Floyd-Warshall algorithm to find all shortest paths is $O(V^2)$ because it requires only a single $V \times V$ adjacency matrix. On the other hand, using Dijkstra's algorithm to solve the all-pairs shortest path problem requires a $V \times V$ adjacency matrix for the computation of the shortest paths from each source vertex, resulting in a total space complexity of $O(V^3)$ (Cormen et al. 2009: 591). The use of C in SciPy may also contribute to the fact that the SciPy Floyd-Warshall approach used less memory than the pure Python NetworkX approaches. C uses manual memory management, which allows the programmer to optimize memory use for their specific application, whereas Python manages memory automatically and does not allow programmers to optimize their memory use (Detlefs et al. 1994: 527). It is possible, though not necessarily the case, that the SciPy developers were particularly conscientious in allocating and deallocating memory, which gave them a performance advantage.

The multiprocess Dijkstra approach used almost twice as much memory as the single-process approach. This result is unsurprising given that the multiprocess approach calculates shortest paths for multiple examples simultaneously, unlike the single-process approach. I also used the Python mprun module to attempt to further isolate causes of difference in memory usage, but the results were not informative given that all approaches called a number of third-party functions which I was not able to optimize.

3.3.3 Programming difficulty and readability:

The single-processor Dijkstra approach was the most straightforward to implement and read, followed by the multiprocess Dijkstra and then the Floyd-Warshall algorithm. For the actual calculation of the shortest paths, the single process Dijkstra implementation required only two lines of code: one to calculate the weighted shortest paths, and one to calculate the unweighted ones. The NetworkX shortest_path() method returns the path as a list of strings, which could then be directly inserted into a Pandas DataFrame.

The multiprocess Dijkstra approach was somewhat more challenging to implement. It required splitting the original DataFrame of paths into its component rows, creating a tuple of the row and the relevant graph for each row, and then unpacking this tuple in the function that was parallelized to calculate the shortest path. However, as with the previous approach, the paths were conveniently returned as a list of strings.

While the actual calculation of the shortest paths with the Floyd-Warshall algorithm was not difficult, getting the results into an interpretable format was more complex than with the other two approaches. Before calculating the shortest paths, the graph had to be converted to a SciPy sparse matrix format.The SciPy implementation of Floyd-Warshall returns a matrix of shortest path distances with numerical indices. These results had to be remapped to their named indices before they could be interpreted. Unlike with the NetworkX implementations, I had to write a separate function to reconstruct the shortest paths using the predecessor matrix.

3.3.4 Overall comparison:

Overall, the Floyd-Warshall algorithm was the best choice for computing the shortest unweighted and semantically weighted paths in the Wikipedia graph. This approach was by far the best performer in terms of memory usage and speed. The fact that nearly half of its runtime came from the overhead required to convert the graph into a SciPy format suggests that this advantage could be even larger for a graph already in this format. Its only downside was that it required more work to convert its results into a human-readable format than with the single-process and parallelized Dijkstra implementations. However, this difference was comparatively minor. If similar analyses were to be conducted on a dataset that was orders of magnitude larger, such as real-time Wikipedia navigation data, this would likely be the only viable approach of the three presented in this paper.

4 Results:

Overall, the results of my analyses indicated that the weighted shortest path lengths were weakly correlated with the human path lengths, though other evidence indicates that these lengths may be useful in predicting different aspects of human search behavior. This weak correlation can be seen in the correlation matrix below. The unweighted shortest path lengths were much more positively correlated with the weighted and unweighted human path lengths than the weighted path lengths. As seen below, both the weighted and unweighted shortest path lengths were positively correlated with human perceptions of difficulty.

The full correlation matrix of these variables is shown below. Note that human path length is represented as "HPL" and shortest path length as "SPL" and that "rating" refers to the optional difficulty rating that users assigned to each mission:

Table 3: Correlation matrix

	Rating	HPL (unweighted)	HPL (weighted)	SPL (unweighted)	SPL (weighted)
Rating	1.00	0.43	0.42	0.24	0.22
HPL (unweighted)	0.43	1.00	0.89	0.49	-0.01
HPL (weighted)	0.42	0.89	1.00	0.46	-0.01
SPL (unweighted)	0.24	0.49	0.46	1.00	-0.04
SPL (weighted)	0.22	-0.13	-0.13	- 0.04	1.00

In addition to examining the correlations above, I also compared the first move of the semantically weighted and unweighted shortest paths to the players' actual first moves as another method of answering the question of whether semantic distance measures between page titles guide players' moves. To do so, I created an additional variable in my DataFrame that indicated whether the first move of the player's path matched that of the shortest weighted and unweighted paths. I then conducted a two-sample z-test to determine whether the proportions of matches differed significantly for the weighted and the unweighted paths using the Python statsmodels package (Seabold and Perkold 2010). I found that the weighted paths were a significantly better match for the players' first moves, with a matching proportion of .22, compared to the unweighted paths' matching proportion of .19 ($z = 5.24$, $p = 1.60E-7$).

5 Discussion:

These results provide support for the idea that humans generally take optimal paths in web navigation. The semantically weighted shortest paths were not highly correlated with the weighted or unweighted human path lengths, while the unweighted shortest paths were. This latter result confirms West and Leskovec's (2012: 619) finding that human navigation behavior in information networks is generally efficient. However, this result is somewhat surprising given that it relies on information that humans do not have access to when they play the Wikispeedia game.

The results of the z-test suggest that semantically weighted shortest paths are a better predictor of humans' first moves than the unweighted shortest-path distances. Taken in tandem with the previous result that semantically weighted path lengths are not highly correlated with human path lengths, this could indicate that humans use semantic relatedness to guide them early in their games and change strategies as the game progresses. Such a theory would be consistent with West and Leskovec's (2012: 623) idea that players tend to change their tactics partway through the game, first navigating to "hub" vertices with a high in-degree and then narrowing in toward their targets.

5.1 Avenues for further research:

Possibilities for further research include building more sophisticated predictive models and extending this work to new datasets. Additional investigations could include analysis of the unfinished paths in terms of the semantic distances that users traveled and examining where humans typically deviate from the semantically weighted and unweighted shortest paths. Future work could also model additional information that is available to humans when they play the Wikispeedia game, such as leveraging the text on the pages they have seen up to their current one and extending existing predictive models of navigation in Wikispeedia to the larger and more generalizable set of Wikipedia click trails.

6 Conclusion:

This study investigated the question of whether human navigation paths in the Wikispeedia game were more closely related to semantically weighted or unweighted shortest paths in the Wikipedia graph. It approached this question with a focus on methodology of determining shortest paths using three computational approaches: a single-process implementation of Dijkstra's algorithm, a parallelized implementation of Dijkstra's algorithm, and the Floyd-Warshall algorithm. Methodologically, the study showed that the SciPy implementation of the Floyd-Warshall algorithm was the best performer in terms of speed and memory usage, though additional work was required to turn its results into an interpretable format. Analysis of the original research question using output from these approaches showed that the unweighted path lengths were more highly correlated with human path lengths than those of the semantically weighted paths. However, the weighted paths beat the unweighted paths in predicting players' first moves. Deeper investigation is needed to answer the question of how semantic distance measures can be leveraged to model human web navigation, but these insights could be used to improve web search experiences well beyond the bounds of the Wikispeedia game.

References

Ageev, M., Guo, Q., Lagun, D., & Agichtein, E. (2011). Find it if you can: a game for modeling different types of web search success using interaction data. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information - SIGIR '11* (p. 345). Beijing, China: ACM Press. Retrieved 2019-01-12, from `http://portal.acm.org/citation.cfm?doid=2009916.2009965` doi: 10.1145/2009916.2009965

Bird, S., Klein, E., & Loper, E. (2009). *Natural Language Processing with Python.* (OCLC: 938844428)

Budanitsky, A., & Hirst, G. (2001). Semantic distance in wordnet: An experimental, application-oriented evaluation of five measures. , 6.

Coleman, T., & Moré, J. (1983). Estimation of Sparse Jacobian Matrices and Graph Coloring Problems. , 24.

Cormen, T. H. (Ed.). (2009). *Introduction to algorithms* (3rd ed ed.). Cambridge, Mass: MIT Press. (OCLC: ocn311310321)

Detlefs, D., Dosser, A., & Zorn, B. (1994, June). Memory allocation costs in large C and C++ programs. *Software: Practice and Experience*, *24*(6), 527–542. Retrieved 2019-01-13, from `http://doi.wiley.com/10.1002/spe.4380240602` doi: 10.1002/spe.4380240602

Hagberg, A. A., Schult, D. A., & Swart, P. J. (2008). Exploring Network Structure, Dynamics, and Function using NetworkX. , 5.

Johnson, D. B. (1977, January). Efficient Algorithms for Shortest Paths in Sparse Networks. *Journal of the ACM*, *24*(1), 1–13. Retrieved 2019-01-12, from `http://portal.acm.org/citation.cfm?doid=321992.321993` doi: 10.1145/321992.321993

Miller, G. A., Beckwith, R., Fellbaum, C., Gross, D., & Miller, K. J. (1990). Introduction to WordNet: An On-line Lexical Database *. *International Journal of Lexicography*, *3*(4), 235–244. Retrieved 2019-01-12, from `https://academic.oup.com/ijl/article-lookup/doi/10.1093/ijl/3.4.235` doi: 10.1093/ijl/3.4.235

Pedersen, T., Patwardhan, S., & Michelizzi, J. (2004). WordNet::Similarity: measuring the relatedness of concepts. In *Demonstration Papers at HLT-NAACL 2004 on XX - HLT-NAACL '04* (pp. 38–41). Boston, Massachusetts: Association for Computational Linguistics. Retrieved 2019-01-12, from http://portal.acm.org/citation.cfm?doid=1614025.1614037 doi: 10.3115/1614025.1614037

Scaria, A. T., Philip, R. M., West, R., & Leskovec, J. (2014). The last click: why users give up information network navigation. In *Proceedings of the 7th ACM international conference on Web search and data mining - WSDM '14* (pp. 213–222). New York, New York, USA: ACM Press. Retrieved 2019-01-12, from http://dl.acm.org/citation.cfm?doid=2556195.2556232 doi: 10.1145/2556195.2556232

Seabold, S., & Perktold, J. (2010). Statsmodels: Econometric and Statistical Modeling with Python. , 6.

Singh, N., Browne, L.-M., & Butler, R. (2013, August). Parallel Astronomical Data Processing with Python: Recipes for multicore machines. *Astronomy and Computing*, *2*, 1–10. Retrieved 2019-01-12, from http://arxiv.org/abs/1306.0573 (arXiv: 1306.0573) doi: 10.1016/j.ascom.2013.04.002

West, R. (2009). Wikispeedia: An Online Game for Inferring Semantic Distances between Concepts. , 6.

West, R., & Leskovec, J. (2012). Human wayfinding in information networks. In *Proceedings of the 21st international conference on World Wide Web - WWW '12* (p. 619). Lyon, France: ACM Press. Retrieved 2019-01-12, from http://dl.acm.org/citation.cfm?doid=2187836.2187920 doi: 10.1145/2187836.2187920

White, R. W., & Morris, D. (2007). Investigating the querying and browsing behavior of advanced search engine users. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR '07* (p. 255). Amsterdam, The Netherlands: ACM Press. Retrieved 2019-01-12, from http://portal.acm.org/citation.cfm?doid=1277741.1277787 doi: 10.1145/1277741.1277787

Wulczyn, E., & Taraborelli, D. (2015). *Wikipedia Clickstream.* Retrieved from https://meta.wikimedia.org/wiki/Research:Wikipedia_clickstream