Mikhail Sushkov
msushkov

Decentralized Search in the Wikipedia Graph

Note: Mikhail is working on this project with John Doherty (doherty1@stanford.edu). John is not taking CS 224W, but both Mikhail and John used this project for CS 224N (not the exact same project but some overlap in terms of the data and features).

Introduction

Over the last decade, Wikipedia has become the premier source of knowledge in the world. As the state of the art of natural language understanding is pushed forward, Wikipedia remains one of the most important sources of information in analyzing human knowledge. Perhaps one of Wikipedia's most significant features is the fact that every article represents a unique entity in the world. Not only does each article contain information about a particular entity, but it contains links to other articles, linking entities together.

This project aims to explore the concept of decentralized search in the context of the Wikipedia network. The question we want to answer is: can we come up with a good heuristic to use in decentralized search such that it approximates the relatedness of Wikipedia articles as defined by the Wikipedia graph and the DBPedia ontology (a hierarchy of types) [1]? We explore different heuristics for decentralized search: those related to properties of nodes in the Wikipedia graph, and those related to properties of article text. Each of those heuristics produces a path length for successful runs of decentralized search on a given article pair. We evaluate the different heuristics based on how well they approximate shortest paths in the Wikipedia graph as well as in the ontology.

Prior Work

This work takes inspiration for finding distances between concepts represented in Wikipedia from work by Gabrilovich et al. [2]. In this work distances between arbitrary texts were computed by representing the texts in the Wikipedia concept space. In this project we represent our texts via more general feature vectors, not constrained to the space of Wikipedia concepts, but the general idea of mapping out concepts in the real world to some vector space nonetheless informs the motivation for this project.

The idea of applying decentralized search to this problem comes from a paper that runs decentralized search on networks using homophily and degree disparity as heuristics [3]. We extend this idea by applying a range of different heuristics to see if they help the decentralized search to give more accurate distance results.

Data

We use four related datasets to accomplish this: the Wikipedia article network, the DBPedia ontology, a dataset of instance types (mapping between article name and types), and a corpus of article text.

Wikipedia Graph

In order to perform decentralized search it is necessary to obtain some portion of the Wikipedia graph. We use the Wikipedia page-to-page link database from the course website. The dataset contains a list of edges between Wikipedia articles, where each article is represented by its name.

Ontology

The DBPedia ontology is a hierarchy of types in Wikipedia [1]. It is a tree that consists of approximately 700 types (Person, Place, BasketballPlayer, etc.), and is at most 7 levels deep.

Instance Types

Each Wikipedia article has a number of types associated with it, which are given in the DBPedia instance types data set. We process this dataset to extract a single type for each Wikipedia article; if an article has several types associated with it, as is the case for most articles, we select the type that is furthest down the ontology tree; in other words we choose the most specific type for each article. For example, if an article has the types Person and BasketballPlayer (where BasketballPlayer is a subtype of Person) we choose BasketballPlayer since it is the more specific type.

Wikipedia Article Corpus

In order to extract NLP features that might be useful in making local decisions during decentralized search it is necessary to obtain the full text for each Wikipedia article. This corpus is obtained by downloading the text of about 60,000 articles using the Wikipedia API. Upon downloading the article text we remove all stop words and Wikipedia markup and tokeniz the text.

Additional Processing

It is necessary to combine the data from our four datasets such that for a given pair of article names we could do the following:

- run decentralized search from the first article to the second article, and while running this, obtain the text of each article that the search hits;
- find the length of the shortest path between the articles in the Wikipedia network;
- identify the type of each article
- given the article types find the distance between the articles in the ontology tree through the type that is the least common ancestor (LCA) of the article types;
- the height of the LCA of the article types in the ontology.

In order to do this, we first filter our Wikipedia graph to include only edges between nodes for which we had type information (ensuring that the types were also found in the DBPedia Ontology). This reduced the Wikipedia graph to about 1.3 million nodes. Since we need to be able to find paths between arbitrary article pairs, we restrict ourselves to using nodes in the largest strongly-connected component, which contains about 1 million nodes. This is still too large of a dataset for our computational resources, so we generate a subset of the Wikipedia graph. First, we take a random node from the largest strongly-connected component. Call this node v. We first add v to the graph. We then take all the nodes that are 1 hop away from v and add them to the graph. Then, we add nodes that are 2, 3, and 4 hops away. This gives us a smaller but connected directed graph of approximately 60,000 nodes. This is the graph which we work with.

Figures 1 and 2 show the in- and out- degree distributions, and some key statistics of the 60k-node Wikipedia graph.

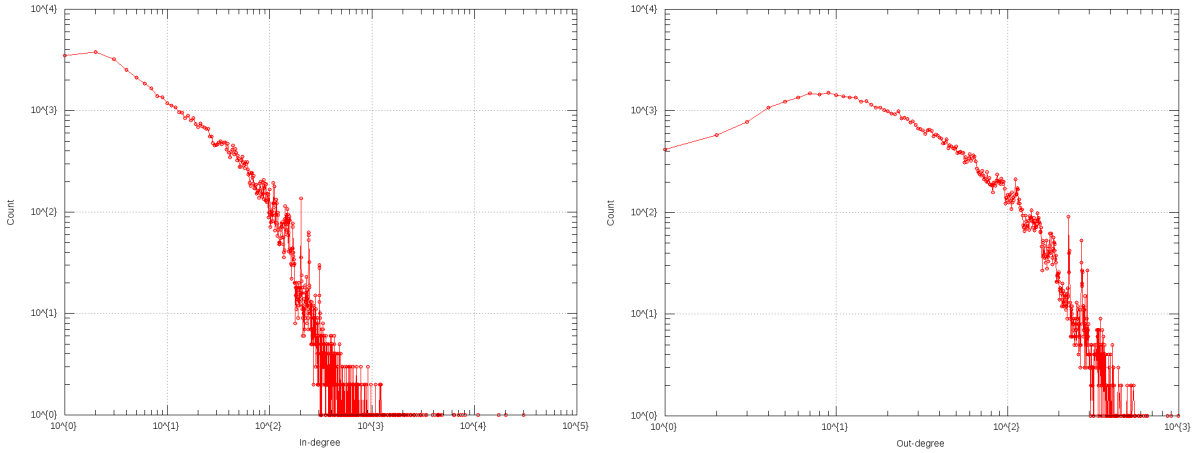

Figure 1. *In- and out- degree distributions of our 60k-node subset of the Wikipedia graph.*

Number of nodes	63887
Number of edges	3250827
Fraction of closed triads	0.03
Approximate full diameter	5
90% effective diameter	2.85
Clustering coefficient	0.45

Figure 2. *Statistics of our 60k-node subset of the Wikipedia graph.*

Approach

Running decentralized search using different heuristics can be summarized as the following algorithm:

```
search(source, destination, heuristic, current_path_length):
    if source == destination:
        return (SUCCESS, current_path_length + 1)
    else:
        if current_path_length > 500:
            return (FAILURE, -1)
        else:
            for each child of source:
                distance = heuristic(child, destination)
            new_source = child with smallest distance (as measured by heuristic)
            return search(new_source, destination, heuristic, current_path_length + 1)
```

At each step of the search the algorithm makes a local decision as to which of a given node's children to explore next. This decision depends on the heuristic: it tells the algorithm how close each child is to the destination, and chooses the child closest to the destination as the node to explore next.

The decentralized search gives some value for the number of steps it took to complete, in the case of success. Since it can often be the case that the decentralized search gets unlucky and ends up pursuing a

path that will not lead to the destination (if somewhere earlier it made a bad decision) we stop the search after 500 steps to make sure the algorithm terminates.

Evaluation

We use this path length given by our search routine to evaluate how well the decentralized search performs for a given pair of articles. We compare this to several values: the path length given a heuristic that makes a random decision at each step, the shortest path between the two articles in our 60k-node subgraph, the length of the path through the ontology tree between the types of the articles, and the height of the LCA of the two articles in the ontology tree. Statistics on the successful completion of the search, as well as results on how well it did against each of four metrics defined above, are presented below in the results section.

Heuristics for Decentralized Search

We use two types of heuristics to run decentralized search in the graph: graph-based and text-based., and we evaluate these using a baseline heuristic that makes a random decision at each step (call this the random heuristic).

Graph-based Heuristics

We use two graph-based heuristics: smallest degree and largest degree. This heuristic simply chooses the neighbor that has the smallest degree (or largest degree) out of all the neighbors - this is the node that the search will jump to next.

Heuristic 1: Smallest-degree neighbor.
Heuristic 2: Largest-degree neighbor.

***Figure 3**. The graph-based heuristics used to represent pairs of articles.*

Text-based Heuristics

We also try several text-based heuristics, which are summarized in figure 3. In order to represent each article as a vector of TF-IDF values we perform additional processing of our Wikipedia corpus using the gensim library in Python [4]. We give a brief overview of TF-IDF below.

Heuristic 1: The inverse of the size of the intersection of the sets of words in the two articles.
Heuristic 2: The inverse of the Jaccard similarity between the sets of words in the 2 articles.
Heuristic 3: Cosine similarity between TF-IDF vectors of words in the 2 articles.

***Figure 4**. The text-based heuristics used to represent pairs of articles.*

TF-IDF

A TF-IDF representation of a document is a vector representation that is designed to assign a value to the importance of each word in an document [5]. As the name implies, it is composed of two components, the term frequency and the inverse document frequency. Essentially the importance of a word in a document increases if it appears more times in the document or fewer times in the rest of the corpus.

Results

The results for our different heuristics are presented in figure 5. For the random heuristic, we used 100 random pairs of articles, and for each pair we ran 1000 trials. For each of the test heuristics we looked at the

same 100 pairs of articles. For the graph heuristics we looked at 1000 pairs of articles since we observed no successes in the first 100 trials.

Figure 6 shows how closely each of the text heuristics approximates a given output variable.

Heuristic	Success Rate
Random	0.0044
Text heuristic 1: set intersection	0.8500
Text heuristic 2: Jaccard similarity	0.7300
Text heuristic 3: TF-IDF cosine similarity	0.8600
Graph heuristic 1: smallest-degree neighbor	0.0010
Graph heuristic 2: largest-degree neighbor	0.0090

Figure 5. *Decentralized search success rates for each heuristic.*

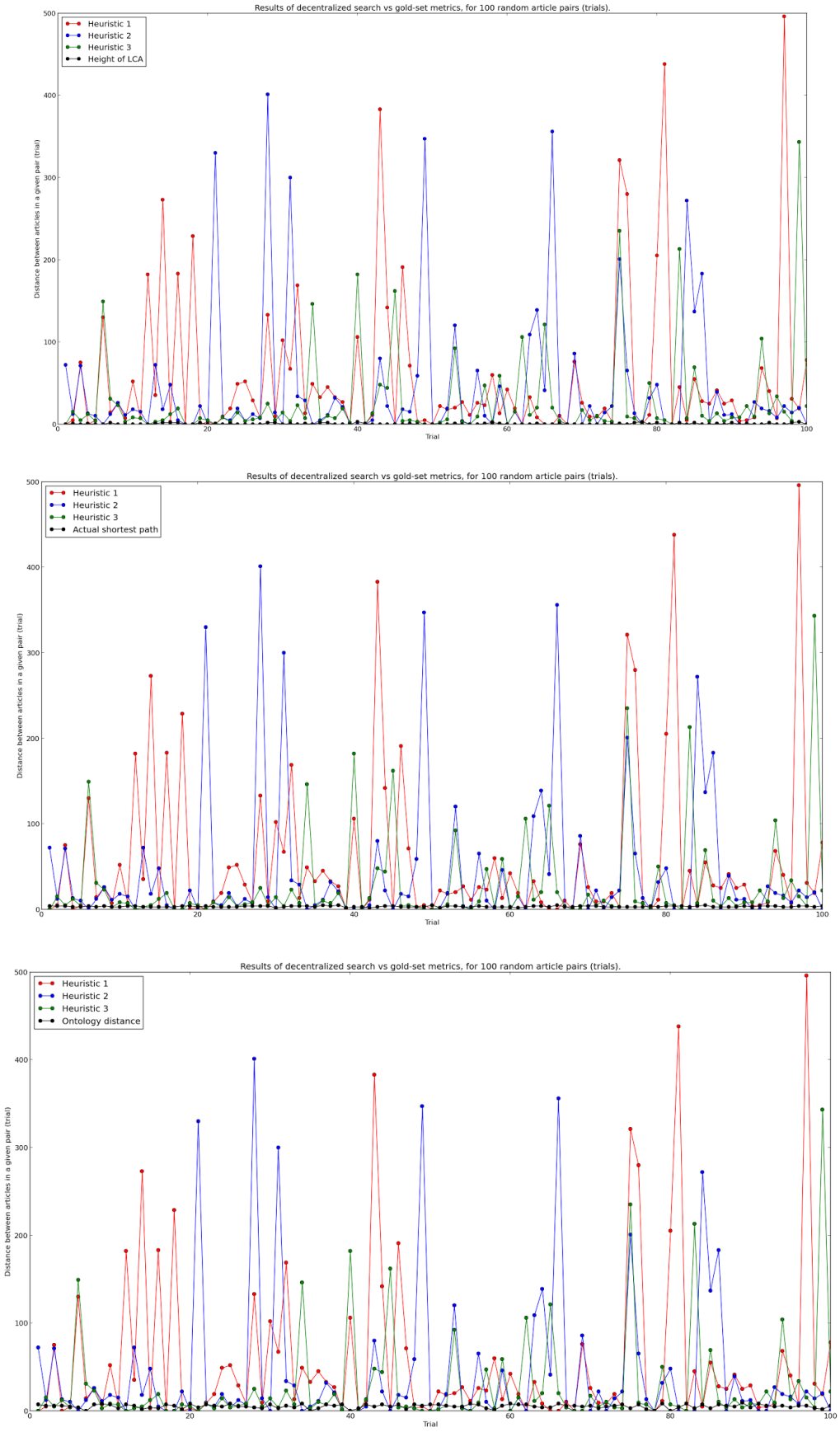

***Figure 6**. the extent to which each text heuristic approximates each output variable.*

Discussion

Figure 5 shows that the graph-based heuristics do not exhibit as high a success percentage as the text-based heuristics. This means that using some local information about the text of two articles, decentralized search can succeed approximately 85% of the time, which is much better than random, and also much better than using only the graph-based heuristics. We note that the best grpah-based heuristic, using the highest-degree neighbor, barely does better than random; as expected, choosing the smallest-degree neighbor does worse than random.

In figure 6 we can see the extent to which each of the text heuristics approximates one of the three output variables. While overall it seems that the errors between the decentralized search-produced path length and the output variables are high, they generally follow the same trends. For all 3 output variables, heuristic 3 outperforms 1 and 2 - thus, TF-IDF is better than Jaccard similarity and the size of the intersection as a measure of the distances between the articles.

Future Work

The first logical direction for this project would be to apply machine learning to learn the best weights for the different features used in a heuristic that combines different graph-related and text-related heuristics. These weights could be learned using using linear regression or support vector regression, with the error defined as the mean squared error between the output of decentralized search and the output variable. The output variable could be any of the following:

- the true shortest-path distance
- the height of the LCA
- the distance in the ontology tree.

One other possible area for improvement could be obtaining better data for shortest-path distances between articles. As mentioned previously we restricted our graph to be ~60,000 nodes. While this may provide some notion of reasonable shortest paths, better quality shortest-path distances would likely be obtained given the entire largest strongly connected component, which consists of 1 million nodes.

In terms of better distance metrics it would be interesting to look at word2vec, which can be used to give similarities between individual words [6]. This could provide another way to calculate distances between pairs of articles.

It would be useful to explore more advanced network features as better heuristics for decentralized search. In our analysis we explored choosing the next neighbor based on the neighbor's degree; it would be interesting to use the betweenness centrality of a given neighbor instead of its degree to make the local decision.

References

[1] http://wiki.dbpedia.org/Ontology
[2] Gabrilovich, E. and Markovich, S. (2007). Computing Semantic Relatedness using Wikipedia-based Explicit Semantic Analysis. In Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07), Hyderabad, In.
[3] Simsek, O. and Jensen, D. (2005). Decentralized search in networks using homophily and degree disparity. International Joint Conference on Artificial Intelligence, 19:304
[4] https://radimrehurek.com/gensim/
[5] Ramos, Juan. "Using tf-idf to determine word relevance in document queries." *Proceedings of the First Instructional Conference on Machine Learning*. 2003.
[6] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. In Proceedings of Workshop at ICLR, 2013.