A Linear Regression Model for Decentralized Search

Kensen Shi
Stanford University
kensens@stanford.edu

Albert Tung
Stanford University
atung3@stanford.edu

Eric Xu
Stanford University
ericxu0@stanford.edu

Abstract

Decentralized search is a classic problem that arises in message-passing between people or P2P networks and in information searching by following hyperlinks or paper citations. Previous approaches use node degree and similarity to perform decentralized search. However, there is an abundance of other locally-available information, such as path history and clustering coefficient, that has not previously been used in decentralized search algorithms. By combining various heuristics (including the recently-developed metric node2vec) in a linear regression model, we increased the success rate and decreased the average path length in both synthetic graphs and real-world networks, compared to previous work. We found that node2vec can enable similarity-based approaches to work in graphs that otherwise do not have a natural notion of similarity between nodes. Furthermore, in graphs (e.g., citation networks) that do have a natural notion of similarity, node2vec often provides a *better* similarity metric than the inherent similarity derived from the real-world meaning of the graph. Finally, we analyze the weights produced by the linear regression model to reveal insights about the decentralized search problem as a whole.

1 Introduction

Decentralized search is a computationally inexpensive algorithm for finding paths between nodes using only local information about the graph. It can model human behavior when searching the Web for information by following links, when exploring scientific literature by following citations, and in message-passing through friends or peer-to-peer computer networks.

We first define the problem of decentralized search on a graph. A decentralized search algorithm begins at a source node s and advances towards a destination node t. In each step of this process, the current node u has knowledge of its local neighborhood, the destination t, and the set of previously-visited nodes. In particular, each node does not have knowledge of long-range contacts or the entire graph. Using the local information, the algorithm chooses a neighbor of u to continue the search. We seek to minimize the resulting path length and maximize the chances of reaching the destination node within a small number of steps.

Various decentralized search algorithms have been extensively studied in the early 2000s, and proposed approaches involve similarity of nodes, degree disparity, hierarchical models, and other strategies tailored to specific network structures. However, there is a wealth of locally-available information contained in the graph that has not been used by prior approaches, including information about a neighbor's neighbors, connectivity among neighbors, relationships between neighbors and previously-visited nodes, and the recently-developed node2vec [3]. Using this additional information can greatly improve decentralized search algorithms.

We propose a method of training a linear regression model to predict the shortest path lengths between nodes, and we show that this model outperforms previous works (especially when using similarity derived from node2vec). We also find that node2vec is an extremely good similarity measure that outperforms other similarity metrics derived from the graph's real-world meaning.

2 Related Work

2.1 Decentralized Search in Networks Using Homophily and Degree Disparity [6]

Simsek and Jensen propose a new algorithm for decentralized search that uses homophily (the assumption that nodes with high similarity are more likely to be connected) and degree disparity (the observation that some nodes have larger degrees than others and can act as hubs). Their algorithm, called Expected-Value Navigation (EVN), aims to minimize the expected path length, which they approximated using the first two terms of the following:

$$E(l_{st}) = \sum_{\forall i} i \cdot \Pr(l_{st} = i)$$

where l_{st} is the path length from a neighbor s to the target t. If $s = t$, then $l_{st} = 0$, and otherwise, minimizing $E(l_{st})$ is approximated by maximizing $\Pr(l_{st} = 1)$, i.e., the next-best value. This probability is then estimated by assuming independent links, so the probability of s having an edge to t is

$$\Pr(l_{st} = 1) \approx 1 - (1 - q_{st})^k$$

where q_{st} is the probability that an individual edge from s leads to t (as a function of their similarity), and k is the degree of s.

Comparing EVN to degree-based and similarity-based approaches using synthetic networks and a scientific citation network, they show that EVN produces higher success rates and lower path lengths (when successful), compared to prior algorithms proposed by [4] and [1].

2.2 Search in Weighted Complex Networks [7]

Thadkamalla, Albert, and Kumara utilizes the idea of local betweenness centrality (LBC) as a new decentralized search strategy for weighted networks. First, they introduce the notion of a local network of a node - the node itself and its first and second neighbors. Then for a neighbor node i in this local network, they define $L(i)$ to be the LBC of i in this local network as

$$L(i) = \sum_{s \neq i \neq t} \frac{\sigma_{st}(i)}{\sigma_{st}}, s, t \in \text{local network}$$

They define σ_{st} as the total number of shortest paths (paths having minimum sum of weights) passing through i. A node with a high LBC would be considered as a node that is important in the local network, but not necessarily the node that has the highest degree because the search is performed in a weighted network and weighted paths are more important to the network. After computing the LBC, the authors compare their model with various other strategies including choosing a random neighbor and choosing the best-connected neighbor. Their algorithm, however, seems limited by the distribution of the weighted edges as running the LBC strategy on a Poisson network rarely gave better results than choosing the highest degree.

2.3 node2vec: Scalable Feature Learning for Networks [3]

Grover and Leskovec propose a computationally inexpensive method that computes a vector embedding of each node in a graph representing both the neighborhood and structure of the node. The algorithm resembles the word2vec skip-gram algorithm that generates word embeddings. Given a graph $G = (V, E)$, they learn a function $f : V \to \mathbb{R}^d$ mapping nodes to a d-dimensional vector. Similar to how the skip-gram model maximizes the probability that the words in a sliding window appear given the source word, for each node $u \in V$, they define $N_S(u) \subset V$ to be the neighborhood of u generated by a sampling strategy S and attempt to maximize $\sum_{u \in V} \log Pr(N_S(u)|f(u))$ over f.

They assume conditional independence of the neighborhood nodes and approximate the probability using softmax. Then, they apply stochastic gradient descent on the objective function to learn the embedding f.

The sampling strategy S proposed is a second order random walk of fixed length l with two parameters p and q. They define the unnormalized probability that a node u visits node v next after previously traveling on edge (t, u) by $\pi_{uv} = \alpha_{pq}(t, v)w_{uv}$ where w_{uv} is the weight of the edge and

$$\alpha_{pq}(t, v) = \begin{cases} \frac{1}{p} \text{ if } d_{tv} = 0 \\ 1 \text{ if } d_{tv} = 1 \\ \frac{1}{q} \text{ if } d_{tv} = 2 \end{cases}$$

with d_{ab} representing the shortest path distance from a to b.

Grover and Leskovec show that this algorithm performs better than previous methods on node classification due to its flexibility in learning a general representation of the node and its ability to explore both local neighborhoods and farther nodes through the random walk. This local and general trade-off is controlled by the parameters p and q. The parameter p determines how much of the sampled neighborhood will be close to the starting node. A small p value will create a BFS-like search during the random walk. On the other hand, q determines how far we allow the path to go away from the starting node. A small q value causes the random walk to behave like a DFS search.

3 Algorithms and Approaches

We implemented several different heuristics for decentralized search. In all of the approaches described below, if the destination node is a neighbor of the current node, the decentralized search will go there directly. Furthermore, we only consider unvisited neighbors of the current node (if all neighbors are visited, we choose a random neighbor instead).

3.1 Baseline

Our baseline algorithm for decentralized search is a random walk that avoids previously-visited nodes. At each step, we choose a random unvisited neighbor of the current node to visit next. Note that this encourages exploration because we avoid unnecessary backtracking.

3.2 Degree Heuristic

To improve the decentralized search, we consider the degrees of the current node's neighbors. Intuitively, nodes with higher degrees are more likely to be linked by a path to the target node. Therefore, we greedily choose the unvisited neighbor with the highest degree.

3.3 Similarity Heuristic

We also consider intrinsic information about the nodes when determining which node to visit next, which depends on the underlying meaning of each network. This node information provides a similarity score $\text{sim}(x, y)$ between any two nodes x and y in the network. Since nodes that have higher similarity scores should have a higher probability of being connected, we can perform a search using similarity scores. To choose the next node in the search, we can greedily choose an unvisited neighbor with the highest similarity score to the target.

We use several different ways of computing similarity. Thus, we normalize all similarity scores such that the minimum similarity score between any two distinct nodes is 0 and the maximum is 1.

In Sections 4.1 and 4.2, we describe how we compute similarity scores for the real-world networks in our experiments. We call these similarity measures "inherent similarity" because they are crafted using the real-world meaning of the graphs (i.e., people in our Facebook graphs and scientific papers in our citation network).

Since node2vec embeddings [3] encode structural information about the node while only using local information from random walks, we can utilize them as an alternative similarity metric (which can be useful in graphs that do not provide inherent similarity data). For each graph, we ran node2vec to obtain a 100-dimensional vector embedding of every node (100 dimensions was discovered to be optimal in the node2vec paper). We then use the embeddings as navigation coordinates. In other words, we can measure the similarity between two nodes by taking the negative l_2 norm of their embeddings. Thus, two nodes have high similarity (close to 1 after normalization) if their l_2 norm is small.

3.4 EVN

We also implemented EVN [6] to compare with successful past work. EVN greedily chooses the unvisited neighbor that maximizes the estimated probability of having a direct link to the target. If all neighbors are visited, one is chosen randomly.

A node s is linked to the target t with an estimated probability $p_{st} = 1 - (1 - q(\text{sim}(s, t)))^k$, where k is the degree of s, and q is an empirical estimate of a single (independent) edge occurring between s and t given their similarity score. To compute q, we find similarity scores between all pairs of nodes and discretize them into the regions

$[0, 0.01), [0.01, 0.02), \ldots, [0.99, 1)$, and $\{1\}$. Then, for each discretized similarity score S, we set $q(S)$ to be the fraction of edges existing among all node pairs (x, y) where $\text{sim}(x, y) = S$.

Note that EVN depends on the particular method of computing similarity. Hence, we test EVN using both similarity derived from the true meaning of our real-world networks (as described in Sections 4.1 and 4.2) and node2vec similarity (negative l_2 norm of embeddings).

3.5 Linear Regression Model

Our final algorithm for decentralized search (and our main contribution) relies on a weighted combination of different metrics. At each step in the decentralized search, we have a potential next node s (a neighbor of the current node), a target node t, and a set of visited nodes. We then create a feature vector containing:

1. Average shortest-path length in the graph (averaged over all pairs of nodes)
2. Similarity: $\text{sim}(s, t)$
3. Degree of s
4. Clustering coefficient of s
5. Whether s is visited (0 if visited, 1 otherwise)
6. Fraction of s's neighbors that are visited
7. EVN probability p_{st}
8. 1 (a constant term)

Using these feature vectors, we trained a linear regression model to predict the actual shortest-path distance from s to t. During the decentralized search, we greedily choose the unvisited neighbor that minimizes the shortest-path distance as predicted by the model.

Note that the first and last features listed above do not actually affect any paricular decentralized search, since those values are constant within a graph. However, we included these to aid the linear regression model. For instance, the average shortest-path length in the graph is provided to help the model distinguish between small and large graphs, so that the other features can be used to differentiate between short and long paths within a particular graph.

We did not include local betweenness centrality (LBC) [7] as a feature because it is almost always equivalent to choosing the highest degree neighbor. Since the degree of nodes is already included as a feature, we felt that LBC would be largely redundant.

Training the model

To obtain a training data point, we need a potential next node, a target node, and a reasonable set of previously-visited nodes. Hence, we use a random walk algorithm to generate training data that closely resembles the situations that would occur in an actual decentralized search. For each graph, we sample 10000 pairs of nodes (s, t). Then, for each sample, we simulate a walk from s to t as follows:

```
path = []
cur = s
add cur to path
while cur != t:
  if t is neighbor of cur:
    cur = t
  else:
    with probability 1/3:
      cur = random neighbor of cur
    otherwise:
      cur = neighbor of cur closest to t
          (choose among ties randomly)
  add cur to path
return path
```

If we are not one step away from the destination, we either choose a neighbor at random or choose a random neighbor among those closest to the destination. While we always want to choose the optimal node, the one-third probability

of randomly walking adds some noise to our data and makes the resulting path more similar to a real decentralized search path. In addition, the randomness helps us compute the features such as number of visited neighbors. (If we always chose the best neighbor, that neighbor would always have exactly one visited neighbor.)

Next, given a path from a source s to destination t representing a simulated decentralized search, we can compute features for a neighbor of a node on this path. First, we choose a node u along the path at random and choose a random neighbor v of u. We can then compute the feature vector of v given t and the visited nodes from s to u. This feature vector is used as input with the true shortest-path length from v to t as output to form one training pair. Given these training input and output pairs, we feed them into a linear regression model to compute the weights of the features.

We use Ridge regression, which adds an l_2 regularization term to ordinary least squares regression.[1] For each real-world graph, we trained two models: one using the intrinsic similarity and one using the node2vec similarity. These models are graph specific—they are trained only on training pairs extracted from that graph. Finally, we trained two general regression models, one with intrinsic similarity and one with node2vec similarity, that computes a set of weights that can be used for all graphs (trained on all training pairs from all real-world graphs). We refer to the general models as "Overall Ridge".

4 Experiments

We tested our various approaches on both synthetic graphs as well as real world datasets. Using built-in Snap functions, we generated an Erdos-Renyi graph, a Power Law graph with exponent 2.6, a Small World graph, and a Preferential Attachment graph, each with 1000 nodes. In addition, we evaluated our algorithms on two real world datasets: Facebook Social Circles [5], and a subset of the High Energy Physics Theory Citation Network (cit-HepTh) [2]. An overview of the networks is shown in Table 1.

We treated all of these networks as undirected graphs, and we only ran the algorithm on nodes in the largest connected component with self-edges removed. This allowed all of the sampled pairs of nodes to be connected so we could compare our average path length to the shortest path length. Furthermore, by eliminating the smaller components in the graph, we reduced the noise that results from pairs of nodes that can never be reached.

Table 1: Overview of Networks

Graph Name	Nodes	Edges	Nodes in Max CC	Edges in Max CC
Erdos-Renyi	1000	10000	1000	10000
Small World	1000	9961	1000	9961
Power Law	1000	1214	752	1074
Preferential Attachment	1000	9945	1000	9945
Facebook 0	333	2519	324	2514
Facebook 107	1034	26749	1034	26749
Facebook 1684	786	14024	775	14006
Facebook 1912	747	30025	744	30023
Facebook 3437	534	4813	532	4812
Facebook 348	224	3192	224	3192
Facebook 3980	333	2519	324	2514
Facebook 414	150	1693	148	1692
Citation Network	915	14739	913	14735

4.1 Facebook Social Circles

This dataset contains data from ten social circles on Facebook. A social circle is defined by an "ego" or central person. The nodes of the social circle are the friends of the ego, and the edges are the friendships between two nodes in the network. Note that the ego is not included in the graph. We only use the social circles with more than 100 nodes (there are eight such networks).

[1] Ridge regression outperforms ordinary least squares, LASSO, and Elastic Net regression.

Furthermore, each node in a circle has a set of features scraped from Facebook (location, work, political affiliation, etc.). The features are described as a 0-1 vector. For example, the 15th entry in the vector may represent whether a person is affiliated with the Democratic Party. Note that a node having a 1 in that spot indicates the person is part of the Democratic Party, but a node having a 0 in that spot does not necessarily mean the person is not part of the Democratic Party; the person instead may have not included that information on the profile.

We calculate similarity between two feature vectors using a standard cosine similarity measure, i.e., the similarity of vector $\mathbf{v}_1$ and $\mathbf{v}_2$ is given by $(\mathbf{v}_1 \cdot \mathbf{v}_2)/(\|\mathbf{v}_1\|_2 \cdot \mathbf{v}_2\|_2)$. We then normalize the similarities as described in Section 3.3.

4.2 High Energy Physics Theory Citation Network

This is a citation network of papers uploaded to the theoretical high-energy physics section of `arXiv.org`, a dataset originally from [2]. A subset of this network was previously used to evaluate EVN [6]. We wanted to mimic the graph in [6] for an easier comparison to the past work. Hence, we only included papers published from 1995 to 2000 (inclusive) with more than 50 non-self citations. Our graph contains 913 nodes and 14735 edges. This is slightly bigger than the graph in [6]; we believe this is due to slight differences in the removal of self-citations.

Our dataset also includes titles and abstracts for each paper, which we use to compute similarity between papers. For each paper, we concatenate the title twice and the abstract once and compute the resulting TF-IDF (term frequency-inverse document frequency) vectors. We measure the similarity between two papers with cosine similarity of the TF-IDF vectors (normalized).

4.3 Experimental Setup

For each graph, we randomly generated 10000 tasks (a source and destination node pair) and computed the average path length for each algorithm on every task. (These tasks were generated independently from the training data.) In addition, we define the notion of a successful search as one whose search path length is at most 100. Hence, we also report the success rate for each algorithm.

5 Results and Discussion

Results of the various methods are presented and analyzed below.

5.1 Synthetic Networks

Table 2: Results on Synthetic Networks

	Erdos-Reyni		Small World		Power Law		Pref. Attach.	
Strategy	Succ.	Avg. Len.	Succ.	Avg. Len.	Succ.	Avg. Len.	Succ.	Avg. Len.
Random	86.57%	34.58	83.23%	36.64	36.38%	36.65	92.86%	27.86
Highest Degree	93.27%	29.00	91.49%	32.51	54.46%	21.76	99.81%	10.78
node2vec	**100%**	3.47	**100%**	3.21	**99.99%**	5.37	**100%**	3.48
EVN (node2vec)	**100%**	**3.35**	**100%**	**3.20**	98.71%	9.84	**100%**	4.01
Overall Ridge (n2v)	**100%**	3.38	**100%**	3.21	99.69%	**5.32**	**100%**	**3.39**
Optimal	100%	2.63	100%	2.70	100%	4.91	100%	2.54

In Table 5.1, we list the performance of various methods on each of the synthetic graphs. Note that we did not use methods involving similarity inherent to the graph, since these graphs do not represent any real-world objects that we can compare. We are, however, able to use node2vec as a similarity metric, which also allows us to apply EVN to these networks (which was previously impossible).

As expected, randomly going to an unvisited node produces the longest path lengths. The highest degree strategy is slightly better than the baseline on the Erdos-Renyi graph and Small World graph, but performs significantly better (in terms of both success rate and average path length) than the baseline on the Power Law graph and Preferential Attachment graph. This is due to the degree distributions of the networks. As shown in Figure 1, most of the nodes in the Erdos-Renyi and Small World networks have more or less the same degree, so a node with relatively higher degree

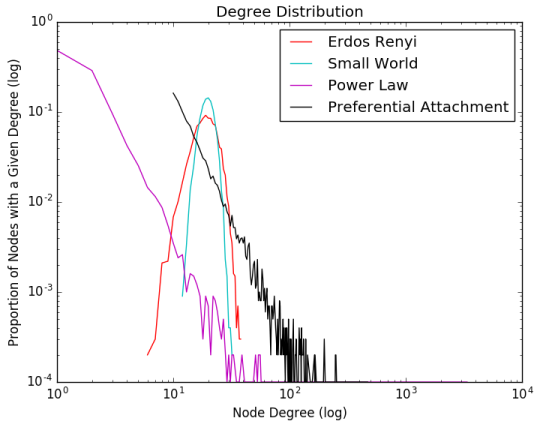

Figure 1: Degree distribution in synthetic networks.

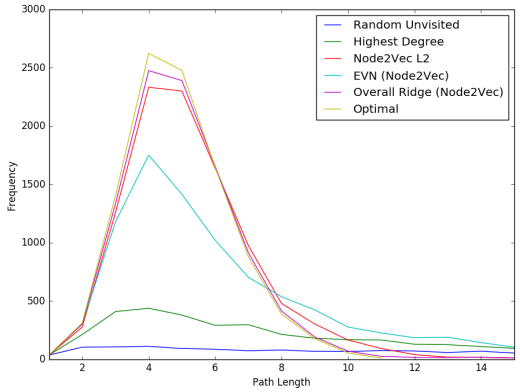

Figure 2: Path length distribution for power law network.

Table 3: Results of Various Strategies on Real-World Networks

	Facebook Circles		Citation Network	
Search method	Success Rate	Avg. Path Len.	Success Rate	Avg. Path Len.
Random	74.93%	24.21	88.25%	25.61
Highest Degree	75.96%	14.60	93.69%	10.137
Similarity	88.68%	19.38	**99.24%**	7.13
EVN (Similarity)	83.45%	16.13	99.1%	**5.85**
Ridge Reg. (Similarity)	87.47%	14.98	98.94%	6.98
Overall Ridge (Similarity)	**92.72%**	**14.48**	98.85%	9.49
node2vec	98.58%	8.37	**100%**	3.95
EVN (node2vec)	96.33%	8.92	99.68%	3.45
Ridge Reg. (node2vec)	97.70%	8.43	99.67%	3.42
Overall Ridge (node2vec)	**98.83%**	**7.29**	99.85%	**3.29**
Optimal	100%	2.92	100%	2.57

does not necessarily imply that it is well-connected to the rest of the graph. On the other hand, in the Power Law and Preferential Attachment networks, most nodes have very low degree and a few of them have extremely high degree. This means that moving to a high degree node gives us access to many other nodes, leading to greatly-improved performance.

Finally, using node2vec embeddings greatly reduces the average path length and maintains a very high success rate for all four synthetic networks. This suggests that the l_2 norm of the embeddings do indeed provide a good indication of similarity between two nodes. By running node2vec, we are able to learn features of the graph itself and construct our own similarity metric, achieving results that are surprisingly close to optimal. Furthermore, the EVN and Ridge regressions that utilize node2vec are able to decrease the average path lengths a bit more.

Figure 2 plots the distributions of path lengths in the Power Law synthetic network. From this visualization, we see that Overall Ridge is extremely close to optimal, with node2vec itself (without other features) close behind.

5.2 Real-World Networks

Table 3 shows the results for the real-world networks (Facebook circles and the citation network). The top section of the table lists methods that do not use node2vec (instead using similarity inherent to the graphs, as discussed in Section 4.1 and 4.2). The bottom section lists methods that have this inherent similarity replaced with node2vec l_2 norm.

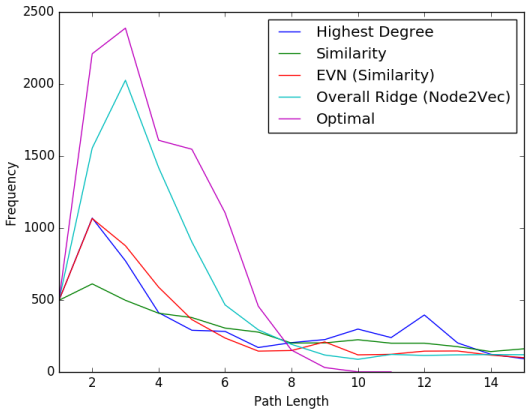

Figure 3: Path lengths in the Facebook-0 network.

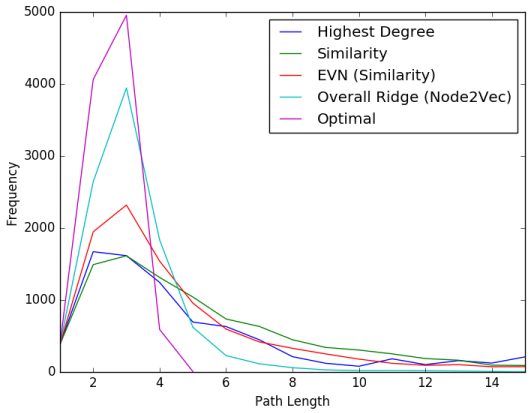

Figure 4: Path lengths in the citation network.

The degree heuristic was better than the similarity heuristic for Facebook circles, but the similarity heuristic was better for the citation network. However, in each case, algorithms using more information performed even better (Overall Ridge for Facebook circles and EVN for the citation network).

We were also able to compare node2vec with the similarity inherent to the graphs. In the real networks, we see that using a node2vec heuristic alone performs significantly better than any other method that only uses inherent similarity. We found this to be quite surprising—node2vec, an algorithm that is oblivious to the real-world objects that nodes actually represent, produces a better similarity metric than hand-tuned similarity metrics drawn from the real meaning of the graph. In every method that uses similarity in some way, replacing the inherent similarity with node2vec results in much lower path lengths and higher success rate. Finally, we observe that the best method is our Overall Ridge regression, using node2vec l_2 norm as the similarity metric. It is interesting to note that the model produced by training on all graphs simultaneously (i.e., Overall Ridge) outperforms the models trained on each graph separately (i.e., Ridge regression).

Figures 3 and 4 provide a visual representation of the data in Table 3, showing the distribution of path lengths for selected methods. These graphs highlight the superior performance of node2vec methods (we only plot Overall Ridge since the four node2vec methods are quite similar) over previous methods that used other forms of similarity. In fact, the Overall Ridge regression trained with node2vec similarity has near-optimal performance.

5.3 Linear Regression Weights

Table 4: Overall Ridge Regression Weights

Feature	Inherent-Similarity Weights	node2vec Weights
Avg. shortest-path length	$+0.430$	$+0.478$
Similarity	-0.605	-1.228
Degree	-0.000558	-0.000198
Clustering coefficient	$+0.0715$	$+0.0209$
0 if visited, 1 otherwise	-0.102	-0.0492
Fraction of visited neighbors	$+0.446$	$+0.541$
EVN probability	-0.400	-0.311
1 (constant term)	$+1.954$	$+2.004$

Table 4 shows the weights learned by the Overall Ridge regressions (one using inherent similarity and one using node2vec similarity instead). Note that the signs of the weights are consistent with intuition. For example, higher similarity generally leads to shorter paths, so the learned weight for similarity is negative. Interestingly, going to a node with high clustering coefficient generally leads to longer paths, since the neighbors of such a node form a tight-knit community that might be difficult to escape. Likewise, a node with low clustering coefficient is more likely to bridge different communities, therefore aiding the decentralized search.

Since the signs of the weights follow natural intuitions, we examine the magnitudes to find the most important features. The magnitude of the degree weights is very small because we did not normalize the degree feature, whereas the other features (except the average shortest-path length) are all between 0 and 1. In particular, we observe that the node2vec model approximately doubles the weight of the similarity feature, compared to the inherent-similarity model. To compensate, the node2vec model puts significantly less weight on the degree, clustering coefficient, and "is visited" features. This further supports our findings that node2vec is a better similarity measure, at least in the context of decentralized search.

We can also use the weights to infer the most important features for decentralized search. While the magnitude for the "fraction of visited neighbors" is higher than that for the EVN probability, we note that the fraction of visited neighbors is generally very small, whereas the EVN probability ranges more uniformly across the range $[0, 1]$. Furthermore, note that neither the average shortest-path length nor the constant term will affect any individual decentralized search task, since they are constant within graphs. Hence, the most important features for decentralized search appear to be similarity, degree, and EVN probability.

6 Conclusions and Future Work

In conclusion, we have found that using many different kinds of locally-available information can improve upon previous algorithms for decentralized search. We furthermore found that node2vec is a surprisingly good similarity metric for decentralized search, better than other inherent similarity metrics drawn from the true meaning of the graph. In general, our linear regression model (trained on all graphs simultaneously, using node2vec as the similarity metric) produces the best results out of all the methods tested.

We have explored various regression models such as ordinary least squares, LASSO, Elastic Net, and Ridge. However, one can continue to explore other machine learning algorithms to predict path length. In particular, reinforcement learning techniques could address a limitation of our supervised learning approach: the paths used to generate the visited nodes for the training instances are not exactly the same as the paths used by the actual decentralized search algorithm. The reinforcement learning framework would eliminate this problem by using the actual decentralized search paths as part of the training data.

We began exploration for neural network strategies by feeding our network the same features as the linear regression model, which allows for learning non-linear relationships between features. We used a two-layer neural network with 100 neurons and 20 neurons respectively. Then we trained on 9000 data points and validated with another 1000 using data generated from the 10 Facebook social circle graphs. However, since training time took nearly an hour and running simulations took another hour, we felt that this model would not be practical in real scenarios since decentralized search is meant to be a computationally-inexpensive algorithm. Our preliminary results showed that the neural network strategy performed slightly better than the other node2vec strategies, but worse than the inherent-similarity strategies (when trained without node2vec). Further exploration into different architectures or hyperparameters may lead to better results and faster runtime.

References

[1] L. A. Adamic, R. M. Lukose, A. R. Puniyani, and B. A. Huberman. Search in power-law networks. *Physical review E*, 64(4):046135, 2001.

[2] D. S. Berman and M. K. Parikh. Confinement and the ads/cft correspondence. *Phys.Lett.*, B483:271–276, 2000.

[3] A. Grover and J. Leskovec. Node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 855–864. ACM, 2016.

[4] J. M. Kleinberg. Navigation in a small world. *Nature*, 406(6798):845–845, 2000.

[5] J. McAuley and J. Leskovec. Learning to discover social circles in ego networks. *NIPS*, 2012.

[6] O. Simsek and D. Jensen. Decentralized search in networks using homophily and degree disparity. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, IJCAI'05, pages 304–310. Morgan Kaufmann Publishers Inc., 2005.

[7] H. P. Thadakamalla, R. Albert, and S. R. Kumara. Search in weighted complex networks. *Physical Review E*, 72(6), 2005.