Question Recommendation On the Stack Overflow Network

Jacob Perricone
Institute of Computational and
Mathematical Engineering
Stanford University
Email: jacobp2@stanford.edu

***Abstract*—The use of Question-Answering Communities in software engineering has skyrocketed in recent years, with the number of posts on Stack Overflow alone nearing 37 million. Stack Overflow allows users to post questions to a large developer community, nearly 8 million users, and receive answers or advice about the issue the user is encountering. As the number of answered questions grows, Stack Overflow is not only useful to the user posting a question but to any developer interested in gathering information. With an ever-increasing network of information, it is important to devise algorithms able to recommend relevant questions (or even answers) to a user. Typically recommendation systems employ some form of collaborative filtering coupled with latent factor models and content analysis. This project utilizes the network structure of Stack Overflow to recommend a set of related questions for a given input question. In particular, this paper employs a modified guided Personalized Page Rank algorithm to generate candidate recommendations and compares the results to those recommended by Stack Overflow. Semantic similarity and tag-overlap are used to asses candidate recommendations.**

***Index Terms*—Recommendation Systems, Stack Overflow, Personalized Page Rank**

I. Introduction

Stack Overflow represents a vast wealth of curated answers to commonly encountered software engineering questions. Indeed, combing through the Stack Overflow archives in search for answers or relevant information is common-place when encountering a bug or exploring a new library. The emergence of Stack Overflow testifies to the push for crowd-sourced developer environments. Often, when writing a piece of software, it is likely that components of the software have already been written. As the amount of software grows, it will become increasingly inefficient to spend time rewriting code that has already been developed. Creating a collaborative development environment where code is recommended for a user given their history of projects and the task at hand is paramount in increasing developer efficiency.

Recommendation systems are present in most of the services we use, ranging from movie recommendation on Netflix, job recommendation on LinkedIn, product recommendation on Amazon, or users on social media. In fact, Stack Overflow even offers a list of related posts for a given question. Despite the prevalence of recommendation systems in every day life, code recommendation systems are not common, likely due to the difficulty of assessing intention and relevance. Nevertheless, the first step in creating a recommendation system, which is integrated into both the user's IDE and the developer community at scale, is to analyze the recommendation system on the largest open-source software network available.

This project utilizes the network structure of Stack Overflow to recommend for a given question a set of related questions. One difficulty that must be addressed is how to determine the quality of the recommendation, since there is no labeled dataset. Real world systems often approximate recommendation quality by examining user interaction with the recommendation. In the off-line setting, however, assessing recommendation quality requires an ad-hoc metrics of "relevance" which may not accurately capture quality.

Lastly, this project focuses on the utility of network algorithms in the recommendation tasks, which places constrictions on the input to the recommendation algorithm (i.e. a node within the network). Recommending questions given only a natural language query is an additional task that is not addressed in this paper.

A. Literature Review

Recommending relevant questions given an input question and user touches on two areas of research: natural language process and network recommendation algorithms. If the input question is taken to be a node in the graph, the question can be modeled as a recommendation algorithm; whereas, if the input query is natural language, inference techniques from natural language processing must be used.

Recommendation problems on networks can be analyzed using variations of the canonical Page Rank algorithm. The Page Rank Algorithm assigns an importance score to each node in the network. However, Page Rank is restrictive in that it initially assigns equal weighting to all nodes and edges in the network. In effect, it is useful as a global measure of relevance but not helpful for finding nodes pertaining to a given source node. Personalized Page Rank (PPR) attempts rectify this short-coming by ranking nodes not just by global popularity but by relevance to a given set of source nodes. A common interpretation of PPR is to take a random walk around the graph, restarting randomly, and rank nodes by proximity [1] [2]. Variations on Personalized Page Rank are leveraged for recommendation in large-scale system such as the Pinterest graph [3]. The system used at Pinterest to recommend pins for

a board given a set of input pins in part utilizes a real time random walk service Pixie [4]. Pixie conducts many random walks on a bipartite graph of pins and boards and aggregates pin counts at each step, which is effectively computes Personalized PageRank on the graph with a given query pin. Pixie has shown to be quite effective at recommendation since it is able extract highly connected pins while as well retaining good coverage of rare pins. A combination of Pixie candidates, heuristic candidates and session co-occurence candidates is then used to augment the recommendation system.

In Q&A communities, there has been a great deal of research on how to recommend to expert users questions they would like to answer. Mathur et al [5] analyzes a restricted set of the Stack Exchange math community, using network features learned through node2vec [6] as inputs for a machine learning algorithm to rank users for a given question. They achieve high quality results, with the correct user being in the top 2 suggestions over 75% of the time. Utilizing network embeddings is a novel and insightful aspect of their work. Gideon et all [7] proposes a recommendation system for Yahoo Answers that combines features extracted from the question's text and user-user interactions to create a score for each user. Impressively, their model is able to capture the underlying state of the question through time and assess the propensity of a user to answer a question in a given state.

Within the Stack Overflow network, additional research has been conducted on Tag Recommendation for a given input question and user. The most relevant algorithm in the tag-recommendation sphere is *NetTagCombine*, which expands upon previous work tagging Software information sites by incorporating network structure into their tag prediction algorithm [8] [9] [8]. Short et al. analyzes three graphs within the Stack Overflow network: a network based on semantic post-similarity created by adding edges between questions if the cosine similarity of the two questions' tf-idf vectors are greater than a set threshold, a network based on user to user interactions ($u \rightarrow v$ if u answers v questions), and a bipartite graph linking users to tags assigning edge weights based on whether the user asked, posted, or commented on the question. Short et al. then implements the *TagCombine* algorithm, which consists of a multi-label learning algorithm, a similarity based ranking component (finds similar post, accumulates the tags within them, assigns probabilities to the tags through empirical likelihood), and a co-occurrence weighting between the words in the question and the words associated with a tag. They then improve the algorithm by leveraging the bipartite graph between users and tags to assign tag i a score proportional to the weighted sum of the edges between the users who interacted with the post and the tag i. In addition they use the `BIGCLAM` algorithm to detect communities in the question to question network and allow the top 50 most similar posts in each community to contribute their tag representatives to the likelihoods of assigning tags to the new question. Their research demonstrate a significant improvement in prediction when utilizing network structure. Utilizing the bipartite graph between users and tags and forming communities of questions is resourceful.

II. Data

A. Data Collection & Attribution

The data used for this project is primarily provided by the StackExchange data-dump [10], though additional data was gathered by scraping the Stack Overflow website. From the StackExchange data-dump, this project uses only data provided for Stack Overflow, in particular the tables `Posts.xml`, `Users.xml`, `PostLinks.xml`, `Comments.xml`. Additional data on Stack Overflow's "related questions" for a given post were mined from the Stack Overflow website itself.

B. Data Processing

Six SQL tables were created form the raw-data dump file: **Questions, Answers, Users, PostLinks, RelatedLinks, Tags**. Since the ultimate goal is to improve the granularity of recommendation, this project restricts the Stack Overflow Posts to only those posts relevant to questions pertaining to python. This involved a number of processing steps detailed below.

The **Tag** table contains columns *TagId, TagName, Count* and was parsed directly from the XML file. The post data provided by the data-dump is a 56-gigabyte xml file of all the posts, including both questions and answers[1] on site. The **Question** table contains the columns: *Id, AcceptedAnswerId, CreationDate, Text, Code, Score, ViewCount, Title, Tags, AnswerCount, FavoriteCount, OwnerUserId, CommentCount*. A post was added to the **Question** table if a) the post was a question and b) any version of the string `python` could be found in *Tags, Body* or *Title*. Next the data-dump attribute *Body*, containing the raw HTML of the post, was parsed into *Text* and *Code* columns. The total number of questions accumulated in the first step was $\sim 800,000$. Now, to take into account that some posts pertaining to python are incorrectly mislabeled by only a subcategory[2], the graph was further augmented by fetching all linked questions from the `PostLinks.xml` file[3]. Finally, to ensure the validity of the processing and that all questions pertaining to `python` posts are incorporated into the graph, for each question q in the **Question** database, the list of recommendated questions was scraped from the Stack Overflow web-page of q and stored them in a **relatedlinks** table[4]. The related links serve as our baseline for comparison in the ranking algorithms discussed below. Since the scale of this scraping procedure was massive, on the order of 800,000 questions, the procedure employed multi-threaded calls to AWS lambda functions to extract the relevant information in parallel.

[1]For the purposes of this project Posts denote Questions or Answers

[2]For example, questions pertaining to Django, a python server-side framework, often forgot the parent tag

[3]The postLinks table has two types of links, duplicates and linked. An edge from p_i to p_j is of type link if p_j is explicitly referred to by a user in p_i

[4]**relatedlinks** table has columns origin *PostId RelatedPostId*

After taking the union of the ids listed in **relevantlinks** table, the total number of questions was $\sim 1,100,000$. Using the ids of the **Question** table, an **Answer** table was created that contains all of the answers associated with the questions in the **Question** table. Finally, a **User** table that stores information on all users within both the **Question** and **Answer** tables was created. The total number of answers is $2,226,000$; the total number of unique users in the question and answer table is $\sim 620,000$; the total number of distinct tags for python-related questions is $23,160$.

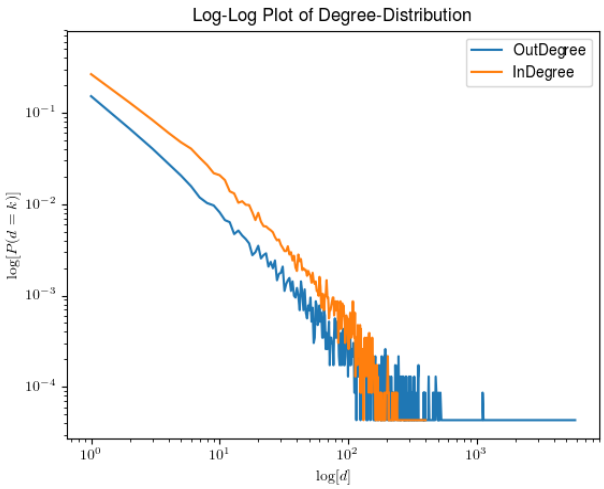
III. Network Analysis

A variety of graphs were created and explored from the Stack Overflow data. Not all of them proved useful in recommendation, but nonetheless the exploration provides insight into the Stack Overflow network. In particular, this paper explores the $Tag \rightarrow Tag$ graph, $\mathbb{TT}$, the bipartite $Question \rightarrow Tag$ graph, $\mathbb{QT}$, the $User \rightarrow Tag$ graph, $\mathbb{UT}$, and the four-way $User \rightarrow Tag \rightarrow Question$ graph, $\mathbb{UTQ}$.

A. Tag-Tag Network

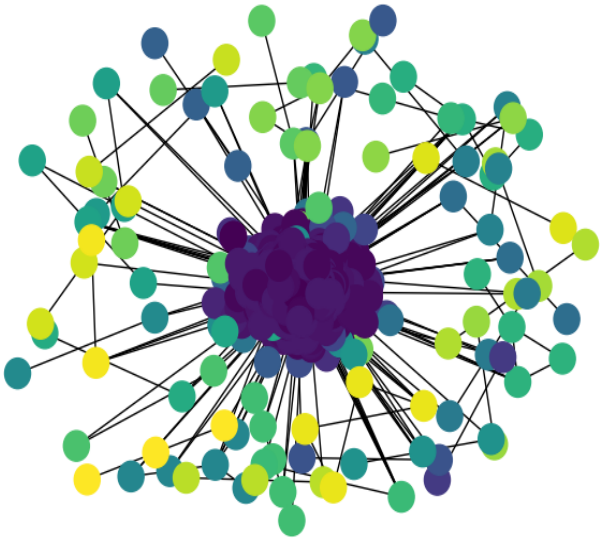
A common way to query software sites is using tags. The initial approach to recommending questions relied upon utilizing the implicit hierarchy in tags to restrict the search space. Tags on Stack Overflow are not intrinsically hierarchical though they retain an implicit structure; that is, if a question pertains to the pandas package in python, it is most frequently given the tags `<python><pandas>`. Therefore, there is a loose ordering of tags such that those that come first represent broader categories than those succeeding it. The $\mathbb{TT}$ is constructed as follows: let t_i denote the ith tag. The tag-tag $\mathbb{TT}$ network contains an edge from $t_i \rightarrow t_j$ if t_j succeeds t_i in ordering for some question q_k. There are 23116 nodes and $290,408$ edges. The initial impetus of exploiting the tag-tag network was to create a tag-tree to effectively group questions by their position in the tree. However since the ordering is not strict, creating a tag-hierarchy proved uninformative. After examining the properties of the tag-tag network, it was clear that the $\mathbb{TT}$ network was a multi-edge digraph with many cycles. The plot below shows the in-degree, out-degree, and total-degree distributions of the tag-tag network on a log-log

Fig. 1. Degree Distribution of $\mathbb{TT}$ network



As seen above, the in and out degree distribution of the Tag-Tag network appear to follow a power-law. Moreover, one can notice that the out-degree distribution has much fatter tails than the in-degree. Nevertheless, there are still many nodes that have in-degree over 100, demonstrating the difficulty of unraveling the network into a tree. In hopes to find a way to partition the question-tag space, the communities within to the tag-tag network[5] in order to determine whether the space could be effectively partitioned. The plot below shows the result of running the Louvian Method for Community detection on the $\mathbb{TT}$ network.

Fig. 2. Communities within the $\mathbb{TT}$ network



In total 60 communities were found, but only 5 of them contain more that 1% of the network. In fact, the top community

[5]Casting the TT graph as a weighted undirected network

contains around 27% of the nodes, and the top 5 communities contain nearly the entire graph. The modularity of the partition is only around .30. Therefore, we are only able to cluster the tag space into a few meaningful groups and definitely not able to partition the graph into a well structured tree.

B. Question-Tag Network

Since the Question-Tag network is the first graph upon which the random walk algorithm will be tested, it is pertinent to examine its network characteristics. The Question-Tag network, $\mathbb{QT}$, is an undirected bi-partite graph from questions to tags, where q_i is linked to t_j if q_i contains t_j. There $\mathbb{QT}$ network is quite large, with $800,000$ nodes and 3 million edges. It is evident by construction that there will be tags that are linked to nearly all the questions of the graph, i.e. the tag `<python>`. Furthermore, it is expected that the degree distribution of questions to be very contained, i.e. have small support, since questions are rarely linked to a massive number of tags. Indeed, examining the plots below, we can see that there exists tags for which hundreds of thousands of questions are linked. The network characteristics above immediately suggest that the vanilla personalized page-rank algorithm on a bi-partite graph of questions and tags may have trouble achieving granularity in results. The reasoning is that there exists tags to which nearly all of the questions are linked. Therefore taking a step to one of these high-degree tags from a question node will broaden the search too widely. Indeed the semi-hierarchical structure of the tags themselves do not lend themselves well to vanilla PPR algorithm. Therefore, in order to recommend related questions, it is important to weight the edges of the graph and modify the page rank algorithm. The methodology used will be discussed in the next section.

Fig. 3. Degree Distribution of Tags within $\mathbb{TQ}$ network

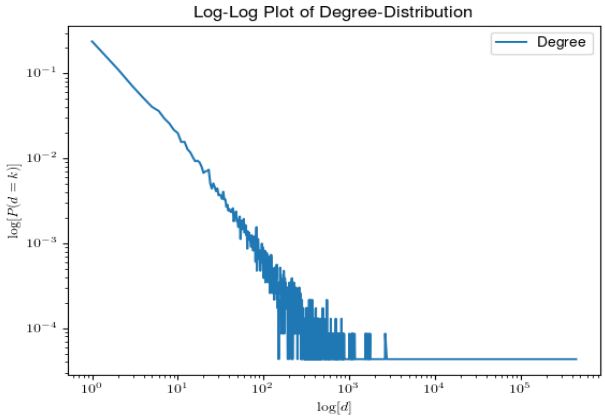
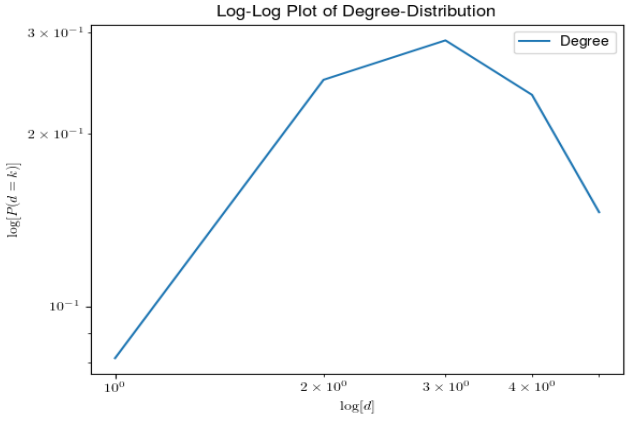


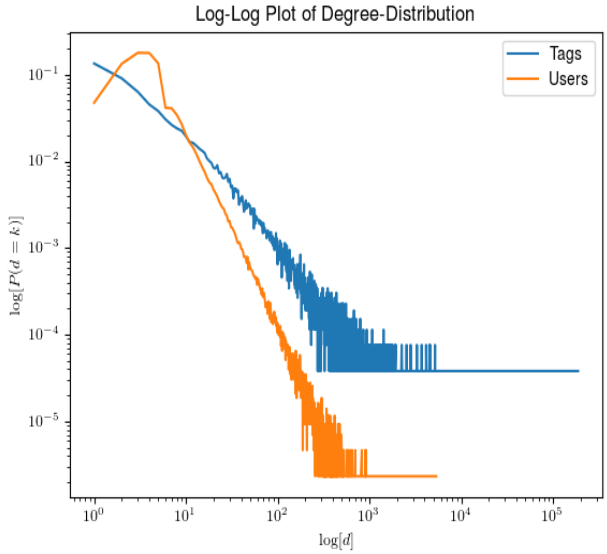
Fig. 4. Degree Distribution of Questions within $\mathbb{TQ}$ network



C. User-Tag Network

The User-Tag network,$\mathbb{UT}$, is a bipartite network between users and tags where an edge from u_i to tag t_j is drawn if u_i answers some question q_k where $t_j \in \text{Tags}[q_k]$. This graph has $|V| = 450,521$ and $|E| = 3,690,086$, which is much larger than the $\mathbb{TT}$, network. The impetus behind examining this graph is that if we can locate a set of users that are authorities for a group of tags, we can prioritize questions that are answered by those users in a random-walk. Furthermore, it is of value to determine whether users typically answer questions in a relatively small subset of the tag-space.

Fig. 5. Degree Distribution of $\mathbb{UT}$ network



Examining the figure above, notice that the degree-distribution of users decays much more quickly than the tag distribution. The graph shows that the majority of users answer questions within a relatively small subset of the tag-space, with a spike in density at around 6. Those users that answer questions over a large tag-domain, greater than 300, on average answer more than 3 times as many questions as those users confined within the tag-space.

In order to create some structure for the massive $\mathbb{UT}$ network, Louvian community detection was applied to the graph. A total of 59 communities were found, with 12 communities containing over 1% of the total nodes in the network. The top 5 communities contain around 70% of all of the nodes

and the partition achieved a modularity score of 42%. The results indicate that we are able to create structure within the user-tag space. This is of value since one can utilize the network structure to restrict the search space. If user u_i is within a community that is commonly associated with t_j, we can weight more heavily questions that contain t_j. For example, community c_0 has mainly tags associated with lower level systems engineering, i.e. questions with tags such as `linux, binary, declarative-pipeline`. Given a question with tags in c_0, one could recommend the questions answered by users of c_0, since they are strongly associated in the user-tag space.

IV. Evaluation

The goal is to recommend related questions given an input question. In order to quantitatively assess the quality of the recommendations, it is first necessary to devise metrics of similarity. The primary four metrics used in this paper are Title Similarity, Text Similarity, and Tag Overlap. Recall that given an input question $\tilde{q}$, the algorithm returns a set of recommended questions Q_r.

A. Word Embeddings

Given $\tilde{q}$ and Q_r, there are a variety of methods to compute semantic similarity between the text in $\tilde{q}$ and the text in the questions within Q_r. A baseline method uses a frequency based approach and assigns each document within Q_r a score based on the cosine similarity of the TF-IDF vector for q_j and the TF-IDF vector of $\tilde{q}$, where the TF-IDF matrix is computed on $\tilde{q} \cup Q_r$[6]. However this approach does not take into account the larger corpus of questions within the Stack Overflow and yields quite restrictive representations of words.

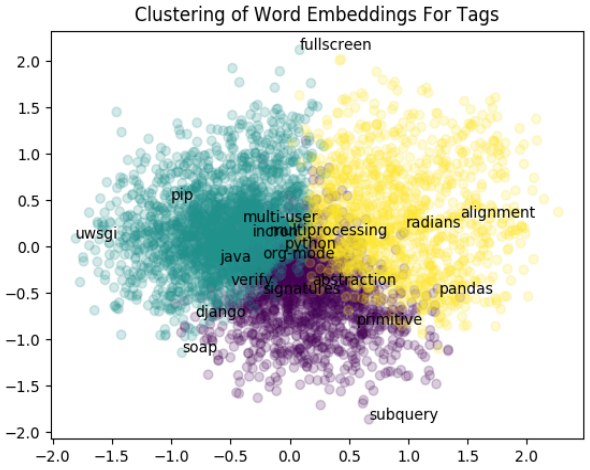
Word embeddings provide a mapping from natural language to a rich real-valued vector space and are commonly used in natural language processing techniques. The `word2vec` algorithm maps a word to a fixed-length, dense and real-valued vector by training a neural network to predict a target word given the context words[7], learning the word representations in the process. Google has pre-trained word2vec embeddings available for download, however, many of the hyper-specific programming words are not contained within the embedding space. Therefore, using the data collected from Stack Overflow, three sets of word embeddings were trained using gensim's word2vec implementation [11]. One set of embeddings trained on only the titles, another on only the text, and lastly one where each document was composed of the concatenation of the title and body. The embeddings using the concatenation of title and body were used to score the results.

These embeddings give a mechanism to cluster questions by semantic meaning. For example the plot below shows the result of a PCA projection of the word embeddings for all tags within the dataset. The colors represent clusters found using `K-Means` on the word embedding space with $k = 3$.

[6]Term-Frequency Inverse Document Frequency.

[7]This is the CBOW implementation of `Word2vec|`

Fig. 6. Word Embedding Clustering



For a given question q_i, the average of the word embeddings within q_i was used to represent the document. The semantic similarity between q_i and q_j was approximated by the cosine similarity of q_i and q_j's vector representations. Formally, let t_i,b_i denote the vector representation of the ith questions title and body, respectively. For both the title and the text body

$$S_{title} = \frac{\langle t_i, t_j \rangle}{||t_i||||t_j||}$$
$$S_{body} = \frac{\langle b_i, b_j \rangle}{||b_i||||b_j||}$$

The tag overlap between $\tilde{q}$ and the questions in Q_r was also calculated. Let $T(q)$ be the tags generated by a question q. For a given recommendation question q_r, the tag overlap is defined as

$$O_T = \frac{T(q) \cap T(q_r)}{T(q) \cup T(q_r)}$$

The metrics above are averaged across the q_r in the recommendation set. Let $\bar{O}, \bar{S}_{title}, \bar{S}_{body}$ denote the average scores of the recommendation sets Q_r^i for a set of experiments $i \ldots, N$. These metrics are then compared to the average scores of Stack Overflow's recommended questions. [8].

V. Experiments

All algorithms were coded in NetworkX python library. Computationally expensive tasks or obviously parallel tasks were performed on an AWS server with 32 cores. The experiments were run for $10,000$ starting nodes within the $\mathbb{QT}$ network of size $|V| \sim 800,000, |E| \sim 3$ million. For a given recommendation set, the average text, title, and tag-overlap scores were calculated. These scores were then averaged across all trials of the experiment to yield a final score. Unless otherwise stated, the restart parameter of Personalized Page

[8]These recommended questions are in the **relatedlinks** table mined from the Stack Overflow site

rank was set to $\alpha = .3$. All variations of the standard PPR run for 10000 iterations or until the $10th$ recommended question has a count greater than 30.

A. Vanilla Random Walk with Restarts

The first algorithm implements the standard Personalized Page Rank algorithm on the $\mathbb{QT}$ graph. It follows closely the Pinterest algorithm introduced in Jure Leskovec's [1]. The pseudocode for the algorithm is shown below:

```
INPUT: queryQuestionsSet, G, NUMSTEPS, alpha
queryQuestion = queryQuestionsSet.random()
for i in range(NUMSTEPS):
  tagnode = getRandomTag(queryQuestion)
  queryQuestion = getRandomQuestion(tagnode)
  count[queryQuestion] += 1
  if random.rand() <= alpha:
    queryQuestion = queryQuestionsSet.random()
```

Starting at an input question, it takes a random walk to a tag-node, and then randomly walks to the following question.

Recalling Figure 2, one can see where this algorithm will fail. If from a given question, each tag-node is equally likely, often the algorithm will choose a tag that is far too broad. For example if the input question contains the tags `<python> <django> <django-models> <sql>`, with probability $\frac{1}{4}$, the algorithm will jump to `<python>`, which links to nearly a million questions! Indeed, the algorithm should have some way to restrict its search. Otherwise, if the algorithm does not take into account the hierarchical ordering of tags, it will with high probability jump to an incredibly general tag, thereby expanding the random walk space and reducing the specificity of results. It is as if there exists a Pinterest board that encompassed all of the pins. Another issue also stemming from the implicit hierarchical behavior is that the ordering of the tags matters. Say, for example the algorithm jump to the tag node `<sql>`. `<sql>` is linked to a huge body of questions that do not at all pertain to `<django>`. Similarly `<django>` is linked to a huge number of questions that do not contain `<sql>`. In theory, the overlapping links for questions with both `<django> <sql>` should cause the pertinent questions to rank higher, but there is a high probability that this will not happen [9]. The issue stems from the out-degree of the tag-nodes. If there is no way to prioritize question nodes linked to both sql and django, it is difficult to restrict the search space. The results detail the average Tag-OverLap score, title semantic-similarity score, and text semantic similarity score of the Naive PPR method run on 10000 random starting nodes. The average semantic similarity for both the text and question were calculated using the word embeddings described in the previous section. I also report the average standard deviation of the scores across each recommended set. [10]

[9]due to the number of questions linked to both tag groups

[10]Starting nodes were processed in parallel. Due to the size of the graphs the hyperparameter α was not optimized for

TABLE I
RANDOM WALK W/ RESTART ALGORITHM $\alpha = .3$

Relevance Metric	PPR	Stack Overflow Baseline
$\bar{O}_T$	0.5361	0.6509
$\bar{S}_{title}$	0.5205	0.7382
$\bar{S}_{text}$	0.7397	.8184
$\sigma_{\bar{S}_{title}}$	0.0665	0.0449
$\sigma_{\bar{S}_{text}}$	0.0273	0.0182

As shown above the vanilla PPR algorithm under-performs as compared to the Stack Overflow "related" questions across all categories. This is likely attributed to the fact that some tags have very high degree, thereby expanding the search space of the algorithm too widely. Examining this further, the experiment was rerun restricting the $\mathbb{QT}$ to those questions with more than 2 tags (i.e. questions tagged more specifically). The idea here is that there is a greater likelihood of the algorithm jumping to tags with higher specificity. The results are shown below.

TABLE II
RANDOM WALK W/ RESTART ALGORITHM ON RESTRICTED $\mathbb{QT}$ GRAPH $\alpha = .3$

Relevance Metric	PPR	Stack Overflow Baseline
$\bar{O}_T$	0.5425	0.6109
$\bar{S}_{title}$	0.674971	0.7199
$\bar{S}_{text}$	0.79680	.8232
$\sigma_{\bar{S}_{title}}$	0.0856	0.0624
$\sigma_{\bar{S}_{text}}$	0.0246	0.0209

The results returned by PPR, though improved, are still too general. This is likely attributed to the fact the PPR algorithm jumps to all tags with equal probability, thereby ignoring the loose ordering of tag specificity.

B. Guided Random Walk with Restart

In order to restrict the search space, a modified weighted Personalized Page Rank algorithm was adapted. In particular, given a question q_i, the probability of transitioning to a tag-node t_j is inversely proportional to the out-degree of t_j. This makes the event of transitioning to a tag linked to a massive number of questions more unlikely. Furthermore, I weighted the edges based on their ordering within the question. The ultimate tag had the highest weighting, followed by the penultimate tag, and so forth. To prevent the algorithm from overly-weight tags with the fewest number of linked questions, a maximum probability of a transition was set to be equal to .8. Formally,

$$\mathbb{P}[q_i \to t_j] \propto \max\left(c\frac{r^i_{t_j}}{d_{t_j}}, .8\right)$$

where $r^i_{t_j}$ is the rank of tag j in question i, d_{t_j} is the degree of tag j and c is a normalizing constant.

The next modification made assigns a higher probability of traveling to a question-node from a tag-node if the question node weights the tag similarly to the source question-node. Formally, given that algorithm jumps from $q_i \to t_j$ with probability p_{ij}, the algorithm assigns a higher probability to

the transition $t_j \to q_k$ for nodes q_k that have p_{kj} close to p_{ij}. Formally,

$$\mathbb{P}[t_j \to q_k] \propto c\frac{F_k + V_k}{|\mathbb{P}[q_k \to t_j] - \mathbb{P}[q_i \to t_j| + \epsilon]}$$

where c is a normalizing constant and F_k, V_k are the favorite count and view count respectively. Intuitively, this is guiding the algorithm to question that have more similar tag structure to the source question.

The results of the running the guided PPR algorithm on 10000 random nodes are shown below:

TABLE III
GUIDED RANDOM WALK W/ RESTART $\alpha = .3$

Relevance Metric	Guided PPR	Stack Overflow Baseline
$\bar{O}_T$	0.7432	0.5962
$\bar{S}_{title}$	0.7187	0.7215
$\bar{S}_{text}$	0.9057	0.8628
$\sigma_{\bar{S}_{title}}$	0.1125	0.0741
$\sigma_{\bar{S}_{text}}$	0.0314	0.0311

The results above suggest a significant boost in performance as compared to unweighted Random Walk algorithm across all categories. Indeed, the weighted PPR algorithm achieves better performance than Stack Overflow's related set for both Tag-Overlap and Text-Similarity.[11] These results make sense since the algorithm essentially guides the walk to tag-nodes with more specificity. This is ideal if the goal is to recommend questions within a general category, but the ultimate goal is to recommend questions that may have the answer. Furthermore, it is important to note that the weighted PPR is slower than the vanilla algorithm due to the weight updates at each tag node. Furthermore, since this algorithm is jumping to questions through Tags, it is unlikely that it will give high weight to questions with a very different tag set. Sometimes a question is erroneously tagged. Since this algorithm explores via a guided walk through the tag-space, it is unlikely to find the answer if the questions is too generally tagged or tagged erroneously.

To improve results one could rank the recommendations of the PPR algorithm by their semantic similarity to the input question. Indeed, when taking the top 100 results from PPR and ranking each question by semantic similarity to the input question, the algorithm achieves better performance scores. However, this process isn't fair since it uses the evaluation metric to guide the procedure and therefore is not included in the results.

C. Attempted Variations

1) Incorporating User Information in User-Tag-Question Graph: I attempted to incorporate user information using a four-way graph between questions users and tags. Question q_i links to u_j if u_j asked question q_i. Question q_i is linked to t_k as in the $\mathbb{UT}$ graph. User u_j is linked to tag t_k with probability proportional to the number of times the user asks a question tagged with t_k. The links between q_i and t_k were calculated as in the guided-random walk. At each iteration i, the algorithm can choose to jump from a question to a tag or from a question to a user and then to a tag. The results, on preliminary analysis, were more diffuse, in that they recommended questions from a large tag-subset. This algorithm, however, did not scale well, since at each step, the algorithm must filter the edges based on their grouping within the graph. This is computationally expensive for such a large graph, and thereby was excluded from the analysis.

D. Incorporating Community Structure

The last variation of the PPR algorithm, which remains partially implemented [12], utilizes the communities of the $\mathbb{UT}$[13] to restrict the tag-space of the $\mathbb{QT}$ to those induced by the community to which the input question belongs. Formally, for a given q_i, let the community to which the owner of q_i belongs as c_i. For a given input question q_i, find the subgraph induced by $t_j \in c_i$ within the the $\mathbb{QT}$. In particular drop all $t_k \notin c_i$ and all $q_k : \nexists T(q_k) \in c_i$, where $T(q_k)$ is the tag set of question k.

VI. EXTENSIONS & FUTURE WORK

This paper utilizes variations on the Personalized Page Rank algorithm for question recommendation on a subset of the Stack Overflow Network with around 1 million nodes and 3 million edges. This paper implements variations of the PPR on the Question→ Tag graph to recommend "related" questions. This paper finds that guided PPR works well, achieving better results on average than Stack Overflow in terms of semantic similarity and tag overlap. Nevertheless, there appears to be a limit on the specificity of results able to be achieved using random walk algorithms. Using natural language processing on the text within a candidate set after PPR recommendation is an interesting avenue of exploration to increase the specificity of results.

As addressed in the introduction, an obvious limitation of the experiments is that it is not clear whether the metrics used (semantic similarity of text/title and tag overlap) to assess the quality of a recommendation set are actually useful in practice. It may be the case that PPR performs moderately well using the metrics defined in this paper but fails miserably at another metric such as user overlap. Furthermore, if one is optimizing for text similarity, it may be more accurate and faster to use elastic search over the raw database[14].

Another limitation is that the algorithms described above are restricted to nodes on the network. They do not generalize to natural language. Therefore, in order to transfer from input question node to input query while at the same time utilizing network information requires additional processing.

[11]The PPR and guided PPR did not receive the same input set, which may raise concerns about drawing comparisons between them. However, since both sets of random trials are compared to the baseline and the number of random starting nodes is large, I this is unimportant. The goal is to evaluate comparison to the baseline not between the two algorithms

[12]It proved too computationally expensive to run large simulations on the graph

[13]It is in fact a slight variation of the graph discussed. There is a link $u_i \to t_j$ if user i asks a question with tag j

[14]This is true and it is faster and more intuitive

One potential extension is to recast the problem to a semi-supervised setting. One could use `node2vec` [6] to learn feature representations of the nodes within the $\mathbb{UT}$ and $\mathbb{TT}$ graph [15]. One could then, for a given input user, use the word embeddings of the input query, the node embedding of the user, and the tag embeddings of the user's previous questions to predict a set of tags or even a set of questions. In the case of tags, labels are given, whereas for questions one could use Stack Overflow's `PostLinks` table. Indeed, utilizing learned node embeddings to predict the recommendation set using semi-supervised learning methods, though beyond the scope of this paper, would be a great avenue of exploration.

APPENDIX A
CODE

Code for the project can be found at https://github.com/jacobperricone/224w.

ACKNOWLEDGMENT

I would like to thank Poorvi Bhargava with her help throughout the project and Professor Leskovec for an enjoyable and informative class

REFERENCES

[1] Lure Jeskovec. Link analysis: Pagerank and hits, Nov 2017.

[2] Jon Kleinberg David Easley. *Networks, Crowds, and Markets: Reasoning about a Highly Connected World*. 2010.

[3] Raymond Shiau Dmitry Kislyuk Kevin C. Ma Zhigang Zhong Jenny Liu Yushi Jing David C. Liu, Stephanie Rogers. elated pins at pinterest: The evolution of a real-world recommender system. 2017.

[4] J. Z. Liu Y. Liu R. Sharma C. Sugnet M. Ulrich Eksombatchai, P. Jindal and J. Leskovec. Pixie: A system for recommending 1+ billion items to 150+ million pinterest users in real-time. 2011.

[5] Dibyajyoti Ghosh Priyank Mathur, Siamak Shakeri. User recommendation for stack exchange. 2016.

[6] Jure Leskovec Aditya Grover. node2vec: Scalable featuring learning for network. 2016.

[7] Gideon Dror, Yehuda Koren, Yoelle Maarek, and Idan Szpektor. I want to answer; who has a question?: Yahoo! answers recommender system. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, pages 1109–1117, New York, NY, USA, 2011. ACM.

[8] Bogdan Vasilescu Shaowei Wang, David Lo and Alexander Serebrenik. Entagrec: An enhanced tag recommendation system for software information sites. 2014.

[9] Christopher Wong Logan Short and David Zeng. Tag recommendations in stackoverflow. CS224W Project, 2014.

[10] StackExchange. https://archive.org/details/stackexchange.

[11] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA.

[15] I actually explored this avenue, creating node embeddings, but did not have time to implement the prediction step