# Final Project Report
# Link Recommendation in collaborative coding platforms like GitHub using biased random walks
# Group 35

Aditya Palnitkar
aditpal@stanford.edu

Vikesh Khanna
vikesh@stanford.edu

December 11, 2013

## 1 Introduction

In this paper, we address the classical problem of *link prediction* in social networks. Link prediction is the problem of identifying interactions between the nodes in the near future given the network snapshot at a point in time. Whereas the problem is well-studied in traditional social networks and co-authorship graphs, the concept of collaborative programming is fairly new and unexplored. There are several applications of link recommendation in this kind of a setting, like recommending potential collaborators on GitHub, making smarter hiring decisions, among others. To formalize, given the snapshot of a network at time $t$ and a node $u$, we predict the nodes with whom $u$ is likely to collaborate at time $t' > t$, on an online collaborative coding platform like GitHub.

In particular, we claim that the link formation process on collarobative networks like Github follows different dynamics than social networks like Facebook. We analyze the structural properties of the GitHub network to reason about these claims. We also propose a simpler variant of the supervised random walks algorithm that will bias the random walk based on a pre-computed model. Also, we introduce a tunable parameter in the algorithm ($\beta$) that will enable us to vary the sensitivity of the random walk's bias towards the features of the nodes, thus allowing it to adapt to networks similar to either Facebook or Github.

## 2 Review and Critique of prior work

Link prediction has been explored in some detail for traditional social networks of human interaction and co-authorship graphs.

### 2.1 The Link-Prediction Problem for Social Networks -David Liben-Nowell and Jon Kleinberg [5]

Nowell and Kleinberg's study explores whether we can understand the evolution of social networks based on features that are intrinsic to the network itself. It considers various approaches of scoring nodes using features based mostly on network topology only (like common neighbours, Jacard's coefficients, SimRank [4] etc.). Thus, the methods are computing a notion of similarity or proximity between the two nodes. Whereas these features are interesting, they do not account for various structure-agnostic properties of the nodes and edges that may considerably impact the results.

### 2.2 Link prediction using supervised learning: Mohammad Al Hasan, Vineet Chaoji, Saeed Salem and Mohammed Zaki [2].

This study claims that for social networks like Facebook, several non-topogical aspects (like age, hometown etc.) can substantially improve the link prediction results. In particular, they have used a mix of semantic (keyword match) and purely topological features (clustering index) for a supervised machine learning approach to link prediction in a co-authorship network.

### 2.3 Supervised random walks: Predicting and Recommending Links in Social Networks- Backstrom L., Leskovec J. [3]

This paper explores a middle ground between the purely network topology oriented and edge feature oriented approaches taken in the previous papers. It introduces the idea of supervised random walks. Supervised random walks take into account the network topology while calculating the page ranks. However, the algorithm does not assign uniform probabilities for the decision of taking a randomized step from one node to another. Rather, it assigns an edge strength to each edge based on the features of that edge. Influencing the probabilities of steps between pairs of nodes in this way gives the algorithm the ability to control the random walk to prefer certain nodes over others while taking individual steps. This influences the final stationary distributions and biases the page ranks to be higher for nodes which we know the initial node forms an edge to (in the learning setup

Although this paper manages to succesfully find an algorithm that takes into consideration the network topology and the edge features, the paper does not provide any tunable parameter to vary the sensitivity of the algorithm towards either of the two factors being considered.

## 3 Challenges

Link Prediction and recommendation are challenging from at least two points of view [3]. First, real networks are very sparse, i.e. nodes have connections to only a small fraction of nodes. Thus, an algorithm that predicts no new edges reaches near perfect accuracy. This challenge is furthur accentuated in the GitHub network since formal collaboration is extremely sparse, as we observed through data analysis. The second challenge is to understand the features, apart from the network topology itself, that motivate the formation of new links. There may be many exogenous factors that lead to link formation, but the network data itself often captures the essense of these factors. For instance, if two people become friends on Facebook because they met at a party, they are likely to have same age and belong to the same town. Similarly, we believe that GitHub-specific features like watching and forking a repository and following users capture the motivation of future link formation.

## 4 Claim

We claim that in networks like GitHub, features based on user activity influence link formation more than network topology. The reason for this is that users on Facebook are more likely to close triangles and befriend people with a high number of mutual friends. However, on Github, links would rather be formed based on user's interest in another user's work, i.e. link formation on GitHub is driven primarily by user interests and preferences. This does not mean that link formation is agnostic to topology, but that it is heavily influenced by user activity. We use this fact to develop a simpler variant of supervised random walks that biases the surfer (at node $u$) based on the probability of link formation with the source node ($s$) rather than the interactions with its neighbours (as in the original implementation).

## 5 Data collection

Our primary source of data was GitHub archive [1]. The data is available both in the form of raw data files (json) and through Google Big Query. We chose to parse the raw data files and build our own schema. This is because Google Big Query has data limitations. Morever, our own parsing scheme allows us to do online stream parsing (as opposed to batch download of data) making it computationally inexpensive. The raw data files are categorized by events, such as FollowEvent, MemberEvent etc, representing a single event in the GitHub timeline. Our parser analyzes each event type, extracts the most important information from the JSON object and populates the database.

Since the Github API allowed us access to events like pull, fork, watch and follow, and the API allows us to download the events which happened post April 2012, the data we obtained represents only a subgraph of the actual GitHub network.

## 6 Data Analysis

We use the following notation when referring to graphs in our paper -

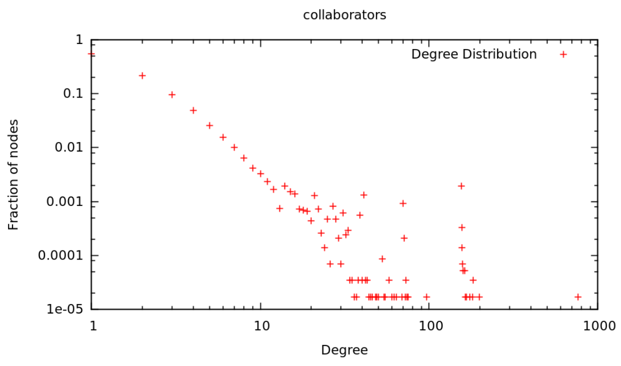
1. $G_{collab}$ - There is an undirected edge $(u, v)$ from $u$ to $v$ if $u$ and $v$ have worked on a common repository.
2. $G_{pull}$ - There is a directed edge $(u, v)$ from $u$ to $v$ if $u$ sent $v$ a pull request.
3. $G_{follow}$ - There is a directed edge $(u, v)$ from $u$ to $v$ if $u$ follows $v$.
4. $G_{watch}$ - There is a directed edge $(u, v)$ from $u$ to $v$ if $u$ watches $v$'s repository.
5. $G_{fork}$ - There is a directed edge $(u, v)$ from $u$ to $v$ if $u$ forks $v$'s repository.

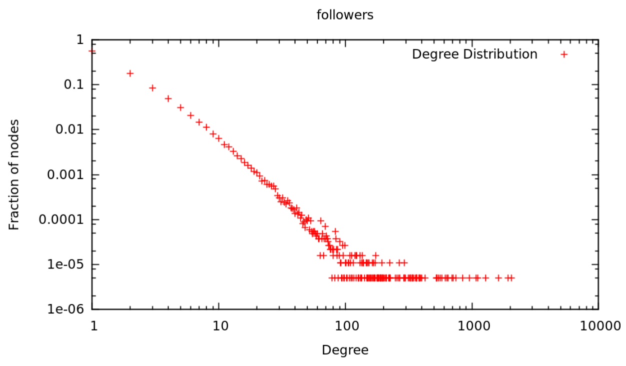
The objective of preliminary analysis of data was to understand the structural properties of GitHub, and how they differ from other social networks like Facebook. These properties play an important role in link formation. All data analysis was performed on a 4 month snapshot of the graph. We start off by analyzing the $G_collab$ graph and all the feature graphs, and compare them with other collaboration networks.

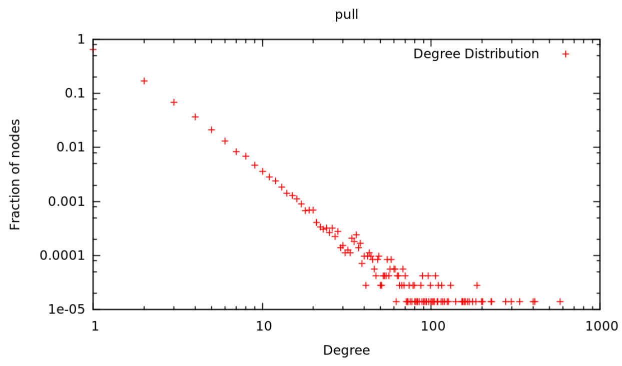
| **Property** | $G_{collab}$ | $G_{pull}$ | $G_{follow}$ | HepPh | Facebook |
|---|---|---|---|---|---|
| MLE estimate (alpha) for degree | 2.615 | 3.12 | 2.61 | 1.54 | 1.31 |
| Clustering coefficient | 0.394 | 0.028 | 0.054 | 0.611 | 0.56 |
| Fraction of nodes in largest WCC | 0.1436 | 0.76 | 0.794 | 0.933 | 0.993 |
| Average degree | 3.92 | 2.54 | 3.399 | 19.73 | 44.07 |

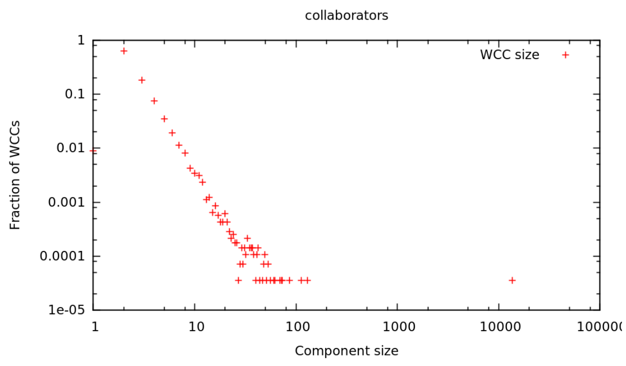
We also analyze the degree distribution of $G_{collab}$ and observe that GitHub graphs obey the power law. The degree distribution and component size distribution plots are shown below.

This analsysis helps us make crucial design decisions for our algorithm and verifies our belief of the GitHub network. From the network properties we can see that the average clustering coefficient of GitHub collaboration network is lesser than that of Facebook and HepPh. This supports our initial claim that there is less clustering in online social collaboration platforms as compared to real-life human interaction. We also have a dramatically low largest component size. From the data, we observe that, of the 68826 users added as collaborators to a repo, 45268 only worked on a single repository. The fraction of nodes in the largest WCC is merely 0.14 for formal collaboration. In fact, 65% of the users collaborate on a single repository only, i.e. 65% of all the components consist of just two nodes. Clearly, the GitHub network is extremely sparse and random walk in its original form will not perform well here. Note, however, that the largest connected component of the $G_pull$ graph has 0.76 fraction of all the nodes. We will use this fact later to expand the random surfer's territory.

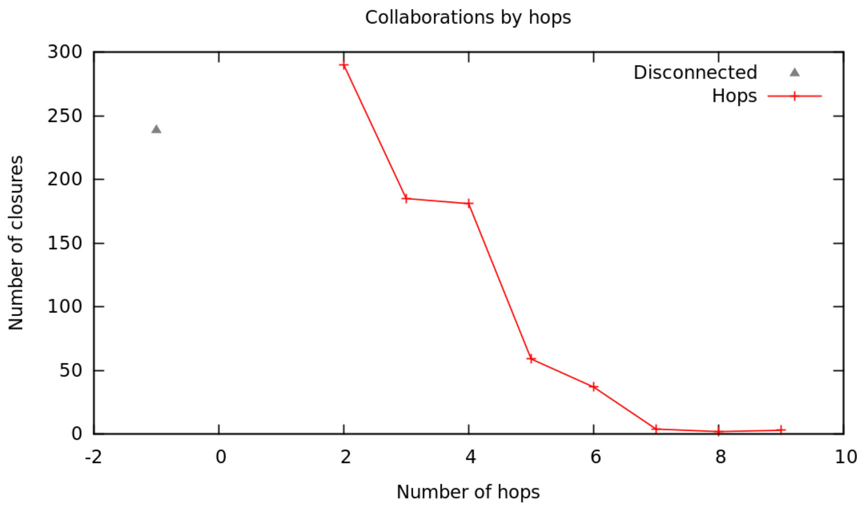








Besides the clustering coefficient, to get an intuituion of how the probability of a collaboration being formed between two nodes varies with the shortest distance between the two nodes, we plot the number of collaborations against the distance between the two nodes just before edge formation :



We can see that majority of the collaborations were formed between nodes that were two hops away. Also, a significant number of edges were formed between nodes that were in different WCCs, again highligiting the large number of WCCs in the graph, and the associated problems with using PageRank on such a fragmented graph.

## 6.1 Evaluation of link formation through network coefficients

Our goal here is to understand how well we can solve the link prediction problem using network characteristics alone. We do not use GitHub-specific features for this analysis. We took a snapshot of the graph at a time $t_1$ (2 months, 04/12 - 06/1/12). Using this snapshot as the base graph, we calculated the values of network coefficients like the Common neighbor coefficient, the Jaccard score and the Adamic score. We then considered the new edges added to the graph until time $t_2$ (15 days, 6/1/12 - 6/15/12). If number of new edges added between the times $t_1$ and $t_2$ was found to be k, we output the edges having the top k scores for these coefficients from the base graph, and considered them our predictions for the edges formed in between the times $t_1$ and $t_2$. We tested our predictions against the actual edges formed, and tabulate the results in terms of the recall value for the predictions:

| Feature used | GitHub network | Small World | HepTh network | HepPh network |
|---|---|---|---|---|
| Common Neighbor | 0.4115 | 0.1448 | 0.0370 | 0.0631 |
| Jaccard score | 0.0615 | 0.1853 | 0.0227 | 0.0798 |
| Adamic score | 0.46 | 0.1436 | 0.05211 | 0.0723 |

The Adamic score gave us the best results, which is in line with the expectations based on the results discussed in [5]. However, the Jaccard score gave results that were not as good as expected. The Common Neighbor also performs surprisingly well on the GitHub network, which when viewed in light of the small WCC sizes indicates the presence of localized clusters in the network.

## 6.2 Implementation of learning algorithms

We used three network features to run a learning algorithm on the Github Collaboration network. The features used were the Jaccard coefficient, the Common Neighbors coefficient, and the Adamic coefficient. The learing algorithm was run on three networks- the GitHub collaboration network, a synthetically generated small world network with similar degree distribution and a HepTh collaboration network. The algorithm used was logistic regression. The results obtained have been tabulated below.

| Metric used | GitHub Collaboration network | Small world graph | HepTh collaboration network |
|---|---|---|---|
| Recall value | 0.2924 | 0.025 | 0.1625 |
| Precision | 0.4909 | 0.43 | 0.63 |

We see that the recall value for Github network was the best amongst the three, when only the network coefficients are used. However, on an absolute scale, the recall value is still pretty low. We use the feature graphs discussed above, along with the random walk model, to significantly improve on these results.

# 7 Supervised Random Walk - Description and modification

The supervised random walk algorithm calculates the weights to be assigned to the different edges in the graph to bias the probability of the random walk such that the final PageRanks of the nodes to which a given source node forms an edge are higher than the PageRanks of all the other nodes. The optimization function can be expressed as:

$$\min_{\omega} F(\omega) = \|\omega\|^2 \tag{1}$$

subject to

$$\forall d \in D, l \in L : p_l < p_d \tag{2}$$

where $\omega$ are the weights of the features $\phi_{u,v}$ of the edges in the edge strength function $f_w(\phi_{u,v})$, and $p_d$ and $p_l$ are pageranks of nodes which formed and did not form links with the node $s$ being considered, respectively. We can convert this minimization objective to

$$\min_{\omega} F(\omega) = \|\omega\|^2 + \lambda \sum_{d \in D, l \in L} h(p_l - p_d) \tag{3}$$

This algorithm requires us to optimize a function that includes calculation of Pageranks. Due to this, we have to calculate the pageranks and the gradients of the PageRanks for every iteration of the gradient descent that is run on $F(\omega)$.

## 7.1 Adjusting the Supervised Random Walk algorithm for GitHub

The supervised random walk algorithm described above biases the surfer (at node $u$) based on the the edge strength with its neighbours (nodes $v$), i.e. it learns the edge weights $w$. However, as per our belief, link formation in GitHub is heavily influenced by user interests and preferences. Therefore, we propose a model of a random walk in which the transition probabilities are biased based on a pre-computed model, where the edge strengths are assigned based on the probability of link formation with the source. This results in a much simpler and computationally inexpensive algorithm that performs well for GitHub. We also introduce a parameter $\beta$ that can help us bias the random walk to be more or less sensitive towards the features of a pair of nodes being considered. Our algorithm has two stages - the supervised learning phase and the biased random walk phase.

1. **Supervised Learning**:
   We run a supervised learning algorithm taking into consideration the base and delta graphs. Specifically, for every pair of nodes at most $k$ hops away in the base graph, we extract features from $G_{follow}$, $G_{fork}$ and $G_{watch}$. Then we observe the delta graph and see which nodes each source node forms an edge with. We mark these edges as positive class labels and the others as negative. We use a binary classifier to learn a model that can predict edges based purely on the feature graphs. The following table lists the features used from each of the feature graphs:

   | **Feature used** | **Comment** |
   |---|---|
   | Edge | Marked as 0 or 1 depending on whether there is an edge in the feature graph. For instance, an edge between two users in the fork graph may indicate collaboration in the near future |
   | Common neighbours | Number of common neihbours indicates the proximity of two nodes in the corresponding feature graph. For instance, if the two nodes follow the same users, they may have similar interests and may collaborate in the future. |
   | Jaccard Coefficient | Gives a measure of similarity between two nodes in the feature graph. |
   | Adamic score | Gives a measure of similarity between two nodes in the feature graph. |

   Note that all features were scaled to have zero mean and unit variance.

2. **Biased Random Walk** :
   As we have already seen, one issue with the GitHub collaboration network is the extreme sparseness that makes random walks practically ineffective. To solve this problem, we add edges between two nodes in $G_{collab}$ if there is a corresponding edge in $G_{pull}$. These edges have a edge type of 1 whereas the formal collaboration edges have a type of 2. This allows us to expand the random surfer's territory signifcantly while also maintaining the difference in the type of edges.

   (a) For this phase, we select a source node from the base graph for which we predict the links it will form in the delta graph. We start with constructing a subgraph of *Gcollab* containg nodes that are at most 10 hops away from the source node. We do this to improve the run time of the algorithm - This should not affect the results because, as we saw in the data analysis, negligible collaborations happen for nodes that are more than 10 hops away.

   (b) For this particular source node $s$, we extract the features of every pair of nodes $\phi_{sv}$. Then we bias the probabilities of the random walk such that for every node $v$, the probability in the transition matrix $Q_{uv}$ for all neighbor $u$ of $v$, u is biased to be higher if the features $\phi_{sv}$ are predictive of an edge being formed between $s$ and $v$. The function we use allows a parameter $\beta$ to vary this amount of bias. Note that this probability is assigned based on the features of the source node $s$ and $v$, whereas the orignal implementation would calculate the edge strength, $a_{uv}$ based on the interactions between $u$ and $v$. This allows us to give more importance to the similarity of $s$ and $v$'s interests in predicting a link.

(c) We calculate the PageRanks using this biased transition probability matrix.

(d) Then, we consider the 20 nodes with the highest PageRanks, and considering these nodes to be the predicted positive samples, evaluate our prediction by calculating the recall and precision.

Formally, during the computation of the transition probability matrix $Q$, for every edge $(u, v)$ in the base graph, we calculate the function $g(\omega, \phi_{u,v}, \beta)$:

$$g(\omega, \phi_{u,v}, \beta) = exp(\beta * f(\omega, \phi_{s,v})) \tag{4}$$

where $\omega$ is the model learnt by the training phase, $\phi_(s, v)$ are the features for the pair $(s, v)$ and $f(\omega, \phi_{s,v})$ is the probability of an edge being formed according the supervised learning algorithm. This probability will depend on the algorithm being used. For logistic regression, $f$ will be the sigmoid function.

We normalize and calculate $Q$ as follows:

$$Q_{uv} = \begin{cases} \frac{g(\omega,\phi_{u,v},\beta)}{\sum_{w \in N(u)} g(\omega,\phi_{u,w},\beta)} & \text{if } (u, v) \in E \\ 0 & \text{if } otherwise \end{cases} \tag{5}$$

where $N(u)$ is the set of nodes in the neighorhood of $u$. In essence, for every node $v$ for which logistic regression suggests a good probability of edge formation, we favourably bias the probability of all the edges incoming for $v$ from all its neighbors.

If all the weights are kept as 1, we will be simulating a pure random walk. So, more the departure of the $g(\omega, \phi_{u,v}, \beta)$ values from the constant value of 1, i.e. more the variance of the $g(\omega, \phi_{u,v}, \beta)$ values for a given value of u, more biased is the probability of taking a particular edge in the supervised random walk. So, we need an edge strength function that will give us a high sensitivity to the values of the edge features.

One function that can serve this purpose is the exponential function $e^{\beta * f(\omega\phi_{uv})}$. By varying the value of $\beta$, we can control the variance of the $a_{uv}$ values. By having $\beta = 0$, we can force all the $g(\omega, \phi_{u,v}, \beta)$ values to 1, and hence perform a pure random walk on the graph. By increasing the value of $\beta$, we can have an algorithm that is very sensitive towards edge features in the graph.

## 8 Experimental Setup

We perform all experiments on $G_4$ and $G_6$, referring to a 4 month and 6 month snapshot of the GitHub data respectively. $G_4$ extends from 04/12 to 08/12, while $G_6$ extends from 08/13 to 12/13. The last month of each graph was used to construct the delta graph. The rest of the timeline was used to construct The base graphs for both collaboration and features. We tested the algorithm for a range of $\beta$ values on both the graphs. We first selected a set of random source nodes. This set of source nodes was kept constant for all values of $\beta$. After selecting the source node, we construct the sub graph with nodes at most 10 hops away from the source. For each pair of nodes in this subgraph, we extract the features from the feature graphs. Thereafter, we construct the biased transition matrix as described ealier. Based on this matrix, we perform a biased random walk. Finally, we took the 20 nodes with the highest PageRanks and considered them to be our predictions for an edge formation with the source node.

## 9 Results

We calculated the recall and Precision@20, the number of nodes in the top 20 nodes to which the source node actually formed an edge. We also analysed the average index of the nodes to which an edge was actually formed in the descending page rank array. A smaller value for the average index points to the nodes which actually formed an edge being higher in the page rank array, thus having relatively higher PageRanks.
We ran the implementation of the algorithm varying the value of $\beta$. By starting with $\beta = 0$, we first derived the performance of an unbiased Random Walk, which we treated as a baseline. Then, we gradually increased the value of $\beta$, and evaluated the performance of the Random walk in terms of the three metrics described above.

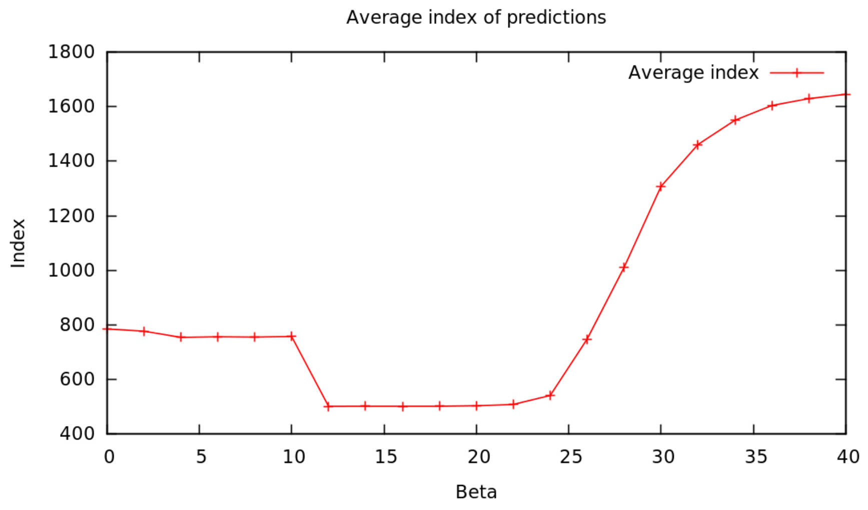
Our algorithm performed better than a baseline of simple random walk on all the three metrics tested for. The results for $G_4$ are tabulated as follows:

| $\beta$ value | Recall | Average index of positive samples | Precion@20 |
|---|---|---|---|
| 0.000000 | 0.518421 | 1001.986765 | 1.400000 |
| 1.000000 | 0.518421 | 999.086765 | 1.400000 |
| 2.000000 | 0.518421 | 996.836765 | 1.400000 |
| 3.000000 | 0.518421 | 993.986765 | 1.400000 |
| 4.000000 | 0.518421 | 991.836765 | 1.400000 |
| 5.000000 | 0.518421 | 990.686765 | 1.400000 |
| 6.000000 | 0.518421 | 987.586765 | 1.400000 |
| 7.000000 | 0.518421 | 986.136765 | 1.400000 |
| 8.000000 | 0.518421 | 984.936765 | 1.400000 |
| 9.000000 | 0.518421 | 982.736765 | 1.400000 |
| 10.000000 | 0.518421 | 981.486765 | 1.400000 |
| 11.000000 | 0.518421 | 979.736765 | 1.400000 |
| 12.000000 | 0.518421 | 978.536765 | 1.400000 |
| 13.000000 | 0.568421 | 969.686765 | 1.450000 |
| 14.000000 | 0.568421 | 968.236765 | 1.450000 |
| 15.000000 | 0.568421 | 967.486765 | 1.450000 |
| 16.000000 | 0.568421 | 966.586765 | 1.450000 |
| 17.000000 | 0.568421 | 965.886765 | 1.450000 |
| 18.000000 | 0.568421 | 964.936765 | 1.450000 |
| 19.000000 | 0.568421 | 964.186765 | 1.450000 |
| 20.000000 | 0.568421 | 963.736765 | 1.450000 |

Here is a comparison of the various methods we implemented (as tested on $G_6$) -

| **Method** | **Recall** | **Precision@20** | **Average Index** |
|---|---|---|---|
| Learning from network coefficients only | 0.29 | - | - |
| Random Walks with Restart (Biased walk with $\beta$=0) | 0.518421 | 1.40 | 785.159291 |
| Biased random walks with restart ($\beta$=12) | **0.568421** | **1.45** | **501.597928** |

We also plotted the average index of positive samples in the Pagerank array. The average index decreases with increasing $\beta$. As we increase $\beta$, we are assigning higher PageRanks to nodes which actually formed an edge with the source node in $G_{delta}$. The plot shows that the average index reduces with $\beta$, reaches a minima at 12 and then increases. This is expected, as for sufficiently high values of $\beta$, we start biasing the edge probabilities too sensitively, and thus the network topology starts playing lesser of a role in determining the PageRank of a node. Thus, for this GitHub network, we can observe the optimal value of $\beta$ is around 12. We observed that several nodes formed a few edges quite randomly, probably motivated by factors completely external to the state of the network. Considering the scale of the network, the index takes a drastic hit even if a single edge is formed randomly. Therefore, the absolute value of index is not a good metrics for performance, but, a decrease in the value of the average index on increasing $\beta$ for the same source nodes (hence, same random edge formations) indicates our ability to predict links based on our bias from feature graphs.



## 10 Future work

In this study, we have implemented a variant of random walks that biases the transition probabilities based on a supervised learning model. From the experiments, we see an improvement in both recall and precision@20. Although these improvements are marginal, they suggest that feature graphs are a good indicator of link formation on GitHub.

In the future, we would like to test the case where all the edges leading from a source node to this particular node $v$ to have biased probabilities. This will lead to a higher PageRank for $v$. However, this change will also increase the pageranks for all the nodes that lie on the path from the source node to $v$, regardless of the feature match with the source node. Experiments that compare both the results need to be performed before it can be said which of the two effects will be more significant.

Although we do believe that user interests play a larger role in link formation on GitHub, we would like to experiment with the implementation of the supervised random walk as in [3]. Combining our algorithm with this implementation might signficantly improve results. For instance, we can better learn the edge strength parameter $\rho$ (edge type in the unified graph). We can also use the recency of the edge and number of interactions in the collaboration graph as edge strength parameters.

Currently, due to reasons of runtime efficiency, for a given source node, we are predicting links only among nodes which are at most 10 hops away. As we can see from the Analysis section, this restriction is not very limiting in the sense that most of the edges added are within 3 hops away. In the future, we would like to remove this restriction. In addition, we would like to introduce a factor $\alpha$, which is the probability of a random surfer jumping to a random node in the whole graph (based on an intelligent criteria). This will mitigate the problem of having a large number of disconnected WCCs in the graph, and allow the random walker to span territories that might be good potential targets for future edges. Although unifying the $G_{collab}$ and $G_{pull}$ graphs mitigates the sparseness, we would also like to improve our learning algorithm to be more robust to the class imbalance in GitHub.

## References

[1] http://www.githubarchive.org.

[2] Mohammad Al Hasan, Vineet Chaoji, Saeed Salem, and Mohammed Zaki. Link prediction using supervised learning. In *SDM06: Workshop on Link Analysis, Counter-terrorism and Security*, 2006.

[3] Lars Backstrom and Jure Leskovec. Supervised random walks: predicting and recommending links in social networks. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 635–644. ACM, 2011.

[4] Glen Jeh and Jennifer Widom. Simrank: a measure of structural-context similarity. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 538–543. ACM, 2002.

[5] David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. *Journal of the American society for information science and technology*, 58(7):1019–1031, 2007.