2014

An Adaptive Hybrid Method for Link Prediction in Multi-Modal Directed Complex Networks Using the Graph Traversal Pattern

William Lyon

An Adaptive Hybrid Method For Link Prediction In Multi-Modal Directed Complex Networks Using The Graph Traversal Pattern

By

William James Lyon

B.A. Economics, University of Montana, Missoula, Montana, 2006
B.S. Business Administration, University of Montana, Missoula, Montana 2006

Thesis

presented in partial fulfillment of the requirements for the degree of

Master of Science
in Computer Science

The University of Montana
Missoula, MT

December 2014

Approved By:

Sandy Ross, Dean of the Graduate School
Graduate School

Jesse Johnson
Department of Computer Science

Robert Smith
Department of Computer Science

Douglas Dalenberg
Department of Economics

Lyon, William - M.S. - December 2014 Computer Science

An Adaptive Hybrid Method For Link Prediction In Multi-Modal Directed Complex Networks Using The Graph Traversal Pattern

Chairperson: Jesse Johnson

The paper examines the link prediction problem for directed multi-modal complex networks. Specically, a hybrid method combining collaborative filtering and Triadic Closeness methods is developed. The methods are applied to a sample of the GitHub network. Implementation details are discussed, with a focus on design of a scalable system for handilng large data sets. Finally, results of this new method are discussed with no significant improvement over current methods.

Declaration

I hereby declare that this thesis is my own work and effort and that it has not been submitted anywhere for any award. Where other sources of information have been used, they have been acknowledged.

William Lyon

Contents

CHAPTER 1

Introduction

Many social, biological and information systems can be modeled as complex networks (or graphs) where the nodes are individuals and the edges (or links) between nodes represent interaction between the nodes. The term complex network is used to describe graphs (consisting of nodes and edges) that exhibit non-trivial topological structure. These networks often consist of real-world interactions, such as computer, social and collaboration networks that cannot be modeled as a random graph[Ste10]. With the growing popularity of online social networks (such as Facebook and Twitter) in recent years, much research has been devoted to understanding these types of complex networks. One area of that research is recommender systems: the problem of recommending interesting content/users in the network. Examples of these systems used in production include Twitter's "Who To Follow" system [Pan13], LinkedIn's "People You Might Know", and similar recommendation in FaceBook. When modeled as a graph, recommender systems become link predictors answering the question can the system predict removed links in the remaining network? In this paper we explore the problem of link prediction in complex networks, and apply the methods to a data set from the GitHub online social collaboration network. A novel system for generating link predictions is discussed, implemented and evaluated.

1.1 Online Social Collaboration Networks

A new type of social network that has been gaining popularity is the online social collaboration network. Online social collaboration networks map social interactions allowing users to collaborate toward some common goal. An example of this type of network is GitHub [1]. GitHub is a software collaboration web service built around the git version control system. Software developers use GitHub to collaborate on software projects, to share their projects, to interact with other developers/users, and to follow what other users are working on. GitHub is perhaps the largest community of software

1 http://www.github.com

developers in the world, with perhaps the largest collection of open source software under active development. As we will see, this provides a very rich dataset for analysis. In a more practical sense, improving recommender systems for the GitHub network can help users find software projects and users with whom to collaborate faster and more easily. This could lead to a more productive software development ecosystem.

1.2 Link Prediction For Directed Multi-Modal Networks - An Adaptive Hybrid Method

The goal of this thesis is to explore the link prediction problem, as applied to online social collaboration networks. Much of the previous literature in this area focuses on homogenous (single relationship) undirected social networks. We extend this research to focus on heterogenous (multiple relationship type) or multi-modal directed social collaboration networks, specifically the GitHub network. We duplicate the work done previously in this area and develop a novel approach to link prediction, specifically an adaptive ensemble method.

The remainder of this paper is outlined as follows:

Chapter 2 - Previous Work. A review of the literature in this field. We show how link prediction fits into the recommender system literature and discuss how link prediction methods have adapted as they are applied to evolving types of networks.

Chapter 3 - Methods. An examination of the data used for this project as well as an in-depth explanation of the algorithms used for link prediction, in the context of the graph traversal pattern, which is also explained in this chapter. A novel approach for link prediction is proposed, a combined similarity and network structure method. Implementation details involving graph data modeling and graph databases are discussed.

Chapter 4 - Evaluation. This new recommender system is evaluated relative to similarity based methods and network structure methods. Challenges of the present system are discussed.

Chapter 5 - Summary. Areas of further research are discussed.

CHAPTER 2

Previous Work

The link prediction problem for complex networks was best formalized by Kleinberg: "Given a snapshot of a social network, can we infer which new interactions among its members are likely to occur in the near future?" [LN07] It is this problem which we address in this paper. We discuss how link prediction fits into the world of recommender systems and provide a brief overview of the methods commonly used for the link prediction program. We identify two distinctive types of link prediction algorithms: similarity based algorithms and network based algorithms.

2.1 Recommendation As Link Prediction

Consider a bipartite graph of Users and Items where the edges of the network represent purchase links: if a link exists between User 1 and Item x then User 1 has purchased Item x. Recommender systems modeled on this type of data take the form of "users who bought x also bought y", as often seen on online retail sites such as Amazon. For example, in Figure 2.1 we see a bipartite graph as described above. An item recommendation for User 1 takes the form of a predicted link in the network. This is an example of a User-Item link prediction, however we could also predict links between Users based on the structure of the network, similarity between User-Item preferences, or both. An example of user recommendation is LinkedIn's "Do You Know..." feature, which suggests Users. These types of link prediction/recommender systems can be generalized as "Who To Follow" recommender systems and it is this type of recommendation on which this paper will focus [Pan13].

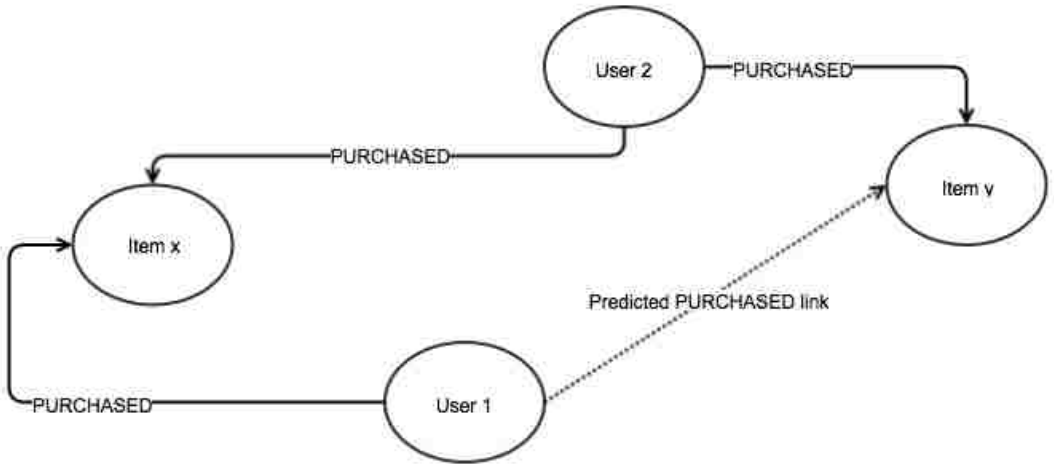


Figure 2.1: Here we see an example of an item recommender system modeled as a graph where the nodes are Users and Items and the edges indicate a purchse of an Item by a User. When modeled this way recommendation takes the form a predicted link in the graph.

2.2 Similarity-based Algorithms

Similarity based methods rely on the computation of a user-user similarity metric, which is then used to make recommendations. This is based on the homophily principle, that users are more likely to be interested in users similar to them.[Ric11] Each pair of nodes in the network is assigned a score S_{uv}, which represents the strength of the similarity between u and v. To generate predicted links, all unobserved links u,v are ranked based on S_{uv}, with the highest ranked selected as predicted links. Similarity metrics are often calculated based on observed links in the networks, with the concept of overlapping neighbors in the network being a common distinguishing characteristic. [Lu10]

Table 2.1: Common similarity metrics. Defintion of several commonly used similarity metrics. Note that Pearson correlation and Cosine similarity make use of weighted edges, while Jaccard is calculated without taking edge weights into account.

Metric	Definition	Description
Jaccard	$\mathrm{J(A,B)} = \frac{\lvert A \cap B \rvert}{\lvert A \cup B \rvert}$	Size of the intersection of neighbors divided by the union of the sets of neighbors
Pearson correlation	$\frac{\sum_{i=1}^{n}(X_i-\bar{X})(Y_i-\bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i-\bar{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i-\bar{Y})^2}}$	A mean adjusted correlation coefficient.
Cosine similarity	$\frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n}(A_i)^2} \times \sqrt{\sum_{i=1}^{n}(B_i)^2}}$	Cosine distance between two vectors.

If edge weights (often a User-Item rating) are available, the Pearson Coefficient or Cosine similarity are commonly used. For binary ratings the Jaccard metric is often used. Table 2.1 shows the definition for these metrics.

When user preferences are taken into account (such as the User-Item purchase network shown in Figure 2.1) to calculate the similarity metric the system is said to make use of *collaborative filtering*. Collaborative filtering systems produce user specific recommendations based on patterns of behavior observed from other users. Typically this involves observing ratings of items and using either latent factor models or a neighborhood based approach to generate item recommendations[Ric11]. With the rise of social network analysis however, often instead of user-item recommendations, we are more interested in generating user-user recommendations. User-user recommendations are the focus of this project. The underlying assumption of collaborative filtering is that of homophily: similar users like similar things. Collaborative filtering implementations can be problematic when applied to a large dataset. Most methods require a large sparse matrix for computation, the use of which is not always performant. Instead, the problem can be modeled as a graph, and make use of the graph traversal pattern as an alternative to the construction of a large sparse matrix [Rod10]. Consider for a moment the time complexity involved in calculating the all-pairs User-User similarity metric necessary for collaborative filtering. This is at best an $\mathcal{O}(n^2)$ computation, depending on the similarity metric being calculated. This obviously does not scale well to large networks and so implementation details must be taken into account.

2.3 Network-based Methods

Network-based methods analyze the structure of the network to develop recommendations. Example of this include PageRank [Pag], HITS [LN07], SALSA [Lem01], and Triadic Closeness. The PageRank and HITS algorithms attempt to rank nodes in the network by their relative importance, or centrality in the network. PageRank does this by computing an eigenvector centrality, while HITS focuses on identifying nodes that can be classified as authorities, forming important hubs in the graph. SALSA and related methods use a more probabilistic approach and use random walks through the graph to generate link predictions. Indeed the Twitter Who To Follow system is based on such a method. [Pan13]. Other methods such as Triadic Closeness use patterns in the network to generate recommendations.

2.3.1 Triadic Closeness

Triadic Closeness is based on the graph theory concept of triadic closure. Triadic closure is the hypothesis that for two nodes in a network u and v, the existence of an edge between u, v is highly correlated with the overlap of u and v direct connections (neighbor overlap). The Triadic Closeness method uses triad pattern detection to determine the likelihood that a given triad pattern is likely to close in the network (that an edge will

form from u to v). Triadic Closeness can be summarized as:

$$\text{Triad Closeness} = \frac{\text{Number of closed triads}}{\text{Number of potentially closed triads}} \tag{2.1}$$

2.4 Directed Social Networks

There is an important distinction to note between undirected social networks and directed social networks. Much of the literature has focused on undirected social networks only [Lu10]. In fact the similarity metrics shown above are all based on undirected social networks. A notable exception is the Triadic Closeness method described by Schall [Sch14].

2.5 The case for a hybrid method

There is a clear gap in the literature making use of combined network based methods with similarity based methods [Lu10]. My contribution to this field is to explore how these methods can be combined to improve the accuracy of such recommendations. A linear combination of a collaborative filtering similarity based approach and a network based approach leveraging the use of directed networks is proposed, based somewhat on the work developed in [Can08]. By combining a network-based method with a similarity-based method we are able to capture more information about the structure of the network and information about specific user preferences and actions. By taking into account the proportion of edge types available for each user we are able to adjust the weights for each method in the hybrid metric, making the method adaptive for each user, based on the quantity of information available that describe each user's actions in the network. To enhance the effectiveness of the combined method we focus exclusively on multi-modal networks. Multi-modal networks are a type of complex network that contain multiple relationship types and/or multiple node types. Finally, we focus here on making use of the graph traversal pattern. By modeling our data as a graph we can efficiently implement the methods discussed by traversing the graph, focusing on a local portion of the graph, rather than complex and expensive calculations for the entire network.

CHAPTER 3

Methods

Here we describe the data used and detail the implementation of the link prediction experiment. We first examine in detail two methods for link prediction using a sample network: collaborative filtering using the Jaccard similarity metric and the Triadic Closeness method. We then show how these two methods can be used together in a hybrid predictor using adaptive weights. An experiment using data from the Github social collaboration network is discussed. The data used in this experiment is examined. Finally, implementation is discussed. We focus on the link prediction problem for a partially observed network. We assume certain links are missing from the network and attempt to predict the missing link(s), focusing on User-User edges.

3.1 Sample Network

Consider the network shown in figure 3.1. This is a sample network which was randomly generated and does not represent any real world observed data. However, we shall refer to this network to demonstrate the techniques used in this project. The sample network contains two types of nodes: Users and Items. Each User can *FOLLOW* other Users. This is represented as a User-User directed edge with the label *:FOLLOWS*. Similarly, Users can express their interest in an Item with the *:LIKES* relationship (or edge). This type of network structure is similar to those observed in social networks (such as Facebook, or Twitter), but also in collaboration networks, such as Github. Since the network has multiple types of nodes and edges it is referred to as a **multi-modal network**[Ste10].

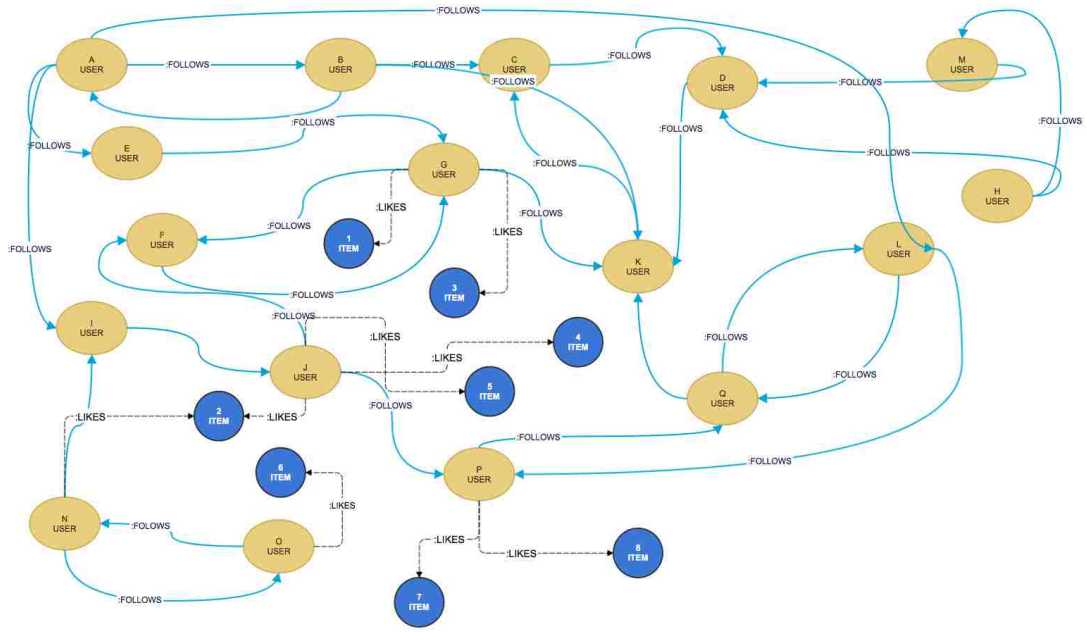


Figure 3.1: This sample network will be used to demonstrate the methods used for link prediction in this paper. This network demonstrates a random multi-modal network with multiple types of nodes and edges.

3.2 Algorithms

For illustrative purposes we will work through three examples of link prediction algorithms for the sample network shown above. First, using the collaborative filtering method with the Jaccard similarity metric. We will use User-Item edges to identify similar uses and generate recommendations based on those similarities. Next, we walk through the Triadic Closeness method as described in [Sch14]. Using probabilities observed from triad patterns we will generate link predictions and compare to those created using collaborative filtering. Finaly, we propose a hybrid method that combines collaborative filtering and Triadic Closeness using an adaptive weighting system. In the context of the sample network we focus on predicting User-User *:FOLLOWS* edges only.

For the purposes of the next three sections we will consider link prediction for user J. We proceed through each algorithm manually, ignoring some implementation details for now that will explored in depth in the proceeding section.

3.2.1 Collaborative Filtering

Collaborative filtering is a method of generating recommendations based on the homophily principle: users who are similar are likely to be interested in similar items. It is implemented by finding similar users, based on some similarity metric [Ric11]. Here we will use User-Item edges as an indication of a User's binary rating of an Item. The

Jaccard metric is used to show a proportion of overlapping neighbors.

Similarity Metrics

The Jaccard index is used to identify similar users. For two users, a and b, let A and B denote the sets of all users being followed by a and b, respectively. The Jaccard index is therefore as defined in Equation 3.4.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{3.1}$$

In this context, Jaccard is defined as the intersection of the Items liked by a and b divided by the union of the items likes by a and b. This results in a number between 0 and 1, indicating the strength of similarity between users a and b.

To generate recommendations for user J, we first must identify all friend-of-friend nodes, that is nodes that share a neighbor Item in common with J. That gives us the set $\{N\}$. Our possible recommendations are now reduced to N. We will now compute the Jaccard similarity metric for the pair (J, N):

$$J(J,N) = \frac{|J \cap N|}{|J \cup N|} \tag{3.2}$$

The intersection of J and N here is defined as all items that have an incoming *:LIKES* edge from both J and N. Looking at the graph we can see that the intersection is *Item2*. Similarly, we can look at the graph to find the items that compose the union of J and N.

$$J(J,N) = \frac{|\{Item2\}|}{|\{Item2, Item4, Item5\}|} \tag{3.3}$$

We are only interested in the size of the two sets, so we simply count the elements.

$$J(J,N) = \frac{1}{3} \tag{3.4}$$

We can now predict the edge $J \rightarrow N$ with weight $1/3$. [1]

As you can see, the collaborative filtering link prediction process for a given user x

1 If we were interested in predicting User-Item links, we could allow each similar user to *vote* for other Items in which J might have an interest. We now take the top k nodes that have the highest Jaccard score and allow each to vote for new outgoing links to form from J. Here we will select L and recommend any outgoing links from L as destination nodes for predicted links emminating from J. However, we are only concerned here with User-User edges.

involves finding other users most similar to user x, then finding items those similar users are most interested in. In this sense collaborative filtering can be thought of as very similar to k-nearest neighbors, where the distance calculation is based on some similarity metric.

3.2.2 Triadic Closeness

Graph theory proposes the concept of triadic closure, the hypothesis that the creation of an edge between u and v is related to the degree of overlapping neighbors in u and v's respective networks. [Ste10] The concept of Triadic Closeness is an application of the theory of triadic closure, specifically taking into account the directed nature of social networks. For a given fully observed network, Triadic Closeness can be thought of as the ratio of the number of closed triads to the number of potentially closed triads[Sch14]. A triad consists of three nodes u, z, v where edges (ignoring direction) u,z and z,v exist. Edges between u and v may exist, however the concept of triadic closure posits that an implicit connection exists between u and v.

Algorithm 1 Link prediction algorithm for Triadic Closeness

```
input : G(U, E), x, N
usersSample ← getRandomUsers(U,x)
results ← {}
for eachuserinusersSample do
    validationEdge ← getRandomEdge(G, user)
    removeEdge(validationEdge,G)
    triads ← getTriads(user, G)
    pred ← {}
    for u, v in triads do
        tc ← calcTC(u,v,G)
        pred ← pred + {tc, u, v}
    end for
    predictions ← topXSortedByTC(pred, N)
    hit ← isvalidationEdgeinpredictions?
    addEdge(validationEdge, G)
    results ← results + {hit, pred, u, v, validation_edge}
end for
return results
```

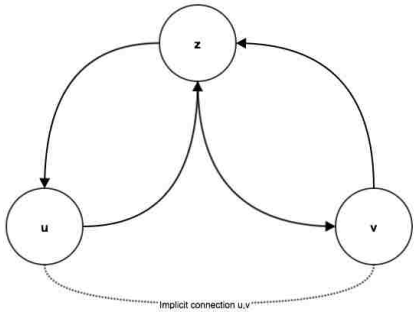


Figure 3.2: For any given triad (u, z, v) there is an implicit link between u and v. Triadic Closeness is a measure of the strengt of this implicit link.

In a directed network there are 27 distinct configurations, or patterns that a triad can take on. Figure 3.3 shows the 9 unique open triad patterns. Table 3.1 shows triad patterns that are open, that is no connection exists between nodes u and v. The pattern identifications ($T01, T02...$) are taken from[Sch14]. Any open triad can be closed in one of three possible ways: $u \leftarrow v$, $u \rightarrow v$, or $u \leftrightarrow v$.

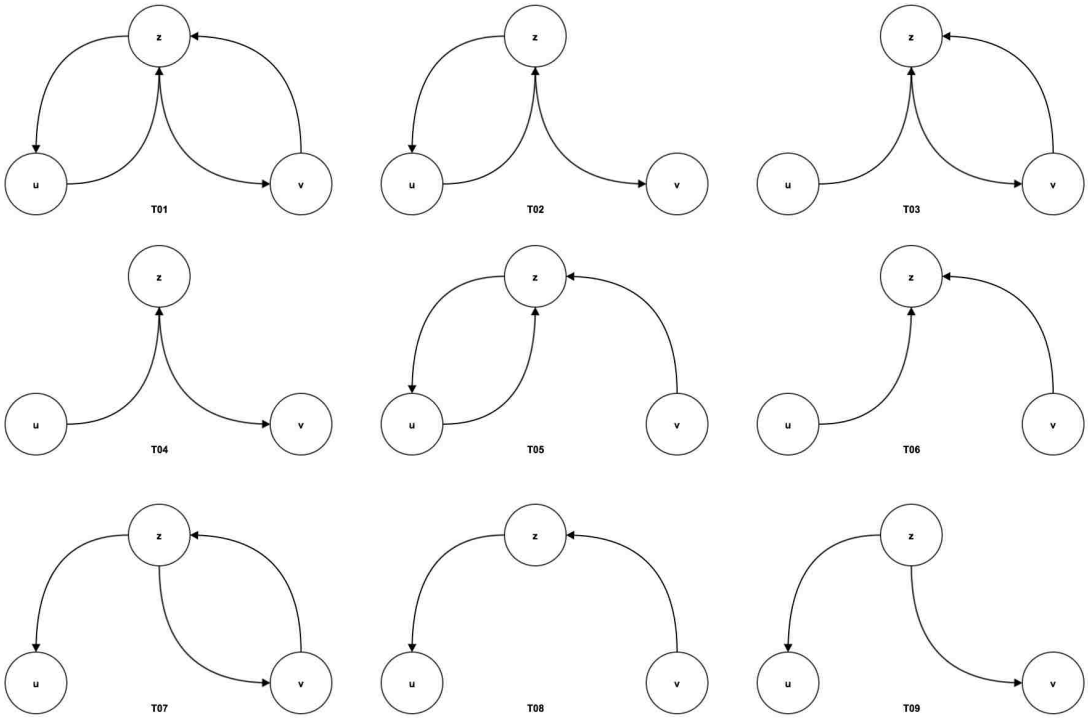


Figure 3.3: Here all possible open triad patterns are shown. Each of the nine unique triad patterns are labled using an identifier Txy where y indicates which of the nine open patterns the triads corresponds and x indicates if the triad is open (0), if the triad is closed with an edge $u \to v$ (1), if the triad is closed with an edge $u \leftarrow v$ (2) or if the triad is closed with two edges $u \leftrightarrow v$ (3). This triad pattern identification scheme is used when computing Triadic Closeness.

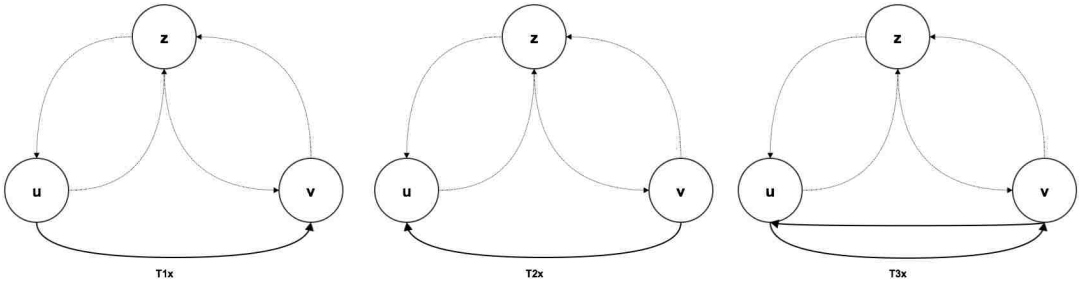


Figure 3.4: A triad is considered to be closed if an edge exists between u and v. In the triad pattern ID, Txy, x identifies how the triad is closed.

Figure 3.4 shows the three distinct ways in which an open triad can be closed. Either the creation of an edge $u \to u$, the creation of an edge $u \leftarrow v$, or the creation of two edges $u \to v$ and $u \leftarrow v$. In terms of the triad pattern id, the first digit indicates if the

triad is open (0), closed with an edge $u \rightarrow v$ (1), closed with an edge $u \leftarrow v$ (2), or closed with two edges $u \rightarrow v$ and $u \leftarrow v$ (3). With this nomenclature we are now able to represent each possible triad pattern with a two digit identifier.

$$TC_{uv} = \sum_{z \in \Gamma(u) \cap \Gamma(v)} w^P(u,v,z) \times w(z) \tag{3.5}$$

As shown in Equation 3.5, triadic closeness for u,v is defined as the sum of w^P times $w(z)$ over all triads in which u is a member in the network. The weights w^P and $w(z)$ are defined below.

$$w^P(u,v,z) = \frac{F(T(u,v,z)+10) + F(T(u,v,z)+30)}{F(T(u,v,z))} \tag{3.6}$$

The function $T(u,v,z)$ retrieves the triad pattern ID (as shown in Figures 3.3 and 3.4) of the triad (u,v,z), while the function $F(...)$ retrieves the frequency of a given triad pattern in the network. Thus, w^P can be be thought of as the proportion of triads of a given pattern that were closed with a link $u \rightarrow v$.

$$TC_{uv} = \sum_{z \in \Gamma(u) \cap \Gamma(v)} w^P(u,v,z) \times \frac{1}{k_z} \tag{3.7}$$

$w(z)$ is defined as the inverse of the degree of z. This weight is meant to express the significance of edges $u \leftrightarrow z$ and $z \leftrightarrow v$ relative to the number of edges connecting z throughout the entire network.

Having collected these triad pattern frequencies, we can now generate recommendations as we did above for user J.

The first step is to identify all open triads of the form u,z,v where J is u and no link between u,z exists. Those triads are:

- (J,F,G)
- (J,I,N)
- (J,P,Q)

Thus the three possible recommendations that we might generate are $J \rightarrow G$, $J \rightarrow N$, and $J \rightarrow Q$. To determine the rank of the predictions, we must calculate the triadic closeness metric for the pairs (J,G), (J,N) and (J,Q).

$$TC_{JG} = \sum_{z \in \Gamma(J) \cap \Gamma(G)} w^P(J,G,z) \times w(z) \tag{3.8}$$

Table 3.1: Open triad pattern frequency in the sample network. This frequencies are used to compute Triadic Closeness.

ID	Pattern	Count
T06	$u \to z \leftarrow v$	20
T04	$u \to z \to v$	14
T08	$u \leftarrow z \leftarrow v$	14
T02	$u \leftrightarrow z \to v$	8
T07	$u \leftarrow z \leftrightarrow v$	8
T09	$u \leftarrow z \to v$	8
T03	$u \to z \leftrightarrow v$	3
T05	$u \leftrightarrow z \leftarrow v$	3

Table 3.2: Closed triad pattern frequency in the sample network.

ID	Pattern	Count
T18	$u \leftarrow z \leftarrow v \leftarrow u$	3
T14	$u \to z \to v \leftarrow u$	2
T16	$u \to z \leftarrow v \leftarrow u$	2
T19	$u \leftarrow z \to v \leftarrow u$	2
T15	$u \leftrightarrow z \leftarrow v \leftarrow u$	1
T17	$u \leftarrow z \leftrightarrow v \leftarrow u$	1
T24	$u \to z \to v \to u$	3
T26	$u \to z \leftarrow v \to u$	2
T28	$u \leftarrow z \leftarrow v \to u$	2
T29	$u \leftarrow z \to v \to u$	2
T22	$u \leftrightarrow z \to v \to u$	1
T23	$u \to z \leftrightarrow v \to u$	1
T34	$u \to z \to v \leftrightarrow u$	1
T38	$u \leftarrow z \leftarrow v \leftrightarrow u$	1

$$TC_{JG} = w^P(J, G, F) \times w(F) \tag{3.9}$$

$$w^P(J,G,F) = \frac{F(T(J,F,G) + 10) + F(T(J,F,G) + 30)}{F(T(J,F,G))} \tag{3.10}$$

We can see that T(J,F,G) = T03, so we have:

$$w^P(J,G,F) = \frac{F(T03+10)+F(T03+30)}{F(T03)} \tag{3.11}$$

$$w^P(J,G,F) = \frac{F(T13)+F(T33)}{F(T03)} \tag{3.12}$$

$$w^P(J,G,F) = \frac{0+0}{3} \tag{3.13}$$

$$w^P(J,G,F) = 0 \tag{3.14}$$

$$TC_{JG} = 0 \tag{3.15}$$

Next for J,N:

$$TC_{JG} = \sum_{z \in \Gamma(J) \cap \Gamma(G)} w^P(J,N,z) \times w(z) \tag{3.16}$$

$$TC_{JG} = w^P(J,I,N) \times w(I) \tag{3.17}$$

$$w^P(J,G,F) = \frac{F(T(J,I,N)+10)+F(T(J,I,N)+30)}{F(T(J,I,N))} \tag{3.18}$$

We can see that T(J,I,N) = T08, so we have:

$$w^P(J,G,F) = \frac{F(T08+10)+F(T08+30)}{F(T08)} \tag{3.19}$$

Using tables 3.1 and 3.2 we can substitute the triad frequencies.

$$w^P(J,G,F) = \frac{3+1}{14} \tag{3.20}$$

$$w^P(J,G,F) = 0.286 \tag{3.21}$$

$$w(I) = 1/3 \tag{3.22}$$

$$TC_{JN} = 0.286 \times 1/3 = 0.095 \tag{3.23}$$

and finally, calculate TC(J,Q):

$$TC_{JQ} = \sum_{z \in \Gamma(J) \cap \Gamma(Q)} w^P(J,Q,z) \times w(z) \tag{3.24}$$

$$TC_{JG} = w^P(J,P,A) \times w(Q) \tag{3.25}$$

$$w^P(J,P,Q) = \frac{F(T(J,P,Q) + 10) + F(T(J,P,Q) + 30)}{F(T(J,P,Q))} \tag{3.26}$$

We can see that T(J,P,Q) = T04, so we have:

$$w^P(J,P,Q) = \frac{F(T04 + 10) + F(T04 + 30)}{F(T04)} \tag{3.27}$$

$$w^P(J,P,Q) = \frac{2 + 0}{14} \tag{3.28}$$

$$w^P(J,P,Q) = 0.143 \tag{3.29}$$

$$w(P) = 1/3 \tag{3.30}$$

$$TC_{JQ} = 0.143 \times 1/3 \tag{3.31}$$

$$TC_{JQ} = .048 \tag{3.32}$$

We can then sort our recommendations by TC and the most likely edge we will recommend is $J \rightarrow Q$.

3.2.3 An adaptive hybrid method

Each of the two methods described above do not fully capture the information needed to make robust recommendations:

- Collaborative filtering captures similar items to identify nodes that a user might find interesting, however the model does not capture any probabilistic information to inform how likely the link is to form, given the relevant/similar node.
- While Triadic Closeness captures probabilistic information that informs how likely certain triad patterns are to close, it is not informed by any user rating observations. This results in predictions based solely on patterns and ignoring content similarity.

We next examine how these two methods can be combined to improve link prediction. [**Lu2010**] identifies such hybrid methods as a way to improve accuracy of link prediction beyond what any one algorithm might be able to obtain.

While such an hybrid method could be defined as a simple weighted average with fixed weights as described in [Can08], we instead propose an adaptive weighting mechanism to take into account the information available for each component of the hybrid metric.

Consider:

$$AEM_{u,v} = \sum_{c \in components} c(u,v) \times w_c(u,v) \tag{3.33}$$

Using Jaccard and Triadic Closeness, this becomes:

$$AEM_{u,v} = J(u,v) \times w_J(u,v) + TC(u,v) \times w_{TC}(u,v) \tag{3.34}$$

Both Jaccard and Triadic Closeness metrics are in the range {0,1} so we do not need to normalize. $J(u,v)$ is described above in Equation XX and $TC(u,v)$ in Equation 3.34, but what values to assign the weights w_J and w_{TC}? Rather than assigning equal weights, the weights should be assigned according to the proportion of our confidence in each metric. Since we are dealing with a multi-modal network, each metric is calculated using a certain relationship type. Here Jaccard is calculated using User-Item *:LIKES*

relationships, while Triadic Closeness is calculated using only User-User *:FOLLOWS* relationships. For a given user, u we can use the proportion of total out-edges of each relationship type as the weight for the corresponding metric.

For example:

$$w_J(u,v) = \frac{count(u - \{:LIKES\} - >)}{count(u - \{*\} - >)} \tag{3.35}$$

$$w_{TC} = \frac{count(u - \{:FOLLOWS\} - >)}{count(u - \{*\} - >)} \tag{3.36}$$

Where $count(u - \{:LIKES\} - >)$ is the number of outgoing *:LIKES* edges for User u and $count(u - \{*\} - >)$ is the total number of outgoing edges (both *:LIKES* and *:FOLLOWS*).

We can think of each outgoing edge as a rating or vote from that user, expressing their interest. If for a given user we observe 10 *:FOLLOWS* edges and only 1 *:LIKES* edge, we have greater confidence in the accuracy of the Triadic Closeness metric since we have more information about User u's preferences.

3.3 Data

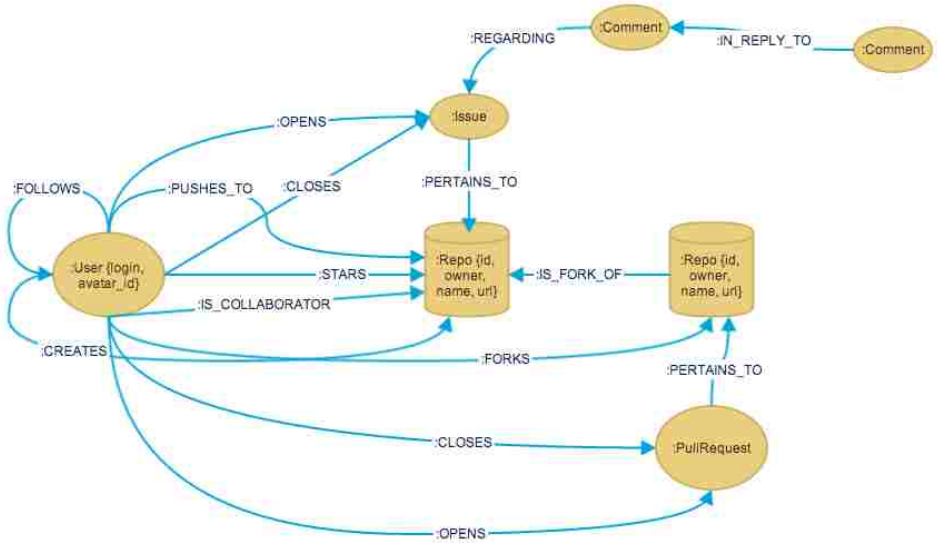


Figure 3.5: GitHub data model as a labeled property graph.

GitHub [1] is an online social collaboration network built around the Git version control system. GitHub allows software developers to share and collaborate on software projects. Many open-source software projects are hosted on GitHub and use GitHub as their primary development and distribution platform. GitHub also has a social component: users are able to *follow* other users to receive updates about user activity. This combination of social and collaboration components make GitHub an excellent example of a multi-modal complex network.

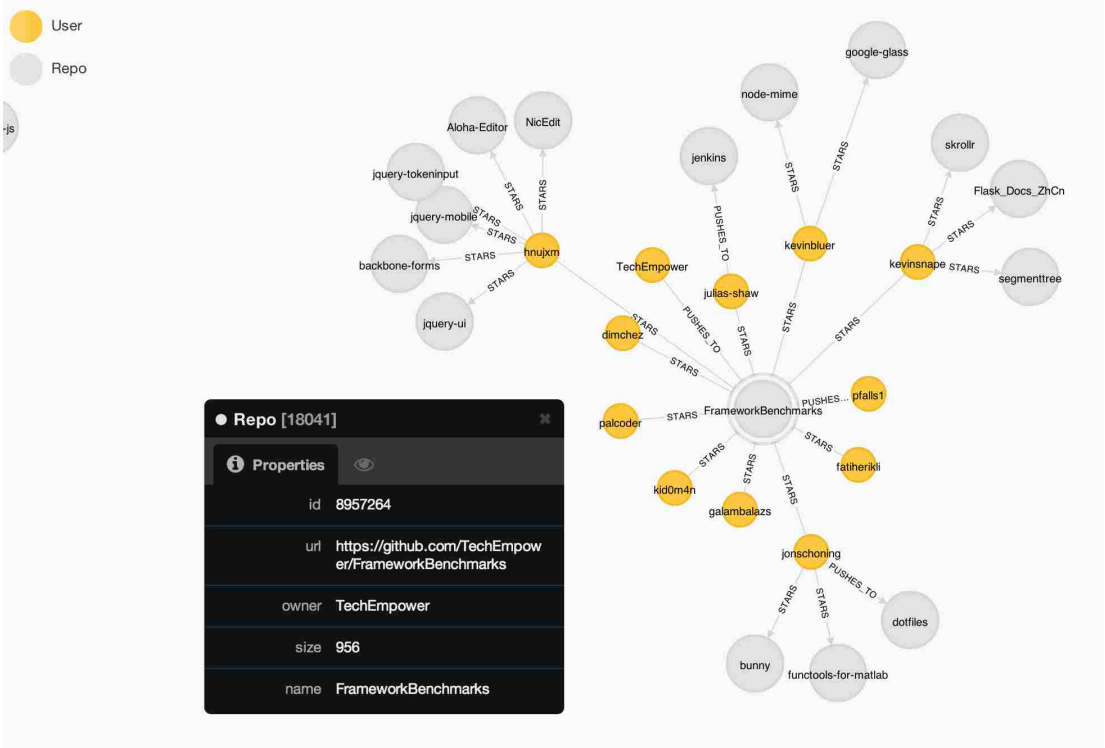


Figure 3.6: GitHub data model as a property graph. Screenshot from Neo4j graph database interface.

The GitHub network is multi-modal in that there are multiple nodes types (primarily Users and Repositories, or software projects) and multiple edge types (User-User follows, User-Repository Stars, etc). Figure 3.5 shows a portion of the GitHub data modeled as a graph. This data model is quite rich, however for the purposes of this paper we will only concern ourselves with User-User *:FOLLOWS* and User-Repository *:STARS* edges. That simplifies the data model to that shown in Figure 3.8.

1 http://github.com

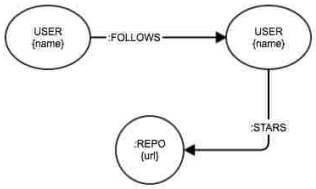


Figure 3.7: The GitHub follow graph is a simple graph with User nodes and Follows edges.

3.3.1 Github Archive

Data was collected from GitHub Archive[Gita], a service that maintains an archive of all public events emitted by the GitHub API[Gitb]. These include events such as creation of new repositories, pushes to repositories, repository stars, and user follows. Data was collected for the time period April 1st, 2013 - April 1st, 2014. Table 3.3 shows summary statistics about the size of the network built from this data. It is important to note that this data represents only a sample of the network, not the complete GitHub network.

A graph data model is used to represent this data as the data is highly connected: it is describing entities (users and repositories) and their interactions (stars, follows, pushes, etc). Figure 3.6 shows an example of a subgraph of user and repository nodes and the interactions among those entities, modeled as a graph.

FollowEvent

The data collected from GitHub Archive is in the streaming JSON format. An example of a User-User folow event is shown below:

```
{
  "created_at": "2013-07-11T15:03:05-07:00",
  "payload": {
    "target": {
      "id": 4602587,
      "login": "smarquez1",
      "followers": 1,
      "repos": 1,
      "gravatar_id": "42eb6556201588fa7641bf2f0bf615e6"
    }
  },
  "public": true,
  "type": "FollowEvent",
  "url": "https://github.com/smarquez1",
  "actor": "matiasalvarez87",
  "actor_attributes": {
    "login": "matiasalvarez87",
    "type": "User",
```

```
    "gravatar_id": "0ee1a5bec013545c91ad05c451fb9715",
    "name": "Matias Alvarez Duran",
    "company": "NaN Labs",
    "blog": "http://ar.linkedin.com/pub/matias-emiliano-alvarez-duran/17/39b/a96",
    "location": "Argentina",
    "email": "matiasalvarez87@gmail.com"
  }
}
```

Listing 3.1: JSON document example of data point - Follow event

WatchEvent (Stars)

User-Repository *stars* event data is formatted as in this example:

```
{
  "created_at": "2013-07-11T15:01:56-07:00",
  "payload": {
    "action": "started"
  },
  "public": true,
  "type": "WatchEvent",
  "url": "https://github.com/CamDavidsonPilon/Probabilistic-Programming-and-Bayesian-Methods-for-Hackers",
  "actor": "cebe",
  "actor_attributes": {
    "login": "cebe",
    "type": "User",
    "gravatar_id": "2ebfe57beabd0b9f8eb9ded1237a275d",
    "name": "Carsten Brandt",
    "company": "cebe.cc",
    "blog": "http://cebe.cc/",
    "location": "Berlin, Germany",
    "email": "mail@cebe.cc"
  },
  "repository": {
    "id": 7607075,
    "name": "Probabilistic-Programming-and-Bayesian-Methods-for-Hackers",
    "url": "https://github.com/CamDavidsonPilon/Probabilistic-Programming-and-Bayesian-Methods-for-Hackers",
    "description": "aka \"Bayesian Methods for Hackers\": An introduction to Bayesian methods + probabilistic programming with a computation/understanding-first, mathematics-second point of view. All in pure Python ;)  ",
    "homepage": "http://camdavidsonpilon.github.io/Probabilistic-Programming-and-Bayesian-Methods-for-Hackers",
    "watchers": 3353,
    "stargazers": 3353,
```

```
    "forks": 444,
    "fork": false,
    "size": 1264,
    "owner": "CamDavidsonPilon",
    "private": false,
    "open_issues": 21,
    "has_issues": true,
    "has_downloads": true,
    "has_wiki": true,
    "language": "Python",
    "created_at": "2013-01-14T07:46:28-08:00",
    "pushed_at": "2013-07-04T17:08:47-07:00",
    "master_branch": "master"
  }
}
```


Listing 3.2: JSON document example of data point - Watch Event

3.3.2 Data Analysis

Table 3.3 shows some descriptive statistics about the GitHub dataset used for this project. The observed network consists of 1.7 million nodes (this includes both Users and Repositories) and 10.7 million edges (both User-User *follows* edges and User-Repository *stars* edges.

Table 3.3: Summary statistics for the data collected from GitHub Archive

	Count
Num nodes	1,751,605
Num edges	10,740,463
Mean degree	6.13
Num $USER$ nodes	871,382
Num $REPO$ nodes	880,223
Num : $FOLLOWS$ edges	1,120,069
Num : $STARS$ edges	9,620,394
Mean : $FOLLOWS$ degree	1.29
Mean : $STARS$ degree	11.04

As an initial sanity check for exploring the data we compute PageRank for the observed GitHub network. PageRank is a link analysis algorithm that assigns weights to nodes in a network based on that nodes relative importance, or centrality in the network. [Pag] It is similar to a measure of eigenvector centrality. PageRank has the weakness of only being applicable to a homogenous or single mode network, therefore we apply the

algorithm to the User-User *follows* subgraph and the User-Repository *stars* subgraph separately. The GraphLab PageRank algorithm implementation is used [**GraphLab**].

Table 3.4: Here we see the most "central" Users per their PageRank rankings. This is based on the graph created by user-user follows edges.

User	PageRank
funkenstein	413.14
mojombo	300.01
torvalds	248.21
rippleFoundation	220.29
visionmedia	140.52
paulirish	129.59
BYVoid	114.29
schacon	112.17
JakeWharton	110.55
defunkt	106.86
mattt	99.38
worrydream	87.33
hakimel	83.05
pjhyett	80.89
addyosmani	80.59
mbostock	75.63
mdo	70.40
LeaVerou	66.92
tekkub	62.24
nf	60.93

Tables 3.4 and 3.5 show the highest ranked Users and Repositories for the GitHub data collected.

Table 3.4 shows the results of the PageRank algorithm using User-User *follows* relationships only. As a sanity check of the data, we expect the highest ranked Users to be influential developers in the open-source software development community - the nodes with highest relative importance in this context. A cursory evaluation of the list of GitHub usernames confirms that this is indeed the case:

mojombo Tom Preston-Werner, a co-founder and developer of GitHub

torvalds Linus Torvalds, maintainer of the linux operating system kernel

Table 3.5: Top 20 central GitHub repositories by PageRank.

Repository	PageRank
https://github.com/vhf/free-programming-books	455.07
https://github.com/twbs/bootstrap	335.46
https://github.com/jquery/jquery	289.42
https://github.com/resume/resume.github.com	251.99
https://github.com/mandatoryprogrammer/Octodog	233.41
https://github.com/angular/angular.js	202.18
https://github.com/mbostock/d3	149.55
https://github.com/torvalds/linux	133.48
https://github.com/FortAwesome/Font-Awesome	121.47
https://github.com/twitter/bootstrap	111.42
https://github.com/laravel/laravel	106.75
https://github.com/papers-we-love/papers-we-love	102.25
https://github.com/joyent/node	101.27
https://github.com/rethinkdb/rethinkdb	92.34
https://github.com/neovim/neovim	91.52
https://github.com/libgit2/libgit2	90.99
https://github.com/rogerwang/node-webkit	88.23
https://github.com/github/gitignore	88.08
https://github.com/dypsilon/frontend-dev-bookmarks	86.73
https://github.com/zurb/foundation	84.25

paulirish Paul Irish, a well known Google developer

mbostock Mike Bostock, core developer of d3.js a popular JavaScript data visualization library

mdo Mark Otto, one of the developers of Bootstrap the widely used frontend CSS/-Javascript framework

These are indeed software developers that could be considered very influential to the open source software community.

Similarly, Table 3.5 enumerates the 20 most central GitHub software repositories, according to PageRank. Here we expect to see influential and widely used open-source software projects. Scanning through the list we are able to identify several widely used and important projects:

/twbs/bootstrap Twitter Bootstrap, a popular frontend framework

/angular/angular.js A widely used JavaScript framework developed by Google

/torvalds/linux The linux operating system kernel

/laravel/laravel An enterprise PHP framework

/joyent/node The infamous node.js project

3.4 Implementation

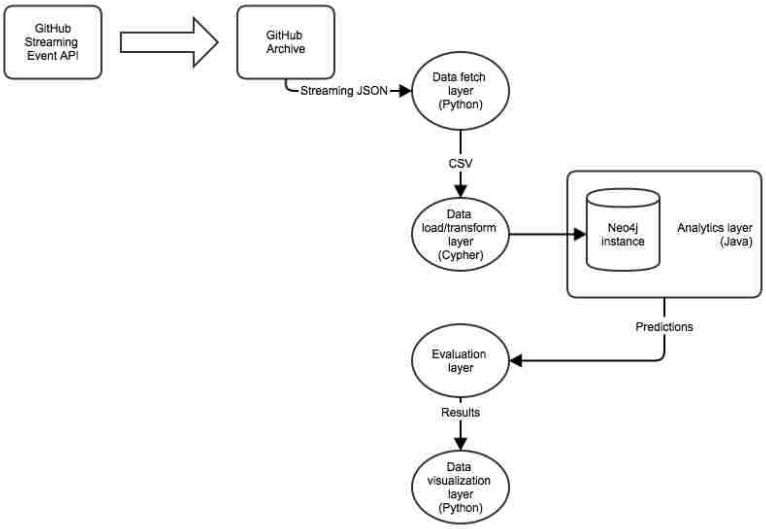


Figure 3.8: The GitHub follow graph is a simple graph with User nodes and Follows edges.

We implement a system capable of generating recommendations using each of the three methods described above (Jaccard similarity, Triadic Closeness, and an adaptive hybrid method). This system is written in Java and makes use of the Neo4j graph database.

3.4.1 Architecture

We design the system to handle large amounts of data, beyond what will fit into memory on a single machine. When developing such a system the tradeoffs become:

- *Complexity of the system.* A distributed cluster allows for more efficient in-memory processing performance, however at the expense of complexity. Designing

algorithms for distributed systems is complicated, as is running and maintaining such systems. If data is to be distributed across multiple instances, ineviatably there is a tradeoff between data duplication and network latency as highly connected graph data such as complex networks typically cannot be partitioned completely.

- *Memory access.* If we make the assumption that the data will not fit into memory, then we need some sort of persistence layer. Accessing this persistence layer will dramatically reduce processing performance relative to in-memory computation and so this persistence layer must be designed to optimize our specific data access use-cases.

Data collection layer

As mentioned in the previous section, the data for this experiment is collected from GitHub Archive. One year of events are downloaded for analysis. The data collection layer handles querying GitHub archive streaming JSON data, filtering for events by type and converting the data into flat CSV formatted files. These CSV files are then efficiently loaded into a Neo4j graph database instance. The data collection layer is implemented in Python.

Analytics layer

The analytics layer is capable of running link prediction experiments with validation or generating ad-hoc link predictions for a given user node in the network. Queries to the embedded Neo4j instance are done using the Cypher query language and are listed below. The analytics layer is implemented in Java.

```
MATCH (u1:User {name: {u1}}), (u2:User { name:{u2}})
MATCH (u1)-[:STARS]->(x:Repo)<-[:STARS]-(u2) WITH x, u1, u2
WITH count(x) as intersect, u1, u2
//MATCH (u1)-[r:STARS]->(intersection:Repo)<-[:STARS]-(u2) WITH u1, u2,
intersection LIMIT 10
//WITH count(intersection) as intersect, u1, u2
MATCH (u1)-[r:STARS]->(rest1) WITH u1, u2, intersect, collect(DISTINCT rest1)
AS coll1
MATCH (u2)-[r:STARS]->(rest2) WITH u1, u2, collect(DISTINCT rest2) AS coll2,
coll1, intersect
WITH u1, u2, intersect, coll1, coll2, length(coll1 + filter(x IN coll2 WHERE
NOT x IN coll1)) as union
WITH u1, u2, (1.0*intersect/union) as jaccard
RETURN jaccard
```

Listing 3.3: Cypher query for computing Jaccard similarity

```
// Identify triad pattern
// u<->z<->v
// to calc TC for u<->v
//MATCH (u:User {id: 'a'})
```

```
//MATCH (z:User {name: 'c'})
//MATCH (v:User {name: 'd'})
MATCH (u:User), (z:User), (v:User) WHERE u<>v AND v<>z AND z<>u // for all 3 node combos
MATCH (u)--(z)--(v) // find triads only
//MATCH (u)-->(v) // closed triads only
WITH DISTINCT u, z, v
OPTIONAL MATCH t01=(z)-->(u)-->(z)-->(v)-->(z) WHERE NOT (u)--(v)
OPTIONAL MATCH t02=(z)-->(u)-->(z)-->(v) WHERE NOT (u)--(v)
OPTIONAL MATCH t03=(u)-->(z)-->(v)-->(z) WHERE NOT (u)--(v)
OPTIONAL MATCH t04=(u)-->(z)-->(v) WHERE NOT (u)--(v)
OPTIONAL MATCH t05=(v)-->(z)-->(u)-->(z) WHERE NOT (u)--(v)
OPTIONAL MATCH t06=(u)-->(z)<--(v) WHERE NOT (u)--(v)
OPTIONAL MATCH t07=(u)<--(z)-->(v)-->(z) WHERE NOT (u)--(v)
OPTIONAL MATCH t08=(v)-->(z)-->(u) WHERE NOT (u)--(v)
OPTIONAL MATCH t09=(u)<--(z)-->(v) WHERE NOT (u)--(v)
WITH
CASE
 WHEN t01 IS NOT NULL THEN 't01'
 WHEN t02 IS NOT NULL THEN 't02'
 WHEN t03 IS NOT NULL THEN 't03'
 WHEN t04 IS NOT NULL THEN 't04'
 WHEN t05 IS NOT NULL THEN 't05'
 WHEN t06 IS NOT NULL THEN 't06'
 WHEN t07 IS NOT NULL THEN 't07'
 WHEN t08 IS NOT NULL THEN 't08'
 WHEN t09 IS NOT NULL THEN 't09'
END
AS type
WITH collect(type) AS types
WITH types//, length(types) as triadcount
WITH
  [x IN types WHERE x = 't01' | x] AS t01_c,
  [x IN types WHERE x = 't02' | x] AS t02_c,
  [x IN types WHERE x = 't03' | x] AS t03_c,
  [x IN types WHERE x = 't04' | x] AS t04_c,
  [x IN types WHERE x = 't05' | x] AS t05_c,
  [x IN types WHERE x = 't06' | x] AS t06_c,
  [x IN types WHERE x = 't07' | x] AS t07_c,
  [x IN types WHERE x = 't08' | x] AS t08_c,
  [x IN types WHERE x = 't09' | x] AS t09_c
RETURN
  // divide by triadcount for frequency?
  1.0*length(t01_c) AS t01,
  1.0*length(t02_c) AS t02,
  1.0*length(t03_c) AS t03,
  1.0*length(t04_c) AS t04,
```

```
  1.0*length(t05_c) AS t05,
  1.0*length(t06_c) AS t06,
  1.0*length(t07_c) AS t07,
  1.0*length(t08_c) AS t08,
  1.0*length(t09_c) AS t09;
```

Listing 3.4: Cypher query for computing graph triad pattern frequencies

```
// Identify triad pattern
// u<->z<->v
// to calc TC for u<->v
MATCH (u:User {name: {u} })
MATCH (z:User {name: {z} })
MATCH (v:User {name: {v} })

OPTIONAL MATCH t01=(z)-->(u)-->(z)-->(v)-->(z)
OPTIONAL MATCH t02=(z)-->(u)-->(z)-->(v)
OPTIONAL MATCH t03=(u)-->(z)-->(v)-->(z)
OPTIONAL MATCH t04=(u)-->(z)-->(v)
OPTIONAL MATCH t05=(v)-->(z)-->(u)-->(z)
OPTIONAL MATCH t06=(u)-->(z)<--(v)
OPTIONAL MATCH t07=(u)<--(z)-->(v)-->(z)
OPTIONAL MATCH t08=(v)-->(z)-->(u)
OPTIONAL MATCH t09=(u)<--(z)-->(v)

WITH
CASE
 WHEN t01 IS NOT NULL THEN 't01'
 WHEN t02 IS NOT NULL THEN 't02'
 WHEN t03 IS NOT NULL THEN 't03'
 WHEN t04 IS NOT NULL THEN 't04'
 WHEN t05 IS NOT NULL THEN 't05'
 WHEN t06 IS NOT NULL THEN 't06'
 WHEN t07 IS NOT NULL THEN 't07'
 WHEN t08 IS NOT NULL THEN 't08'
 WHEN t09 IS NOT NULL THEN 't09'
END
AS type
RETURN type
```

Listing 3.5: Cypher query for triad pattern identification

Evaluation and visualization layer

Prediction results from the analytics layer are fed into the evaluation and visualization layer. Evaluation metrics are computed and data visualizations are created for evaluation. This layer is implemented in Python.

CHAPTER 4

Evaluation

The result of the algorithm is a set of user IDs that are predicted destination nodes, given a specific source node. These are recommended users that the specified user might be interested in following.

4.1 Evaluation Metrics

Each method described above in Chapter 3 is evaluated using the same evaluation metrics for comparison. They are as follows.

4.1.1 Precision

In this context, precision is defined as the number of relevant links predicted divided by the total number of link predicted. A link is said to be relevant if it is one of the existing but removed links.

4.1.2 HitRatio@N (Recall)

The other evaluation metric we will use is the *HitRatio@N*, where N is the number of predicted links for a given user (at one iteration of the validation). This metric is defined as the number of users for which at least one correct link was predicted, divided by the total number of users for which predictions were generated.

4.1.3 Evaluation

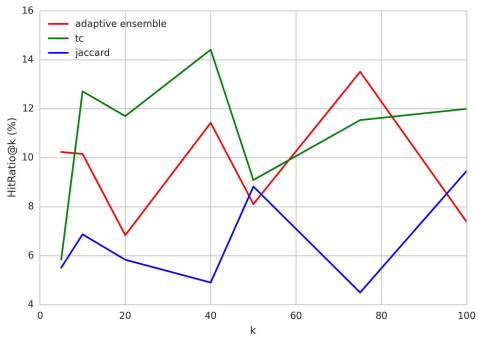
The system is evaluated using 200 iterations for validation with different configurations for N, where N is the number of predicted links generated for each user sampled. At each validation run a user is selected at random (cross-validation is used to avoid a statistical bias from users being selected multiple times). Tables 4.1 and 4.2 summarize the results. We can see that all three methods perform much better than the random selection baseline. Jaccard similarity does not perform as well as triadic closeness, which is expected based on the results presented in [Sch14]. Our adaptive ensemble method however, does not perform any better than the triadic closeness method and for some configurations performs more poorly than triadic closeness.

Table 4.1: HitRatio@N (%) results for Jaccard similarity (JS), Triadic Closeness (TC) and adaptive ensemble (AE) methods. Random probability is shown for comparison.

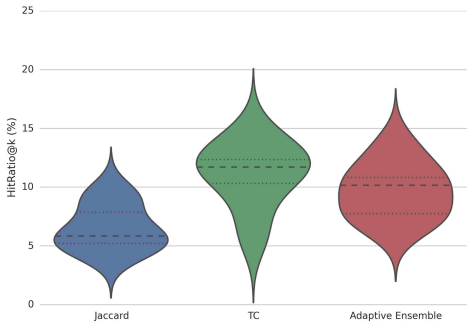
N	JS (%)	TC (%)	AE (%)	Random (%)
5	5.51	5.84	10.24	5.738×10^{-4}
10	6.87	12.71	10.15	1.1476×10^{-3}
20	5.83	11.70	6.84	2.2952×10^{-3}
40	4.90	14.41	11.43	4.5904×10^{-3}
50	8.82	9.09	8.11	5.738×10^{-3}
75	4.49	11.54	13.52	8.607×10^{-3}
100	9.47	12.00	7.37	1.14×10^{-2}
250	8.86	10.71	8.33	2.869×10^{-2}
500	12.86	22.86	12.70	5.738×10^{-2}

Table 4.2: Precision (%) results for Jaccard similarity, Triadic Closeness and adaptive ensemble methods.

N	JS	TC	AE
5	1.178×10^{-2}	1.2084×10^{-2}	2.113×10^{-2}
10	7.43×10^{-3}	1.346×10^{-2}	1.092×10^{-2}
20	3.16×10^{-3}	$6.232\,29 \times 10^{-3}$	3.6363×10^{-3}
40	1.366×10^{-3}	3.9206×10^{-3}	3.2867×10^{-3}
50	2.046×10^{-3}	2.2596×10^{-3}	$1.905\,97 \times 10^{-3}$
75	6.97×10^{-4}	$1.757\,21 \times 10^{-3}$	2.0024×10^{-3}
100	1.152×10^{-3}	$1.405\,81 \times 10^{-3}$	7.989×10^{-4}
250	4.329×10^{-4}	5.798×10^{-4}	4.1694×10^{-4}
500	3.856×10^{-4}	5.874×10^{-4}	3.608×10^{-4}

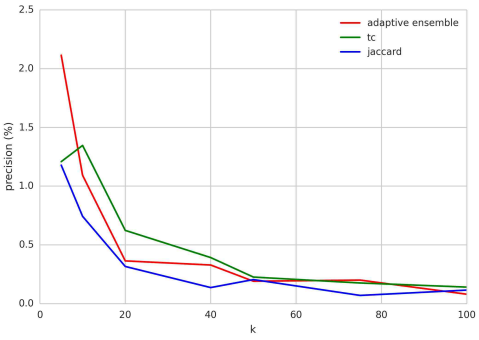


(a) Plot of precision results.

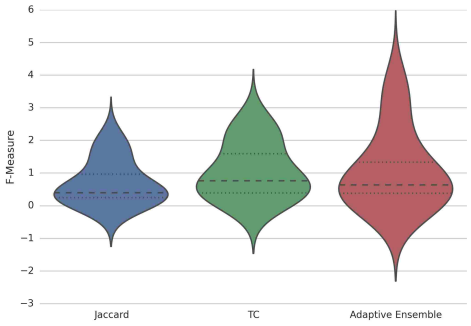


(b) Violin plot of precision results.

Figure 4.1: Plots comparing precision results for Jaccard similarity, triad closeness and adaptive ensemble methods.



(a) Plot of HitRatio@N results.



(b) Violin plot of HitRatio@N results.

Figure 4.2: Plots of HitRatio@N results for Jaccard similarity, triadic closeness and adaptive ensemble methods.

CHAPTER 5

Summary and Outlook

This paper has explored the link prediction problem as applied to complex networks. An adaptive hybrid method is developed and evaluated against data from the GitHub network. We find that Jaccard similarity performs much better than the random benchmark and that Triadic Closeness performs better than Jaccard similarity. The adaptive hybrid method developed here performs no better than triadic closeness, and in some cases performs worse.

The relatively poor performance of the adaptive hybrid model could potentially be improved by adding a normalization process to the weights of each component and by adding additional components. Due to the structure of the network data used, the number of relationships observed for each type of relationship (*follows* and *stars*) are disproportional. There are approximately ten times as many *stars* relationships in the data set as *follows* relationships. This skew should be taken into account with a normalization process to avoid one component of the hybrid model skewing the metric. Finally, additional components could be added to the hybrid model to improve accuracy.

5.1 Further research

The GitHub network provides a rich data set for analysis. More data from this network should be included in the data model for the adaptive hybrid method. This could perhaps improve performance, but also link prediction for other edge types should be explored.

The methods presented here should be extended to other datasets. Online social collaboration networks such as AngelList and CrunchBase provide open API access and data exports for research. These networks provide a rich dataset with data about the founding of startup companies, employment and venture funding for startups. The application of the methods presented here could allow for predictions about who will leave their current jobs to found startups and who will fund them. In fact, data analysts are already exploring these type of predictions [**mattermark**].

The system implemented for this paper was designed to be as scalable as possible, making the assumption that a single machine would be used for analysis (instead of a

distributed cluster). This design consideration severely limited the performance of such a system. While it was possible to complete the analysis using the system as designed, for larger datasets a distributed graph processing engine such as SparkX [**sparkx**] or the more recently developed techniques of on-disk processing which take advantage of performant solid state drives, such as GraphChi [**graphChi**] should be considered.

Bibliography

[Gita] URL: http://www.githubarchive.org/ (cit. on p. 20).

[Gitb] URL: https://developer.github.com/v3/activity/events/types/ (cit. on p. 20).

[Agg11] Charu C. Aggarwal: *Social Network Data Analytics*. London: Springer, 2011.

[Aie12] Luca Maria Aiello, Alain Barrat, Rossano Schifanella, Ciro Cattuto, Benjamin Markines, and Filippo Menczer: 'Friendship prediction and homophily in social media'. In*ACM Transactions on the Web* (May 2012), vol. 6(2): pp. 1–33.

[Brz09] Michael J Brzozowski and Daniel M Romero: 'Who Should I Follow ? Recommending People in Directed Social Networks '. In*Proceedings of the Fifth International AAAI Conference on Weblogs and Social Media*. Brzozowski. Associate for the Advancement of Artificial Intelligence, 2009: pp. 458–461.

[Can08] Laurent Candillier and Frank Meyer: 'Designing Specific Weighted Similarity Measures to Improve Collaborative Filtering Systems'. In*Advances in Data Mining. Medical Applications, E-Commerce, Marketing, and Theoretical Aspects Lecture Notes in Computer Science* (2008), vol. 5077: pp. 242–255 (cit. on pp. 6, 17).

[Che] Charalampos Chelmis, Viktor K Prasanna, and Southern California: 'Social Link Prediction in Online Social Tagging Systems'. In*ACM Transactions on Information Systems* (), vol. V(212): pp. 1–27.

[Chi] Nitin Chiluka, Nazareno Andrade, and Johan Pouwelse: 'A Link Prediction Approach to Recommendations in Large-Scale User-Generated Content Systems'. In (), vol.

[Dav11] Darcy Davis, Ryan Lichtenwalter, and Nitesh V. Chawla: 'Multi-relational Link Prediction in Heterogeneous Information Networks'. In*2011 International Conference on Advances in Social Networks Analysis and Mining* (July 2011), vol.: pp. 281–288.

[Du10] Nan Du, Hao Wang, and Christos Faloutsos: 'Analysis of Large Multimodal Social Networks : Patterns and a Generator'. In*Machine Learning and Knowledge Discovery in Databases Lecture Notes in Computer Science* (2010), vol. 6321: pp. 1–16.

[Faw06] Tom Fawcett: 'An introduction to ROC analysis'. In*Pattern Recognition Letters* (June 2006), vol. 27(8): pp. 861–874.

[Lem01] R Lempel and S Moran: 'SALSA: The Stochastic Approach for Link-Structure Analysis'. In*ACM Transactions on Information Systems* (2001), vol. 19(2): pp. 131–160 (cit. on p. 5).

[LN07] David Liben-Nowell and Jon Kleinberg: 'The Link-Prediction Problem for Social Networks'. In*Journal of the American Society For Information Science and Technology* (2007), vol. 58(7): pp. 1019–1031 (cit. on pp. 3, 5).

[Lic10] Ryan N Lichtenwalter, Jake T Lussier, and Nitesh V Chawla: 'New Perspectives and Methods in Link Prediction'. In*KDD 2010*. ACM, 2010: pp. 243–252.

[Lin13] Jimmy Lin and Dmitriy Ryaboy: 'Scaling Big Data Mining Infrastructure : The Twitter Experience'. In*SIGKDD Explorations* (2013), vol. 14(2): pp. 6–19.

[Lu10] Linyuan Lu and Tao Zhou: 'Link Prediction in Complex Networks: A Survey'. In*arXiv:1010.0725v1* (2010), vol. (cit. on pp. 4, 6).

[LÖ9] Linyuan Lü and Tao Zhou: 'Role of weak ties in link prediction of complex networks'. In*Proceeding of the 1st ACM international workshop on Complex networks meet information & knowledge management - CNIKM '09* (2009), vol.: p. 55.

[Pan13] Reza Zadeh Pankaj Gupta, Ashish Goel, Jimmy Lin, Aneesh Sharma, DongWang: 'WTF: The Who to Follow Service at Twitter'. In*International World Wide Web Conference Committee*. 2013 (cit. on pp. 1, 3, 5).

[Ric11] Franceso Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor: *Recommender Systems Handbook*. London: Springer, 2011 (cit. on pp. 4, 5, 8).

[Rod09] Marko A Rodriguez, David W Allen, Joshua Shinavier, and Gary Ebersole: 'A Recommender System to Support the Scholarly Communication Process'. In*arXiv preprint* (2009), vol. arXiv 0905.1594v1.

[Rod10] Marko A Rodriguez and Peter Neubauer: 'The Graph Traversal Pattern'. In*arXiv preprint* (2010), vol. arXiv:1004.1001v1: pp. 1–18 (cit. on p. 5).

[Sch14] Daniel Schall: 'Link prediction in directed social networks'. In*Social Network Analysis and Mining* (Feb. 2014), vol. 4(1): p. 157 (cit. on pp. 6, 8, 10, 11, 29).

[Ste10] Maarten van Steen: *Graph Theory and Complex Networks: An Introduction*. 2010 (cit. on pp. 1, 7, 10).

[Sym11] Panagiotis Symeonidis, Eleftherios Tiakas, and Yannis Manolopoulos: 'Product recommendation and rating prediction based on multi-modal social networks'. In*Proceedings of the fifth ACM conference on Recommender systems - RecSys '11* (2011), vol.: p. 61.

[Pag] 'The PageRank Citation Ranking: Bringing Order to the Web'. In (1998), vol.: pp. 1–17 (cit. on pp. 5, 22).

[Wan] Xi Wang and Gita Sukthankar: *Link Prediction in Heterogeneous Collaboration Networks*.

[Xin13] Reynold S Xin, Joseph E Gonzalez, Michael J Franklin, Ion Stoica, and U C Berkeley: 'GraphX : A Resilient Distributed Graph System on Spark'. In. 2013.

List of Figures

List of Tables

Listings