An Adaptive Ensemble Method For Link Prediction In Multi-Modal Directed Social Collaboration Networks Using The Graph Traversal Pattern

Masters Thesis
University of Montana

William Lyon

October 14, 2014

This page is probably the biographical bs stuff

William Lyon

Link Prediction In Directed Collaboration Networks

Keywords keywords here, probably remove this section

Abstract
William Lyon

Abstract here. 350 words I guess.

Key words: <insert key words>

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.

<insert data and location> <insert full name>

Contents

CHAPTER 1

Introduction

Social networks are everywhere. Facebook, Twitter, LinkedIn, etc...

1.1 Online Social Collaboration Networks

Social networks based on our activity rather than who we know. Consider GitHub: Github is a software collaboration web application based on the git version control system. Using GitHub software developers can collaborate on projects, share software projects and follow what other developers are working on. GitHub can be thought of as a social collaboration network, a type of network that has both collaboration and social interactions between users. [Goo94]

1.2 This project

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) 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.

The remainder of this paper is outlined as follows:

Chapter 2 - Previous Work A review of the literature in this field

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.

Chapter 5 - Summary Areas of further research are discussed.

Figure **??** shows the GitHub network modeled as a property graph [Rod]. This project is concerned with only the *FOLLOWS* relationship: GitHub users can follow other users to express interest in another user and receive updates about that user's actions in the GitHub system. A user recommendation system is implemented which generates recommendations of other users to follow on GitHub. Since user recommendations are a prediction of future action, this can be thought of as a link prediction problem: given a vertex in the graph, can we predict which edges will be added in the future? Specifically, we are predicting *FOLLOWS* edges for specific users. A user-user collaborative filtering algorithm is implemented using the graph traversal pattern [Rod] and evaluated using a leave-one-out method.

CHAPTER 2

Previous Work

Recommender system literature can typically be divided into two categories: 1) content (or item) recommendation and 2) user recommendation. An example of content recommender systems is Amazon product recommendations. These take the form of "users who bought x also bought y". An example of user recommendation is LinkedIn's "Do You Know..." feature, which is suggestingThe latter can be generalized as "Who To Follow" recommender systems and it is this type of recommendation on which this paper will focus.

2.1 Recommendation As Link Prediction

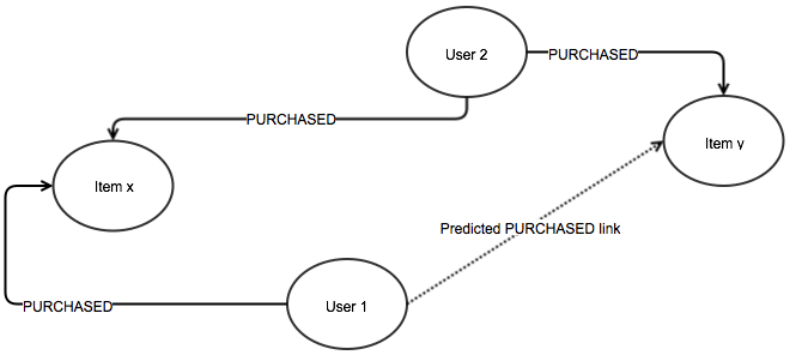


Figure 2.1: Here we see an example of an item recommender system modeled as a graph where the nodes are Users and Items and the edges indicate a purchse of an Item by a User. When modeled this way recommendation takes the form a predicted link in the graph.

2.2 User-User Recommender Systems

There are essentially two types of User-User recommender system algorithms: 1) Similarity based methods and 2) Network based methods.(CITATION NEEDED) 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 homopholy principle, that users are more likely to be interested in users similar to them. Network based methods analyze the structure of the network to develop recommendations. Example include PageRank, HITS, and SALSA (CITATION NEEDED).

2.2.1 Similarity Based Methods

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[**cf**]. 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 [Rod].

2.2.2 Network Based Methods

User-User recommender systems are often generalized as "Who To Follow" recommender systems, as made popular by the "Follow" social network action in the Twitter social network. A Follow action indicates a user expressing interest in another user in the network. This often involves

The Twitter Who To Follow system is an example of a link prediction system running in production. [**wtf**] details the design and implementation considerations for such a large scale system. The Twitter system operates on a custom in-memory graph processing engine which implements a PageRank-type algorithm known as Stochastic Approach for Link-Structure Analysis (SALSA). SALSA uses a random walk of the graph to generate link predictions.

2.3 Directed Social Networks

There is an important distinction to note between undirected social networks and directed social networks. Examples/explanation. Much of the literature has focused on undirected social networks only (CITATION NEEDED). In fact the similarity metrics shown above are all based on undirected social networks. A notable exception in the Triadic Closeness method described by (CITATION NEEDED).

2.4 My Approach

There is a clear gap in the literature making use of combined network based methods with similarity based methods. (CITATION Ensemble paper) 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 levering the use of directed networks is proposed, based somewhat on the work developed in (CITATION paper about novel similarity metric, weighted averaging).

CHAPTER 3

Methods

Here we describe the data used and detail the implementation. We first examine in detail two methods for link prediction: collaborative filtering using the Jaccard similarity metric and the Triadic Closeness method. We then show how these two methods can be used together in an ensemble predictor using adaptive weights. 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 form 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, as we will see in more detail). Since the network has multiple types of nodes and edges it is referred to as a **multi-modal network**. (CITE GRAPH THEORY / COMPLEX NETWORK BOOK).

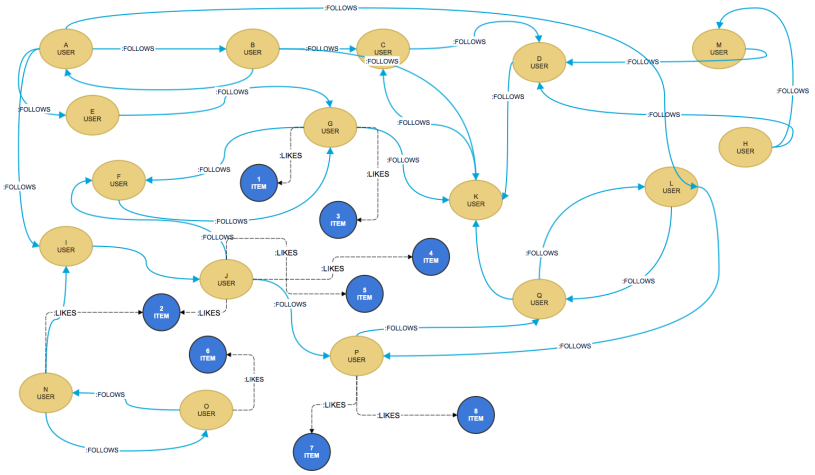


Figure 3.1: This sample network will be used to demonstrate the methods used for link prediction in this paper. This network demonstrates a random multi-modal network with multiple types of nodes and edges.

Some basic descriptive statistics about this sample network are shown in Table 3.1.

Table 3.1: Descriptive statistics for sample network

Metric	value
Num nodes	17
Num edges	28
Avg degree	1.67
OTHERS	??

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 ensemble 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. [**cf**] 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.

FIXME: THIS NEEDS TO BE UPDATED FOR NEW DATA MODEL

To generate recommendations for user J, we first must identify all friend-of-friend

Algorithm 1 Collaborative filtering algorithm

```
1: input : G(U, E), x, N                                   ▷ explain inputs here
2: usersSample ← getRandomUsers(U,x)
3: results ← {}
4: for eachuserinusersSample do
5:     validationEdge ← getRandomEdge(G, user)
6:     removeEdge(validationEdge,G)
7:     fofs ← getFOFS(user, G)
8:     fofRanks ← {}
9:     for each fof in fofs do
10:        j ← jaccard(user, fof) fofRanks ← results + {j : fof}
11:    end for
12:    knn ← topk(fofRanks, k)
13:    aggregated ← {}
14:    for each kn in knn do
15:        possible ← getFollows(kn)
16:        for each p in possible do
17:            if p in aggregated.keys then aggregated[p]+ = 1
18:            else
19:                aggregated[p] = 1
20:            end if
21:        end for
22:    end for
23:    predictions ← topXSortedByTC(pred, N)
24:    hit ← isvalidationEdgeinpredictions?
25:    addEdge(validationEdge, G)
26:    results ← results + {hit, pred, u, v, validation_edge}
27: end for
28: return results
```

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):

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

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

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

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

As you can see, the collaborative filtering link prediction process for a given user x 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. (CITATION NEEDED and expand on this - see one of the network analysis books I have) 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.

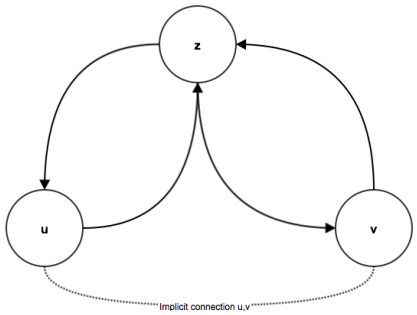


Figure 3.2: A triad is considered to be closed if an edge exists between u and v.

In a directed network there are 27 distinct configurations, or patterns that a triad can

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.

Algorithm 2 Triadic Closeness link prediction algorithm

```
1: input : G(U, E), x, N                                   ▷ explain inputs here
2: usersSample ← getRandomUsers(U,x)
3: results ← {}
4: for eachuserinusersSample do
5:     validationEdge ← getRandomEdge(G, user)
6:     removeEdge(validationEdge,G)
7:     triads ← getTriads(user, G)
8:     pred ← {}
9:     for u, vintriads do
10:        tc ← calcTC(u,v,G)
11:        pred ← pred + {tc, u, v}
12:    end for
13:    predictions ← topXSortedByTC(pred, N)
14:    hit ← isvalidationEdgeinpredictions?
15:    addEdge(validationEdge, G)
16:    results ← results + {hit, pred, u, v, validation_edge}
17: end for
18: return results
```

take on. See (CITATION TABLE of triad patterns). Table 3.3 shows triad patterns that are open, that is no connection exists between nodes u and v. The pattern identifications ($T01, T02...$) are taken from (CITE PROPERLY) Schall 2013. Any open triad can be closed in one of three possible ways: $u \leftarrow v$, $u \rightarrow v$, or $u \leftrightarrow v$.

INSERT IMAGE OF 3 CLOSED TRIAD PATTERNS

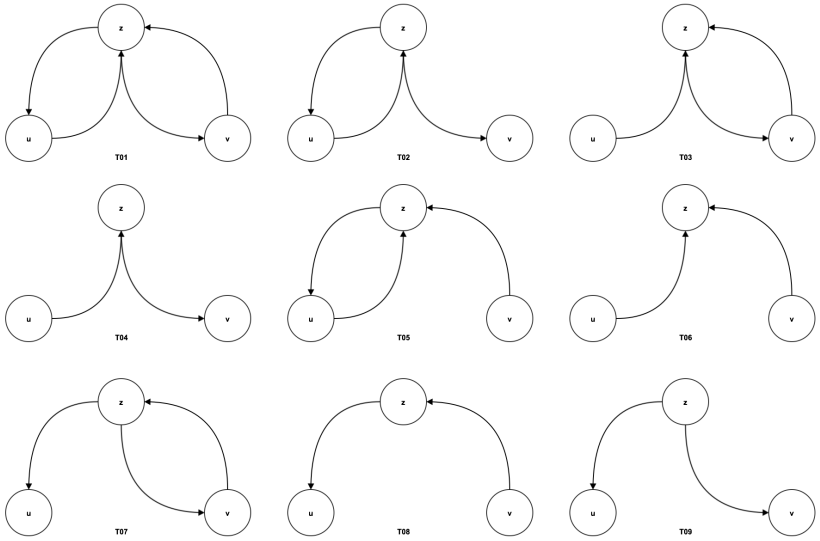


Figure 3.3: All possible open triad patterns are shown. A triad is said to be closed is a connection is established between u and v.

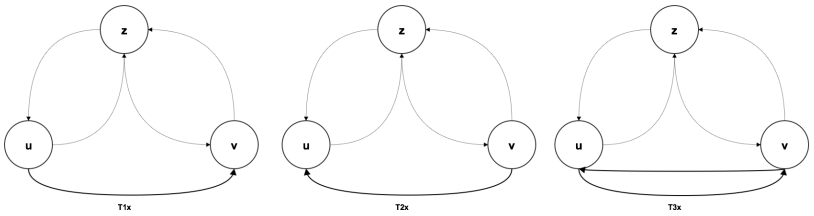


Figure 3.4: A triad is considered to be closed if an edge exists between u and v.

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

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

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

Table 3.2: Open triad pattern frequency in the sample network shown in Figure CITATION NEEDED

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

Table 3.3: Closed triad pattern frequency in the sample network shown in Figure CITATION NEEDED

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

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

Having collected these triad pattern frequencies, we can now generate recommendations as we did in section (REFERENCE SECTION COLLABORATIVE FILTERING) for user J.

The first step is 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)

(INSERT GRAPH HERE OF THE THREE TRIADS AND THEIR PATTERNS)

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

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

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

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

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

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

$$w^P(J,G,F) = \frac{0+0}{3} \tag{3.12}$$

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

$$TC_{JG} = 0 \tag{3.14}$$

Next for J,N:

$$TC_{JG} = \sum_{z \in \Gamma(J) \cap \Gamma(G)} w^P(J,N,z) \times w(z) \tag{3.15}$$

$$TC_{JG} = w^P(J,I,N) \times w(I) \tag{3.16}$$

$$w^P(J,G,F) = \frac{F(T(J,I,N)+10) + F(T(J,I,N)+30)}{F(T(J,I,N))} \tag{3.17}$$

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

$$w^P(J,G,F) = \frac{F(T08+10) + F(T08+30)}{F(T08)} \tag{3.18}$$

Using table (TRIAD FREQ PATTERN REF TABLE)

$$w^P(J,G,F) = \frac{3+1}{14} \tag{3.19}$$

$$w^P(J,G,F) = 0.286 \tag{3.20}$$

$$w(I) = 1/3 \tag{3.21}$$

$$TC_{JN} = 0.286 \times 1/3 = 0.095 \tag{3.22}$$

and finally, calculate TC(J,Q):

$$TC_{JQ} = \sum_{z \in \Gamma(J) \cap \Gamma(Q)} w^P(J,Q,z) \times w(z) \tag{3.23}$$

$$TC_{JG} = w^P(J,P,A) \times w(Q) \tag{3.24}$$

$$w^P(J,P,Q) = \frac{F(T(J,P,Q)+10) + F(T(J,P,Q)+30)}{F(T(J,P,Q))} \tag{3.25}$$

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

$$w^P(J,P,Q) = \frac{F(T04+10)+F(T04+30)}{F(T04)} \tag{3.26}$$

$$w^P(J,P,Q) = \frac{2+0}{14} \tag{3.27}$$

$$w^P(J,P,Q) = 0.143 \tag{3.28}$$

$$w(P) = 1/3 \tag{3.29}$$

$$TC_{JQ} = 0.143 \times 1/3 \tag{3.30}$$

$$TC_{JQ} = .048 \tag{3.31}$$

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

3.2.3 An adaptive ensemble 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. [Lu10] 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 ensemble method could be defined as a simple weighted average with fixed weights as described in [**Candillier**], we instead propose an adaptive weighting

mechanism to take into account the information available for each component of the ensemble metric.

Consider:

$$AEM_{u,v} = \sum_{c \in components} c(u,v) \times w_c(u,v) \tag{3.32}$$

Using Jaccard and Triadic Closeness, this becomes:

$$AEM_{u,v} = J(u,v) \times w_J(u,v) + TC(u,v) \times w_{TC}(u,v) \tag{3.33}$$

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 XX, 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:

$$w_J(u,v) = \frac{count(u - \{: LIKES\} - >)}{count(u - \{*\} - >)} \tag{3.34}$$

$$w_{TC} = \frac{count(u - \{: FOLLOWS\} - >)}{count(u - \{*\} - >)} \tag{3.35}$$

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 Implementation

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

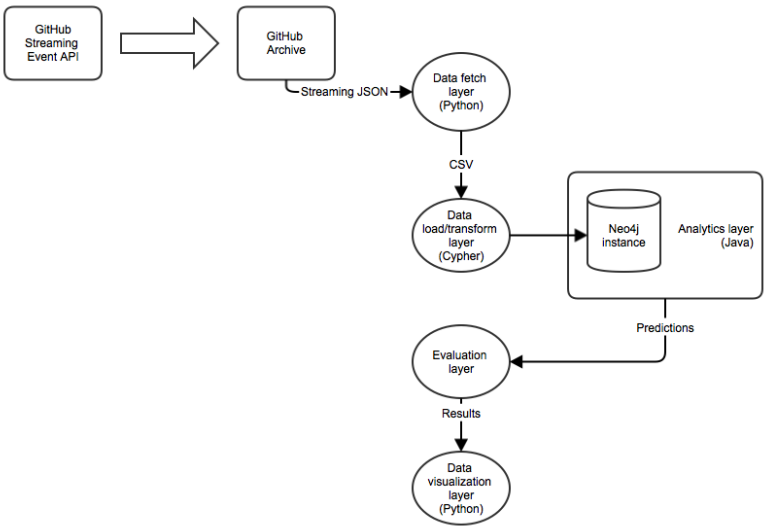


Figure 3.5: The GitHub follow graph is a simple graph with User nodes and Follows edges.

3.3.1 Graph Data Model

We model the data as a graph, $G = (V, E)$

3.3.2 Graph Traversal Pattern

The graph traversal pattern allows us to design algorithms where the answer to our question is a traversal through the graph.

3.3.3 Graph Database And Querying Traversals

Why using a graph database makes sense. Performance benefits of using graph database. Compare to how this would be modeled in a RDBMS or a document DB.

3.3.4 Triad detection

Triad detection involves scanning the entire graph to identify triads in the network, both open and closed.

3.4 Data

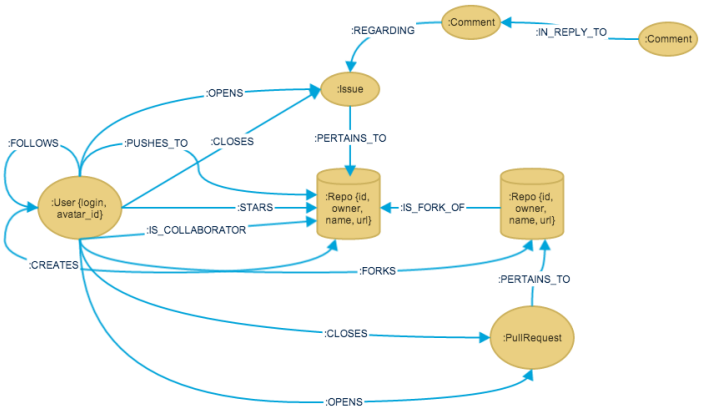


Figure 3.6: GitHub data model as a labeled property graph.

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.

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 3.6 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.

3.4.1 Github Archive

Data was collected from GitHub Archive[Git], a service that maintains an archive of all public events emitted by the GitHub API[**github:Online**]. 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. For the purposes of this project, only *FOLLOWS* events were considered. This resulted in a total of 539893 users and 919489 follows. In terms of a graph, that equates to 539893 nodes and 919489 vertices between nodes.

A graph data model is used to represent this data as the data is highly connected:

1 http://github.com

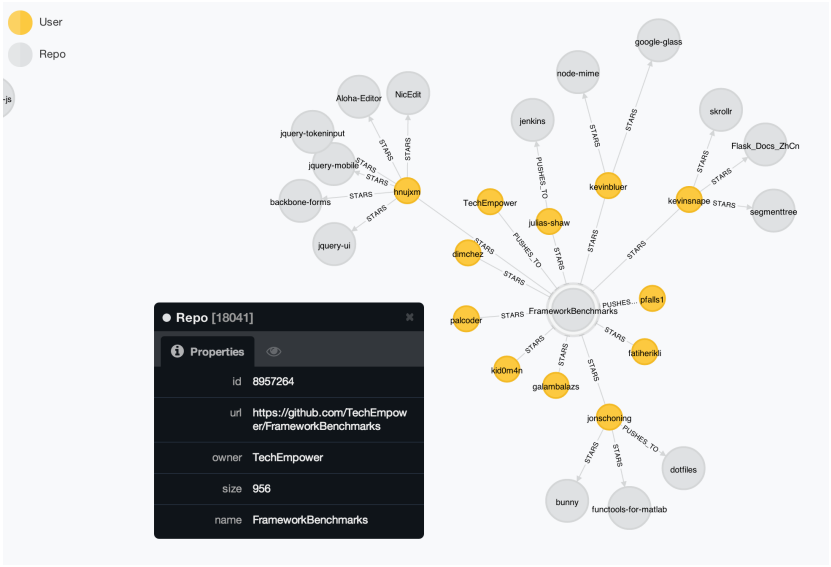


Figure 3.7: GitHub data model as a property graph. Screenshot from Neo4j graph database interface.

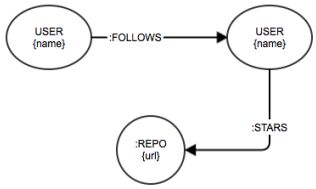


Figure 3.8: The GitHub follow graph is a simple graph with User nodes and Follows edges.

it is describing entities (users and repositories) and their interactions (stars, follows, pushes, etc). Figure 3.7 shows an example of a subgraph of user and repository nodes and the interactions among those entities, modeled as a graph.

FollowEvent

```
{
  "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: LaTeX Typesetting By Example

WatchEvent (Stars)

```
{
  "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: peep this JSON object dawg

3.4.2 Data Analysis

Table 3.4 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.

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. [**pagerank**] 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**].

Tables 3.5 and 3.6 show the highest ranked Users and Repositories for the GitHub data collected.

Table 3.5 shows the results of the PageRank algorithm using User-User *follows*

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.

	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

relationships only. As a sanity check of the data, we expect the highest ranked Users to be influential develops 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

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

Similarly, Table **??** enumerates the 20 most central GitHub software repositories, according to PageRank. Here we expect

Table 3.5: 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

Table 3.6: 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

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 section (PROPER SECTION CITATION) is evaluated using the same evaluation metrics for comparison.

4.1.1 Precision

4.1.2 HitRatio@k (Recall)

4.1.3 F-Measure

?? on **??** demonstrate the creation of a pleasant appearing table, which helps to read the table without attracting to much attention by the use of shaded colors. The caption uses the additional short caption in square brackets `[ ]`, which is used in the list of tables, see page 39.

All possibilities of grouping pictures side by side, on top or in matrices can be realized. Each subfigure is created in the same way as a graphic inside a figure, just enclosed by a figure environment, as shown in fig. 4.3.

For complex subfigure constructs and correct alignment of the subcaption the `floatrow` provides powerful commands.

4.1.4 Evaluation

The system is evaluated using several iterations for cross-validation with different configurations. Table 4.1 shows the results. Two methods of evaluation are identified as the standard for evaluating link prediction systems: precision and area under the receiver operating characteristic curve (AUC). Precision is defined as the ratio of the relevant items selected to the total number of items selected. AUC is a more involved calculation that requires an all-pairs comparison of the rank of the predicted link and *a*ll other possible unobserved links in the network. Due to the size of the GitHub network,

Table 4.1: Evaluation metrics for cross fold validation runs with varying parameters. K is number of neighbors allowed to vote. N is number of predicted links. Precision and accuracy are evaluation metrics, expressed in percentages. Random is the probability of at least one of the N links being relevant if N links are chosen at random.

K	N	Precision (%)	Accuracy (%)	Random (%)
5	25	0.61	15.0	0.0046
5	50	0.18	8.0	0.0093
5	100	0.19	15.0	0.0185
10	25	0.16	4.0	0.0046
10	50	0.14	7.0	0.0093
10	100	0.19	18.0	0.0185
25	25	0.24	6.0	0.0046
25	50	0.12	6.0	0.0093
25	100	0.11	11.0	0.0185
50	25	0.12	3.0	0.0046
50	50	0.14	7.0	0.0093
50	100	0.10	0.10	0.0185
100	25	0.28	7.0	0.0046
100	50	0.16	8.0	0.0093
100	100	0.05	5.0	0.0185
1000	25	0.32	8.0	0.0046
1000	50	0.24	12.0	0.0093
1000	100	0.14	14.0	0.0185
2000	25	0.24	6.0	0.0046
2000	50	0.18	9.0	0.0093
2000	100	0.10	10.0	0.0185

calculating this metric was not seen as feasible. Instead of AUC, a simple accuracy metric is calculated: if the removed link is present in the array of predicted links we count the test run as correct, if not we consider it incorrect. Accuracy is therefore the number of correct predictions divided by the total number of runs. Due to the large number of possible links we expect these accuracy metrics to be quite low, relative to other recommender system results.

Table 4.2: Numbers of Computers in the department, By Type.

Mac (Apple)	2
Windows XP, 7	60
Linux (Server)	10

Table 4.3: Evaluation results for Jaccard similarity, for various values of N, where N is the number of predictions generated for each user.

N	HitRatio@N	Precision	F-Measure
5	0.0494	0.0115	1.2
10	0.0329	0.0039	1.2
20	0.0409	0.0025	1.2
40	0.0389	0.0011	1.2
50	0.0392	0.0010	1.2
75	0.0285	5.0994	1.2

Table 4.4: Evaluation results for Triadic Closeness, for various values of N, where N is the number of predictions generated for each user.

N	HitRatio@N	Precision	F-Measure
5	0.0364	0.0115	1.2
10	0.0410	0.0039	1.2
20	0.0479	0.0025	1.2
40	0.0526	0.0011	1.2
50	0.0392	0.0010	1.2
75	0.0522	5.0994	1.2
100	0.0392	0.132	1.2
250	0.0285	0.3242	1.2
500	0.0135	0.3234	1.2
1000	0.033	0.3211	1.2
5000	0.0166	0.4431	1.2

Table 4.5: Evaluation results for Adaptive Ensemble, for various values of N, where N is the number of predictions generated for each user.

N	HitRatio@N	Precision	F-Measure
5	0.0364	0.0115	1.2
10	0.0410	0.0039	1.2
20	0.0479	0.0025	1.2
40	0.0526	0.0011	1.2
50	0.0392	0.0010	1.2
75	0.0522	5.0994	1.2
100	0.0392	0.132	1.2
250	0.0285	0.3242	1.2
500	0.0135	0.3234	1.2
1000	0.033	0.3211	1.2
5000	0.0166	0.4431	1.2

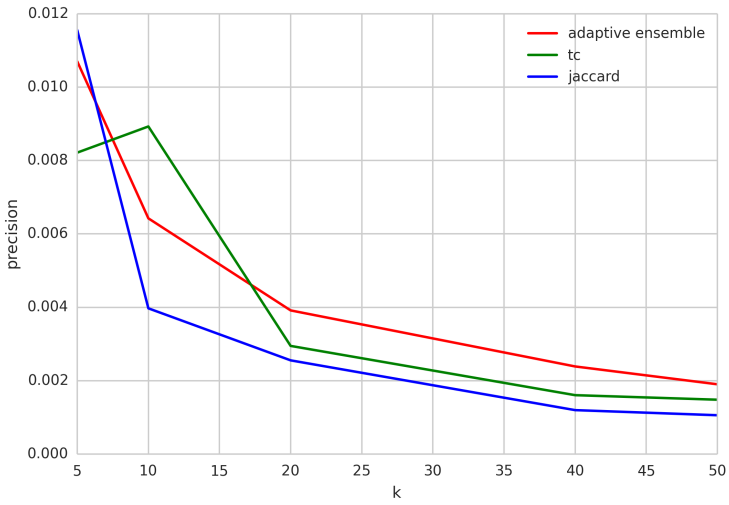
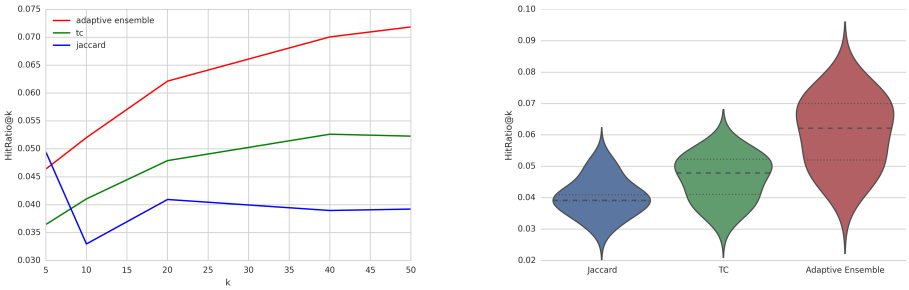
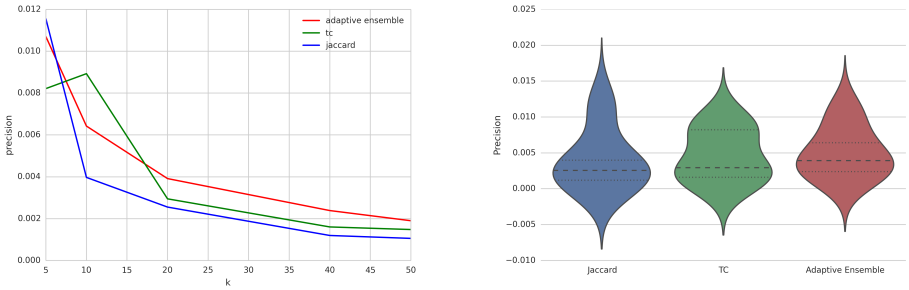


Figure 4.1: Test image for television (Origin of the image: http://de.wikipedia.org/wiki/Testbild).



(a) The first subfigure. **(b)** The second subfigure.

Figure 4.2: Demonstration of the *subfigure* environment inside a figure environment



(a) The first subfigure. **(b)** The second subfigure.

Figure 4.3: Demonstration of the *subfigure* environment inside a figure environment

4.2 Discussion

Table 4.1 details the results of running our algorithm for various configurations of the parameters k and n, where k is the number of neighbors considered to generate recommended links and n is the number of predicted links generated for each user. We calculate both precision and accuracy for each run, where accuracy is defined as the number of users for which the removed link was observed to be in the set of predicted links divided by the total number of users in that run. In other words, we count a prediction as "correct" if v, the removed *F*OLLOWS edge is in the set of predicted edges. Using this same definition of correctness, we calculate the probability of accuracy if the links were chosen at random from our graph of existing GitHub users. We can see that for almost all configurations, the results are significantly (approximately 3000X) better than choosing links at random. Surprisingly, the best performing configurations (as measured by both accuracy and precision) are those with the lowest values of K (the number of neighbors used to generate recommendations).

CHAPTER 5

Summery and Outlook

Bibliography

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

[Goo94] Michel Goossens, Frank Mittelbach, and Alexander Samarin: *The LaTeX Companion*. 1st ed. Reading, Mass.: Addison-Wesley, 1994. 528 pp. (cit. on p. 1).

[Lu10] Linyuan Lu and Tao Zhou: 'Link Prediction in Complex Networks: A Survey'. In*arXiv:1010.0725v1* (2010), vol. (cit. on p. 17).

[Rod] Marko A Rodriguez and Peter Neubauer: 'The Graph Traversal Pattern'. In (), vol.: pp. 1–18 (cit. on pp. 2, 4).

[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. 9, 11).

List of Figures

List of Tables

Listings

APPENDIX A

First chapter of appendix

A.1 Parameters

Acknowledgments

I thank ?? and ?? for giving me the opportunity to write this bachelor/master/phd thesis at ??, and for their professional advise.

I thank in particular the ?? team who readily/willingly provided information at any time and ??.

I would also like to than all people who supported me in writing this thesis.