Guided Walk: A Scalable Recommendation Algorithm for Complex Heterogeneous Social Networks

Roy Levin
Microsoft
Herzliya, Israel
rolevin@microsoft.com

Hassan Abassi
Technion
Haifa, Israel
hassan@cs.technion.ac.il

Uzi Cohen
IBM Research
Haifa, Israel
uzic@il.ibm.com

ABSTRACT

Online social networks have become predominant in recent years and have grown to encompass massive scales of data. In addition to data scale, these networks can be heterogeneous and contain complex structures between different users, between social entities and various interactions between users and social entities. This is especially true in enterprise social networks where hierarchies explicitly exist between employees as well.

In such networks, producing the best recommendations for each user is a very challenging problem for two main reasons. First, the complex structures in the social network need to be properly mined and exploited by the algorithm. Second, these networks contain millions or even billions of edges making the problem very difficult computationally.

In this paper we present Guided Walk, a supervised graph based algorithm that learns the significance of different network links for each user and then produces entity recommendations based on this learning phase. We compare the algorithm with a set of baseline algorithms using offline evaluation techniques as well as a user survey. The offline results show that the algorithm outperforms the next best algorithm by a factor of 3.6. The user survey further confirms that the recommendation are not only relevant but also rank high in terms of personal relevance for each user.

To deal with large scale social networks, the Guided Walk algorithm is formulated as a Pregel program which allows us to utilize the power of distributed parallel computing. This would allow horizontally scaling the algorithm for larger social networks by simply adding more compute nodes to the cluster.

CCS Concepts

•Human-centered computing → Social recommendation;

RecSys '16, September 15-19, 2016, Boston , MA, USA

DOI: http://dx.doi.org/10.1145/2959100.2959143

Keywords

Recommendations, Heterogeneous Networks, Social Networks, Apache Spark, GraphX, Pregel

1. INTRODUCTION

In the last decade levels of activity in online social networks has proliferated and is still rapidly growing. Every minute Facebook users share nearly 2.5 million pieces of content [6]. Due to the sheer amount of activity, manually sifting through the content is a complex and Sisyphus task. This problem has given birth to a recommendation system, first introduced by Facebook in 2006 [6], that generates a personalized news-feed for each user. Other social networks such as Twitter, LinkedIn and Google+ have followed suit and have adopted a similar concept.

Recently, such recommendation systems have also been developed for Enterprise Social Networks (ESN) such as Tibbr, Yammer, Jive and IBM Connections. The goal of these systems is also to filter and rank news items according to their usefulness to a specific employee. Rather than reporting other employees' actions in the form of a news-feed, a similar task is recommending specific social entities to an employee [14]. For example, raising the user's awareness about a specific blog post. However, in addition to the actions employees perform on the different social entities ESNs typically also contain complex structural information. For instance, in IBM-Connections [11] hierarchies exist between social entities as well as between employees. A community contains blogs which contains articles which, in turn, can contain comments. An employee's manager may 'like' any of these social entities. This kind of structure is illustrated in Figure 1.

Producing a recommendation for "user 1" in the example from Figure 1 is a difficult task as it requires taking into account (a) the relationships between users, (b) the relationships between entities and (c) the interactions between users (employees) and entities. Such networks form complex heterogeneous networks as they contain various types of relationships between entities and they also encompass complex structures. One common recommendation technique is Collaborative Filtering (CF) [24]. CF makes predictions about user's interests based on preferences of other users. However, CF is generally designed for bipartite graphs which model interactions between users and items hence it cannot be easily applied over complex heterogeneous networks such as in our example.

Recommendations for more complex network structures can be generated based on a similarity search which finds the

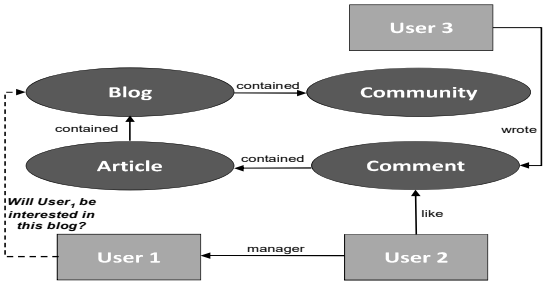

Figure 1: How to recommend entities to an employee taking into account the complex network structure?

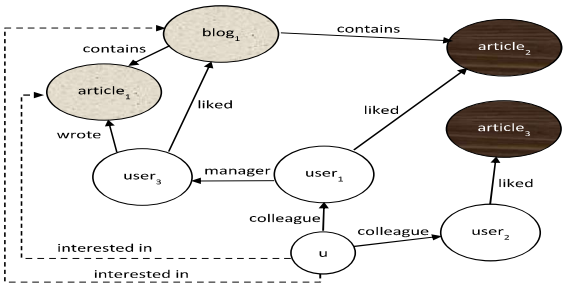

Figure 2: In the example, user u is interested in $article_1$ and $blog_1$. The task is to recommend article-2 or article-3 based on the known user interests and network structure.

most proximate node to a given node. Personalized PageRank [13] and SimRank [12] have been shown to be useful similarity measures to use for this technique. In this paper we propose a novel approach which is a **supervised** version of a Simple Random Walk on a Graph [18] called *Guided Walk* (GW). In the simple and directed case of a random walk all outgoing edges are assumed to have equal probabilities and the probabilities of reaching each vertex in the graph from a given starting point is calculated. This is similar to the Personalized PageRank [13] approach mentioned above. However, in our case, the edges are not assigned uniform probabilities but are rather learned and determined based on the user's actions. Edges in the graph are assigned probabilities based on their likelihood to lead to entities that are known to interest the user. To explain the intuition behind this approach consider the example depicted in Figure 2. In the example, employee u already knows about $article_1$ and $blog_1$ hence the task is to recommend a new entity, i.e., either $article_2$ or $article_3$. Based on the graph in the example it is more likely that the user would be interested in $article_2$ than in $article_3$. The intuition for this is due to the fact that there is a path from user u via $user_1$ to $article_2$ and from $blog_1$ to $article_2$ and we see that other paths from $user_1$ and $blog_1$ lead to items that interest user u. On the other hand, the only path from u to $article_3$ is via $user_2$ which does not lead to any entities that are of interest to u. Hence, it makes sense to assume that $article_2$ is a better recommendation than $article_3$ for user u.

One key challenge in implementing such an approach is in efficiently calculating the edge probabilities. This must be done without enumerating every possible path from the source in a given graph. Furthermore, as social networks nowadays tend to be extremely large, loading the entire graph into main memory and running a sequential process on a single machine is usually not a feasible option. In recent years, data-parallel systems like MapReduce and Spark [4, 34] have been designed to process large amounts of data. These systems achieve horizontal scalability by allowing computations to be distributed over a cluster of machines using various fault-tolerance strategies. However, they are not designed for graph processing thus making it difficult for programmers to express graph processing tasks. Furthermore, it may also lead to suboptimal data processing performance over the cluster which involves too much data shuffling [31]. To cope with this challenge, large-scale distributed graph-parallel frameworks such as Pregel have lately been introduced [20, 31]. These make graph algorithms such as PageRank, Connected Components, Triangle Counting and others relatively easy to express and process efficiently in parallel using the power of distributed computing.

For the purpose of our research we implemented the Guided Walk algorithm as a set of Pregel algorithms. The Pregel algorithms are applied both for the learning phase, during which we learn the edge transition probabilities, and for the the later phase that assigns recommendation scores to entities per-employee. Our graph is modeled as a Resilient Distributed Graph (RDG), and then, using Spark's GraphX [31] layer to apply the set of Pregel algorithms. In addition to scalability, this approach allows us to fully utilize the cluster's RAM for processing the graph.

2. RELATED WORK

For the purpose of mining useful information from networked data, various types of networks have been categorized and studied. From homogeneous networks which have only a single type of object and a single type of link through multi-relational networks which have a single type of object but multiple types of links and finally heterogeneous networks which have both multiple types of objects and of links [26]. Each of these types of networks can be further categorized based on the complexity of the structures formed in the graph. For example bipartite networks are heterogeneous networks which are widely used to construct interactions among two types of objects, such as user-item[33], and document-word [17]. Alternatively in a star-schema network there is a target object that is connected to objects around it which form its attributes [22, 29]. Multiple-hub networks further extend this idea and form star-schema networks around multiple types of target objects [30]. Besides these widely studied network types, many real systems form yet more complex heterogeneous networks which do not natuarully fit into either of these sub-categories [26].

In this paper we study the problem of personalized recommendations of entities in the enterprise social network (ESN). As illustrated in Section 1, an ESN forms a complex heterogeneous information network, having different types of entities with various relationships between them as well as hierarchies among entities and users. Recommendation tasks in complex heterogenous information networks have lately been the focus of several studies [28, 32, 15, 27]. A common approach employed in these works involves three basic steps. First, a "bootstrapping" step is employed in which a set of Meta-Paths are manually selected to be con-

sidered by the recommendation algorithm. Each Meta-Path may capture the similarity semantics between two entities connected corresponding paths which are encoded according to the various entity types and relationship types that are included in that path. The next step generally involves measuring the similarity between each pair of entities by exploiting path information using similarity measures such as PathSim [29], HeteSim [25] and PCRW [16]. The final step, performs the actual recommendation based on entity-similarity information. This final recommendation step can be classified into two main types, either Random-Walk based [16] or CF and Matrix-Factorization based [19].

More specifically some research has been focused on the problem of recommending entities in a social network setting [14, 5]. Rather than recommending specific entities to a user some papers have studied the problem of producing a news-feed for the user [23, 9, 1]. A news-feed includes actions performed by other users rather merely the entities themselves. Specific aspects of producing news-feeds in the ESN have also been studied recently [3, 7, 6].

Finally Supervised Random Walks have also been studied (e.g. [2]). There are a few key differences however w.r.t. this work. First, using the algorithm requires domain knowledge since it involves a feature extraction phase for nodes and edges. Next, a supervised random walk is employed to learn their importance. The recommendation itself is then based on these learned features and not on the paths leading to the recommended entity. Second, the algorithm is not designed to run in a distributed environment or on large graphs. In fact, the algorithm is tested on graphs with no more than 175,000 nodes while limiting recommendations to nodes that are no more than 2-hops away from the source. This further reduces the number of candidate recommendations that need be examined.

Compared to previous works, this work is not tailored for a specific domain nor does it require schema knowledge or manual extraction of meta-paths and/or other features. Much like other works, in this work we also utilize a random walk to form a similarity metric between nodes however, in our case, it is preceded by a training phase which learns edge transition probabilities, per-user, based on specific user's preferences as embodied in the relationships. A novel supervised learning algorithm is used to guide the walk based on those preferences. Finally, the algorithm is based on the Pregel paradigm allowing it to run in a distributed environment over many machines, thus enabling it to run on large scale graphs.

3. PROBLEM DEFINITION

In this Section we describe our framework and layout the model for the Guided Walk recommendation method.

3.1 Data Model

Let V denote the set of all social entities and users (e.g., employees, blogs, wikis, articles, etc.) and E be a set of all relationships between them. The relationships may include actions between users and entities, structural information between entities (e.g., a community containing a blog) and relationships between users. Note that some of the relationships may be directed and some may be undirected. For instance, when an employee creates a blog post, it forms a directed relationship from the employee to the blog post, whereas if two employees are colleagues, it forms an undirected relationship.

Let U be a set of users (in our case employees) for which we would like to produce recommendations. For each $u \in U$ let $G_u(V_u, E_u)$ be a *Personalized Graph* such that E_u is a set of directed edges derived from E where each edge represents the relationships from the point of view of user u. Hence, under this definition all the colleagues of u are represented as outgoing edges from u (in our dataset other types of edges remain unchanged). The set of vertices V_u are the vertices in V that are reachable from u via the directed edges in E_u. Figure 2 depicts an example of the Personalized Graph of u. An additional limitation here is that $G_u(V_u, E_u)$ must be a DAG. This limitation is enforced during the construction of the personalized graph for each user from V and E. Note that in the dataset we used (details in Section 5) cycles do not exist. Among the set of social entities V_u we define the subset of entities called *Recommendation Candidates* which may potentially appear in user u's list of recommendations. In our case this is done based on entity types that we wish to focus our recommendations on.

Recall from the description in Section 1 that our goal is to learn edge probabilities based on the actual interests of the user for guiding the recommendation algorithm. For this purpose we take a subset of entities from V_u which are known to be of interest to user u (based on user activities such as 'liked', 'clicked', etc.). We refer to this subset of entities as the *Known User Interests (KUI)* , denoted $K_u \subseteq V_u$. These entities are removed from the Recommendation Candidates to avoid recommending entities the user is already aware of.

3.2 Problem Formulation

We now describe the problem as follows. Given an employee $u \in U$ we generate her personalized graph $G_u(V_u, E_u)$ from V and E and select the user's known interests set KUI K_u. The task is then to generate a ranked list of social entity recommendations for user u such that the entities are ranked based on how likely they are to interest u.

3.3 Guided Walk Problem Formulation

The Guided Walk method involves a two phase process. The first phase is a training phase in which we determine edge weights. These edge weights represent the "importance" of an edge to the user u. This is modeled by estimating the likelihood of a specific edge to be on a path leading from u to an entity that interests u. The second phase is then to produce a list of top-k recommendations based on their expected likelihood based on the edge weights. We next describe these two phases in detail.

3.3.1 Determining Edge Probabilities

We define a *Full Path* between two vertices $v, l \in V_u$ in a given G_u, as an edge path which starts at v, ends at l where l is a leaf, i.e., it has an out-degree of 0. For every $v \in V_u$ let Ψ_v be a set of all Full Paths staring at v. Let $X_v : \Psi_v \to \mathbb{N}$ be a random variable representing a *Full Path Gain (FPG)* defined on a given Full Path $\psi \in \Psi_v$ as the number of vertices in ψ that are also in the KUI set K_u, i.e., $X(\psi) = \left|K_u \cap \psi\right|$ (this is a slight abuse of notation as ψ here should be interpreted as the set of vertices spanned by the path of edges). The expected value of the path gain random variable is equal to the FPG multiplied by the probability

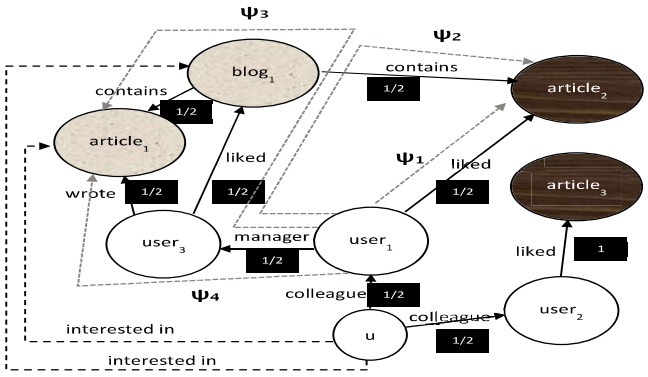

Figure 3: Full path score calculation illustration

of uniformly picking that path out of the set of all possible paths. The expected FPG is given in Equation 1.

$$E[X_v] = \sum_{\psi \in \Psi_v} X(\psi) \cdot p(\psi) \tag{1}$$

In Figure 3 we illustrate an example of how the expected value of the FPG $E[X_{user_1}]$ is calculated. The set of Full Paths $\Psi_{user_1} = \{\psi_1, \psi_2, \psi_3, \psi_4\}$ as well as the edge transition probabilities are depicted for each edge. Based on the figure, the FPGs starting from $user_1$ are as follows:

- $X_{user_1}(\psi_1) \cdot p(\psi_1) = 0 \cdot \frac{1}{2} = 0$
- $X_{user_1}(\psi_2) \cdot p(\psi_2) = 1 \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{8}$
- $X_{user_1}(\psi_3) \cdot p(\psi_3) = 2 \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$
- $X_{user_1}(\psi_4) \cdot p(\psi_4) = 1 \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$

Hence, $E[X_{user_1}] = \sum_{i=1}^{4} X_{user_1}(\psi_i) \cdot p(\psi_i) = \frac{5}{8}$ is the expected FPG for $user_1$. Note that in practice calculating the expected FPG for each vertex by iterating over all possible full paths in the graph may be impractical or very inefficient at best. In Section 4 we describe how it can be calculated efficiently and how it can be applied over large scale data by utilizing distributed and parallel computation technologies.

The next step is to determine edge transition probabilities for all the edges based on their target vertex FPG. This will assign higher transition probabilities to edges leading to vertices with higher expected gains. However, before doing so, note that some FPGs may be 0. This happens for vertices which do not have any Full Paths leading from them to entities that are of interest to u. This is a problem since lack of information about the user being interested in some entity does not really indicate that the likelihood of interest to the user is really 0. Treating it as such would be a classic case of over-fitting. To deal with this problem we employ an Additive Smoothing approach in a similar fashion to how its used in Naive Bayes Classifiers [8]. Intuitively, instead of setting edge transition probabilities to 0 they are set to some predefined (small) ε value and the probability function is adjusted for the other edges accordingly. The final transition probabilities for the edges is given in Equation 2.

$$P(e = (v_1, v_2)) = \frac{E[X_{v_2}] + \varepsilon}{\sum_{e' \in E} (E[X_{v_2}] + \varepsilon)} \tag{2}$$

3.3.2 Recommendation Scores

We now describe how we use Equation 2 to calculate the recommendation score for each $v \in V_u$ in the Personalized Graph of user u. The score of a vertex $v \in V_u$ should reflect the probability of reaching it from u using the edge traversal probabilities defined in Equation 2. As a result, a vertex with a higher score generally reflects the fact that full paths leading from u via v to leaves in G_u contain items of interest to u. This is true since ε in Equation 2 is generally kept small. Note that the higher the value assigned to ε the greater the significance given to the number of full paths passing through v in G_u. As a result, the smaller the value assigned to ε the more the Guided Walk will fit the recommendation to the user specifically. At the extreme, setting $\varepsilon = 0$ will over-fit the recommendations to the user giving absolutely no weight to other user actions. Hence, selecting the best value for ε is a matter of tuning (more details in Section 5). Equation 3 formalizes the recommendation score of an entity $v \in V_u$ w.r.t. the user u where Γ represents the set of all edge paths between user u and entity v.

$$s(v) = \sum_{\gamma \in \Gamma} \left(\prod_{e \in \gamma} P(e) \right) \tag{3}$$

Note that this recommendation score function is calculated per-user, i.e., for each G_u separately. Once more, in Section 4 we describe how Equation 3 is computed efficiently and in a scalable manner.

4. THE SOLUTION

In this Section we present our Guided Walk algorithm and explain in detail how it is implemented efficiently and in a scalable manner using the Pregel paradigm [20] over the Apache Spark GraphX framework [34, 31]. In addition we also describe a set of benchmark algorithms used to evaluate our algorithm experimentally as further detailed in Section 5.

4.1 Learning Edge Weights Using Pregel

In Subsection 3.3.1 we formulated the edge traversal probabilities. We now show how they are computed efficiently and in a scalable fashion. We next explain how we calculate the expected FPG for each vertex $v \in V_u$ which is depicted in Equation 1.

We begin with a few definitions and notations. Given a DAG, $G(V, E)$ and a subset of vertices $S \subseteq V$, we define a *Path Count (PC) Function* denoted $\#_{G,S} : V \to \mathbb{N}$ as a function mapping each vertex $v \in V$ to the number of paths starting at any $s \in S$ and leading to v via the edges in E. We use the notation $L(G) \subseteq V$ to denote the leaf vertices of $G(V, E)$ (i.e., all vertices with an out degree of 0). Let G^{rev} denote the the graph G with its edges reversed. Following the definitions above, it is easy to see that for any $v \in V_u$, $\#_{G_u^{rev}, L(G_u)}(v)$ is equivalent to the number of Full Paths in G_u starting at vertex v (since $L(G_u)$ are the roots of G_u^{rev}). Furthermore, the number of Full Paths starting at v but which pass via user u's known interest entities is given by $\sum_{s \in K_u} \#_{G_u^{rev}, L(G_u)}(s) \cdot \#_{G_u^{rev}, \{s\}}(v)$. Since the number of full paths is counted once per entity in K_u it is equivalent to the FPG as defined in Section 3.3.1. Also note that, $\frac{1}{\#_{G_u^{rev}, L(G_u)}(v)}$ is the probability of uniformly selecting a specific full path among all possible full paths starting at

v. Hence, equation 4 is equivalent to the expected FPG as defined in Equation 1 of Section 3.

$$E[X_v] = \frac{\sum_{s \in K_u} \#_{G_u^{rev}, L(G_u)}(s) \cdot \#_{G_u^{rev}, \{s\}}(v)}{\#_{G_u^{rev}, L(G_u)}(v)} \quad (4)$$

However, to compute equation 4 we need the function $\#_{G_u^{rev}, \{s\}}$ for each $s \in K_u$. For users with a lot of activity this would require more computations. An examination of our data reveals that entities in K_u are either leafs or are distant a single edge away from a leaf. We can therefore closely approximate Equation 4 more efficiently as follows.

$$E[X_v] \cong \frac{\#(G_u^{rev}, K_u)(v)}{\#(G_u^{rev}, L(G_u))(v)} \quad (5)$$

It follows then that we can closely approximate the expected FPG $E[X_v]$ by computing $\#_{G_u^{rev}, K_u}$ and $\#_{G_u^{rev}, L(G_u)}$ and combining their values over each vertex as illustrated in Equation 5. We now explain the Pregel algorithm which computes PC Functions. The description is based on Apache Spark's GraphX APIs [1]. In general, the Pregel paradigm is an iterative vertex-centric process. Each iteration a.k.a. super-step has three main steps which it applies (in our case we only explain the directed graph case):

1. **Vertex Program (vprog):** A vertex receives a combined message of all the messages sent to it in the previous super-step (or an initial message if this is the first super-step). A new state is computed based on the current state and the combined message by applying the provided 'vprog'.

2. **Send Messages: (send):** Outgoing edges [2] of vertices that received messages in the current super-step apply the given 'send' message and can optionally send a message via these edge.

3. **Merge Messages:: (merge):** Each vertex that appeared as a destination in the Send Messages step applies a provided 'merge' method on the messages sent to it to combine them into a single unified message.

A Pregel algorithm terminates when the latest super-step does not send any more messages or when a some predefined maximum number of super-steps is reached.

Before applying our Pregel algorithm using GraphX, we first replace the content of the vertices with content that represents a vertex's state in the current super-step. The content of each vertex is a pair containing: (a) the PC updated according to current super-step i and (b) its value before the update took place (i.e., the PC value at super-step $j < i$). The latter is needed to keep track of the change done to the PC in the current super-step by vprog. Initially these values are set to $1, 0$ respectively for vertices in S and $0, 0$ for all other vertices — this represents the starting values, i.e., in super-step 0. An initial message of 0 is sent to all vertices in super-step 1. The algorithm, as well as the types used to represent vertex content, message types and the 3 methods 'vprog', 'send' and 'merge' are detailed in Table 1.

[1] http://spark.apache.org/docs/latest/graphx-programming-guide.html#pregel-api

[2] These are actually triplets (not plain edges) which are edges with embedded vertex content.

Types

ID *description* *notations*	vertex id *(id)*
VD *description* *notations*	vertex content tuple of the form *(new-count, old-count)* (nc, oc)
A *description* *notations*	message containing the *change-count* Δ

Methods

vprog	$(\mathrm{ID}, \mathrm{VD}, \mathrm{A}) : \mathrm{VD}$ $(id, (nc, oc), \Delta) \rightarrow$ if $(\Delta > 0)$ then $(nc + \Delta, nc)$ else (nc, oc)	update PC based on changes done in the previous super-step
send	$((ID, VD), (ID, VD)) : (ID, A))$ $((sid, (snc, soc)), (did, (\_, \_))) \rightarrow$ if $(snc - soc > 0)$ then $(did, (snc - soc))$	if vprog changed value of its PC then send change via outgoing edges
merge	$A^K : A$ $\{\Delta_1, \Delta_2, \ldots, \Delta_K\} \rightarrow \sum_{k=1}^{K} \Delta_k$	combined change in PC is sum of PC changes

Table 1: Pregel program for calculating PCs for each vertex

4.2 Recommendations From Edge Weights

Now that edge weights are available for a given user, recommendation scores can be produced as described in Section 3.3.2. Recall that the recommendation scores produced by the Guided Walk algorithm represent, for each vertex, the expected probability of reaching that vertex from the source user u using the transition probabilities calculated during the training phase. To calculate these recommendation scores we can reuse the Pregel program (defined in Table 1) with some minor changes as follows. First, we apply the algorithm on G_u (not on G_u^{rev}) and we set $S = \{u\}$. Second, since in this phase we compute expectancies rather than plain counts we also need to change the way messages are produced. For this purpose, we multiply the delta count in the send method (see the 'send' method in Table 1) by the transition probability of the edge. That is, the send message method is changed as follows. Let $m = P(sid, did) \cdot (snc - soc)$ and if $(m > 0)$ then $(did, m))$. When the program completes we produce recommendations for user u and rank the candidate entities in descending order based on the computed recommendation scores.

4.3 Baseline Algorithms

To measure the effectiveness of our Guided Walk algorithm we implemented a set of baseline algorithms for comparison. We next explain each one of these algorithms.

4.3.1 Simple Directed Random Walk (RW)

To see if the learning phase produces benefit a good baseline is to create a version of the Guided Walk algorithm that skips the learning phase and assigns edge probabilities uniformly to every outgoing edge of every vertex. To implement this algorithm we reuse the algorithm for producing recommendations from edge weights described in Section 4.2 where all outgoing edges of a vertex v are assigned a probability equal to $\frac{1}{|out(v)|}$.

4.3.2 Collaborative Filtering (CF)

Collaborative Filtering is a widely used algorithm for producing personalized recommendations and, as such, is a good

baseline for comparison. We used Spark's implicit feedback collaborative filtering implementation [3] implemented according to [10]. In our case an implicit feedback is given whenever a user performs any type of operation on an entity, e.g., "like", "comment", "create", "vote", etc. In other words, any operation that indicates an entity should be in a user's known interests is counted as an implicit feedback by a user on that entity. This is the only input necessary for the algorithm and it produces a ranked list of recommendations for each user based on the implicit feedback of all users. We describe how we choose the parameters for CF in Section 5.

4.3.3 Popular in Neighborhood (PN)

Given a personalized graph G_u, the Popular in Neighborhood (PN) algorithm ranks the candidate entities based on the number of actions all users performed on them within the context of G_u. Hence, the algorithm ranks the candidate entities that appear in G_u based on how popular they are. Recall that G_u only contains vertices and edges reachable from u, therefore this algorithm is still personalized in that it disregards entities outside the reach of u.

5. EXPERIMENTAL EVALUATION

In this Section we describe the experimental setting, our dataset and our evaluation methods. We analyze our results and provide evidence to support the usefulness of our Guided Walk recommendation algorithm for complex heterogeneous social networks.

5.1 Dataset

Our dataset was imported from a deployment of IBM Connections (IC) [11] within a large global enterprise. IC is a social media application suite for the enterprise. We used data collected over a period of 6 month, from October 2015 to March 2016 (inclusive). The data was exported as a textual CSV file whose size is roughly 100GB. The data consists of different types of social media entities which include: Communities, Blogs, Blog-entries, Wikis, Wiki-entries, Comments, Files, Forums, Forum-topics and Microblogs (similar to Tweets). Employees can create. edit, like, add comments and bookmark any of these entities and also add other employees as their colleagues (similar to friending in Facebook), tag them or follow them. Furthermore, hierarchical information about the structure between the entities also exists, e.g., a community can contain blogs that contain blog entries which contains user comments. For further information about IC and the dataset that it exports see [11, 6].

5.2 Environment

We used a shared multi-tenant Hadoop YARN cluster with 11 data/compute nodes with a total of 3.18TB of RAM and 260 virtual cores. The application framework we used is Apache Spark version 1.6.0. The graph based algorithms described in Section 4 were implemented using the GraphX framework and the Collaborative Filtering algorithm was implemented by using the MLlib implementation provided by Apache Spark.

[3]http://spark.apache.org/docs/latest/mllib-collaborative-filtering.html#explicit-vs-implicit-feedback

5.3 Offline Experiments

We selected 1100 of the most active IBM-Connections users within the above time period. We split the dataset into a training set and a test set. For the training set we used 4 months worth of data, from October 2015 to January 2016 (inclusive) and for the test we used the remaining 2 months of data after January 2016. We evaluated the algorithms described in Section 4 by training them on the training data and then having each algorithm produce the top 10, 50 and 100 recommendations for the selected users over the test data. The relevant documents for the information needs of a specific user were determined as any entity the user performed an action on according to the test dataset. The type of actions that were available to us in the dataset were 'like', 'comment', 'edited', 'joined a community', etc. However, one limitation is that we did not have access to passive view data, i.e., which entities users only viewed. To make the comparison between all the algorithms simple we further filtered this list by removing entities that were not present in the training set, e.g., new blogs, wikis, etc. which were not available to the algorithms during their training phase.

5.3.1 Parameter Tuning

We arbitrarily chose 100 of the 1100 users to tune the parameters of each of our tested algorithms by training the algorithm on the training set with different parameters and then examining the Mean Average Precision (MAP) [21] results on the test set for these same 100 users (as described above). Based on these results, we then chose the optimal parameters for each algorithm to produce recommendations for the remaining 1000 users. The parameters we chose for CF are rank = 10, $\alpha = \frac{1}{10000}$, $\lambda = \frac{1}{10000}$ and the number of iterations was set to 100. For GW we chose $\varepsilon = \frac{1}{1000000}$.

5.3.2 Results

To determine the quality of the recommendations produced by the algorithms described in Section 4, we calculated the MAP@10, MAP@50 and MAP@100 results for each algorithm over the 1000 users that were not part of the parameter tuning phase described in Section 5.3.1. The MAP results are reported in Figure 4. We added one more algorithm just to serve as a a minimum baseline for all other algorithms. This baseline algorithm is called Random-Rank (RR) (not to be confused with Random Walk (RW)) and it simply produces a randomly ranked list of recommendations from the training dataset for each user. Its MAP scores are barely noticeable in the figure since they are orders of magnitude lower than the scores of the other algorithms. Based on the figure, when comparing MAP@100 results, GW outperforms CF, PN, RW, RR by the factors of 3.6, 13.7, 16.3 and 7138 respectively. Especially interesting is the comparison between GW and RW since, as described in Section 4.3.1, RW works just like GW only without the learning phase. The fact that MAP results are improved $16-$fold provides clear evidence that the learning phase works well.

To find out how diverse the top recommendations produced by each algorithm are, we report the Jaccard Similarity between the top-100 recommendations produced by each algorithm in Table 2. First, from the comparison between all the algorithms to PN we can learn which algorithms tend to recommend entities that are more popular, i.e., entities that are not necessarily tailored to a specific user. It should

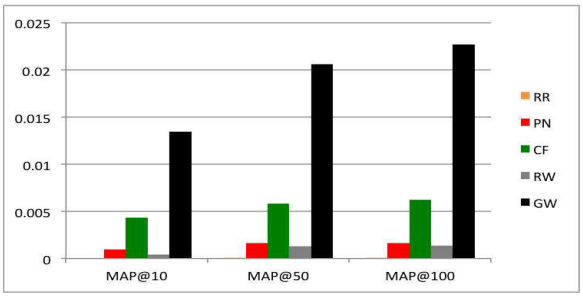

Figure 4: Offline experiments MAP results. The tested algorithms are Guided Walk (GW), Simple Directed Random Walk (RW), Popular in Neighborhood (PN), Alternating Least Squares Collaborative Filtering (CF) and RandomRank (RR). Note that MAP values for RR are barely noticeable since they are very close to 0.

(PN,CF)	(PN,RW)	(PN,GW)	(CF,RW)	(CF,GW)	(RW,GW)
0.187	0.387	0.230	0.139	0.220	0.132

Table 2: Jaccard similarity ratios between the top-100 entities produced by each pair of algorithms averaged over all users. (0/1 indicates that sets are disjoint/identical).

not be surprising that the similarity between PN and RW is relatively high since entities that many users found interesting also tend to have more paths passing through them (i.e., via those users). Also interesting is that all similarity scores to PN tend to be somewhat high. We can learn from this that all the algorithms gave some natural preference to popular entities. This is a good sign since indeed popular entities tend to be of interest. Finally, notice that similarity between CF and GW is higher than CF and RW or RW and GW since both CF and GW try to better tailor the recommendations personally to the user.

We also tracked the running times of each of the algorithms when running in our environment. Note that the average per-user personalized graph G_u contained around 2 million edges and some vertices contained tens-of-thousands of adjacent edges making them hypernodes. The absolute running times are specific to our setting, however we report them in Table 3 to provide a comparison between the algorithms. To enhance user-recommendations throughput for GW which has a relatively high runtime we further utilized parallelism by splitting the set of 1000 users into 6 disjoint sets and running a Spark job for each of them separately. This allowed us to calculate the MAP values for GW over all 1000 users within less than 2 days. Each job was run on 11 Spark Executors with 2 cores each and 8 GB RAM per Executor.

5.4 User Survey

	GW	CF	RW	PN
Training	670 (per-user)	1590	-	-
Recommending	402	13	294	0.5

Table 3: Average running times in seconds for each algorithm. Note that RW and PN have no training phase and for GW the training phase is applied per-user whereas for CF it is applied once for the entire dataset.

One drawback of the offline experiment is that only entities which the user actually performed actions on during the test period are determined as relevant. Hence some entities may have been incorrectly categorized as irrelevant merely because the user was unaware of their existence. Furthermore, the degree of personal relevance between the entities and the users could not be directly assessed.

To address these problems we ran a user survey. We arbitrary selected 373 highly active IC users and sent each of them a list of recommendations based on their activity in the last two months (March and April 2016). The recommendations were produced by having the algorithms PN, CF, RW and GW each contribute their top-3 recommendations. The final list of recommendations was randomly shuffled. Out of the 373 selected users 94 chose to participate.

We sent the list of shuffled recommendations to each of the participants and asked them to rate the entities based on personal relevance on a scale of 0 to 3 (0 meaning irrelevant and 3 indicates the highest degree of personal relevance). We report the average ratings in Table 4.

	GW	CF	RW	PN
Rating	2.40	2.05	2.11	2.09

Table 4: Survey ratings for each algorithm averaged over all participants. (We ran ANOVA with $\alpha = 0.05$ which yielded an F-Value of 6.78 where the critical point value is 2.61 — this indicates that GW's score is statistically significant).

The results show that all algorithms achieve very high relevance ratings from the participants. Table 4 shows that GW achieves an average rating at least %14 higher than that of every other algorithm. Interestingly, RW in the survey slightly outperforms CF which does not correspond well with the offline experiment MAP results reported in Figure 4. We speculate that this has to do with the fact that RW recommends entities that the user was unaware of (hence the lower MAP values) but are none-the-less relevant to her since there are many paths in the social network from the user to those entities. As CF is oblivious to the social network structure, it is not as effective in finding such entities.

6. CONCLUSION

In this work we studied the problem of social entity recommendation in the context of a complex heterogeneous ESN. We devised a supervised learning algorithm called Guided Walk (GW) which builds on existing similarity based recommendations methods. However, unlike the previous methods its learning phase is fully automated and requires no domain knowledge or manual configuration. Moreover, the algorithm is implemented as a Pregel program allowing it to utilize the power of parallel distributed computing. As a result, the algorithm can cope with ever growing social networks by adding more compute nodes to the the cluster.

To evaluate the quality of recommendations we ran an offline experiment as well as a user survey. We show that GW's MAP scores are better than the other baseline algorithms by a factor of, at least, 3.6. Furthermore, the user survey results also confirm that recommendation are not only relevant but also rank high in terms of personal relevance for each user.

7. REFERENCES

[1] D. Agarwal, B.-C. Chen, R. Gupta, J. Hartman, Q. He, A. Iyer, S. Kolar, Y. Ma, P. Shivaswamy,

A. Singh, et al. Activity ranking in linkedin feed. In *SIGKDD*, pages 1603–1612, 2014.

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

[3] E. M. Daly, M. J. Muller, L. Gou, and D. R. Millen. Social lens: Personalization around user defined collections for filtering enterprise message streams. In *ICWSM*, 2011.

[4] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[5] W. Feng and J. Wang. Retweet or not?: personalized tweet re-ranking. In *WSDM*, pages 577–586, 2013.

[6] I. Guy, R. Levin, T. Daniel, and E. Bolshinsky. Islands in the stream: A study of item recommendation within an enterprise social stream. In *SIGIR*, pages 665–674, 2015.

[7] I. Guy, T. Steier, M. Barnea, I. Ronen, and T. Daniel. Finger on the pulse: the value of the activity stream in the enterprise. In *Human-Computer Interaction*, pages 411–428. 2013.

[8] P. Harrington. *Machine learning in action*. 2012.

[9] L. Hong, R. Bekkerman, J. Adler, and B. D. Davison. Learning to rank social update streams. In *SIGIR*, pages 651–660, 2012.

[10] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *ICDM*, 2008.

[11] IBM. Ibm-connections. http://www-03.ibm.com/software/products/en/conn, 2007.

[12] G. Jeh and J. Widom. Simrank: a measure of structural-context similarity. In *SIGKDD*, pages 538–543, 2002.

[13] G. Jeh and J. Widom. Scaling personalized web search. In *WWW*, pages 271–279, 2003.

[14] I. Konstas, V. Stathopoulos, and J. M. Jose. On social networks and collaborative recommendation. In *SIGIR*, pages 195–202, 2009.

[15] P. Kouki, S. Fakhraei, J. Foulds, M. Eirinaki, and L. Getoor. Hyper: A flexible and extensible probabilistic framework for hybrid recommender systems. In *ACM RecSys*, pages 99–106, 2015.

[16] N. Lao and W. W. Cohen. Fast query execution for retrieval models based on path-constrained random walks. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 881–888, 2010.

[17] B. Long, Z. M. Zhang, and P. S. Yu. Co-clustering by block value decomposition. In *SIGKDD*, pages 635–640, 2005.

[18] L. Lovász et al. Random walks on graphs: A survey. *Combinatorics, Paul Erdos is Eighty*, 2:353–398, 1996.

[19] C. Luo, W. Pang, Z. Wang, and C. Lin. Hete-cf: Social-based collaborative filtering recommendation using heterogeneous relations. In *ICDM*, pages 917–922, 2014.

[20] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 135–146, 2010.

[21] C. D. Manning, P. Raghavan, H. Schütze, et al. *Introduction to information retrieval*, volume 1. 2008.

[22] R. S. Marcus and J. F. Reintjes. A translating computer interface for end-user operation of heterogeneous retrieval systems. ii. evaluations. *Journal of the American Society for Information Science*, 32(4):304–317, 1981.

[23] T. Paek, M. Gamon, S. Counts, D. M. Chickering, and A. Dhesi. Predicting the importance of newsfeed posts and social network friends. In *AAAI*, volume 10, pages 1419–1424, 2010.

[24] F. Ricci, L. Rokach, and B. Shapira. *Introduction to recommender systems handbook*. 2011.

[25] C. Shi, X. Kong, Y. Huang, P. S. Yu, and B. Wu. Hetesim: A general framework for relevance measure in heterogeneous networks. *Knowledge and Data Engineering*, 26(10):2479–2492, 2014.

[26] C. Shi, Y. Li, J. Zhang, Y. Sun, and P. S. Yu. A survey of heterogeneous information network analysis. *TKDE*, 2015.

[27] C. Shi, Z. Zhang, P. Luo, P. S. Yu, Y. Yue, and B. Wu. Semantic path based personalized recommendation on weighted heterogeneous information networks. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 453–462, 2015.

[28] Y. Sun and J. Han. Meta-path-based search and mining in heterogeneous information networks. *Tsinghua Science and Technology*, 18(4):329–338, 2013.

[29] Y. Sun, J. Han, X. Yan, P. S. Yu, and T. Wu. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *VLDB*, 2011.

[30] R. Wang, C. Shi, S. Y. Philip, and B. Wu. Integrating clustering and ranking on hybrid heterogeneous information network. In *SIGKDD*, pages 583–594. 2013.

[31] R. S. Xin, J. E. Gonzalez, M. J. Franklin, and I. Stoica. Graphx: A resilient distributed graph system on spark. In *First International Workshop on Graph Data Management Experiences and Systems*, page 2, 2013.

[32] X. Yu, X. Ren, Y. Sun, Q. Gu, B. Sturt, U. Khandelwal, B. Norick, and J. Han. Personalized entity recommendation: A heterogeneous information network approach. In *WSDM*, pages 283–292, 2014.

[33] X. Yu, X. Ren, Y. Sun, B. Sturt, U. Khandelwal, Q. Gu, B. Norick, and J. Han. Recommendation in heterogeneous information networks with implicit user feedback. In *ACM RecSys*, pages 347–350, 2013.

[34] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2, 2012.