Bi-directional Joint Inference for User Links and Attributes on Large Social Graphs

Carl Yang*†‡, Lin Zhong‡, Li-Jia Li‡, Luo Jie‡
†University of Illinois, Urbana Champaign, 201 N. Goodwin Ave, Urbana, IL 61801, USA
‡Snap Research, 64 Market St, Venice, CA 90291, USA
†jiyang3@illinois.edu
‡{lin.zhong, jia.li, roger.luo}@snap.com

ABSTRACT

Users on social networks primarily do two things, connect to existing or new friends and exchange information. Recently, social media apps have become the primary information source for users to consume news stories. Users are inundated with a flood of information from social media networks shared by their friends and other content providers. For social media networks, it is important for us to study users' friendships and interests in order to best serve their need. However, given the nature of online applications, information on both sides is highly incomplete and noisy. Inspired by the well-known homophily phenomenon which found the two properties strongly interleaving, we propose to jointly learn them by leveraging their data redundancy and mutual reinforcement. Specifically, we exploit homophily by iteratively addressing smoothness on the graph in two directions, *i.e.*, from closeness to similarity (stronger links lead to more similar attributes), and vice versa. The two processes are done in a unified probabilistic framework through label propagation and graph construction. The refined user links and attributes are immediately useful for various tasks including link recommendation and content targeting on social networks.

Keywords

attribute profiling, link prediction, social network analysis

1. INTRODUCTION

The problems of *link prediction* and *attribute inference* have been attracting intense research attention ever since the emergence of online social networks. On the one hand, predicting non-existing links in large networks directly enables services like *link recommendation* that add value for both users and the network businesses. The typical goal is to complete the networks, with predicted links that can form but have not formed [1, 2, 10, 18, 29]. On the other hand, inferring user attributes such as location and interest is valuable in tasks like *content targeting*, which improves business efficiency as well as user experience. The basic objective is also to complete the networks, with inferred labels assigned to originally unlabeled users [3, 5, 8, 24, 28].

While each of the two problems is challenging due to the nature of incompleteness and noise in social networks, we note that user links and attributes are highly interleaving, according to the famous concept of *homophily* in network data [5, 15, 28]. We further emphasize that homophily is *bi-directional*, *i.e.*, 1) users linked to each other tend to share similar attributes, and 2) users with more similar attributes tend to link more closely. Based on this concept, we propose to jointly learn user links and attributes on social graphs. However, unlike existing methods that combine links and attributes in a static way by augmenting the networks [5, 22, 28, 29], we propose to dynamically grow the social graph by iteratively learning user links *w.r.t.* inferred user attributes, and inferring user attributes based on learned user links, aiming to fully leverage the interactions and mutual reinforcement between links and attributes.

The challenges lie in coherently combining links and attributes and properly constructing the graph. To address them in a principled way, we develop a unified probability framework. Inspired by the widely used *affinity graphs* in traditional graph learning [11, 31], we model two proximities on social graphs: the *similarity* on attributes and the *closeness* on links. Then we implement homophily by addressing *smoothness* on social graphs, *i.e.*, aligning similarity and closeness. The framework allows us to rigorously infer attributes based on links via label propagation (LP) and recover links based on attributes via graph construction (GC). With properly designed interfaces and pipelines, we propose *Bi-directional joint inference for user Links and Attributes* (BLA), which iteratively addresses smoothness on social graphs in two directions through LP and GC, and fully leverages the mutual reinforcement between links and attributes. During the BLA iterations, we carefully maintain the probability interpretations of attribute assignment and link existence through proper normalizations and avoid over-construction of the graph through regularizations.

Figure 1 gives an intuitive example of BLA that jointly learns user links and attributes. (a) is the original graph, where nodes $\{v_i\}_{i=1}^{6}$ are the users and solid lines are their observed links. v_2, v_3 and v_4 have attribute $\mathcal{A}$, *e.g.*, the same graduating school. Since neither attributes nor links are complete, precise prediction on each side is hard. *E.g.*, it

*The work was done when the first author was on a fulltime internship at Snap Research.

WWW '17 Perth, Australia
ACM 978-1-4503-4914-7/17/04.
http://dx.doi.org/10.1145/3041021.3054181

is hard to directly tell either v_1 or v_5 is more likely to possess $\mathcal{A}$, and either e_{36} (between v_3 and v_6) or e_{45} (between v_4 and v_5) is more likely to exist. (b) and (c) illustrate the joint inference process of BLA that learns both attributes and links. Specifically, after the first round of LP, v_1 and v_5 are inferred with similar attribute probabilities, and then after the first round of GC, a weak link e_{45} is predicted between v_4 and v_5. Later on, due to the existence of e_{45}, $\mathcal{A}$ can be assigned to v_5 with a higher probability than v_1 and v_6. In consequence, e_{45} is predicted as more likely to exist than e_{36}. As we can see, BLA starts from learning the most probable links and attributes and then continues to approximate the complete and precise graph step by step. Note that through proper regularization, we require BLA to converge at graphs like (c) rather than an over-construction, where $\mathcal{A}$ is assigned to all nodes and a link is predicted between every pair.

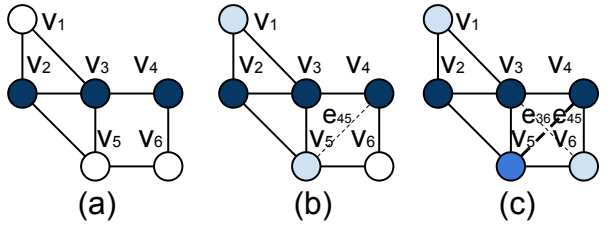


Figure 1: A toy example of a simple social graph.

Besides the challenges of incomplete and noisy graphs, large scales of real-world social networks severely prohibit the direct application of most traditional GC and LP algorithms. *E.g.*, on a large scale real-world social network with millions of users and their links, simply counting the mutual friends among each pair of users takes days to finish on the most advanced PC. To address the efficiency of GC and LP on large social graphs, we design a novel *edge sampling* algorithm that significantly reduces computation without sacrificing performance. We also implement a scalable map-reduce pipeline with Spark GraphX [30], utilizing the efficient *aggregate and pregel* functions.

The framework is designed with the flexibility to plug in various probability adjustment and regularization based on insights about the data. To explore more interesting challenges and opportunities, we perform data analysis on an anonymous subset of Snapchat's social graph including about 100K users with their links and attributes. We utilize the insightful results to develop intuitive sub-models and seamlessly build them into the BLA pipeline. Two examples of effective data-driven sub-models are presented in Sec 5 and the improvements they make are evaluated in Sec 6.

Summary. We propose BLA to learn user links and attributes on social graphs. Our contribution can be summarized as follows:

1. We develop a unified probabilistic framework to iteratively learn user links and attributes, which leverages the data redundancy on each side and the mutual reinforcement between the two (Sec 3).
2. We implement an efficient BLA pipeline on real-world large social graphs with a novel edge sampling method and optimize it to fully utilize the scalable GraphX and MapReduce functions of Spark (Sec 4).
3. We design an adaptive label significance model and a dynamic user activeness model based on real network data analysis and seamlessly integrate them into the BLA framework (Sec 5).
4. We conduct extensive experiments on two public social networks and a large Snapchat dataset to show the supreme efficiency and effectiveness of BLA (Sec 6).

2. RELATED WORK

BLA is closely related to three groups of algorithms: graph construction, label propagation and graph augmentation.

Graph construction (GC) is essential for modeling network data [10]. It basically constructs a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathcal{A}\}$, where $\mathcal{V}$ is the set of nodes (users), $\mathcal{E}$ is the set of edges (links) and $\mathcal{A}$ is the set of attributes (labels) associated on $\mathcal{V}$. However, in social networks, $\mathcal{E}$ is severely sparse and incomplete, *i.e.*, many links that can form do not form, which motivates the task of link prediction. Traditional methods are mostly based on the topology of existing links, leveraging quantities such as edge density and node degree [9], while some also utilize random walks or spectral algorithms [19, 32]. They fail to leverage the rich information in $\mathcal{A}$. A few recent methods exploit user attributes like age and location to alleviate the sparsity of $\mathcal{E}$ [1, 2, 29]. However, since $\mathcal{A}$ itself is also sparse and incomplete, predictions on $\mathcal{E}$ may actually suffer from the noises brought by $\mathcal{A}$.

Label propagation (LP) is a common method for inferring node attributes (labels) based on link structure [33, 34]. However, literature in LP focuses on either efficient propagation of attributes [11, 26] or fast approximation of affinity graphs [6, 12], rather than learning precise graphs with noisy data. As pointed out in [3], in social networks, the lack of precise and complete link structure puts unique challenges to the inference task, which usually leads to poor performance. Specifically, LP enforces the nodes to share similar attributes if they are measured as close on the graph. Thus, it works well only when the closeness is systematically enumerated everywhere, *i.e.*, links between any pair of nodes exist and are properly weighted. This is obviously not the case in social networks, where $\mathcal{E}$ is incomplete and usually unweighted. Therefore, propagating attributes directly on $\mathcal{E}$ can hardly lead to satisfactory results.

As it is intuitive to simultaneously learn $\mathcal{A}$ and $\mathcal{E}$, algorithms based on this idea have also been developed recently, most of which utilize the technique of graph augmentation (GA). With a distance function or network embedding learned on a heterogeneous network augmented by attribute nodes, they are then able to predict new links among unconnected nodes as well as attributes of unlabeled nodes [5, 22, 28, 29]. However, since the links and attributes they use to learn the distances or embeddings are both static, the interactions and mutual reinforcement between $\mathcal{A}$ and $\mathcal{E}$ are not fully exploited.

3. THE BLA FRAMEWORK

3.1 Problem definition

Inspired by the popular affinity graphs in traditional machine learning literature, we use link probabilities and attribute probabilities to formulate *social graphs* which can better model user links and attributes under noisy environment. Instead of using binary values, probabilities pro-

file the quantities in a finer scale. Under the probabilistic framework, relative strengths of user links and attributes are naturally comparable. We formally define our problem as follows:

Input. Given a social network $\mathcal{S}$, we represent its set of users as $\mathcal{V}$, their links as $\mathcal{E}$, and their attributes as $\mathcal{A}$. Therefore, $\mathcal{S}$ can be represented as a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathcal{A}\}$. For most common social networks, $\forall e_{ij} \in \mathcal{E}$ is usually binary, where $e_{ij} = 1$ indicates the existence of a link between v_i and v_j and 0 otherwise. $\forall \boldsymbol{a}_i \in \mathcal{A}$ records the L attributes of v_i. Each component of $\boldsymbol{a}_i$ is a binary value representing whether v_i is associated with a specific attribute value.

Output. Our objective is to learn a complete and precise social graph $\mathcal{G}^* = \{\mathcal{V}, \mathcal{W}, \mathcal{Y}\}$, where each pair of users v_i and v_j is connected by a link with a weight $w_{ij} \in [0, 1], \forall w_{ij} \in \mathcal{W}$, which encodes the probability of v_i and v_j to share a link, and each user v_i is associated with a refined attribute vector $\boldsymbol{y}_i \in [0, 1]^L, \forall \boldsymbol{y}_i \in \mathcal{Y}$, where each component of $\boldsymbol{y}_i$ encodes the probability of v_i to possess a specific attribute (label). The inferred probability measures in $\mathcal{W}$ in the framework effectively differentiate the strengths of existing links and predict the existence of missing links (when $e_{ij} = 0$ and w_{ij} is large), while those in $\mathcal{Y}$ work similarly on existing and missing labels for each user. The probability constraints then effectively avoid the dominance of certain attributes and the explosion of link weights.

3.2 Learning paradigm

BLA is motivated by the bi-directional homophily theory:

1. Users are more likely to link with whom they share more similar attributes with.
2. Users are more likely to share similar attributes with whom they link more closely with.

In the proposed framework, we implement the two properties in a principled way by addressing smoothness, *i.e.*, aligning *similarity* and *closeness* on social graphs in two directions. On the one hand, we use $\mathcal{W}$ to encode the predicted probabilities of links, where w_{ij} can be interpreted as the closeness between v_i and v_j on $\mathcal{G}$, differentiating close friends (with stronge connections) and ordinary friends (with weak connections). On the other hand, we use $\mathcal{Y}$ to record the inferred probabilities of attributes. Thus, $\boldsymbol{y}_i$ and $\boldsymbol{y}_j$ can be used to compute the similarity between two users v_i and v_j. With the two proximities well defined, the first property of BLA can be implemented through computing closeness based on similarity, while the second addressed by constraining similarity *w.r.t.* closeness.

3.3 Learning $\mathcal{W}$

We first introduce our Graph Construction (GC) method for learning $\mathcal{W}$. In social networks, link structures are usually incomplete and edge weights are often missing. *E.g.*, a typical Facebook user is connected to about 100 out of 1 billion users, and thus many links that can be possibly formed are missing [5]. In addition, links between different pairs of friends are in binary form from observation, although strengths of links which indicate the closeness among friends should be different. To properly construct a complete and precise link matrix $\mathcal{W}$ of a social network, we leverage paths on the graph. For each pair of nodes, we want to account for two quantities: the number of paths between the nodes and the importance of these paths. Moreover, we hope that $\mathcal{W}$ should have the proper probability meanings that differentiate the strengths of links.

The Random Walk Theory [4, 29] provides us with a principled solution. Specifically, the one-step transition probability p^1_{ij} of a random walker from v_i to v_j is defined as $p^1_{ij} = w_{ij}/d_i$, where $d_i = \sum_j w_{ij}$. It measures the direct closeness between v_i and v_j in v_i's view. Therefore, the importance of different edges can now be distinguished– it is often the case that the more links one node has, the less important each link is to it on average. Since direct closeness is extremely sparse and incomplete in a large social graph, we intuitively extend the random walk length and use the K-step transition probabilities as a measure of the complete closeness, *i.e.*, $p^K_{ij} = \sum_{\iota \in I} p^K_\iota$. I is the set of all K-step paths between v_i and v_j and $p^K_\iota = \prod_{k=1}^K p^1_k$, where p^1_k is the one-step transition probability on the kth edge passed by path ι. The multiplication of probabilities differentiates the importance of each path and the summation over multiple paths takes the number of paths into consideration.

Thus, the objective of GC can be formalized as follows,

$$J_{GC} = (1-\beta)\sum_{i,j}(w_{ij} - e_{ij})^2 + \beta \sum_{i,j}(w_{ij} - p_{ij})^2, \quad (1)$$

where $e_{ij} \in \mathcal{E}$ is the binary value indicating the existence of a direct edge between v_i and v_j, and p_{ij} is the path-wise transition probability from v_i to v_j. β controls the trade-off between the two terms, depending on how much adjustment we want to make on the original link structure.

Setting $K = 2$ allows us to leverage the individual closeness of v_i and v_j to their mutual friends to infer their actual closeness. It also leads to efficient computations through aggressive reduction. Since data analysis on typical social networks suggests that links between users without any mutual friends are extremely rare [5, 10], the completeness of the constructed graph can still be guaranteed.

While measuring two-step transition probabilities efficiently considers mutual friends, we aim to incorporate user attributes into the process of computing $\mathcal{W}$ as suggested by the first property of the bi-directional homophily theory. For this purpose, before computing the two-step transition probabilities, we manually adjust each p^1_{ij} by multiplying an attribute similarity scaler $s_{ij} = \text{sim}(\boldsymbol{y}_i, \boldsymbol{y}_j)$ (different kinds of similarity measures can be used here, such as the cosine similarity) and then re-normalizing $\{p_{i\cdot}\}$ on each node. In this way, the random walker will 'prefer' edges connecting nodes with more similar attributes.

Note that, while attribute similarity is an intuitive factor leading to properly adjusted transition probabilities, other insightful factors can also be leveraged. We will discuss more about the flexibility of BLA in Sec 5.

3.4 Inferring $\mathcal{Y}$

One standard way to infer node properties on a graph is through Label Propagation (LP), which aims to label all data based on a limited number of labeled ones and an affinity graph describing the closeness among all data [31, 33].

We find that our task of improving the quality of $\mathcal{Y}$ based on $\mathcal{A}$ and $\mathcal{E}$ (or $\mathcal{W}$) is intrinsically similar to the objective of LP. Inspired by the second property of the bi-directional homophily theory, we like the users to get larger influences from friends with stronger links, and the final attributes

to be based on what we originally know about the user as well as the influences from the linked friends. Therefore, we adopt LP to infer attributes for users by constraining attribute similarity according to link closeness on the graph. The objective of LP can be formalized as following,

$$J_{LP} = (1-\alpha)\sum_i ||\boldsymbol{y}_i - \boldsymbol{a}_i||^2 + \alpha \sum_{i,j} w_{ij}||\boldsymbol{y}_i - \boldsymbol{y}_j||^2, \quad (2)$$

where $\boldsymbol{y}_i$ and $\boldsymbol{a}_i$ can be a single attribute value or a vector of multiple attribute values. If multiple values are concerned, propagation can be done separately for each of them. This formulation provides an efficient and principled way to infer user attributes, where the first term leverages the available $\mathcal{A}$ in $\mathcal{S}$, and the second term exploits the local consistency of attributes among linked users. The propagation strength parameter α controls the trade-off between the two terms.

4. SCALABLE BLA LEARNING

Overall, BLA solves the following optimization problem:

$$\hat{\mathcal{W}}, \hat{\mathcal{Y}} = \arg\min_{\mathcal{W},\mathcal{Y}} \big(J_{GC} + J_{LP}\big). \quad (3)$$

To fully leverage the interaction and mutual reinforcement between $\mathcal{Y}$ and $\mathcal{W}$, we adopt an iterative learning process that is similar to the block coordinate descent approach [27]. It works for the optimization of Eq. 3 because if $\mathcal{W}$ is fixed, then Eq. 2 is convex and $\hat{\mathcal{Y}}$ can be obtained, and vice versa. We introduce the inference framework of BLA as below:

- Step 0: Extract the links $\mathcal{E}$ and attributes $\mathcal{A}$ from the given data. Initialize the link probabilities $\mathcal{W}^0$ as $\mathcal{E}$ and attribute probabilities $\mathcal{Y}^0$ as $\mathcal{A}$. Initialize the iterator $t = 0$ and set the maximum iterator T.
- Step 1: Using the current link structures implied by $\mathcal{W}^t$, optimize the attribute probabilities as $\mathcal{Y}^{t+1}$.
- Step 2: Using the current attribute assignments implied by $\mathcal{Y}^{t+1}$, optimize the link probabilities as $\mathcal{W}^{t+1}$. Increase the iterator $t = t + 1$.
- Step 3: Repeat Step 1-2 until $t = T$ or convergence.

We propose to implement Step 1 by LP based on Eq. 2, and Step 2 by GC according to Eq. 1. In the following subsections, we will explain each step in detail.

4.1 Social Graph Initialization

We describe how to compute $\mathcal{A}$ and $\mathcal{E}$ from a given dataset. In social networks, user attributes (labels) can be extracted from various interesting items. *E.g.*, existing user profiles, topically modeled posts (tweets, pictures, articles), semantically clustered contents (links clicked, pages viewed) and so on. We assume there are L such interesting items in a particular network, and would like to obtain an L-dimensional attribute vector $\boldsymbol{a}_i$ for each user v_i. For categorical profiles like those on Facebook[1] and G+[2], we can generate a label l_j for each common value (*e.g.*, value Snap for profile company). For each v_i, we assign 1 to a_{ij} if v_i has the specific profile value and 0 otherwise. For topical posts and clustered contents on social media sites like Twitter[3] and Snapchat[4], we could generate a label l_k for each topic or cluster, and count the number of times t_{ik} that v_i interacts with l_k (e.g., posting a tweet with a specific topic or clicking a link within a specific content group). We assume that for each label l_k, $\{t_{ik}\}_{i=1}^{|\mathcal{V}|}$ follows the power law distribution, where most users interact with l_k in an average frequency while a small portion of users interact in very high frequencies [17]. Therefore, we demote the stress on large volumes of interactions made by the few users and focus on small volumes around the average by using the sigmoid function adjusted to the range of $[0, 1)$ as below.

$$a_{ij} = \sigma\Big(\frac{t_{ij} - \min(t_{\cdot j})}{\bar{t_{\cdot j}} - \min(t_{\cdot j})}\Big), \quad (4)$$

where $\sigma(x) = \frac{1-e^{-x}}{1+e^{-x}}$. t_{ij} is shifted to $[0, \infty)$ by deducting $\min(t_{\cdot j})$ and then normalized by dividing $\bar{t_{\cdot j}} - \min(t_{\cdot j})$.

The computation of $\mathcal{E}$ is similar. *E.g.*, consider the communication frequency c_{ij} among v_i and v_j in chatting Apps like Snapchat and Messenger[5], where more frequent communication indicates stronger connection. Similarly, we still assume a power law distribution for c_{ij}, and compute e_{ij} as

$$e_{ij} = \sigma\Big(\frac{c_{ij} - \min(c_{i\cdot})}{\bar{c_{i\cdot}} - \min(c_{i\cdot})}\Big). \quad (5)$$

We can also consider simpler types of link, *e.g.*, explicit friendship, where we set e_{ij} to 1 if v_i and v_j are friends and 0 otherwise.

4.2 Attribute Inference Via Label Propagation

Based on the graph structure described by link probabilities $\mathcal{W}$, inferring user attributes (labels) can be done through LP on the graph. LP is a well-studied problem in graph-based semi-supervised learning [31, 33]. According to Eq. 2, we aim to learn a labeling function f that applies on all labeled and unlabeled nodes. The value of f should be close to the true labels on labeled nodes, while changing smoothly among all nodes. In this paper, we resort to one specific LP technique described in [31], which directly implements the idea of transductive learning on graphs. The algorithm is closely related to the famous PageRank algorithm [16].

Algorithm 1 describes the LP process. In Steps 4-8, the L labels we consider are propagated one by one, and $y_{\cdot k}$ is the row vector describing the probabilities of every user to possess label l_k. On small graphs where inverting a $|\mathcal{V}| \times |\mathcal{V}|$ matrix is possible, Step 5-7 can be replaced by $y^t_{\cdot k} \leftarrow (I - \alpha S)^{-1} y^0_{\cdot k}$, which directly yields the convergence solution. However, in graphs with millions of nodes as we consider, the inversion is too expensive. We will discuss an efficient implementation of LP In Sec 4.5.

As indicated by Step 6, the learned attributes $\boldsymbol{y}_i$ of each node v_i are coherent to attributes on the neighbors of v_i, and the larger w_{ij} is, the more similar $\boldsymbol{y}_i$ and $\boldsymbol{y}_j$ are. Therefore, the LP process well preserves smoothness on the graph from link closeness in $\mathcal{W}$ to attribute similarity in $\mathcal{Y}$.

To maintain the interpretation of attribute probability, we normalize the results of LP *w.r.t.* each attribute value to maintain the interpretation of attribute probability, *i.e.*, we divide the maximum propagated score for each dimension of $\boldsymbol{y}_i$, $\forall \boldsymbol{y}_i \in \mathcal{Y}$. In this way, the importance of different labels on each user is properly differentiated. Such normalization

[1] https://www.facebook.com/
[2] https://plus.google.com/
[3] https://twitter.com/
[4] https://www.snapchat.com/
[5] https://www.messenger.com/

Algorithm 1 Label Propagation

procedure LP
▷ Input
$\mathcal{W}$: the current graph with weighted user links;
$\mathcal{Y}^0$: the observed user labels to be propagated;
α: the decay factor;
T: the number of maximum iterations.
▷ Output
$\mathcal{Y}^T$: the inferred labels through propagation.
▷ Symmetrically normalize $\mathcal{W}$
$D \leftarrow$ the diagonal matrix with (i,i)-element equal to the sum of the i-th row of $\mathcal{W}$
$S \leftarrow D^{-1/2}\mathcal{W}D^{-1/2}$
for $k = 1 : L$ **do**
  for $t = 1 : T$ **do**
    $y^t_{\cdot k} \leftarrow (1-\alpha)y^0_{\cdot k} + \alpha S y^{t-1}_{\cdot k}$
  end for
end for
end procedure

also avoids the dominance of extremely large label probabilities due to some occasional abnormality, such as the excessive consumptions of contents from certain channels made by a few individuals.

4.3 Link Learning Via Graph Construction

Based on attribute probabilities $\mathcal{Y}$, we aim to learn user links by reconstructing the graph and refining edge weights using transition probabilities as described by Eq. 1. By setting the derivative of w_{ij} to zero, we can directly get

$$w_{ij} = (1-\beta)e_{ij} + \beta p_{ij}, \tag{6}$$

where we firstly update $\mathbf{P}$ with $\mathbf{W}$ fixed, and then update $\mathbf{W}$ with $\mathbf{P}$ fixed. It is done only once in each iteration of GC. The computation of p_{ij} is non-trivial, which involves joining of all edges on the graph. While every path counts in traditional graph theory, in large graphs with billions of edges, it is usually unnecessary and too expensive to retrieve every path. Moreover, paths are of different importance for individual nodes. It is possible to just consider the top most important paths and deliver good performance.

To improve both the efficiency and effectiveness of GC, we design a novel edge sampling method for computing the transition probabilities, which leverages the different importance of paths. Unlike [23] that samples edges by whether it should exist on a graph *w.r.t.* its weight, we sample paths by whether the random walker will pass a specific component edge e_{ij} at each step *w.r.t.* a certain probability θ_{ij}. Since in our GC process, each edge on the graph can be visited by random walkers for multiple times and thus become a part of multiple paths, sampling every time upon constructing each path gives more accurate approximations to the true path-wise transition probabilities. Furthermore, the sampling probability θ can be a fixed value for every considered edge, or any functions on the node-edge triples, providing the flexibility of preferring certain paths for specific tasks. In Sec 6 Figure 6, we show experimental results on the impact of different settings of θ.

We summarize the GC process in Algorithm 2. Step 3 gives the flexibility of adjusting the one-step probabilities with various intuitions and sub-models. We will give two examples of them in Sec 5.

Algorithm 2 Graph Construction

1: **procedure** GC
▷ Input
$\mathcal{Y}$: the current user labels;
$\mathcal{W}^0$: the user links to be reconstructed;
β: the trade-off factor.
▷ Output
$\mathcal{W}^T$: the learned links through reconstruction.
▷ Adjust one-step transition probabilities in $\mathcal{W}^0$
2: **for** each w^0_{ij} **do**
3: $w^1_{ij} \leftarrow w^0_{ij} \cdot \mathrm{sim}(\boldsymbol{y}_i, \boldsymbol{y}_j)$
4: **end for**
▷ Re-normalize the adjusted probabilities
5: **for** each v_i **do**
6: $z_i \leftarrow 0$
7: **for** each $v_j \in \mathcal{N}(v_i)$ **do**
8: **if** $w^1_{ij} > z_i$ **then**
9: $z_i \leftarrow w^1_{ij}$
10: **end if**
11: **end for**
12: **for** each $v_j \in \mathcal{N}(v_i)$ **do**
13: $w^1_{ij} \leftarrow w^1_{ij}/z_i$
14: **end for**
15: **end for**
▷ Reconstruct the graph $\mathcal{W}^T$
16: $\mathcal{W}^T = (1-\beta)\mathcal{W}^0 + \beta\mathcal{W}^1 \cdot \mathcal{W}^1$
17: **end procedure**

In Step 3, as attribute similarity is used to adjust the one-step transition probabilities, the random walker is required to 'prefer' edges connecting similarly labeled nodes. In this way, the GC process well preserves smoothness on the graph from attribute similarity in $\mathcal{Y}$ to link closeness in $\mathcal{W}$.

The normalization in Step 5-15 maintains the link probability interpretation of $\mathcal{W}$, which effectively avoids the explosion of edge weights. The normalized weights still differentiate the closeness of different friends. Links with ignorable normalized weights are removed to keep the graph sparse. In this work, we empirically remove the links around each node with weights that are smaller than 1% of the largest weight.

Step 16 involves the multiplication of two sparse $|\mathcal{V}| \times |\mathcal{V}|$ matrices, which is computational expensive on graphs with millions of nodes. We design an efficient map-reduce pipeline with Spark to implement it efficiently (see Sec 4.5).

4.4 Optimality and Convergence

The decomposition of the objective function into J_{LP} and J_{GC} largely simplifies the optimization problem. In this subsection, we discuss the influence of this decomposition on the overall optimization objective.

The four quadratic terms in Eq. 3 are all convex in $\mathcal{W}$ and $\mathcal{Y}$, respectively. Therefore, following the linearity and composition rules of convexity, the overall objective function is convex. In the iterations of LP and GC, we update $\mathcal{W}$ while fixing $\mathcal{Y}$ and vice versa. Although solving the optimization in iterations might break the convexity, under the smoothness assumption, the two processes should keep minimizing Eq. 3. Specifically, during the LP process, Eq. 2 clearly decreases. If we assume that in $\sum_{i,j}(w_{ij} - p_{ij})^2$, w_{ij} is positively related to p_{ij}, according to smoothness from $\mathcal{Y}$

to $\mathcal{W}$, then Eq. 1 should not increase. Similarly, during the GC process, Eq. 1 clearly decreases. If we assume that in $\sum_{i,j} w_{ij}(y_i - y_j)^2$, $(y_i - y_j)^2$ is negatively related to w_{ij}, according to smoothness from $\mathcal{W}$ to $\mathcal{Y}$, then Eq. 2 should not increase. During the experiments, we also observe quite stable performance of BLA during multiple runs with random initializations on the same data.

The speed of convergence is influenced by the decay factor α in LP and trade-off factor β in GC. During experiments (see Sec 6), we observe that BLA usually achieves more than 80% optimal performance after 3 iterations and converges within 8 iterations on very large data.

4.5 Efficient Spark Implementation

On social graphs consisting of millions of nodes, storing the data in simple matrices and running the algorithm on a single machine are no longer feasible. We implement BLA on Spark with elaborately designed pipelines, which fully leverage its MapReduce and GraphX functions [30].

For the LP process, as we discuss in Sec 4.2, inverting the $|\mathcal{V}| \times |\mathcal{V}|$ matrix becomes intractable. Therefore, we resort to the iterative propagation mechanism as described in Step 5-7 of Algorithm 1. Since a large amount of time is spent on repeatedly joining the vertex RDDs and edge RDDs when propagating each label, we interchange Step 4 and Step 5 and propagate a map of all weighted labels as we consider at each time. This technique effectively improves the efficiency of LP by a factor almost similar to the total number of labels L on the graph. Moreover, we apply early stop and find that the number of iterations does not significantly affect the performance, as shown in Sec 6 Figure 7 (a).

For the GC process, as we discuss in Sec 4.3, multiplying the $|\mathcal{V}| \times |\mathcal{V}|$ matrices is intractable. In this case, we design a propagation based pipeline similar to that of LP to efficiently compute the two-step transition probabilities. Specifically, for each node, we generate a unique *identity label* and propagate it on the graph just like a common attribute. After two iterations of propagation, an identity label l_i found on a node v_j indicates a two-step path from v_i to v_j. Like LP, we use a map to store the identity labels with edge weights, so the transition probabilities can be simply computed by adding up the values of the same keys. The edge sampling method can be easily incorporated into the process by randomly sampling on the edges to propagate the identity labels at each step. In Sec 6 Figure 6, we show that the sampling probability θ has a large impact on both the performance and efficiency of the GC process.

We analyze the complexity of BLA in traditional matrix computation on local machines, which may not be applicable in real large networks. The BLA framework basically consists of three steps: social graph initialization, label propagation and graph construction. The complexities of preprocessing labels and links are $O(|\mathcal{V}|)$ and $O(|\mathcal{E}|)$, respectively. The major computation of LP lies in inverting the Laplacian matrix, which is usually $O(|\mathcal{V}|^3)$, while can be improved to approximately $O(|\mathcal{V}|^2)$ by leveraging link sparsity. GC takes $O(|\mathcal{E}|^2)$ to compute the two-step transition probabilities. Since the maximal iteration can be set to expect certain optimality, the overall computation complexity of BLA is $O(|\mathcal{V}|^2 + |\mathcal{E}|^2)$, where $|\mathcal{V}|$ and $|\mathcal{E}|$ are the numbers of users and links in the network, respectively.

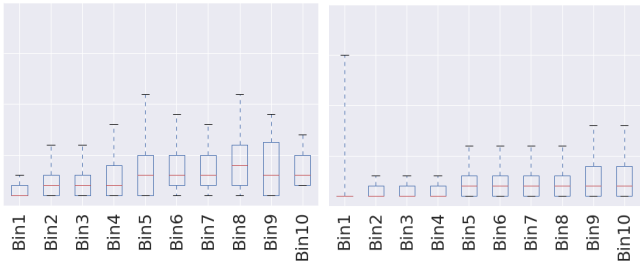
In real large networks with millions of nodes, the BLA framework is implemented on Spark clusters with hundreds of nodes (we use 300 in our largest experiments). Our novel graph propagation pipelines and edge sampling method further improve the efficiency of BLA.

5. DATA DRIVEN BLA SUB-MODELS

In this section, we present data analysis on an anonymous subset of Snapchat's social network. Based on the insights, we develop a dynamic activeness model which takes in user activeness in friend making behaviors, and an adaptive significance model which considers the significance of various attributes (labels). Both models can be seamlessly incorporated into our BLA framework.

The dataset used in this analysis consists of about 100K users those are direct or 2-step friends (friends of friends) of 1K seed nodes randomly sampled from a set of daily active Snapchat users. The user attributes are generated from their interactions with popular Snapchat's Discover channels and top public accounts[6]. The data is fully anonymous.

Dynamic activeness model. One intuitive idea to improve the GC process is to consider user activeness in making friends. While we believe that users' future activeness should be related to their past friend making behaviors, it is unclear how the two sides should be connected. Making more friends in the past might indicate making more friends later, because activeness is lasting. But it may also lead to fewer new friends, because enough friends have been made.



(a) June vs. May (b) June vs. April

Figure 2: User activeness analysis results.

To better understand how past links influence the formation of new links, we take snapshots of links among the same set of users made in each month and perform data analysis on them. Figure 2 shows some insightful results we get. In this study we count the number of links made by each user in April, May and June 2016. Figure 2 (a) shows the number of links made in June grouped by the number in May, and Figure 2 (b) shows the June number grouped by April. From the figures, we can get the following conclusions: 1) users' activeness changes over time; 2) activeness in the past is positively correlated to that in the future; 3) activeness in the more recent past is more related to that in the future.

Inspired by these observations, we model user's dynamic activeness by assuming 1) the more links a node made in the past, the more active it will be later and 2) the older the existing link is, the smaller influence the link has. To this end, we design a dynamic activeness model based on the

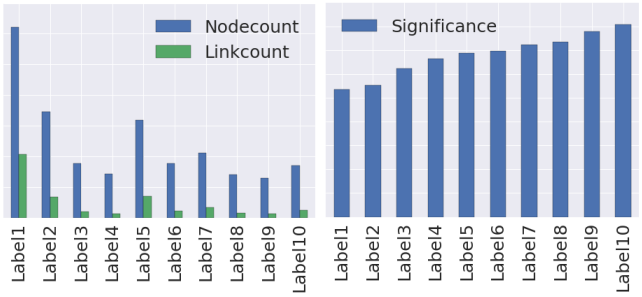
[6]accounts of celebrities, internet personality, organizations and brands who make their story public available and has a large number of followers

exponential family [25], and formulate the user activeness as follows.

$$\Phi(v_i) = \sum_{j:\, v_j \in \mathcal{N}(v_i)} exp\Big(-\frac{\Delta t_{ij}}{\tau}\Big), \tag{7}$$

where $\mathcal{N}(v_i)$ is the set of neighbors of v_i, Δt_{ij} is the time difference between the current time and the time of the creation of link e_{ij} or the last communication between v_i and v_j, and τ is a bandwidth parameter, controlling the decaying speed of the influence of old links. Δt's are float numbers in the unit of a day, which are dynamically computed before each GC process. The influence of old links dies out exponentially as the time difference gets large.

Adaptive significance model. The performance of BLA can be further improved by considering the significance of various attributes (labels). In our Snapchat dataset, we consider various Discover channels as the dimensions of user interests, so different channels may not contribute in the same way to a user's friend making behaviors. *E.g.*, people highly interested in a video game channel may be more likely to make friends with each other compared with people interested in a general news channel.



(a) #nodes and #links (b) Significance

Figure 3: Label significance analysis results (labels anonymized and Y-ticks hidden for privacy issues).

We analyze the Snapchat data to validate our intuition about the significance of label types. Users who read stories of a Discover channel or follow a public account are assigned with a unique label. Figure 3 shows some insightful results we get. In Figure 3 (a), each blue bin is the number of users that have a specific label and each green bin is the number of links made among that group of users. As we can see, some labels potentially lead to more links than the others. *E.g.*, users having Label1 (a popular entertaining channel) make more links than those having Label8 and Label10 (two celebrity public accounts). However, Label1 is so popular as shared by lots of users, so they in fact should not contribute much to the formation of friend links. On the other hand, Label8 and Label10 with smaller numbers of followers but relatively larger numbers of links should be more significant.

The situation reminds us of the widely used TF-IDF in information retrieval [7]. TF-IDF weights the importance of a term T within a document D by multiplying a term frequency (TF) of T in D and dividing an inverse document frequency (IDF) as the number of all documents having T. Inspired by it, we formulate label significance as

$$\Theta(l_i) = \frac{2\mathcal{L}(l_i)}{\mathcal{N}(l_i)^2}, \tag{8}$$

where $\mathcal{N}(l_i)$ is the number of users having label l_i, and $\mathcal{L}(l_i)$ is the number of links made among the $\mathcal{N}(l_i)$ users. $\mathcal{N}(l_i)^2/2$ is about the number of all possible links that can form among the $\mathcal{N}(l_i)$ users. Upon input data, our significance model works similarly as TF-IDF, where the number of links contributes positively to the weight, and the number of nodes contributes negatively. Figure 3 (b) shows the significance computed for the same group of labels.

Integrating sub-models. Our BLA framework is designed with the flexibility for integrating various sub-models developed under validated intuitions. In Algorithm 2 Step 3 as we adjust the one-step transition probabilities on graph, the activeness model is integrated by adding $\Phi(v_i)\Phi(v_j)$ after $sim(\boldsymbol{y}_i, \boldsymbol{y}_j)$, and the significance model is integrated into the computation of $sim(\boldsymbol{y}_i, \boldsymbol{y}_j)$ through re-weighting each dimension of $\boldsymbol{y}$ according to $\Theta(l_\cdot)$. We evaluate the sub-model performances comprehensively in Sec 6 Figure 8.

6. EXPERIMENTS

In this section, we comprehensively evaluate the performance of BLA on three real-world datasets.

On the public Google+ (G+) and Facebook (FB) datasets from [5] and [14], we compare BLA with several state-of-the-art algorithms on the tasks of link prediction and attribute inference.

On our internal Snapchat (SC) dataset, we are able to scale BLA up to millions of users and billions of links and finish within reasonable time (*e.g.*, several hours), which is impossible for most existing algorithms that combine links and contents. On this dataset, we analyze the efficiency of our Spark pipelines and study the impact of various model and parameter settings.

6.1 Experimental Settings

Datasets. Statistics of the three datasets we use are shown in Table 1. More details about the G+ and FB datasets can be found in the original works [5, 14]. In our SC dataset, 1M seed nodes are randomly sampled from a set of daily active users in US and then all of their direct and two-step friends are included. The links are friendships among them, and the attributes are generated based on their interactions with the most popular Discover channels (clicks) and public accounts (follows). For link prediction, the evaluations are done on the new links made in one month; for attribute inference, we uniformly sample 10% of the users and remove their attributes for evaluation.

Dataset	#nodes	#links	#attributes
SC	32,674,735	2,336,875,877	273,742
FB	4,039	88,234	1,282
G+	5,200	8,100	9,539

Table 1: Summary of 3 real-world network datasets.

Compared algorithms. Our proposed BLA algorithm is compared with the following baselines. The parameters of all methods are set via standard 5-fold cross validation.

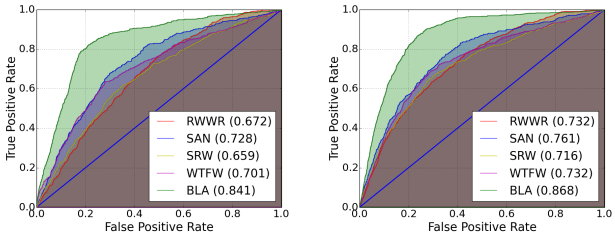
- **RWWR** [29]: a joint link prediction and attribute inference algorithm based on random walk.

- **SAN** [5]: a joint link prediction and attribute inference algorithm based on link features computed by Adamic-Adar and Low-Rank Approximation.
- **SRW** [1]: a link prediction algorithm based on supervised random walk guided by user labels.
- **WTFW** [2]: a link prediction algorithm based on a probabilistic generative model that simultaneously learns and explains social links.
- **RNC** [13]: a simple attribute inference algorithm that leverages labeled neighbors without learning.
- **EdgeExp** [3]: an attribute inference algorithm that leverages a softmax function to solve for both user attributes and relationship types.

Metrics. We use ROC and Precision (Pr@K) to measure the performance of link prediction and attribute inference, respectively, which is commonly done in related literature [5, 29]. Attribute predictions are made on unlabeled nodes with higher inferred label probabilities than the average value on the labeled nodes. We also use minute to measure the runtime.

6.2 Performance evaluation

BLA effectiveness. We evaluate the effectiveness of BLA on the public G+ and FB datasets.



(a) Performance on G+ (b) Performance on FB

Figure 4: Link prediction evaluation in ROC.

Algorithm	G+ Dataset		
	Pr@2±*std*	**Pr@3**±*std*	**Pr@4**±*std*
RWWR	0.433±*.011*	0.564±*.008*	0.738±*.009*
SAN	0.417±*.020*	0.549±*.021*	0.714±*.018*
RNC	0.291±*.006*	0.374±*.005*	0.472±*.005*
EdgeExp	0.454±*.016*	0.591±*.016*	0.770±*.015*
BLA	**0.494**±*.008*	**0.642**±*.009*	**0.833**±*.008*
Algorithm	**FB Dataset**		
	Pr@2±*std*	**Pr@3**±*std*	**Pr@4**±*std*
RWWR	0.465±*.009*	0.611±*.010*	0.823±*.010*
SAN	0.476±*.018*	0.632±*.018*	0.830±*.019*
RNC	0.328±*.004*	0.433±*.005*	0.578±*.004*
EdgeExp	0.523±*.014*	0.678±*.014*	0.886±*.014*
BLA	**0.542**±*.006*	**0.702**±*.007*	**0.931**±*.008*

Table 2: Attribute inference evaluation in Pr@K.

Figure 4 and Table 2 show the link prediction and attribute inference results, respectively. Our experiments were run 10 times over different random training/testing set splits. Besides the average values of the metrics, Figure 4 also shows the ROC curves belonging to the scores that are the closest to the averages, while Table 2 also shows the standard deviations (*std*). The results of all baselines in comparison with BLA have passed our paired t-test with p value $p < 0.01$.

As we can see, BLA achieves the best performance in both tasks. It is 16% and 12% better than the second runner method in link prediction, and also significantly better than all baselines in attribute inference, on both datasets. BLA excels on networks with sparser links and labels like G+. The second runners (*i.e.* SAN and EdgeExp) on the two tasks both learn links and attributes simultaneously, which shows the effectiveness of joint learning. The better performance of BLA further indicates the advantage of the bi-directional inference between user links and attributes in iterations.

BLA efficiency. We show the efficiency of our BLA Spark pipelines in Figure 5. We randomly sampled 10 subnetworks of different sizes from our internal SC dataset and ran BLA on a local PC versus Spark clusters. The local PC we use has two 2.5 GHz Intel i7 processors and 8GB memory. For Spark, we use a cluster of 4 n1-highmem-16 (16 vCPU, 104GB memory) machines from the Google Cloud Platform. As we can see, the runtime of BLA on Spark is much less and grows much slower than that on the local PC as the size of network increases. Notice that, the largest network we consider here only has about 100K users, but BLA on a local PC takes several hours to finish. The situations are similar for other baselines, and most of them cannot be trivially scaled up to real-world networks.

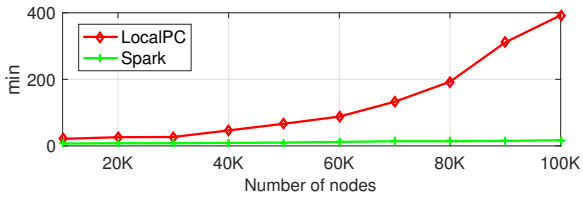
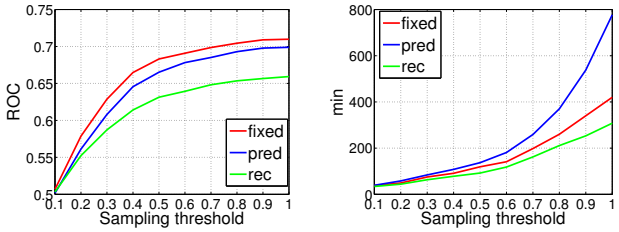


Figure 5: Runtimes of BLA on a local PC vs Spark with different sizes of user.

BLA model selection. On our internal large SC dataset, we conduct comprehensive experiments on different parameter and sub-model settings to further study the performance of BLA. Due to space limit, we only show the evaluation results on link prediction.



(a) Performance (b) Runtime

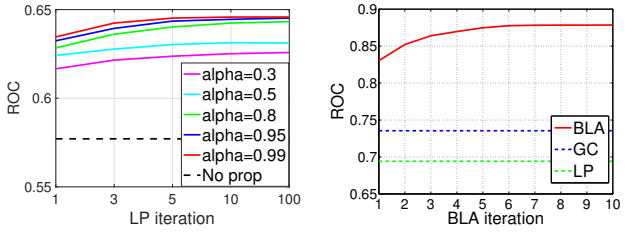
Figure 6: GC evaluation with varying settings of θ.

As discussed in Sec 4, for the GC process, we study how different settings of the edge sampling probabilities θ influence the performance. We experiment on three strategies:

- *fixed*: we set all θ_{ij} to be a fixed sampling threshold ρ.
- *pred*: we set θ_{ij} as ρ scaled by a convex half sigmoid function parameterized by the number of out links of v_i. In this way, we hope to get better prediction results by sampling more paths for nodes with more links.
- *rec*: we set θ_{ij} similarly as for *pred*, but scaled by a concave half sigmoid function. This might be better for the recommendation task, where we sample more paths for nodes that do not have a lot of links.

As we can see in Figure 6, the performance always converges fast as ρ increases to 0.6 (which corresponds to about 85% of the best performance), and the runtime goes up fast as ρ increases. The *fixed* strategy performs the best. It could be explained as it approximates the true two-step transition probabilities by granting the same chance for every edge to survive the sampling process. Such empirical results lead us to believe that fixing θ to a value around 0.6 yields a good trade-off between efficiency and effectiveness.

For the LP process, we study how the propagation strength α may influence the convergence and performance, and how early we can stop the propagation to save computation without significantly sacrificing performance. As we can see in Figure 7 (a), LP achieves optimal performance almost immediately after very few iterations with various values of α, which indicates that the attributes of close neighbors are most effective in the inference. As α increases to a value closer to 1, the performance gets better, which is consistent with previous studies on LP [26, 31].



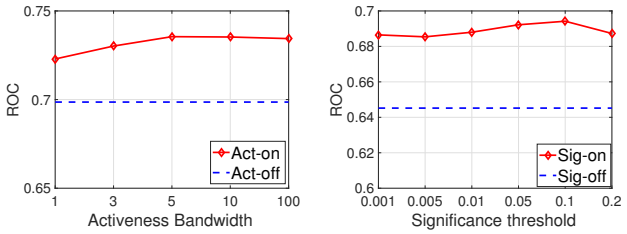
(a) LP performance (b) BLA performance

Figure 7: LP and BLA(LP+GC) evaluation with varying iteration numbers.

We compare the overall BLA framework against the best settings of GC and LP processes alone. As we can see in Figure 7 (b), BLA converges rapidly with a few iterations and the improvement over GC and LP is significant, i.e., around 20% and 27% respectively. The results indicate that by leveraging the mutual reinforcement between user attributes and links, BLA is advantageous in predicting new links.

Figure 8 (a) presents the improvements made by the dynamic activeness model. We evaluate the GC performance with varying activeness bandwidth τ in the unit of a day. The larger τ is, the longer past friends making behaviors count. As we can see, the activeness model significantly improves the GC performance by around 4.2% and different τ values do not influence the performance much. Users' friend making behaviors in the recent 5-10 days are the most informative when computing their current activeness.

Figure 8 (b) demonstrates the improvements brought by the adaptive significance model. The LP performance is measured against varying significance thresholds θ. Attributes with significance lower than θ are eliminated in the computation of similarity. As can be seen, we achieve about 5.5% better LP performance by incorporating the significance model and θ does not influence the performance much in the specific range. The best performance is achieved with $\theta = 0.1$, when less important labels are properly eliminated.



(a) Activeness model (b) Significance model

Figure 8: Sub-model evaluations with varying τ, θ.

7. CONCLUSION

BLA is a general framework that effectively completes and refines social graphs in terms of links and attributes. On each side, the data redundancy is leveraged to refine the existing information and infer the missing. The close loop of GC and LP then utilizes the mutual reinforcement between links and attributes.

Recently, research in community detection indicates that learning the complete graphs might also be a necessary prelude for efficient network clustering [20], while research in security and smart policing also suggests that the predicted graphs can be compared with the existing networks to detect outliers as potential offenders [21]. Beyond social networks, BLA can also be used to effectively complete and refine information in various domains such as research bibliographic networks, where future co-authorships as links and potential research interests as attributes can be jointly predicted, and biomedical interaction networks, where non-observed protein-protein interactions as links and biological pathways as attributes can be mutually reinforced.

8. REFERENCES

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

[2] Barbieri, N., Bonchi, F., and Manco, G. Who to follow and why: link prediction with explanations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining* (2014), ACM, pp. 1266–1275.

[3] Chakrabarti, D., Funiak, S., Chang, J., and Macskassy, S. Joint inference of multiple label types in large networks. In *ICML* (2014), pp. 874–882.

[4] Fang, Y., Chang, K. C.-C., and Lauw, H. W. Graph-based semi-supervised learning: Realizing pointwise smoothness probabilistically. In *ICML* (2014).

[5] Gong, N. Z., Talwalkar, A., Mackey, L., Huang, L., Shin, E. C. R., Stefanov, E., Shi, E. R., and

Song, D. Joint link prediction and attribute inference using a social-attribute network. *ACM Transactions on Intelligent Systems and Technology (TIST) 5*, 2 (2014), 27.

[6] Jebara, T., Wang, J., and Chang, S.-F. Graph construction and b-matching for semi-supervised learning. In *Proceedings of the 26th Annual International Conference on Machine Learning* (2009), ACM, pp. 441–448.

[7] Leskovec, J., Rajaraman, A., and Ullman, J. D. *Mining of massive datasets*. Cambridge University Press, 2014.

[8] Li, R., Wang, C., and Chang, K. C.-C. User profiling in an ego network: co-profiling attributes and relationships. In *WWW* (2014), pp. 819–830.

[9] Li, Z., Fang, X., and Sheng, O. R. L. A survey of link recommendation for social networks: Methods, theoretical foundations, and future research directions. *Theoretical Foundations, and Future Research Directions (October 28, 2015)* (2015).

[10] Lichtenwalter, R. N., Lussier, J. T., and Chawla, N. V. New perspectives and methods in link prediction. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining* (2010), ACM, pp. 243–252.

[11] Lin, B., Yang, J., He, X., and Ye, J. Geodesic distance function learning via heat flow on vector fields. In *Proceedings of the 31th International Conference on Machine Learning* (2014).

[12] Liu, W., He, J., and Chang, S.-F. Large graph construction for scalable semi-supervised learning. In *Proceedings of the 27th international conference on machine learning (ICML-10)* (2010), pp. 679–686.

[13] Macskassy, S. A., and Provost, F. A simple relational classifier. Tech. rep., DTIC Document, 2003.

[14] McAuley, J. J., and Leskovec, J. Learning to discover social circles in ego networks. In *NIPS* (2012), vol. 2012, pp. 548–56.

[15] McPherson, M., Smith-Lovin, L., and Cook, J. M. Birds of a feather: Homophily in social networks. *Annual review of sociology* (2001), 415–444.

[16] Page, L., Brin, S., Motwani, R., and Winograd, T. The pagerank citation ranking: bringing order to the web.

[17] Reed, W. J. The pareto, zipf and other power laws. *Economics Letters 74*, 1 (2001), 15–19.

[18] Rodriguez, M. G., Balduzzi, D., and Schölkopf, B. Uncovering the temporal dynamics of diffusion networks. In *Proceedings of the 28th International Conference on Machine Learning* (2011).

[19] Rosvall, M., and Bergstrom, C. T. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences 105*, 4 (2008), 1118–1123.

[20] Ruan, Y., Fuhry, D., and Parthasarathy, S. Efficient community detection in large networks using content and links. In *WWW* (2013), pp. 1089–1098.

[21] Shaabani, E., Aleali, A., Shakarian, P., and Bertetto, J. Early identification of violent criminal gang members. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2015), ACM, pp. 2079–2088.

[22] Tang, J., Qu, M., and Mei, Q. Pte: Predictive text embedding through large-scale heterogeneous text networks. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2015), ACM, pp. 1165–1174.

[23] Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., and Mei, Q. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web* (2015), ACM, pp. 1067–1077.

[24] Tang, L., and Liu, H. Relational learning via latent social dimensions. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining* (2009), ACM, pp. 817–826.

[25] Welling, M., Rosen-Zvi, M., and Hinton, G. E. Exponential family harmoniums with an application to information retrieval. In *Nips* (2004), vol. 4, pp. 1481–1488.

[26] Xu, B., Bu, J., Chen, C., Cai, D., He, X., Liu, W., and Luo, J. Efficient manifold ranking for image retrieval. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval* (2011), ACM, pp. 525–534.

[27] Xu, Y., and Yin, W. A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion. *SIAM Journal on imaging sciences 6*, 3 (2013), 1758–1789.

[28] Yang, S.-H., Long, B., Smola, A., Sadagopan, N., Zheng, Z., and Zha, H. Like like alike: joint friendship and interest propagation in social networks. In *Proceedings of the 20th international conference on World wide web* (2011), ACM, pp. 537–546.

[29] Yin, Z., Gupta, M., Weninger, T., and Han, J. A unified framework for link recommendation using random walks. In *Advances in Social Networks Analysis and Mining (ASONAM), 2010 International Conference on* (2010), IEEE, pp. 152–159.

[30] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. Spark: cluster computing with working sets. *HotCloud 10* (2010), 10–10.

[31] Zhou, D., Weston, J., Gretton, A., Bousquet, O., and Schölkopf, B. Ranking on data manifolds. *Advances in neural information processing systems 16* (2004), 169–176.

[32] Zhou, D., Zhu, S., Yu, K., Song, X., Tseng, B. L., Zha, H., and Giles, C. L. Learning multiple graphs for document recommendations. In *Proceedings of the 17th international conference on World Wide Web* (2008), ACM, pp. 141–150.

[33] Zhu, X., and Ghahramani, Z. Learning from labeled and unlabeled data with label propagation. Tech. rep., Citeseer, 2002.

[34] Zhu, X., Ghahramani, Z., Lafferty, J., et al. Semi-supervised learning using gaussian fields and harmonic functions. In *ICML* (2003), vol. 3, pp. 912–919.