StructInf: Mining Structural Influence from Social Streams

Jing Zhang†, Jie Tang‡, Yuanyi Zhong‡, Yuchen Mo†, Juanzi Li†, Guojie Song‡, Wendy Hall‡, and Jimeng Sun‡

†Information School, Renmin University of China
‡Department of Computer Science and Technology, Tsinghua University
†Computational Science and Engineering at College of Computing, Georgia Institute of Technology
‡Key Laboratory of Machine Perception, Peking University
‡Electronics and Computer Science, University of Southampton

zhang-jing@ruc.edu.cn, {jietang, lijuanzi}@tsinghua.edu.cn, wh@ecs.soton.ac.uk, gjsong@pku.edu.cn, jsun@cc.gatech.edu

Abstract

Social influence is a fundamental issue in social network analysis and has attracted tremendous attention with the rapid growth of online social networks. However, existing research mainly focuses on studying peer influence. This paper introduces a novel notion of structural influence and studies how to efficiently discover structural influence patterns from social streams. We present three sampling algorithms with theoretical unbiased guarantee to speed up the discovery process. Experiments on a big microblogging dataset show that the proposed sampling algorithms can achieve a 10× speedup compared to the exact influence pattern mining algorithm, with an average error rate of only 1.0%. The extracted structural influence patterns have many applications. We apply them to predict retweet behavior, with performance being significantly improved.

Introduction

Social influence occurs when one’s behaviors or opinions are affected by others. It forms a fundamental mechanism governing the dynamics of social networks, and recently, has attracted tremendous attention with the availability of large online social behavior data. For example, a field experiment conducted on Facebook shows strong evidence of social influence on political mobilization (Bond et al. 2012). The theory of three degrees of influence (Fowler, Christakis, and others 2008) claims that our behavior can influence people we have never met. However, the underlying mechanism is still unclear and a thorough investigation is thus needed.

We show an example of retweet behavior influence in three different structures on a dataset of Sina Weibo† in Figure 1. The red node represents a neighboring user (followee in Weibo) who retweets a message before time \( t \); the white node denotes the target user to be studied. The general question is how likely it is that the target user will retweet this message at time \( t' \) (0 < \( t' - t \) ≤ \( \tau \), \( \tau \) is a short time interval), conditioned on different influence structures. We can see several interesting patterns. First, the conditional probability in Figure 1(b) increases to 150% higher than that of Figure 1(a), suggesting more active neighbors can improve the retweet likelihood. On the other hand, the probability in Figure 1(c) increases to 300% higher than that of Figure 1(b). The difference between them is the relationship between the neighbors. Please note that the target user may be unaware of such a relationship. How does the network structure formed by friends matter in influencing users’ behavior changes? What are the most significant influence structures hidden in the huge volume of streaming behavior data? Although social influence has been extensively studied before (Anagnostopoulos, Kumar, and Mahdian 2008; Kempe, Kleinberg, and Tardos 2003; Tang et al. 2009), most of the existing work focused on peer influence, and ignored the effect of such influence structures. Ugander et al. (2012) presented the idea of structural diversity and showed that diverse structures of neighboring users can influence user’s behaviors. However, they only focused on investigating qualitative effects of social influence, but ignored quantitative estimation.

In this work, we formalize the problem of mining structural influence from the social stream. Specifically, given large user behavior logs and the network structure among users, how can we discover influence structures with high confidence? The discovered influence structures can be applied in many different applications, e.g., to be used as features in a paper’s citation network to predict whether the paper will be extensively cited (Shi, Leskovec, and McFarland 2010), or to predict one’s “following” behaviors in a user’s “following” network (Zhang et al. 2015).

We address the above issue and make the following contributions: (1) we formally define a novel notion of structural influence; (2) to handle large streaming behavior data, we propose an exact and several sampling algorithms to quickly estimate structural influence; we provide theoretical unbiased guarantees for the sampling algorithms; (3) our empirical study on a large Weibo dataset, show that the proposed sampling algorithms achieve a 10× speedup and re-
sult in an average error rate of only 1.0%, compared to our exact counting algorithm; (4) finally, we demonstrate the effect of structural influence on retweet behavior prediction.

**Problem Formulation**

Let \( G = (V, E) \) denote a social network, where \( V \) is a set of users and \( E \subset V \times V \) is a set of relationships. We use \( v_i \in V \) to represent a user and \( e_{ij} \in E \) to represent a relationship between \( v_i \) and \( v_j \). A relationship can be either directed or undirected. With a directed relationship \( e_{ij} \), we only consider one-way influence from \( v_i \) to \( v_j \). An undirected relationship can be divided into two directed relationships, \( e_{ij} \) and \( e_{ji} \), and the influence is also two-way. We use \( N(v_i) \) to indicate the neighbors of user \( v_i \). Another input of our problem is a stream of user action logs.

**Definition 1 Action Logs.** Let \( L \) denote a stream of action logs, where each log entry \( l \in L \) is a triple \((v, a, t)\), representing user \( v \in V \) performed action \( a \in A \) at time \( t \). Here \( A \) is a set of action types.

For example, in a microblogging site, a retweet behavior is an action, and each tweet can be considered as an action type. We build an action diffusion graph from \( G \) and \( L \).

**Definition 2 Action Diffusion Graph.** An action diffusion graph, \( G^p = (L, E^p) \), is a directed graph, where each node is a log entry in \( L \), and each edge \( e_{ij}^p \in E^p \) indicates two log entries \((v_i, a, t_i)\) and \((v_j, a, t_j)\) satisfy \( 1) (v_i, v_j) \in E \) and \( 2) 0 < t_j - t_i - \tau \leq \tau \), where \( \tau \) is a short time interval.

We use \( N^p(l) \) to indicate the neighbors in \( G^p \) that point to \( l \). In an action diffusion graph, an edge \( e_{ij}^p \) represents that when user \( v_i \) performs action \( a \) at time \( t_i \), \( v_i \) has a potential influence on her neighbor \( v_j \) to perform \( a \) at \( t_j \), a short time interval \( \tau \) after \( t_i \). We define \((v_i, a, t_i)\) as influencing action and \((v_j, a, t_j)\) as target action. If later \((v_j, a, t_j)\) actually happens in \( L \), we call it as active target action; otherwise inactive target action. Different from conventional research that mainly decomposes the influence from neighbors into peer influences, we propose a new notion as:

**Definition 3 Structural Influence.** When the target user \( v_j \) performs action \( a \) at \( t_j \), and her \( \gamma \)-degree (\( \gamma \geq 1 \)) friends who already performed the action before \( t_j \) \((0 < t_j - t_i \leq \tau, t_i \) is the latest time when a friend performed the action\), we define structural influence as the combination effect of peer influences exerted by those \( \gamma \)-degree active friends on the target user, when the friends and target user form an influence structure. Table 1 lists all the structures when considering 1, 2, and 3 active \( \gamma \)-degree friends. We also name influence structure as structural influence pattern and use \( C_k \) to represent the \( k \)-th pattern. An instance of \( C_k \) for any action is named a pattern instance, and is denoted by \( c_k^i \).

Structural influence can be formulated as a conditional probability, \( IP_k = \frac{x_k}{x_k + y_k} \), where \( x_k \) represents the frequency of the instances with pattern \( C_k \) and target action \( l_i \) being active in \( L \), while \( y_k \) is that of \( C_k \)’s instances with \( l_i \) being inactive in \( L \). Given the above definition, the key question we want to answer is to efficiently discover the structural influence patterns with high influence probability. Note that a node may be influenced by multiple pattern instances, and this combination effect can be modeled by a partial credit model or a cascade model (Goyal, Bonchi, and Lakshmanan 2010). This paper simply follows the assumption of Bernoulli distribution (Goyal, Bonchi, and Lakshmanan 2010) to estimate structural influences from different pattern instances independently. Other assumptions will be studied in the future. Note also that when enumerating an instance, we assign the maximal matched pattern for it. For example, if an instance can be matched to \( C_4 \) in Table 1, the sub patterns of \( C_4 \), such as \( C_2 \) and \( C_3 \), will not be matched.

Later, we aim at quickly estimating \( IP_k \) for each influence pattern. One potential application is to incorporate the patterns of high probabilities as features to predict user behaviors, and the details will be given in the experimental section.

**StructInf: Structural Influence Estimation**

In this section, we begin by introducing an exact algorithm to estimate the structural influence from social streams, based on which we propose three fast sampling strategies.

**StructInf-Basic**

For each pattern \( C_k \), the goal is to estimate \( x_k \) and \( y_k \), and based on which to calculate structural influence of \( C_k \). Assume the social network is static, while the action logs are very large and arrive in real time. Our approach loads the static network into memory at the beginning and update \( x_k \) and \( y_k \) whenever an action log arrives\(^2\). The key idea of the approach is to 1) identify active and inactive target actions; and 2) enumerate the structural influence patterns from the target actions backwards along the diffusion edges.

**Identifying Target Actions.** We propose Algorithm 1 to estimate \( x_k \) and \( y_k \). We maintain an action diffusion graph \( G^p \), a queue \( Q \) and a hashable \( H \), to record the diffusion edges and action logs within recent time interval \( \tau \). For each newly arrived action, \((v_i, a, t_i)\), we first add it into \( Q \) and \( H \) (Line 4), and then update \( \bar{x} \) (Lines 5-8) and \( \bar{y} \) (Lines 9-19).

To estimate \( \bar{x} \), we need to figure out active target actions. Obviously, each newly arrived action is an active target action. Thus when each action \( l_i = (v_i, a_k, t_i) \) arrives, we first find those neighbors of \( v_i \) who are active in \([t_i - \tau, t_i]\), and create the new diffusion edges from the actions of those active neighbors to \( l_i \) in \( G^p \) (Lines 5-7), and then enumerate the structural influence patterns starting from \( l_i \) (Line 8).

To estimate \( \bar{y} \), we need to figure out inactive target actions. To achieve this, we enumerate the influence patterns when an action is outdated rather than newly arrived, because for any action \((v, a, t)\), we can only know whether user \( v \)'s neighbors are active or not within \([t, t + \tau] \) until time \( t + \tau \), which is exactly the time that \((v, a, t)\) is popped up from \( Q \). In particular, whenever a new action arrives, we remove the outdated actions from \( Q \) and \( H \) (Lines 9-10). For each removed action \( l_r = (v_r, a_k, t_j) \), we identify the neighbors of \( v_j \) that are inactive in \([t_j, t_j + \tau]\) (Lines 11-12), and assign a virtual time \( t_j + \tau \) to each inactive neighbor\(^\dagger\).\(^\dagger\)

\(^2\)In our implementation, storing a static network with millions of nodes and edges costs about 5G memory.
Virtual diffusion edge from we complete the diffusion edges from other active actions to \( V \) \( V \) active or inactive target action into how to enumerate all possible influence patterns from a arbitrary action \( V \) actions in \( V \) algorithm 1), and make sure that, anytime, the labels of the replicate enumeration. To achieve this, we assign each action \( V \) time. According to Algorithm 2, we first add the target ac-

![Figure 2: Illustration of pattern enumeration.](image)

... (Line 14). Finally, we complete the diffusion edges from other active actions to \( l_t \) \( l_t \) (Lines 15-17). After enumerating from \( l_t \) (Line 18), we remove the created virtual diffusion edges (Line 19).

Enumerating Influence Patterns. Algorithm 2 presents how to enumerate all possible influence patterns from a given action log. The basic idea is: we start by adding an active or inactive target action into \( V_{in} \), and their neighboring actions into \( V_{ext} \). Then we extend \( V_{in} \) by selecting an arbitrary action \( l' \) from \( V_{ext} \), and update \( V_{ext} \) by adding all the neighbors of \( l' \) that are not included in \( V_{in} \).

One thing worth noting is the necessary of avoiding duplicate enumeration. To achieve this, we assign each action an incremental (unique) label when it arrives (Line 3 in Algorithm 1), and make sure that, anytime, the labels of the actions in \( V_{ext} \) are smaller than those in \( V_{in} \) (Line 7 in Algorithm 2). Figure 2 illustrates the enumeration process under this strategy. The left part demonstrates an action diffusion graph with all the nodes denoted by incremental labels over time. According to Algorithm 2, we first add the target action \( d_4 \) into \( V_{in} \), and \( d_4 \)'s neighbors, namely \( d_3 \) and \( d_2 \), into \( V_{ext} \), and then extend \( V_{in} \) by selecting the actions from \( V_{ext} \) one by one. We can see that when selecting \( d_2 \) into \( V_{in} \), \( d_3 \) is removed from \( V_{ext} \) because the label of \( d_3 \) is larger than that of \( d_2 \) in \( V_{in} \). A similar idea of enumerating subgraphs was used in (Wernicke 2006). They studied static networks, while we are dealing with the streaming behavior data, and thus the derived action diffusion graph evolves over time.

When invoking Algorithm 2, we first induce a subgraph by including all the edges between nodes in \( V_{in} \), and then determine which pattern the induced graph belongs to (Line 2). This is essentially a problem of graph isomorphism. When the pattern is small, we can use the number of nodes/edges and degree sequences, to uniquely identify a pattern, and leverage approximate solutions when the pattern gets larger (Leskovec, Singh, and Kleinberg 2006). The enumeration stops when the considered maximal number of nodes, i.e., \( N \), is reached (Lines 3-4).

Discussions. In summary, the proposed StructInf-Basic algorithm is a streaming approach that only needs one-time scan of the streaming action logs and can carefully avoid duplicate enumeration by a dynamic labeling mechanism. Some other methods proposed by (Kashtan et al. 2004) and (Yan and Han 2002) extend neighboring edges of the selected edges, rather than extending nodes. However, when extending edges, the speeding up is not easy as that of extending nodes (Wernicke 2006). Regarding the time complexity, in Algorithm 1, the time complexity for enumerating active target actions is \( O(|L|d_{max}) \), and is \( O(|L|d_{max}^2) \) for inactive target actions, where \( d_{max} \) is the maximal degree of \( G \). The complexity of Algorithm 2 is \( O(d_{max}^4)^N \), where \( d_{max}^p \) is the maximal degree of \( G^p \) and \( N \) is the maximal number of nodes in all the considered influence patterns. In summary, the total time complexity is \( O(|L|d_{max}^p(d_{max}^p)^N) \).

StructInf-S: Fast Sampling Algorithms

StructInf-S1. The time complexity of StructInf-Basic is high because it enumerates all possible influence patterns for each target action. To speed up the algorithm, we propose several sampling strategies. The basic requirement is to guarantee unbiased estimation of \( x_k \) and \( y_k \). To achieve this, outside Line 8 and 18 in Algorithm 1, we add judgment statements to determine whether target action \( l_t \) will be enumerated by a probability \( p \) respectively. In the same way, we...
In StructInf-S1, the probability is $p$ to the graph. Specifically, in Algorithm 1, outside Lines 6-7, we then enumerate the influence patterns based on the sampled instances, $C_k$, is selected uniformly according to probability $p^{n_i}$, where $n_i$ is the number of nodes in $C_k$.

**Proposition 1** Let $x_k$ be the exact number by StructInf-Basic, and $\tilde{x}_k$ be the approximate number by StructInf-S1, then $\tilde{x}_k = \frac{x_k}{p^{n_k}}$ is an unbiased estimator for $x_k$.

**Proof 1** The expectation of $\tilde{x}_k$ can be written as

$$E(\tilde{x}_k) = \sum_{n=1}^{S} p(s_n)(\tilde{x}_k/p^{n_k})s_n$$

where $s_n$ represents the $n^{th}$ sample of pattern instances and there are $S$ possible samples in the whole pattern space. Notation $(\tilde{x}_k/p^{n_k})s_n$ represents the value of $\tilde{x}_k/p^{n_k}$ that is estimated in the sample $s_n$. We can factorize $\tilde{x}_k/p^{n_k}$ common to the $i^{th}$ pattern $c_i^k$ and sum over the population:

$$E(\tilde{x}_k) = \sum_{i=1}^{x_k} \frac{1}{p^{n_k}} \sum_{c_i \in s_n} p(s_n)$$

where $\sum_{c_i \in s_n} p(s_n)$ indicates the summation over the probabilities of all samples containing the instance $c_i^k$, which is actually the prior probability that $c_i^k$ is selected in a sample.

In StructInf-S1, the probability is $p^{n_k}$, thus we have

$$E(\tilde{x}_k) = \sum_{i=1}^{x_k} \frac{1}{p^{n_k}} \sum_{c_i \in s_n} p(s_n) = x_k$$

According to the sampling theory (Horvitz and Thompson 1952), the unbiased estimation of the variance of $\tilde{x}_k$ is:

$$\hat{V}(\tilde{x}_k) = \sum_{i=1}^{x_k} \frac{1 - p(c_i^k)}{p^2(c_i^k)} + \sum_{i \neq j} \frac{p(c_i^k c_j^k) - p(c_i^k)p(c_j^k)}{p(c_i^k) p(c_j^k) p(c_i^k) p(c_j^k)}$$

(1)

The above variance is determined by both the approximate estimation, $\tilde{x}_k$, and the sampling probability for a pattern, $p(c_i^k)$. According to the central limit theorem, the sum of a random sample of a large enough size from an arbitrary distribution follows approximately a normal distribution, i.e., $\tilde{x}_k \sim N(x_k, \hat{V}(\tilde{x}_k))$. Thus, the probability of the sampling error with confidence $1 - \alpha$ is:

$$p \left[ \tilde{x}_k - z_{\alpha/2} \sqrt{\hat{V}(\tilde{x}_k)} \leq x_k \leq \tilde{x}_k + z_{\alpha/2} \sqrt{\hat{V}(\tilde{x}_k)} \right] = 1 - \alpha$$

where $z_{\alpha/2}$ represents the number of standard deviations, i.e., $\sqrt{\hat{V}(\tilde{x}_k)}$, by which an observation $\tilde{x}_k$ differs from the mean $x_k$, when the confidence is $1 - \alpha$.

**StructInf-S2.** Another idea is to first sample diffusion edges uniformly to form a sampled action diffusion graph, and then enumerate the influence patterns based on the sampled graph. Specifically, in Algorithm 1, outside Lines 6-7, we add a judgment statement to determine whether $l_s \rightarrow l_t$ will be added into $G^p$ by a probability $q$. Outside Lines 12-19 and 16-17, we also determine whether $l_r \rightarrow l_t$ or $l_s \rightarrow l_t$ will be added into $G^p$ by $q$. We obtain the following lemma:

**Lemma 2** Given an influence pattern $C_k$, any of its instances, $c_i^k$, is selected uniformly according to probability $q^{m_i}$, where $m_i$ is the number of edges in $C_k$.

**Proposition 2** Let $x_k$ be the exact number and $\hat{x}_k$ be the approximate number obtained by StructInf-S2. For the complete graphs such as $C_3$ and $C_{20}$ in Table 1, $\hat{x}_k/q^{m_k}$ is an unbiased estimator of $x_k$, while for the incomplete graphs:

$$\hat{x}_k = \frac{x_k + \sum_{C_i \subseteq C_k, C_j \subseteq n_i, m_i \subseteq n_i} n_i \hat{x}_1}{q^{m_k}}$$

(2)

**Proof 2** When a pattern is a complete graph, the proof is the same as that of Proposition 1. When a pattern is an incomplete graph, $\hat{x}_k/q^{m_k}$ records the number of not only pattern $C_k$, but also the patterns that contain $C_k$ as their subgraph and with the same node size, i.e., $\{C_1 : C_1 \subseteq C_i \subseteq n_i\}$. We name $C_i$ as the parent pattern of $C_k$. In a sampled action diffusion graph, when enumerating an incomplete subgraph, it may be restored to any of its parent patterns. Thus to obtain $\hat{x}_k$, we need to first sum the approximate values of $C_k$ and all its parents, and then divide by the sampling probability, $q^{m_k}$, to get an unbiased estimation (the first part of Eq. (2)), and finally subtract the unbiased value of all the parent patterns (the second part of Eq. (2)). Notation $n_{i,k}$ is the times that $C_k$ is contained in $C_i$. For example, in Table 1, $C_6$ appears two times in $C_1$. The unbiased value, $\hat{x}_i$, of each parent pattern can be estimated in the same way iteratively, until the pattern itself is a complete graph.

**StructInf-S3.** StructInf-S3 combines the above two approaches by not only sampling diffusion edges when building the action diffusion graph, but also sampling the action nodes when enumerating pattern instances.

**Experiment**

The dataset and code are available online now.3

**Experimental Setup.** We perform the evaluation on a dataset collected from Sina Weibo. The network consists of 1,776,950 users as nodes and 308,489,739 "following" relationships as edges, with the maximal degree $d_{max}$ as 2,875. We use 23,755,810 tweet/retweet actions to build action diffusion graphs of 3,472,004 diffusion edges. The action type set $A$ contains 300,000 types (i.e., the original tweets). Please refer to (Zhang et al. 2013) for details.

In the Weibo dataset, we first use the sampling algorithms to estimate the structural influence for the patterns in Table 1. We vary sampling probabilities and compare the error and time trade-off curves. The results of StructInf-Basic are used as ground truth. Please note that no existing methods can be

3http://aminer.org/structinf
directly used to estimate structural influence. Then we apply the extracted influence patterns to help retweet behavior prediction, to demonstrate the effectiveness of influence structures.

**Experimental Results.** We use relative error,

\[ U_{p_k} = \frac{|\hat{IP}_k - IP_k|}{IP_k} \]

where \( IP_k \) is the exact/actual estimation and \( \hat{IP}_k \) is the approximate estimation, to measure how far the approximate estimation is from the exact estimation (Ahmed et al. 2014).

We present the optimal estimation of structural influence, \( IP_k \), with \( k \) from 1 to 20 in Table 1. The results are obtained by executing StructInf-S3 with \( \tau = 25 \) hours, \( q = 0.9 \), \( p_x = 0.6 \) and \( p_y = 0.1 \), where \( p_x \) and \( p_y \) are the node sampling probabilities for estimating \( x_k \) and \( y_k \) respectively, and \( q \) is for sampling edges, and is the same for estimating \( x_k \) and \( y_k \). Because the ratio between active and inactive instances is about 1:700, \( x_k \) can be estimated much faster than \( y_k \). Thus we can first try several parameters to find the optimal values of \( q \) and \( p_x \), and then estimate \( p_y \) approximately by \( p_x \times \sqrt{\frac{x_k}{y_k}} \), that is derived from the variance. We estimate each \( \bar{x}_k \) and \( \bar{y}_k \) by averaging the results of 10 independent runs, and based on which to calculate \( IP_k \). Table 1 shows that each \( IP_k \) is very close to the exact value. Most of the relative errors are around 1.0% and the worst case is about 5.0%. We also find that the top influential patterns are those with more nodes or edges than others ( the numbers that are in boldface). To get the results in Table 1, the exact method (StructInf-Basic) requires 19 hours, while the sampling method (StructInf-S3) requires only 1.8 hours.

**Trade-off between Error and Time.** To compare the performance of different sampling methods, we show how accuracy and speed vary by varying the sampling probabilities. Specifically, we first tune the sampling probabilities. We try \( p \) in the range \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\} for StructInf-S1, try \( q \) in the same range for StructInf-S2, and try all the possible combinations of \( p \) and \( q \) in the same range for StructInf-S3. Second, for each configuration of \( p \) and \( q \), we calculate the average error of \( \bar{x}_k \) over all the influence patterns, and further average them over 10 independent sampling results. Third, we use the number of iterations that Algorithm 2 is invoked as the metric to measure how fast an approximate estimation will be. The more iterations we run, the more influence patterns will be sampled.

Figure 3 plots all the (relative-error, #iterations) pairs when varying \( p \) and \( q \). For each method, the sampling parameters change from 0.1 to 1.0 incrementally from the upper-left to the lower-right points. Because StructInf-S3 has two tunable parameters, the points cannot be connected by one single line. From the results, we can see that first, the error curves of all the methods almost follow exponential distributions. The average error drops dramatically when \( p \) increases at the very beginning, and then changes slowly when \( p \) gets larger than 0.5. Second, StructInf-S1 performs better than StructInf-S2 because its curve is below that of StructInf-S2, implying that it needs less iterations to achieve the same error. Finally, the resultant points of StructInf-S3 are more concentrated in the bottom-left corner than those of StructInf-S1 and StructInf-S2, suggesting that StructInf-S3 is less sensitive to the parameters. This probably because StructInf-S3 combines the power of StructInf-S1 and StructInf-S2, thus achieves a more stable performance.

**Unbiasedness.** We take StructInf-S3 as an example to study the properties of the sampling distribution. Specifically, for each pair of \( q \) and \( p \), we run 20 independent samplings and plot the relative error of each run. From Figure 4, we can see that the sampling distributions of different...
patterns are all centered over the line with relative error as 0, which implies the unbiased property of the estimations. In addition, influence patterns with fewer nodes and edges can achieve better performance with few iteration times (i.e., samples). We see that the simplest pattern $C_1$ obtains 0.5% relative error with few samples, while $C_2$ needs more samples to achieve the same error rate, because the sampling probability $p(c_1) = p^m q^n$ decreases with $m$ and $n$, making the variance larger than that of the simple patterns.

Effect of Time Delay $\tau$. We study how time delay $\tau$ affects structural influence. Figure 5(a) shows the changes of structural influence by varying $\tau$. The Y-axis is the structural influence of $C_1$, which is the basic component of all the other patterns. We can see that social influence increases quickly over time and gradually becomes stable after $\tau$ gets larger than 25 hours, implying that social influence decays over time. Thus, we set $\tau$ to 25 hours on the Weibo dataset.

Application Improvements. We demonstrate how the discovered influence patterns can help improve the application of retweet prediction. The goal is, given an action triple $(v, a, t)$, predict whether user $v$ will retweet the tweet $a$ at time $t$. Basically, for each observed action in the dataset, we treat it as a positive instance. For each positive instance $(v, a, t)$, if a follower $u$ of user $v$ did not retweet $a$ before $t + \tau$, we treat $(u, a, t + \tau)$ as a negative instance. We uniformly sample a balanced dataset with equal number of positive and negative instances and train a binary classifier with logistic regression. We feed some basic features, such as the number of followers/followees, gender, verification status of the user, to the classifier. We aim at investigating whether the structural influence patterns can improve prediction performance based on the basic features. Additionally, we add structural influence patterns as features (the number of each pattern that is counted from the target action $(v, a, t)$). In particular, we divide the patterns into four groups: $C_1$ (i.e., the number of active neighbors), weak ($\hat{IP}_k < 0.1$), moderate ($0.1 \leq \hat{IP}_k < 0.3$), and strong ($\hat{IP}_k > 0.3$). We respectively add the basic features and pattern groups one by one and evaluate the increase of the predictive performance. A larger increase means a higher predictive power. From the results in Figure 5(b), we observe that weak patterns can improve a lot upon basic features and the number of active neighbors (+1.1% in terms of $F_1$), indicating the effect of structural influence on retweet behaviors. Furthermore, significant increase on $F_1$ score is observed when adding strong patterns upon moderate patterns (+1.73%), while no evident increase is observed when adding moderate patterns, implying that the discovered significant structural influence patterns can benefit a lot on predicting retweet behaviors.

Related Work

Structural Influence. A few research examined the structural characteristics of social influence. Ugander et al. (2012) firstly studied structural diversity and found that the possibility that a user joins Facebook is positively affected by the diverse structure of the friends who have joined Facebook. Lately quite a few work has been conducted based on this idea in various scenarios (Fang et al. 2014; Kloumann et al. 2015; Qiu et al. 2016; Zhang et al. 2013). However, all the studies about structural diversity do not distinguish the particular influence structures, and our paper proposes an algorithm to enumerate all influence structures.

Influence Learning. The aim of influence learning is to associate each edge $e_{uv}$ with a probability $P_{uv}$ to represent the strength of influence exerted by $u$ on $v$. (Kimura et al. 2011) were the first to learn influence by maximizing the likelihood of generating historical behavior data. Tang et al. (2009) and Liu et al. (2012) extended the influence probabilities to the topic level. Kutzkov et al. (Kutzkov et al. 2013) approximately estimated the pairwise influence in a behavior stream. In addition to learning pairwise influence, group influence is also studied by (Tang, Wu, and Sun 2013; Zhang et al. 2014). To the best of our knowledge, this is the first attempt to define and estimate structural influence.

Subgraph Mining/Sampling. Traditional research counted triangles (Jha, Seshadhri, and Pinar 2013; Pavan et al. 2013), 4-node subgraphs (Jha, Seshadhri, and Pinar 2015), any subgraphs (Ahmed et al. 2014; Wernicke 2006), or specified ones such as cliques, stars, chains, and cascading patterns (Koutra et al. 2015; Leskovec, Singh, and Kleinberg 2006). However, the methods cannot be directly applied to our problem, because the behavior data is streaming and the structural influence is beyond the frequency of a subgraph.

Conclusion

We present the concept of structural influence and propose a sampling algorithm to quickly estimate the influence probabilities of different structures from social stream, with theoretical proof of unbiasedness properties. Experiments on a large Weibo data show that, compared to the exact solution, the third sampling method can achieve the best performance, with a 10× speedup and an average error rate of only 1.0%. We also demonstrate the effectiveness of the extracted high influential patterns on retweet behavior prediction.

Acknowledgments. The work is supported by the National High-tech R&D Program (No. 2015AA124102, No.2014AA015204), NSSSF (13&ZD190, 12&ZD220), National Key R&D Program(No.2016YFB1000702), NSFC (61272137, 61202114, 61532021), Online Education Research Center, Ministry of Education (2016ZD102), and an Royal Society-Newton Advanced Fellowship Award. Jim Sun was supported by the National Science Foundation, award IIS- #1418511 and CCF-#1533768, Children’s Healthcare of Atlanta, CDC I-SMILE project, Google Faculty Award, AWS Research Award, Microsoft Azure Research Award and UCB. Jie Tang is the corresponding author.
References


Yan, X., and Han, J. 2002. gspan: Graph-based substructure pattern mining. In ICDM’02, 721–724.

