Querying and Tracking Influencers in Social Streams

Karthik Subbian
University of Minnesota,
Minneapolis, MN 55455.
karthik@cs.umn.edu

Charu C. Aggarwal
IBM TJ Watson Research Ctr,
Yorktown Heights, NY 10598.
charu@us.ibm.com

Jaideep Srivastava
University of Minnesota,
Minneapolis, MN 55455.
srivasta@cs.umn.edu

ABSTRACT

Influence analysis is an important problem in social network analysis due to its impact on viral marketing and targeted advertisements. Most of the existing influence analysis methods determine the influencers in a static network with an influence propagation model based on pre-defined edge propagation probabilities. However, none of these models can be queried to find influencers in both context and time-sensitive fashion from a streaming social data. In this paper, we propose an approach to maintain real-time influence scores of users in a social stream using a topic and time-sensitive approach, while the network and topic is constantly evolving over time. We show that our approach is efficient in terms of online maintenance and effective in terms various types of real-time context- and time-sensitive queries. We evaluate our results on both social and collaborative network data sets.

1. INTRODUCTION

The problem of finding influential actors is important in various domains such as viral marketing [7] and political campaigns. The problem was formally defined by Kempe et al. [15] as an optimization problem over all possible subsets of nodes with cardinality k. Subsequently, a significant amount of work [15, 14, 18, 10, 9, 13] has been done on this area. All these approaches are static in the sense that they work with a fixed model of network structure and edge probabilities. In practice, however, the influence of actors are defined by how their messages are propagated in the social network over time. Such propagation can only be observed from the underlying *social stream*, such as a Twitter feed or the sequence of Facebook activities. This problem setting is highly dynamic because the social stream evolves over time, as different actors initiate messages which are propagated in varying degrees over the network. The influence of an actor is best learned by observing the patterns of message propagations over time, as some actors are more successful than others in initiating new cascades of information flow in the network. The use of such information flows in identifying influencers has recently been recognized [24].

A major disadvantage of existing influence analysis methods is that they do not enable the ability to *query* the influencers in a *context-specific* fashion. Ideally, one would like to be able to use search terms to determine influencers that are specific to a given context. For example, the top influencers for the search term "*Egypt Unrest*" would be very different from that of the search term "*PGA Golf Tour*". The inflexibility of existing methods is, in part, because the existing methods [18, 10, 9, 15] decouple the problem of influence analysis from learning content-centric influence probabilities [11]. Clearly, one cannot learn a new set of relevant edge probabilities every time a query is initiated.

The influencers in a social stream are *time-sensitive* and may rapidly evolve [2], as different external events may lead to changes in the influence patterns over time. For instance, a query such as "*winter boots*" may generally have prominent entities associated with shoe stores as the most influential entities, but an (advertisement) statement from a popular figure in the entertainment industry, such as Justin Bieber, on a specific boot style[1] may change this ordering. Important events can often dynamically change the influencers, and this can only be tracked in a *time-sensitive* and *online fashion* from the underlying activities in the social stream.

We address these issues by considering a more flexible *real-time setting* in this paper, in which we enable the ability to *query* a social stream for various types of *context-sensitive* and *time-sensitive* influencers. Any influence query includes four components, corresponding to (a) influencer set, (b) influenced set, (c) query content such as keywords, and (d) time. The last of these components is often (implicitly) assumed to be the current time, although our framework is flexible enough to provide responses to historical queries as well. The influence score $\mathcal{I}(S_1, S_2, \mathcal{Q}, t)$ represents the aggregate influence score of actor set S_1 on actor set S_2 with respect to content $\mathcal{Q}$ at time t. Most of the queries are resolved by evaluating this score and then ranking it over various possibilities in the argument. One or more of the arguments in $\mathcal{I}(S_1, S_2, \mathcal{Q}, t)$ can be instantiated to a "*" (don't care) in order to enable more general queries in which all possibilities for the matching are considered. For instance, the queries $\mathcal{I}$("David", $*$, "Egypt unrest", t) and $\mathcal{I}$("John", $*$, "Egypt unrest", t) can be used to compare the total influence of David and John on the topic "*Egypt unrest*" at time t. Some examples of useful queries that can be resolved with this approach are as follows:

- For a given query context $\mathcal{Q}$, determining the top-k *influencers* at time t can be formulated as: $\max_{X:|X|=k} \mathcal{I}(X, *, \mathcal{Q}, t)$. It is also possible to constrain the query to consider a specific subset Y in the second argument, corresponding to the influenced actors. For example, a winter clothing merchant might decide to consider only those actors whose location profiles correspond to colder regions.

[1] http://money.msn.com/now/post.aspx?post=ea03f857-18c1-414f-97bc-7bc5d0b2ab06

WSDM'16, February 22–25, 2016, San Francisco, CA, USA.

DOI: http://dx.doi.org/10.1145/2835776.2835788

- Determining the top-k *influenced* actors at time t, for a given query context $\mathcal{Q}$, can be extremely useful in context-specific subscriptions and in recommending interesting public content to influenced users.

- Influence queries can also be useful in context-sensitive link recommendation, such as finding the top-k influencer-influenced pairs, for a given query context $\mathcal{Q}$.

Influence queries can provide significant business insights into the content, network, and temporal dynamics of the social stream. In addition, dynamic variants of existing problems can be formulated using this querying framework to leverage our approach. For example, the traditional influence maximization problem [15] is that of maximizing the influence of node set X (of size k) for the entire network, for any content, across all time points. In other words, the traditional problem can be formulated as $\max_{X:|X|=k} \mathcal{I}(X, *, *, *)$. An important difference from traditional influence analysis is that these queries are now resolved in a dynamic and data-driven way with the help of the social stream.

1.1 Contributions and Organization

In this paper, we propose an online approach in which we process the incoming social stream objects to create an *incrementally (lazy) updatable data structure of flow paths*. This data structure implicitly tracks all the influence information corresponding to the different content of all users in real-time. This goal is achieved by tracking information flow patterns in a tree-like data structure across various paths of the network in a context- and time-sensitive fashion. This data structure can then be queried in a flexible way to yield a variety of insights that are not normally possible with a traditional influence-based model. It is important to note that our framework provides the ability to *query influencers* for different contexts without any re-computation of influencers from scratch, unlike existing approaches [10, 9, 15].

Our approach is well-suited for applications where the hardware available for execution is limited. To handle such scenarios, we provide an *approximation technique* (based on the available hardware) and show that our approximation is close enough to actual value. We establish a tight lower bound on the accuracy for this purpose. Our approach also has a well-established relationship with *Katz similarity* measure. Finally, we evaluate our approach using social and collaboration data sets and show several interesting case studies and precision-recall experiments for different types of queries. To the best of our knowledge, this is the first *influence querying and tracking* model for *social streams*.

This paper is organized as follows. The remainder of this section discusses the related work. In the next section, we formalize the problem of online influence analysis for social streams. In section 3, we will provide an efficient algorithm for querying and tracking influencers from social streams. Section 4 contains the experimental results. The conclusions and summary are contained in section 5.

1.2 Related Work

The problem of tracking influencers in a network is often treated as an influence maximization problem [15, 14, 18, 10, 9]. The problem of influence maximization is that of finding the top-k nodes such that the average infection spread is maximized under a specific influence propagation model. There are two popular choices for the influence propagation model, which are referred to as the Independent Cascade (IC) and Linear Threshold (LT) [15] models, respectively. In these models, it is assumed that the edge propagation probabilities for the influence propagation model are already provided. Therefore, it is difficult to query these models for a specific context (or content), unless the edge propagation probabilities are learned as a separate problem in its own right [11]. In the dynamic social stream setting, such a model is too inflexible to *simultaneously track the influencers in different contexts, while also adjusting for the evolution in the flow patterns and influencers over time*.

Information flow mining has been recognized as a useful tool for the problem of influence analysis. A number of basic models for influence analysis in social networks are discussed in [15]. Related work on information flow mining and cascades may be found in [16, 18, 19, 22]. The problem of information flow mining and influence analysis has been related in [12, 24]. There are game theoretic [26] and supervised learning models [25] applied for finding influencers. However, these techniques are applicable only for static graphs and not for streaming social data. There are a few topic-specific influence analysis models [5, 8, 21, 27], but they are either not online, or they cannot be queried in a context- and time-sensitive fashion.

There is a surge of recent literature on social streams [3]. As the content of many social networks, such as *Twitter*, are often available only in the form of social streams of user activity. Social streams have been explored in the context of event detection [1, 23], topic-based browsing [6], classification [29], and influence analysis [24]. There have been some attempts to compute influencers in an online setting [17, 30], but these approaches are not context-specific and cannot be queried. There are other interesting works on topic-specific influence maximization [4]. However, our focus is more on a general keyword-based influence querying scheme for streaming scenarios. As we will see later, our approach is related to a flow-based version of the Katz measure [20], a popular centrality measure.

2. REAL-TIME FLOW-BASED INFLUENCE ANALYSIS

In this section, we discuss the necessary notations and the problem definition to introduce the online influence analysis model for social streams. We assume that we have a social stream $\mathcal{S}(t)$ at time t. It should be pointed out that the social stream $\mathcal{S}(t)$ can be used to construct a temporal network $G(t) = (V(t), E(t))$, which is based on all the edges received so far till time t. It is assumed that all edges in the network are directed, which corresponds to the direction of the information flow along an edge in $E(t)$. Each actor $a_i \in V$ performs a number of *content-based actions* such as sending tweets, exchanging messages, or placing wall posts, etc. Because this process occurs simultaneously over the entire set of actors in the social network, the stream of content created by the different actors can be *globally* treated as a *social stream* of text content. We denote the set of all distinct keywords that appear in the social stream by $\mathcal{K} = \{K_1, \ldots, K_m\}$. Each content-token K_i in the stream is associated with the following meta-data:

- The content-token $K_i \in \mathcal{K}$ represents a component of the content created by an actor in the social stream. This could correspond to the actual string of the tweet/post, or only specific parts of the post, such as URL strings, keywords, or hash tags. In the event that the cascade behavior is being observed broadly in the form of topics, rather than re-posts, the string K_i could also simply be one of a set of a topical dictionary of keywords (contained in the tweet). An interesting feature of our approach is that it can be easily applied to an arbitrary subset of content-tokens or even latent topics to facilitate topic-specific influence in the network (Section

3.3). However, for the purpose of this paper we treat $\mathcal{K}$ as the dictionary of content-tokens and each content token (K_i) denotes a unique hashtag or a keyword. We feel such keyword-based treatment allows for better granularity of analysis.

- It is assumed that the content K_i is created by some actor a_j. The value j is an index drawn from the actor set $V(t)$, and it represents the actor who is responsible for the origination of that particular piece of content.
- It is assumed that the content K_i is received by actor $a_l \in V(t)$, and it represents the recipient of that content. This recipient could be an email recipient in an email network, a follower of the tweet sender in *Twitter*, or any other form of communication in a social network.

The continuous flow of such user content through a network is referred to as social stream ($\mathcal{S}$) [24], and it is often the *only observable aspect of the dynamics of a social network*. In many social networks, such as *Twitter*, this form of social stream data is available using a subscription-based access.

2.1 The Flow-based Query Model

We now set up the model for the influence querying problem, and show its relationship to many of the existing methods in the literature. More specifically, we will discuss how the flow paths are used in the context of influencer querying. The intuition behind the flow-based approach is that influencers that are important to a specific content are often the originators of messages or tweets related to that content. The same observation is true about the pairwise influence between nodes. When a particular actor a_i is influential on actor a_j with respect to a particular content set (e.g., keywords), it will lead to a significant amount of flow of the corresponding content set form a_i to a_j and then to a_j's neighbors, through multiple flow paths. Thus, by tracking the flow paths for different keywords, one can perform effective content-centric analysis.

For the purpose of this paper, we assume that a dictionary of keyword $\mathcal{K}$ is available, and all content flow weights are computed in terms of the transmission of messages containing these keywords. The use of keywords, as opposed to segments of text, actually allows for better granularity of analysis, since the content flows can be computed in terms of these keywords, and then aggregated over different flow paths for a particular topic.

First, we define the concept of a valid flow between a pair of nodes a_j and a_k, with respect to a specific content token K_i. Intuitively, this concept represents the transmitting of a keyword K_i along a specific path $a_j = b_1 \ldots b_r = a_k$ leading from a_j to a_k, such that each transmission between successive nodes occurs in that order. We formally define this concept as a *valid flow*:

DEFINITION 1 (VALID FLOW). *A valid flow $\mathcal{F}$ of keyword K_i from a_j to a_k is an ordered set of nodes $a_j = b_1 \ldots b_s = a_k \in N$, such that the following conditions are satisfied:*

1. *An edge exists between nodes b_r and b_{r+1} in the base network, for all $r = 1 \ldots s-1$.*
2. *The node b_{r+1} transmits a message containing the keyword K_i, after a message is transmitted from node b_r containing the same keyword.*

Thus, a flow from node a_j to node a_k, of the keyword K_i is indicative of the influence that node a_j has over node a_k. Note that the flow of a message occurs over a particular duration that corresponds to the elapsed time since the first transmission of message keyword from node a_j. We refer to this as the flow duration:

DEFINITION 2 (FLOW DURATION). *Consider a valid flow $\mathcal{F}$ of keyword K_i from a_j to a_k. Let t_c be the current time, and t_o be the original time at which the keyword was transmitted from node a_j. Then, the flow duration from node a_j to node a_k is equal to $\delta t = t_c - t_o$.*

The notion of flow duration refers to the *latency* of influence response. When the flow duration (δt) is large, the significance of the content flow decays over time in terms of its influence. This is important, especially if the influence scores are to be made temporally sensitive for more effective analysis. Therefore, we assume a decay factor of λ, and assume exponential decay in order to define the *flow weight* of a particular path.

DEFINITION 3 (DECAYED FLOW WEIGHT). *Let δt be the flow duration of a particular flow $\mathcal{F}$ at a current time t_c, and λ be the decay factor. Then, the decayed flow weight is given by $2^{-(\lambda \delta t)}$.*

The flow $\mathcal{F}$ has a half-life of $1/\lambda$, since the weight of a flow reduces by a factor of 2 every $1/\lambda$ time units. Note that the rate at which the information decays is a constant for all users in our model (i.e. λ). More specifically, the rate of decay before and after the message is transmitted by (the last actor) a_k in flow $\mathcal{F}$ is λ. Thus, the decayed flow weight expression, in Definition 3, is independent of the time of transmission of the message by actor a_k. The decayed flow weight can be further used to define the aggregate path flow.

DEFINITION 4 (AGGREGATE PATH FLOW). *The aggregate path flow $\mathcal{A}(\mathcal{P}, K_i, t_c)$ at current time t_c, for keyword K_i along a particular path $\mathcal{P} = \langle a_j, \ldots, a_k \rangle$, is the sum of the flow weights on that path for the keyword K_i over all different valid propagation of the keyword from node j to node k along path $\mathcal{P}$.*

Note that the aggregate path flow is caused by repeated flow of a keyword K_i along a particular path. For two flows to be considered distinct, the keywords would need to have been transmitted at a distinct time at the source or destination node in the path, and also satisfy all the valid flow constraints.

This path flow can be aggregated across a set of paths that are specific to a particular source and destination node, to define a pairwise value, as opposed to a path-wise value.

DEFINITION 5 (AGGREGATE PAIRWISE FLOW). *The aggregate pairwise flow $\mathcal{V}(a_j, a_k, K_i, t_c)$ at time t_c for keyword K_i from node a_j to node a_k is equal to sum of the value of the flows on the paths from node a_j to node a_k. Therefore, is $\mathcal{S}_{jk}$ be the set of paths from node j to node k, we have:*

$$\mathcal{V}(a_j, a_k, K_i, t_c) = \sum_{\mathcal{P} \in \mathcal{S}_{jk}} \mathcal{A}(\mathcal{P}, K_i, t_c) \quad (1)$$

We now formally introduce the atomic influence function $\mathcal{I}(a_j, a_k, \mathcal{Q}, t)$, which is the influence exerted by node a_j on a_k, for the context $\mathcal{Q}$ (query keywords), at time t. This function can be computed as a sum of the aggregate pairwise flows over all the query keywords $K_i \in Q$. It is important to note that this influence function is asymmetric. In other words, the influence exerted from a_j to a_k can be very different from the influence in the other direction.

DEFINITION 6 (ATOMIC INFLUENCE FUNCTION). *The atomic influence value $\mathcal{I}(a_j, a_k, \mathcal{Q}, t)$ for a node a_j to influence a_k, is defined as the sum of the aggregate pairwise flows over all keywords $K_i \in \mathcal{Q}$ in the data:*

$$\mathcal{I}(a_j, a_k, \mathcal{Q}, t) = \sum_{K_i \in \mathcal{Q}} \mathcal{V}(a_j, a_k, K_i, t) \quad (2)$$

So far, we have explained how the individual content tokens arriving in the social stream $\mathcal{T}$ can be used to model an influence function $\mathcal{I}$, in content, network and time sensitive fashion. It is possible to convert the keywords to other representations in scenarios where it leads to enhanced performance. For example, the content tokens tracked may be URLs or high level topics, while the incoming query may be still a collection of textual tokens. In such cases, a probabilistic latent factor model can be used to compute the association of tracked content tokens to given query tokens. We will revisit this discussion in detail, after we explain a simplified version of the algorithm in the next section.

2.2 General Flow-based Query Processing

The aforementioned computation of the atomic influence function for calculating influence scores in pairwise fashion can be generalized to a wider range of queries easily. An important observation is that the "don't care" conditions, denoted by '*', can be treated as implicit summations. For instance, we have $\mathcal{I}(*, a_k, \mathcal{Q}, t) = \sum_{a_j \in V} \mathcal{I}(a_j, a_k, \mathcal{Q}, t) = \mathcal{I}(V, a_k, \mathcal{Q}, t)$. In this section, we denote the influencer and the influenced as a_j and a_k, respectively. The three main applications of the atomic influence function are as follows:

Influencer Search Queries $(\alpha\,(a_j, \mathcal{Q}, t))$: A typical form of querying is similar to using a search engine. Given a contextual query, $\mathcal{Q} = \{w_1, w_2, ..., w_k\}$, a relevant set of influencers are computed for $\mathcal{Q}$ at the current time t, using the atomic influence function, which is defined as $\alpha(a_j, \mathcal{Q}, t) = \mathcal{I}(a_j, *, \mathcal{Q}, t)/\mathcal{I}(*, *, \mathcal{Q}, t)$. We normalize the scores across all users for the query, before comparing them with each other. We find the top-k (a_j) nodes that have the highest influence scores (using the atomic query $\alpha(a_j, \mathcal{Q}, t)$) and arrange them in the descending order. A user can also query the influencers in a similar way for an earlier time $t' < t$.

Link Prediction Queries $(\beta\,(a_j, a_k, t))$: Social network users tend to connect to the users who influence them highly. Hence, one can use this query to find the top-k users who have the greatest influence on user a_k at current time t. This query is very similar to the link prediction problem where directional (e.g., follower) links are predicted on the basis of context-sensitive influence. Here, the atomic influence function can be defined as $\beta(a_j, a_k, t) = \mathcal{I}(a_j, a_k, *, t)/\mathcal{I}(*, a_k, *, t)$ to find the influence of nodes a_j on a_k across all $K_i \in \mathcal{K}$.

Concept-specific temporal queries $(\gamma\,(Q, [t_1, t_2]))$: These queries can be used to find the evolution of influence of a particular concept or phrase across a time horizon $[t1, t2]$ [2]. Here, a concept or phrase (Q) is given, and its influence over a time window is measured and compared. The atomic influence function is computed as $\gamma(Q, [t_1, t_2]) = \mathcal{I}(*, *, Q, t)/\mathcal{I}(*, *, Q, *), t \in [t1, t2]$.

We will discuss these three applications in detail in our experiments section.

2.3 Relationship with Katz Measure

The aforementioned atomic influence function is closely related to the Katz measure [20]. The Katz measure is defined in terms of the weighted sum of the number of all possible paths between a pair of nodes and the weight decays exponentially with the length of the underlying path. Specifically, if $\mathcal{P}_{ij}$ be the set of paths between nodes a_i and a_j, then the Katz measure $K(a_i, a_j)$ is defined as follows:

$$K(a_i, a_j) = \sum_{P \in \mathcal{P}_{ij}} \beta^{|P|} \tag{3}$$

Here β is the discount factor on the path length, which in our case is analogous to the flow-based temporal decay factor. This is also analogous to the original definition of the Katz measure, in which longer paths are discounted to a greater degree. Thus, our flow-based approach computes exponentially decayed flow weights across different paths, as a more dynamic, time- and content-sensitive way of measuring the importance of nodes. In an indirect sense, this way of computing node importance can be considered a flow-based analogue to the Katz measure in static networks. Because the Katz measure has been shown to be effective for link recommendation in static networks, it lends greater credence to its use in the flow-based streaming scenario. Of course, the Katz measure is used rarely in static networks because of the complexity of enumerating over a large number of possible paths. The important point to understand is that the flow-based measure *significantly* changes in the importance of different paths in the update process, and can also be more efficiently computed in the streaming scenario, because of the availability of a dynamic update process (see Section 3.2). Most paths in the network are used rarely by the different flows, and therefore the flow-based measure provides a better weighting to the different paths in terms of *user actions*. This skew in the distribution of user behavior across different paths also allows for a more efficient computation of the atomic influence function, as compared to the static case, because of the number of paths with significant flow is much smaller. We will use this intuition later to provide an efficient approximation to our algorithm in Section 3.1.

2.4 Challenges and Intuition

The problem of flow-based online influence analysis is extremely challenging because of its computational and real-time constraints. One major challenge is that the number of possible paths between a pair of nodes are exponential and so are the number of flows. Furthermore, real-time tracking becomes extremely difficult, when the number of paths involved is extremely large.

A salient observation is that while the number of possible flows is large, only a small number of them may be significant enough to be maintained. This fact will be exploited in order to create a pruned approximation of the flow paths. This pruned version can keep approximate track of the different paths in an efficient and dynamic way. We provide an analysis that shows our approximation significantly reduces the complexity of the maintaining all flow paths, while retaining the effectiveness of the querying process.

3. INFLUENCER TRACKING ALGORITHM

In this section, we will present the basic flow-based influencer tracking algorithm. As discussed earlier, a key part of this approach is the aggregation of flow paths together with the relevant weights and updating them *in real-time* as the new social stream object arrives. Therefore, the main focus of this approach is to find an efficient way to keep track of the flow paths. For ease in discussion, we will first describe the algorithm with two simplifying assumptions. The first assumption is that the keyword set $\mathcal{K}$ contains a single keyword K. We dropped the subscript i in K_i, since we are dealing with a single keyword assumption. The second simplifying assumption is to set the decay factor $\lambda = 0$ (no decay), so that path weights essentially correspond to the counts of all the messages received so far. Later, we will describe how to modify the algorithm to the general case. This two-stage exposition eases the understanding of the algorithm. Before discussing the algorithm in

[2]Here the time interval is discrete, with a specific time granularity days, hours, minutes, etc.

detail, we will first introduce the concept of the *flow-path tree* that is an essential data structure for keeping track of the flow paths. Our first description will use a very simple version of the path tree, without any latent factor model (such as topic model), no decay and a single keyword. Later, we will systematically discuss the changes required for each of the increasing levels of complexity.

The flow-path tree $\mathcal{T}$ is a compressed representation of all the flow paths that have been encountered so far. This tree is used in order to keep track of all the information flows in the stream so far. For a set of paths $\mathcal{P} = P_1 \ldots P_n$ with flow weights $w_1 \ldots w_n$, along which a flow of the keyword $\mathcal{K}$ has occurred, the flow-path tree is defined as follows:

DEFINITION 7 (FLOW-PATH TREE). *A flow-path tree $\mathcal{T}$ for a set of paths $\mathcal{P} = \{P_1 \ldots P_n\}$ is defined as follows:*

1. *The root of the tree $\mathcal{T}$ is the null node.*
2. *Each path from the root to a leaf* must *correspond to exactly one path $P_i \in \mathcal{P}$, and a path from the root to an internal node* may *correspond to a path in $\mathcal{P}$.*
3. *The weight associated with a node (other than null node) is equal to the flow weight corresponding to the path from root to that node.*

The flow-path tree is different from frequent pattern tree (fp-tree) in several ways. The most important difference is that fp-tree is a representation for *item sets*, where the order does not matter. While in the flow-path tree the ordering of node matters the most. Hence, in our problem, the strategy of using the most frequent node in the root will not help. Moreover, while constructing the fp-tree each transaction in the database is used to update the counts in the tree. While in our scenario, there is no notion of a transaction and multiple paths that may have influenced a node is extracted from graph $G(t)$ and then updated in the flow-path tree. All nodes that propagated a content w will have their own root node (for content w) in flow-path tree, while in fp-tree transactions with same item w will not necessarily be a root node under *null*.

The overall framework for the influencer tracking algorithm (for a single keyword) is illustrated in Fig. 1. This particular description shows only how the flow paths are updated for the arrival of a message containing a single keyword K. In practice, however, all keywords in a given message will need to be extracted, and multiple updates will need to be performed to the flow tree. Furthermore, this procedure will need to be performed every time a message from a user originates at a given node a_i. The input to the algorithm is the network $G(t)$, the originating node a_i, and the current status of the path tree $\mathcal{T}$.

The basic idea in the algorithm is to trace back all the paths in the network $G(t)$ starting at node a_i, at which the keyword K has appeared in the past. For example, if node a_j has an outgoing edge to node a_i, and has propagated the keyword K before a_i, then the path $\langle a_j, a_i \rangle$ is added to the tree $\mathcal{T}$ if non-existent, and the count is increased by 1. Then, the path $\langle a_j, a_i \rangle$ is added to the candidate list $\mathcal{C}$ for further backtracking. Note that all paths in $\mathcal{C}$ will always end with the node a_i, but not all these paths may be relevant because the keyword K may not have had a flow along the immediate prefix (obtained by removing node a_i) of some of these paths. This relevance can be checked by examining whether the immediate prefix of this path occurs in $\mathcal{T}$. More specifically, for any path $P \in \mathcal{C}$ all nodes in the network $G(t)$, which have incoming edges into the first node in the path P are examined, along with the temporal ordering. For any such node a_k, a new path $a_k \oplus P$ is created by appending a_k to the beginning of the path P. Then, it is checked whether

Algorithm *UpdateFlowPaths*(Originating Node: a_i,
Network: $G(t)$, Social Stream: $\mathcal{S}$, Flow-Path Tree: $\mathcal{T}$)
begin
Receive the next message containing keyword K in
in social stream $\mathcal{S}$ originating at node a_i;
Create singleton node a_i in tree $\mathcal{T}$ as child of root
node if it does not already exist;
$\mathcal{C} = \{a_i\}$; { Candidate paths for expansion }
Update weight of singleton path containing only
node a_i in tree $\mathcal{T}$ by 1;
while $\mathcal{C}$ is not empty
begin
Delete the first path P from $\mathcal{C}$ and
denote the first node of P by a_j;
for each $a_k \notin P$ in $V(t)$ with an incoming edge to a_j
if prefix of path $a_k \oplus P$ exists in $\mathcal{T}$ and a_k has
propagated keyword K prior to a_j
if the complete path $a_k \oplus P$ exists in $\mathcal{T}$
Increment weight of last node of path $a_k \oplus P$
by 1 in $\mathcal{T}$;
else
Create last node of P as child for prefix
of path $a_k \oplus P$ in $\mathcal{T}$ with weight as 1;
endif
Add $a_k \oplus P$ to $\mathcal{C}$;
endif
endfor
endwhile
end

Figure 1: Updating the Flow Paths for Single Keyword Simplification

the *prefix of* $a_k \oplus P$ [3] occurs in $\mathcal{T}$. If the prefix does exist, then a new child node of the prefix can be created in $\mathcal{T}$, corresponding to $a_k \oplus P$, if such a child node does not already exist. The count of this child node is incremented by 1. The path $a_k \oplus P$ is added to the candidate set $\mathcal{C}$ for further exploration. Once a path has been explored in $\mathcal{C}$, it is deleted from $\mathcal{C}$. The termination criterion for the algorithm is the case when no more paths in $\mathcal{C}$ remain to be explored. It should be pointed out that while the number of possible paths in the network is exponential, the number of paths relating to a specific keyword is usually much smaller. This number is even smaller in the decay-based scenario discussed later, where very few paths are relevant to a particular keyword at a specific time. Nevertheless, the challenge arises that the tree $\mathcal{T}$ can grow rather large in many scenarios, because the number of children of a node in $\mathcal{T}$ can be as large as the in-degree of the corresponding node in $G(t)$. Since the degrees of nodes may sometimes be large, this can lead to an explosion in the number of nodes in the $\mathcal{T}$. In the next section, we will show how to improve the efficiency of the representation.

3.1 Speeding up by Pruning

Since the flows are skewed across the different paths in the tree, it is possible to use pruning to reduce the tree size and improve the efficiency of representation. Specifically, the tree size can be reduced by pruning out the low frequency leaves in the tree. The overall approach for building the flow-path tree uses an additional pruning phase, in which the low frequency leaves of the tree are removed.

In order to achieve this goal, the number of nodes in the tree always allowed to vary between n and N, where $n = \alpha \cdot N$ for $\alpha \in [0, 1]$. Smaller values of α improves the space complexity, while

[3] Prefix of $a_k \oplus P$ is the path obtained after removing the last node of P.

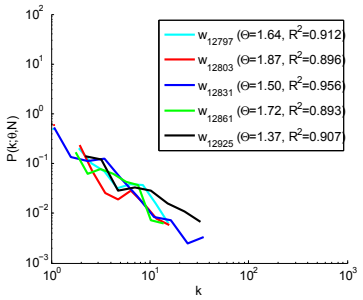


Figure 2: The best-fit estimate of the Zipf parameter θ and corresponding R^2 for few flow-path trees are shown in the legend.

larger values provide better accuracy of estimation. The pruning approach is triggered whenever the total number of leaves in the tree reaches N. The pruning approach sequentially removes the leaves from the tree until the total number of nodes in the tree is equal to $\alpha \cdot N$. A question arises here as to how much of the flow weight is lost by using the pruning approach, since the reduction in the flow weight reduces the accuracy of estimation. We will show that for modestly large values of N, the amount of flow weight lost is very small, in scenarios where the flow weights are skewed. Furthermore, because only the low frequency paths are lost, and the influence scores are mostly relevant to the high frequency paths, the impact of pruning is even lower.

The relative flow weights in a flow-path tree can be approximately modeled to be proportional to the Zipf distribution $1/i^{\theta}$ for $\theta > 1$. We confirm this by verifying the best fit estimates of the distribution of flow weights for several flow-path trees in our experiments on a real-life (Twitter) data set, described in detail in Section 4. In Fig. 3, we show the flow weight distribution for a few trees in rank versus frequency plots. We see that average R^2 is 0.926 across all flow path trees and the fit was statistically significant at 5% significance level. We now show that the reduction in flow weight, drawn from a Zipf distribution, is typically less than $\log(\alpha \cdot N)/\log(N)$.

THEOREM 1. *Let the flow weights on the N nodes in the tree be distributed according to the Zipf distribution $1/i^{\theta}$ for $\theta \geq 1$. Then, the total fraction $F(N, \alpha)$ of the flow in the top $n = \alpha \cdot N$ nodes of the tree is at least equal to:*

$$F(N, \alpha) \geq log(n)/log(N) = 1 - log(1/\alpha)/log(N) \quad (4)$$

Proof: The cumulative flow weight $C(n)$ for the top n flows is given by:

$$C(n) = \sum_{i=1}^{n} 1/i^{\theta} \quad (5)$$

Here, we are trying to determine the value of $C(\alpha N)/C(N)$. The function $C(\alpha N)/C(N)$ is an increasing function of θ for $\alpha < 1$, and $\theta \geq 1$. This is because the derivative of $C(\alpha N)/C(N)$ with respect to θ can be shown to be positive for $\theta \geq 1$. Therefore, the minimum value of the function is achieved at $\theta = 1$. Therefore, we have:

$$F(N, \alpha) \geq \frac{\sum_{i=1}^{n} 1/i}{\sum_{i=1}^{N} 1/i} \quad (6)$$

Note that the value of $\sum_{i=1}^{n} 1/i$ can be asymptotically approximated by $\log(n)$ for large values of n. The result follows. ■

The skew helps significantly in retaining the vast majority of the heavy flows. For example, in a flow tree with $100,000$ paths, discarding half the least frequent paths would result in total flow weight reduction of only $1 - \log(50000)/\log(100000) = 0.06$ of the flow weight. Thus, the vast majority of the heavy flows are retained, which are the ones most relevant for the influence analysis process anyway. This suggests that the pruning process can be used to significantly reduce the complexity of the flow-path tree, while retaining the effectiveness of analysis. This is essentially because of the skew in the flows on different paths in the tree.

3.2 Incorporating Decay

A key issue is the incorporation of decay in the computation of flow weights. At first sight, this looks challenging because it would seem that it is required to continuously update the flow weights at each time stamp because of the decay. However, in practice, it is not necessary to update the flow weights for each time-stamp. Rather, it is sufficient to perform the updates in a *lazy fashion*, as described below. This is because of the *multiplicative property* of decay computations.

LEMMA 1 (MULTIPLICATIVE PROPERTY). *Let $\mathcal{A}(\mathcal{P}, K_i, t)$ be the aggregate flow value at the time t for path $\mathcal{P}$ with destination node a_k. Then, if no new message has been transmitted at destination node a_k in the time interval (t_1, t_2), then the flow values at times t_1 and t_2 are related by the following multiplicative rule:*

$$\mathcal{A}(\mathcal{P}, K_i, t_2) = \mathcal{A}(\mathcal{P}, K_i, t_1) \cdot 2^{-\lambda(t_2 - t_1)} \quad (7)$$

The key pre-condition in the aforementioned rule is that no message should have been transmitted at the destination node a_k of the path. This suggests a lazy approach for performing the multiplicative update, in which the decay-based multiplicative update is performed only whenever a message is transmitted at a node, corresponding to which the flow is added. Therefore, for each path, in the flow tree, each node contains the time-stamp information about the last time that the flow value of the corresponding path was updated. This is referred to as the *last path update time-stamp*. Whenever a path in the flow tree is updated, the first step is to apply the multiplicative decay factor to that node, and then add the newly arriving flow value. The newly arriving flow value (which needs to be added to the current flow value of the path in the tree) is computed according to Definition 3, rather than the simple addition of one unit. The *last path update time stamp* of that node is updated to the current time.

The incorporation of decay into the flow analysis process has the additional advantage of reducing the number of updates. Recall that the algorithm in Fig. 1 performs a backtrack starting at the destination node at which the keyword currently appears. When a particular path being examined has a very low aggregate flow value, then it can be pruned from consideration. In other words, the path does not need to be extended backwards in Fig. 1, when its flow value is below a given threshold. Such paths are therefore not added to the candidate list $\mathcal{C}$. Specifically, a threshold ϵ on the flow path value is imposed for pruning purposes.

3.3 Generalizing beyond Keywords

So far, we have constructed the flow-path tree with the use of a single content token. The generalization to full keyword sets can be easily performed by adding an additional level to the flow-path tree. Specifically, this additional top most level corresponds to the choice of keyword being tracked. The first step in this process is to map each keyword to one of these branches. Subsequently, the flow paths in that branch are updated using the approach discussed earlier.

It is sometimes helpful to track the influence of users on latent topics. While our algorithm is designed to create a flow-path tree for each keyword because of its natural connection to query-processing, it can be very well used for latent topics. To compute the influence scores on latent topics, one can use the posterior of each topic given the query keyword, and compute the atomic influence function for the generalized list of latent topics ($\mathcal{Z}$) as shown in (8).

$$\mathcal{I}(j, k, \mathcal{Q}, t_c) = \sum_{z \in \mathcal{Z}} \sum_{K_i \in \mathcal{Q}} p(z|K_i, t_c)\mathcal{V}(j, k, z, t_c) \qquad (8)$$

This way one can query the topical influencers using latent topics, rather than explicitly using the content tokens in the post. Thus, the proposed approach is general enough to handle individual content tokens, n-gram phrases, and latent topics. We refer to our algorithm as **QUIS**, which is an acronym for **QU**erying **I**nfluencers in social **S**treams.

4. EXPERIMENTAL RESULTS

In this section, we will first evaluate the quality of influencers produced by the real-time flow-based query model compared to several baselines. Then, we will provide sensitivity and efficiency analysis of our querying framework. We also discuss several user- and concept-specific temporal queries to demonstrate the importance of influence query models in various scenarios.

4.1 Data sets

We evaluate the effectiveness and efficiency in two different data sets, belonging to the social network and research collaboration domains. For the social data set, we used Twitter, and for the collaboration data set we used DBLP.

Twitter Data Set: We used tweets over a period of 37 days from March 25, 2014 to May 1, 2014 (on the 10% feed of *Twitter*) and filtered the stream to contain tweets with one of the following set of hash tags related to the blocked news in Turkey for *Twitter* after the corruption allegations: "#turkey, #twitterblockedinturkey, #direntwitter, #twitterisblockedinturkey, #25martkorkusu, #twittericinsokagacikiyoruz, #youtube blockedinturkey, #erdogan, #1mayis". The resulting data set contained 1,919,294 ($\approx$ 1.9 million) tweets. Each extracted post contained the time stamp, user identifier and its content. We extracted all the unique hash-tags from these postings and treated them as the propagated keywords. This entire stream contained 131,574 unique hash-tags and 293,363 unique users. We constructed the relationship network of these users using the *mentions* between them for the past 10 months until 1^{st} of May, 2014. The resulting network contained 293,363 users and 799,605 edges.

DBLP Data Set: The DBLP data set is publicly available and can be downloaded from arnetminer.org [4] [28]. This data set contained a stream of documents (2,084,055 papers), which were temporally ordered based on the year of publication (until 2011). The relationship network G of authors is constructed using their co-authorship relationship. We excluded all papers that did not have any authors, abstract or year of publication in our stream. We considered the abstract of each paper as the message posted by the authors of the paper. Our keyword set ($\mathcal{K}$) was constructed using uni-, bi- and tri-grams of the words appearing in these abstracts. In order to track the interesting keywords, we used the ratio of fraction of keyword occurrences in a specific topic to the fraction of keyword occurrences in the entire corpus. We extracted 24 topics from the field of computer science in the *Microsoft Academic search* Website and constructed the list of top 10 conferences for each topic. If a paper is published in the selected sample of venues for this topic, then the words in the abstract are counted for that topic. By using this approach, we extracted the top-1000 interesting keywords (uni- to tri-grams) for all the 24 topics and created a model based on them. It resulted in 12,974 interesting words across the entire stream. Our final cleaned data stream contained, 330,000 documents with total of 337,081 distinct authors. We set $N = 10^3$ and $\alpha = 0.5$, for our approach as default values, unless specified otherwise.

[4]http://arnetminer.org/citation

4.2 Baselines

We used some of the most popular baselines used in several other recent influence analysis works [15, 10, 9]. The first baseline is the *PMIA* algorithm discussed in [9]. It is the prefix excluded extension of Maximum Influence Arborescence (PMIA) model. This algorithm takes a network structure with predefined edge probabilities as input. We have used the weighted cascade model proposed in [15] to compute the edge probabilities. Our next baseline algorithm is *DegreeDiscountIC*, which is the degree discount heuristic of [10] developed for the uniform IC model. For the PageRank baseline, we used the power-iteration method to compute the page rank values and the restart probability was set to 0.15. The stopping criteria was set 10^{-7}, which is the difference in the L_1 norm between two successive iterations. We also used the degree and weighted-degree as additional baselines.

4.3 Evaluation Measures

We evaluated the effectiveness of our approach in terms of the quality of the influencers returned by different queries. We compared the top-k influencers returned by our approach against the ground truth of influencers for each query. We used the precision and recall evaluation measures, shown in Equations (9) and (10), respectively, to compare against the baselines. The list of top-k influencers returned by each approach is compared with the list of ground-truth influencers. The precision measures the fraction of users retrieved that are relevant, while recall measures the fraction of relevant users that are successfully retrieved.

$$precision = \frac{|\texttt{relevant} \cap \texttt{retrieved}|}{|\texttt{retrieved}|}, \qquad (9)$$

$$recall = \frac{|\texttt{relevant} \cap \texttt{retrieved}|}{|\texttt{relevant}|}. \qquad (10)$$

The ground-truth used for the *DBLP* data set was based on the *citation* counts of authors for each topical area, and the venue was used to decide the topic for each paper. Because the influence analysis needed to be performed in a time-sensitive way, the ground-truth was also computed at various time-points. The ground truth for Twitter data set contained the *retweet* counts for each user until a specific time point (based on the time parameter in the query). There were no specific topics in the *Twitter* data set, because the data set was already focused on the Turkey event. The re-tweet counts were counted only for the tweets in our stream.

4.4 Effectiveness Results

We evaluated the quality of the influencers discovered by each approach in comparison with the ground truth. For our approach, we generated the list of top-1000 influencers using the influencer search queries as described in Section 2.2. The precision and recall values were computed by sweeping through the top-1000 influencers (from 1 to 1000) for each method. For Twitter, our query was a set of hashtags related to the Turkey incident: "#turkey, #twit-

Method	DM	NW	ML	CV	HW	SE
QUIS	**0.096**	**0.092**	**0.042**	**0.084**	**0.146**	**0.028**
PMIA	0.022	0.01	0.014	0.055	0.061	0.014
DDIC	0.02	0.012	0.017	0.037	0.089	0.014
PR	0.035	0.018	0.029	0.067	0.114	0.025
DEG	0.012	0.006	0.013	0.026	0.062	0.006
WTDDEG	0.021	0.009	0.013	0.056	0.064	0.013

Table 1: AUC computed for PR plots for six different computer science fields in 2005

Method	DM	NW	ML	CV	HW	SE
QUIS	**0.097**	**0.101**	**0.037**	**0.083**	**0.121**	**0.057**
PMIA	0.019	0.009	0.008	0.045	0.03	0.006
DDIC	0.021	0.015	0.016	0.034	0.078	0.011
PR	0.036	0.016	0.02	0.068	0.099	0.014
DEG	0.016	0.012	0.015	0.028	0.066	0.006
WTDDEG	0.017	0.008	0.007	0.044	0.032	0.006

Table 2: AUC computed for PR plots for various computer science fields in 2011

terblockedinturkey, #direntwitter, #twitterisblockedinturkey, #25-martkorkusu, #twittericinsokagacikiyoruz, # youtubeblockedinturkey, #erdogan, #1mayis". In the context of *DBLP* we evaluated field-specific influencers, because the influencers were different for each field (such as Data Mining, Networking, etc.). In order to construct the query keywords, that were representative of each field, we selected the ten most cited papers and further filtered the 30 most frequent phrases for each field. These phrases form the query for each area. We computed the influencers for 6 different areas corresponding to Data Mining (DM), Networking (NW), Machine Learning (ML), Computer Vision (CV), Hardware (HW), and Software Engineering (SW). Each of the baseline methods required an interaction network with edge weights. In order to make the networks query-specific, we extracted the co-authorship network for the query by selecting the co-authorship relationships in papers that contained at least one of the query words. The edge weights were constructed by counting the number of query words present in the co-authored publications. This value represents the strength of query-specific co-authorship interactions. The top influencers computed from these methods were used to determine the precision and recall.

The precision-recall (PR) curves for the data mining and networking areas, at the beginning of year 2011 and 2005, are shown in Fig. 3(a) and (b), respectively. The precision-recall plot for the *Twitter* data set computed at the beginning of day May 1, 2014 is shown in Fig. 3(c). Due to limited space, we show the area under the curve (a point-wise value) for PR plots for 2005 and 2011 for six different fields in Tables 1 and 2, respectively. The influencers of the Turkey incident were evaluated on May 1 and April 13, 2014. As per the AUC values, our approach provides *the best performance in context-specific queries*. The main reason for the performance of our approach is that we look at path-level interactions of influence, while the propagation model based methods [10, 9] look only at pairwise interactions. Also, the abstraction of all the query information into a single edge-wise probability score is too unstable. This is especially true when the query-words overlap with more than one area. These issues do not affect our approach as we track specific keywords or phrases and use the phrase-level influence to compute a query-specific influence score. This may also be the reason for the inconsistency in baseline performance. For example, while PageRank performs better than the other baselines in DBLP, PMIA does better in Twitter.

Method	May 1, 2014	April 13, 2014
QUIS	**0.141**	**0.125**
PMIA	0.092	0.102
DDIC	0.095	0.101
PR	0.053	0.053
DEG	0.084	0.088
WTDDEG	0.096	0.104

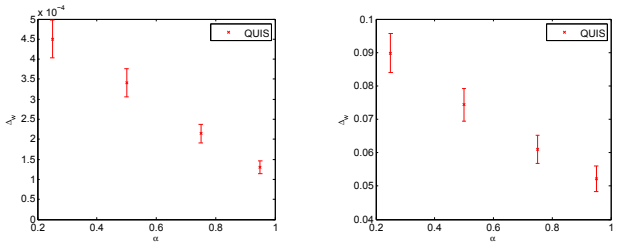
Table 3: AUC computed for PR plots for Turkey incident on *Twitter* at two different time points.

	DBLP		Twitter	
α	mean(Δ_x)	dev(Δ_x)	mean(Δ_x)	dev(Δ_x)
0.25	0.00045	0.0000466	0.08979	0.00585
0.50	0.00031	0.0000355	0.07435	0.00491
0.75	0.00021	0.0000233	0.06101	0.00418
0.95	0.00013	0.0000164	0.05218	0.00378

Table 4: Mean and deviation of Δ_x for DBLP and Twitter data sets computed at 2011 and May 1, 2014 respectively.

4.5 Sensitivity Analysis

The value of α denotes the pruning parameter and ranges from 0 to 1. Larger values of α result in reduced pruning. In order to measure the impact of pruning on effectiveness, we computed the value $\Delta_x(\alpha) = (\mathcal{I}(x,*,*,t) - \mathcal{I}_\alpha(x,*,*,t))/\mathcal{I}(x,*,*,t)$. Here, $\mathcal{I}(x,*,*,t)$ denotes the total influence of user x with no pruning, and $\mathcal{I}_\alpha(x,*,*,t)$ is the total influence computed after pruning with parameter α. Similarly, the quantity $\Delta_w(\alpha) = (\mathcal{I}(*,*,w,t) - \mathcal{I}_\alpha(*,*,w,t))/\mathcal{I}(*,*,w,t)$ computes the fractional loss of influence scores per word w. We reported the mean and deviation of Δ_w in Fig. 4, and those of Δ_x in Table 4. As discussed earlier, by *retaining even a small fraction* $\alpha = 0.25$ of the nodes, we are able to provide *an excellent estimate of the total influence* in both data sets.

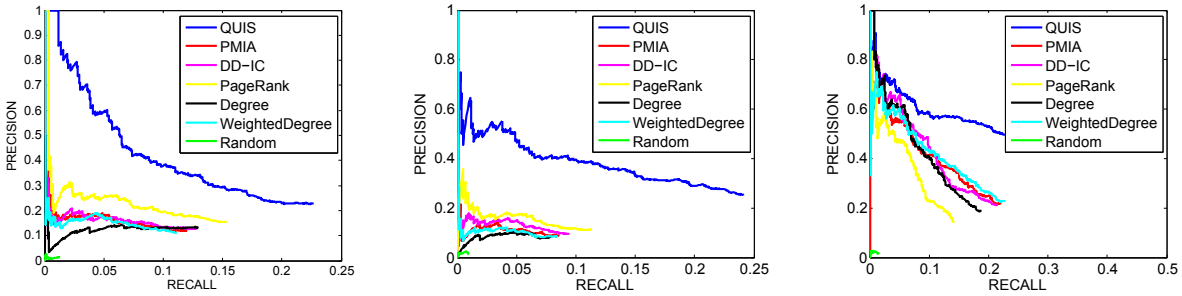


(a) DBLP (2011) (b) Twitter (May 1, 2014)

Figure 4: Mean and deviation of Δ_w for DBLP and Twitter data sets for various α values, computed at the beginning of 2011 and May 1, 2014 respectively.

4.6 Case Study of Context and User-specific Queries

In this section, we show how our framework can be used for querying context and user-specific influence values at different time points. In order to evaluate the context specific queries, we selected the query "frequent itemsets fpgrowth fptree," and we determined the top-3 influencers using our framework for every year from 2000 to 2011. We discuss the top-3 influencers for this keyword in 2000 and 2002, when this query gained traction. In 2000, the top influencers were Philip Yu, Jeffrey Ullman, and Shalom Tsur, because keywords such as fp-growth and fp-tree were not yet recognized. In



(a) Data mining query (2011) (b) Network query (2005) (c) Twitter Turkey query (1st May 2014)

Figure 3: Precision-Recall (PR) plots for two fields in DBLP and Turkey incident in Twitter data set are shown. The quality of influencers found by each approach is evaluated using the PR plot. Our approach performs consistently better in multiple field-specific queries and at multiple time points.

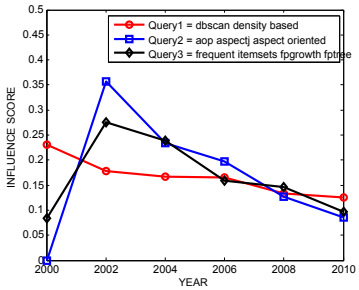
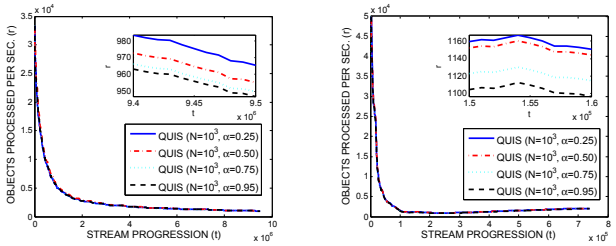


Figure 5: Growth and decay of influence for various concept-based queries.



(a) DBLP (b) Twitter

Figure 6: Number of objects processed per second with stream progression. The inset shows the computational reduction level due to pruning.

2002, the most influential individuals were Jiawei Han, Yiwen Yin and Jian Pei. The temporal aspects of influence queries can be used to understand the historical trends of a specific query in terms of its influence. For instance, in the case of the aforementioned query on frequent pattern mining, the influence values change over time, as illustrated in Fig. 5. Such queries can be used to understand the growth and decay of the influence of specific concepts. We refer to these queries as "*concept-specific temporal queries*" in Section 2.2.

One can also use our framework to *query for the most influential keywords for specific individuals*. For example, we queried DBLP for "Jiawei Han" in 2002 and obtained "fpgrowth" as the most influential keyword. However, in 2006 and 2008, the word "skylines" was the most relevant one. For Hans-Peter Kriegel, the keyword "dbscan" was the most influential one in 2000 and 2002. Such user-centered queries can be used to *understand the change in the user-specific behavior over time*.

4.7 Efficiency Analysis

We evaluate the efficiency in terms of the number of objects processed per second in the stream, and the total running time for each query. The variation in the number of objects processed per second versus the number of stream objects processed (i.e., progression of stream) is shown in Fig. 6. The processing rate reduces initially, because the number of objects indexed in the cascades are initially very small, and they gradually increase over time. As the increase in the number of objects stabilizes over time because of pruning, the processing rate stabilizes. The plot also shows the effect of pruning on the running time as well. As the value of α increases, a larger number of objects are retained in the index. Therefore, the approach is slower in this case.

The efficiency can be evaluated in terms of model building and query execution. Model building is incremental, and therefore we report the incremental time taken to build a model for a year for DBLP, and a day for *Twitter*. These time-spans reflect the time-scales at which these respective data sets were analyzed. The query execution time is the total time required to access the influence scores computed at time t from the incrementally updated tree-index, and determine the top-k influencers for the query. For the baselines, the model-building time is the total time taken to extract the context-specific network structure, and the querying time is equal to that required to list the top-k influencers using its approach. There is no straightforward one-to-one comparison of the running time, because our approach is incremental and query-based, for which there is no other (known) baseline. The degree-centrality measures are computed in linear time and can be extremely scalable for large networks. However, the *model-building* time to make them query-sensitive is large. On the other hand, arborescence-based methods such as PMIA are computationally expensive, and their query-time is very high. *Our approach does much better in terms of total time as it maintains the model incrementally*, and the querying process simply sums several influence scores in a linearly scalable way. The running times in Table 5, are in seconds and computed for DBLP in the beginning of 2011, and for the Twitter data sets on May 1, 2014.

5. CONCLUSIONS

The ability to understand the dynamics of influence in a context-specific way is very important in various applications. In this paper, we address this problem by proposing an influence-query framework. By using this framework, one can query for individual in-

	DBLP			Twitter		
Method	Model	Query	Total	Model	Query	Total
QUIS	65.68	0.15	65.83	12.25	0.051	12.30
PMIA	192.31	43.05	235.36	180.47	1959.36	1959.36
DDIC	192.31	1.05	193.36	180.47	2.25	182.72
PR	192.31	0.09	192.40	180.47	0.68	181.15
DEG	192.31	0.01	192.32	180.47	0.50	180.97
WTDDEG	192.31	0.01	192.32	180.47	0.62	181.09

Table 5: Running time (seconds) for various methods.

fluence values, context-specific influence values, or the temporal influence evolution. We enable a streaming and incremental approach, which suits the social stream scenario. Furthermore, we show that the quality of influencers obtained by our approach is superior to those obtained from the baselines. We anticipate that our scheme will enable broader classes of influence functions, which will be helpful in various industrial applications. This will be the focus of our future work.

6. ACKNOWLEDGMENTS

This research was sponsored by the Defense Advanced Research Project Agency (DARPA) agreement number W911NF-12-C-0028, U.S. Army Research Laboratory (ARL) cooperative agreement number W911NF-09-2-0053 and IBM Ph.D. Fellowship. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the DARPA, ARL, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

7. REFERENCES

[1] C. Aggarwal and K. Subbian. Event detection in social streams. In *SDM*, volume 12, pages 624–635. SIAM, 2012.

[2] C. Aggarwal and K. Subbian. Evolutionary network analysis: A survey. *ACM Computing Surveys (CSUR)*, 47(1):10, 2014.

[3] C. C. Aggarwal. Mining text and social streams: a review. *SIGKDD Explorations*, 15(2):9–19, 2013.

[4] C. Aslay, N. Barbieri, F. Bonchi, and R. A. Baeza-Yates. Online topic-aware influence maximization queries. In *EDBT*, pages 295–306, 2014.

[5] N. Barbieri, F. Bonchi, and G. Manco. Topic-aware social influence propagation models. *Knowledge and information systems*, 37(3):555–584, 2013.

[6] M. S. Bernstein, B. Suh, L. Hong, J. Chen, S. Kairam, and E. H. Chi. Eddi: interactive topic-based browsing of social status streams. In *UIST*, pages 303–312, 2010.

[7] S. Bhagat, A. Goyal, and L. V. Lakshmanan. Maximizing product adoption in social networks. In *WSDM*, pages 603–612, 2012.

[8] B. Bi, Y. Tian, Y. Sismanis, A. Balmin, and J. Cho. Scalable topic-specific influence analysis on microblogs. In *WSDM*, pages 513–522. ACM, 2014.

[9] W. Chen, C. Wang, and Y. Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *KDD*, pages 1029–1038. ACM, 2010.

[10] W. Chen, Y. Wang, and S. Yang. Efficient influence maximization in social networks. In *KDD*, pages 199–208. ACM, 2009.

[11] A. Goyal, F. Bonchi, and L. V. Lakshmanan. Learning influence probabilities in social networks. In *WSDM*, pages 241–250. ACM, 2010.

[12] A. Goyal, F. Bonchi, and L. V. Lakshmanan. A data-based approach to social influence maximization. *VLDB*, 5(1):73–84, 2011.

[13] A. Goyal, W. Lu, and L. V. Lakshmanan. Celf++: optimizing the greedy algorithm for influence maximization in social networks. In *WWW*, pages 47–48. ACM, 2011.

[14] A. Goyal, W. Lu, and L. V. Lakshmanan. Simpath: An efficient algorithm for influence maximization under the linear threshold model. In *ICDM*, pages 211–220, 2011.

[15] D. Kempe, J. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *KDD*, pages 137–146. ACM, 2003.

[16] J. Kleinberg. The flow of on-line information in global networks. In *SIGMOD*, pages 1–2. ACM, 2010.

[17] K. Kutzkov, A. Bifet, F. Bonchi, and A. Gionis. Strip: stream learning of influence probabilities. In *KDD*, pages 275–283. ACM, 2013.

[18] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. Glance. Cost-effective outbreak detection in networks. In *KDD*, pages 420–429. ACM, 2007.

[19] J. Leskovec, M. McGlohon, C. Faloutsos, N. S. Glance, and M. Hurst. Patterns of cascading behavior in large blog graphs. In *SDM*, volume 7, pages 551–556. SIAM, 2007.

[20] D. Liben-Nowell and J. Kleinberg. The link-prediction problem for social networks. *Jour. of the Amer. soc. for info. sci. and tech.*, 58(7):1019–1031, 2007.

[21] L. Liu, J. Tang, J. Han, M. Jiang, and S. Yang. Mining topic-level influence in heterogeneous networks. In *CIKM*, pages 199–208. ACM, 2010.

[22] B. A. Prakash, D. Chakrabarti, N. C. Valler, M. Faloutsos, and C. Faloutsos. Threshold conditions for arbitrary cascade models on arbitrary networks. *KAIS*, 33(3):549–575, 2012.

[23] A. Ritter, O. Etzioni, S. Clark, et al. Open domain event extraction from twitter. In *KDD*, pages 1104–1112, 2012.

[24] K. Subbian, C. Aggarwal, and J. Srivastava. Content-centric flow mining for influence analysis in social streams. In *CIKM*, pages 841–846. ACM, 2013.

[25] K. Subbian and P. Melville. Supervised rank aggregation for predicting influencers in twitter. In *SocialCom*, pages 661–665. IEEE, 2011.

[26] K. Subbian, D. Sharma, Z. Wen, and J. Srivastava. Social capital: the power of influencers in networks. In *AAMAS*, pages 1243–1244, 2013.

[27] J. Tang, J. Sun, C. Wang, and Z. Yang. Social influence analysis in large-scale networks. In *KDD*, pages 807–816. ACM, 2009.

[28] J. Tang, D. Zhang, and L. Yao. Social network extraction of academic researchers. In *ICDM'07*, pages 292–301, 2007.

[29] M.-H. Tsai, C. Aggarwal, and T. Huang. Towards classification of social streams. In *SIAM Conference on Data Mining*, 2015.

[30] H. Zhuang, Y. Sun, J. Tang, J. Zhang, and X. Sun. Influence maximization in dynamic social networks. In *ICDM*, pages 1313–1318. IEEE, 2013.