Scalable Analysis of Information Flows in Networks

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Karthik Subbian

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTORATE OF PHILOSOPHY

Jaideep Srivastava
Arindam Banerjee

November, 2014

Acknowledgements

First, I would like to thank my advisor Prof. Jaideep Srivastava for his continued support and encouragement during my graduate stay at UMN. I thank him for the freedom and belief that he had in me during these years, in allowing me to pursue interesting research questions. I learnt several key qualities from him, that I personally consider very important for a researcher, such as challenging fundamental ideas, asking visionary questions, and being able to relate and apply new scientific results to ongoing research. With no doubt, his technical prowess on data mining and social network analysis has been a strong support for my thesis research. On a personal level, Prof. Srivastava was always supportive and backed me up whenever I needed his time for a friendly discussion.

I consider myself gifted to have a co-advisor like Prof. Arindam Banerjee. His enthusiasm for research and drive for excellence in infectious. Every discussion with Prof. Banerjee has been invaluable, where I learned something, not just machine learning, but the ethics of doing good research and teaching in general. Some of the papers I co-authored with him made me realize how concise and simple communication, can be precise and effective at the same time. To summarize, I would give Prof. Banerjee most of the credit for becoming the kind of scientist I am today.

I am grateful to my mentor Dr. Charu Aggarwal at IBM Research, for his valuable inputs and critical feedback in shaping my thesis research. I am indebted to him forever, for the guidance and mentoring he has offered me throughout my research career. I want to thank Prof. Vipin Kumar and Prof. Ravi Bapna for being a part of my committee and providing me valuable feedback and suggestions at various levels of my PhD, which has helped to shape my work. I thank IBM for supporting my thesis research, during the academic year 2014-15, through a Ph.D. fellowship. I also owe my sincere thanks to Prof. Abhishek Chandra, Prof. Dan Boley, and Prof. Yousef Saad for several hours of valuable discussions. I would also like to

thank my lab mates and friends at UMN for making my graduate stay at Minnesota a fun-filled experience.

Finally, to my family. My parents, Subbian and Valli, have always encouraged me to do whatever would make me happy and satisfied. My wife, Meenal, rejoiced in my success and helped me keep my focus by backing me in my family duties. My dear daugther, Samvi, has been a constant source of energy and motivation throughout my life. I owe all my success to my family and the values they instilled in me.

Dedication

To my parents, wife and daughter for their unduly support and encouragement.

Abstract

Social, collaboration and information networks are rich in interactions through the exchange of different types of content, such as video, audio, text, short and long hyperlinks. These networks are content-rich due to the heterogeneity and size of the content generated by the nodes over long periods. Some of these networks may not necessarily have content, but they may have meta-data, such as synthetic neural networks and climate networks. Despite their differences, all these networks are extremely complex to comprehend and modeling them is even more difficult when they are dynamic and continuously evolving in time. Moreover, the underlying interaction phenomena in these networks, such as information diffusion, can be best understood only at scale and traditional algorithms are either sequential or do not take advantage of content and its temporal dynamics.

In social and collaboration networks, such as Twitter and DBLP, content propagates from one node to another through the influence of the author, nature of the content and the time of posting. Sometimes a message reposted may not be the same, however, sufficiently influenced by the original message. The goal of this task is to understand the causal behavior of topical influence in networks by developing an information flow mining tool. This tool can be used in variety of applications from understanding the most influential authors in social media to finding outlier communication sequences in cyber-crime. Traditionally, mining of content sequences is approached as a sequence mining problem, with no underlying network structure. One can later associate the content sequences with the network structure as a post-processing step. However, the treatment of content and network structure independently is extremely inefficient as the number content sequences extracted in the first step without the knowledge of network structure can be exponentially large and may never finish processing. We propose an integrated approach InFlowMine, for mining information flow patterns by tightly integrating content and network structure during the mining process. The network structure is used to guide the candidate generation process of content sequence extraction. Our approach to mine these patterns is an order of magnitude faster than state-of-the-art sequence mining techniques. We evaluated the information flow patterns discovered in the context of influence analysis application and found the patterns to be extremely useful. We show that using patterns to mine

influencers is equivalent to maximizing a sub-modular function and comes with a $(1 - \frac{1}{e})$OPT guarantee.

When we deal with extremely large graphs, the InFlowMine approach is still inefficient as it is sequential in nature and runs in a single computer and does not scale. Usually parallelization for such problems is dealt at a sub-graph level or at a flow path level, where each sub-graph or flow path is treated as an independent computational unit. Instead, in our approach we provide the highest level of parallelism by treating each node in the network as an independent computational unit. Our approach integrates network, content and time by exchanging a compact summary of content propagated by each node with time information to its neighbors. Each vertex updates its internal flow representations and creates a new set of summary objects to exchange with its neighbors for the next iteration. In each iteration we discover flow paths of one step longer than the previous iteration and this process terminates when there are no further paths to expand. We use vertex centric computational model for mining the information flow patterns in the context of Gather-Apply-Scatter (GAS) framework. Our approach pFlower scales linear with the number of cores and provides three orders of magnitude improvement over the baseline.

Today most of the big social networks, such as Twitter or Facebook feeds, are available in a streaming fashion, where each message that propagated along a set of edges in the network arrives in a data stream with time stamp information. The flow volume and velocity of such data streams are huge, especially in tens of thousands of objects per second. Most of this data has to be processed as they arrive, in order to provide near real-time information on flow patterns. In such scenarios, one needs to maintain the approximate information flow patterns that are more recent and highly frequent for different topics of interest. Current techniques for topic modeling completely ignore the availability of network structure, while the network analysis ignores the content. Our approach integrates both these ends by developing online topic models maintained on evolving approximate flow cascades. Our approach for recommendation in social networks improves the precision and recall measures up to 18% compared to baselines.

In summary, this thesis presents a set of information flow mining algorithms and applications, for understanding large scale social and collaboration networks, using content, network and temporal information in an unified and scalable fashion.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The proliferation of internet and the online digital media, has made its impact by providing a platform to interact, exchange and share content with a larger audience, than ever before. The success of these online medium is mostly evaluated based the reach of the "*voice*" of each user [1]. Such massive propagation of digital information shared by a sequence of users over time is often referred to as *"viral"* and using this strategy for sales and marketing resulted in various applications of *"viral marketing"* [2, 3].

There are several kinds of social and collaboration networks that hosts plethora of knowledge and information in its digital form. In online social networks, such as *Facebook* and *Twitter*, users are allowed to post, mutate, comment, share and like information of their choice. There are specialized social networking sites, such as *FourSquare*, where users share their preferred locations of interests; *Flickr* and *Pinterest*, where users share the preferences and *like*'s in terms of images and tags. On the other hand, there are several collaboration networks of inventors and researchers using their history of collaborations. Examples of such website includes, U.S. Patent Office (USPTO) [4], Database List of Publications (DBLP) [5], ResearchGate [6] and Arnetminer [7]. Though, these networks are similar to friendship and follower networks of Facebook and Twitter, they are different in terms of the rates of activity and the strength of relationships. The academic and invention relationships happen over long term and are often much more stronger than an average friendship relationships in the social networks. Despite these differences, the networks tell us who-is-related-to-who and when did the relationship began.

1.1 Information Flow in Networks

In several of these online social and collaboration platforms, the *underlying network plays an important role in information dissemination, mutation and consumption*. Here are a few examples of networks and their roles in different information propagation scenarios.

- In DBLP the underlying network is the relationships of co-authors, available through their publications. The information shared through the *co-authorship network* could be key research ideas, terms and technologies. One author may initiate the research idea and may spread through their collaborators to different regions of the network.

- Twitter and Digg uses the *followers network* for information sharing [8, 9]. In this network, users can follow any other user of interests to them and can receive content propagated by them in their own wall or news feed. Though, there may be limits on the maximum number of users to follow.

- Some social networks such as Facebook, have a two-sided relationship [10]. Here a user cannot follow another user on his own, instead has to be accepted as a friend in the other user's network. Once a friendship established, private and broadcast messages pass back-and-forth in this *friendship network*.

Most of the social and collaboration network and media sites have an underlying communication network. The network in most of the social media sites are *explicit*, such as friend or follower network. But several other collaboration platforms have networks that are rather *implicit*. An example of implicitly observed network is the co-authorship network as in DBLP or co-inventor networks as in USPTO. An implicit network is the result of activities performed by the users, such as publishing and inventing, respectively, in DBLP and USPTO. Even where there are explicit networks, activities of users can be outside of this explicit network that can result in implicit networks. For instance, in Twitter, *re-tweet* or *mention* activities are used to construct re-tweet and mention networks [11] respectively. In a retweet(mention) network, user 'a' retweets(mentions) 'b', resulting in an edge from user 'b' to 'a'. Note that *the direction of edge denotes the flow of information*. Unobservable relationships can also account for information dissemination and sharing, for instance in DBLP authors could share ideas in a conference or workshop venue, and such information is unavailable and unobserved in most occasions.

Figure 1.1: The figure shows the retweet information flow, plotted on a world map. This was based on the one hour of retweet information obtained after March 2011 earthquake in Japan. The original tweets are shown in red and the retweets are shown in green. Most of the original tweets are from Japan and the rest of the world simply propagates this information further.

Such network pieces can be inferred using existing related work [12, 13] and then included as part of the original network, to recover the complete network structure.

1.1.1 Information Cascades

The flow of information is seen as an avalanche of local sharing behavior and often referred to as "*information cascades*" [14, 15, 2, 16, 8]. The sharing can be restricted to a network such as friend or follower network, or can be shared with anyone, like in email networks [17]. An example of such avalanche of information flow mapped on to the geographic map is shown in Figure 1.1. In an implicit network, such as DBLP, the flow of information from a specific person can be used to visualize the patterns of content flow paths, especially for the research ideas, in their respective neighborhood. Consider Figure 1.2, where the content flow paths originating from Professor Jiawei Han are overlaid to form a information flow network. It clearly shows the high centrality of Prof. Han in his network and leaders evolving in his network neighborhood, such as Jian Pei and Xifeng Yan. We will revisit this information flow graph and analyze it in more detail in Chapter 3.

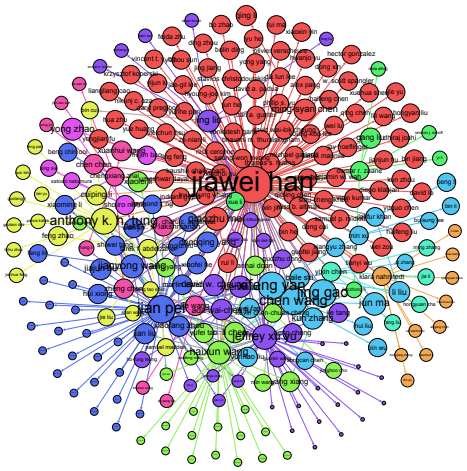


Figure 1.2: The figure shows the dominant information flow paths of Professor Jiawei Han. This size of node denotes the centrality of the nodes in the flow graph and the color denotes the community. It is very clear from the information flow graph that there are several well separated communities led by students and collaborators of Prof. Han. The image can be zoomed in and out using any PDF viewer for more clarity.

1.1.2 Role of Content, Network and Time

Networks have a played a central role in understanding the connected behavior in complex systems and how independent components play together to achieve larger goals (that are beyond their own capacities), such as demand management in power grids [18], emerging success of an growing organization [19], and even synchronized interaction among biological entities [20]. The synchronization [21] between network parts and the emergence of the connected components [22] are some of the classics that illustrate the importance of understanding the collective behavior in large scale systems using networks.

The avalanche of events or cascading behavior in various networks have been well-studied in the last decade [23, 24, 25]. Despite these efforts, on understanding the properties of the cascade and their relationship to the global phenomena, the use of content in such analysis has largely remained unexplored. Recently. some of these network features were used in predicting the future cascading behavior [23], however, most of the analysis have been focused on network-centric aspects and does not use the content features.

Any piece of information that propagates in the network has three important components associated with it:

- **Content**: An integral part of the information is its content. The content can be in natural language [26], or any equivalent semantic representation, such as short URLs or hashtags [27]. Understanding network patterns with out incorporating the content interactions results in flows that are not context-sensitive. We will illustrate this in more detail in a bit.

- **Network**: The content often follows a specific path in the underlying network of relationships, especially due to the authority and trust people express through their relationships, rather than the generic population [8]. Such relationships can be friend, follower or co-author networks and the repeated flow of a content along a certain path or part of the network, signifies the important role played by the vertices in that region of the network.

- **Time**: The rate at which content flows over specific regions of the network is not constant and rather varies by time. This temporal aspect of increase and decrease in demand, for the content, in specific parts of the network, plays a critical role in information flows. This also leads to the emergence of *local and global events* across different parts of the network [28, 27].

The role of content, network and time also plays a critical role in effective pruning, personalization of models, and scaling up of algorithms. Especially in several applications of information flow, which we describe a bit later in this chapter, content plays a crucial role. Consider the task of influence analysis in DBLP, where the goal is to find the top-k influential authors for a specific context. If we ignore the content that describes the context, then the overall influencers in the entire network will be absurd in most scenarios. For instance, we extracted the overall influencers in DBLP and extracted the influencers specific to the data mining context, which is best described by the most frequent 1000 words after removing the stop-words. The results are shown in Table. 1.1.

Time plays a key role in information propagation, as the influence of each content propagated does not last forever. Typically, in a Facebook-wall the posts from the previous day lasts for utmost 24 hours on an average [10]. A key question is how does the life-time of each post affect the future information flow cascade? In other words, given a life-time threshold, after

Ignoring the context	Using the context
Thomas S. Huang	Thomas S. Huang
Wei Wang	Jiawei Han
Jie Yang	Wen Gao
Wen Gao	Qiang Yang
Wei Liu	Michael I. Jordan

Table 1.1: This table illustrates the importance of using content in the task of influence analysis. The most popular thousand unigram keywords specific to the field of data mining was first extracted. These keywords were used to extract the specific information flows, which is then used in the task of finding the influential authors. When the context (of data mining) is ignored the influential authors, as shown in first column, are completely arbitrary, while the second column has the most important authors in the field of data mining.

which the nodes influence on a content is inapplicable to its neighbors, can we extract patterns of information flows preserving this property? What characteristics do such information flows have? These are precisely some of the questions that we address in this thesis in Chapter 3.

Moreover, in several time-sensitive tasks, the content dynamics varies continuously over time. Consider an extension our previous example of influence analysis. The influencers for data mining context in 2002 versus 2008 would be drastically different. Even for the same author the influence of keywords may increase and decrease over time. For example, in DBLP for Jiawei Han in 2002 the most-influential keyword was "fpgrowth". 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 and this keyword vanishes in the top-10 list in 2011. Such temporal dynamics can only be understood by suitable incorporation of time and content along with the network structure. We show how to incorporate time and contextual information along with network in an online fashion in Chapter 4.

1.1.3 Content-centric Analysis

The amount of content available online in social and collaboration media is increasing in an exponential rate. Hence, as we discussed earlier, this content is extremely valuable for the purpose of determining the context-specific information flows. Most of the work in information flow analysis have been closely associated with influence cascade models [29, 30, 31], such as

linear threshold or independent cascade models [29], as they model the underlying process of information diffusion. The main drawback of these models is the use of static edge propagation probabilities, that do not directly capture the dynamic nature of the content flows. Also, the work on actually modeling these transition probabilities is very sparse and almost never uses the *actual* content behavior in the underlying network. Some recent work on estimating influence probabilities [32] uses a number of application-dependent assumptions about the weights of the different edges in the social network. Our content-centric analysis suggest that such static modeling is quite questionable, because the influence behavior of an edge depends on many factors including the actual content of the interaction, the time period, and other factors, *which can only be tracked using the underlying content stream* of the social network. In this thesis, we propose an approach that addresses these problems by tracking the dynamic content flows in the underlying network structure to find information flows and as a result finds the information epicenters or influencers.

1.1.4 Network-centric Analysis

In the content-centric analysis our focus was primarily to make use of the large amounts of content available in online social media to extract information flow patterns in the underlying network. However, we ignored one important fact that such data occurs in large scale and extracting information flow patterns at scale must be done in a distributed and parallel fashion. One could achieve this parallelism at a tree-, path- or a vertex-level, with increasing levels of parallelism in that order. To address this issue, we propose a vertex-centric algorithm that sends and receives messages from its neighbors to extract the information flow patterns. Since we make use of the network structure to address the scalability issue, we refer to this approach as *network-centric analysis*. We use a Gather-Apply-Scatter (GAS) paradigm to implement our approach. Our approach is asynchronous, fueling further to the scalability, and it does not need to wait for the slowest vertex to finish all its computations. We experimentally show that our approach achieves up to *three orders of magnitude* advantage over the state-of-the-art, with an increasing advantage with a greater number of cores. We also study the effectiveness of the discovered content flow patterns by using it in the context of an influence analysis application.

1.1.5 Real-time Analysis

Our focus so far has been to use the content to improve the quality of extracted flows and use the structure to improve the efficiency. However, these approaches cannot handle the data arriving at a high speed in a streaming fashion. As these computations are not incremental in nature, the entire data must be available upfront. This is a serious disadvantage for several industrial applications. For instance, in Twitter the data arrives in terms of continuous stream of user activities, in Flickr a stream of photo shares and comments. It becomes important to extract flow in a streaming fashion, and allow incremental computation to the model, rather than recomputing all the information from scratch every time a new data point arrives. We propose an online algorithm that keeps track of information flow cascades in real-time fashion in Chapter 4. The main difficulty, though, is the availability of limited memory and one-pass constraint – very typical to streaming analysis. Henceforth, we propose an approximation scheme to maintain only the most important part of the cascades, pruning out the rest. We establish mathematical bounds on this flow cascade approximation, and show that the underlying flow cascade follows a Zipf distribution. We evaluate the discovered flows in the context of an influence and link prediction applications.

1.2 Applications

The information flow patterns discovered are valuable and applicable in the context of several industrial applications. In this section, we outline a few important problems and explain their relationship to flow patterns. These applications will be discussed in great detail in the following chapters and evaluated using multiple real-world social and collaboration networks.

1.2.1 Influence Analysis

The problem of finding influential actors is important in various domains such as viral marketing [33] and political campaigns [34]. Kempe et al. [29] formally defined this as an optimization problem over all possible subsets of nodes with cardinality k, where the goal is maximize the overall influence of the k nodes. Subsequently, a significant amount of work [29, 35, 36, 37, 38, 31] has been done on this area. Most of these related work have been closely associated with influence cascade models [29, 30, 31], such as linear threshold or independent cascade

models [29], as they model the underlying process of information diffusion. The main drawback of these models is the use of static edge propagation probabilities, that do not directly capture the dynamic nature of the content flows. Also, the work on actually modeling these transition probabilities is very sparse and almost never uses the *actual* content behavior in the underlying network. Some recent work on estimating influence probabilities [32] uses a number of application-dependent assumptions about the weights of the different edges in the social network. The results of our work, in this thesis, suggest that such static modeling is quite questionable, because the influence behavior of an edge depends on many factors including the actual content of the interaction, the time period, and other factors, *which can only be tracked using the underlying content stream of the social network.* Our proposed approach addresses these problems by tracking the dynamic content flows in the underlying network structure to find information flows and as a result finds the information epicenters or influencers.

The influence of a node is generally characterized by the number of nodes that forward or propagate its content further in the network [29]. This is precisely captured by the set of all information flow paths. There are two main factors that affect the influence of a node in the context of a flow path: (1) the position of the node in a flow path and (2) the number of frequent flow paths in which it occurs. As the node consistently occurs earlier and in more flow paths it is considered as a promising influencer. We use this notion to extract the influential authors for different contexts and at various time periods. We show that our approach is much better than several popular baselines [29, 38, 31] in Chapter 3. Our approach is flexible enough that without any re-computation from scratch, unlike the popular influence maximization approaches, it can be used to query for influence scores for various context, time points, and users, in different ways. This offers immense flexibility for various industrial applications, such as viral marketing.

1.2.2 Link Prediction

An important aspect of many networks is that they are often available in the form of social stream of content interactions between the users. Many networks cannot be easily collected for analysis, and in some cases such as an interaction network, the underlying "social network" is transient without a base static network. Even in the case of more formal social networks such as *Twitter*, the *social stream is the only observable aspect of the dynamics in the social network,* at least from a practical perspective. Furthermore, the implicit network and content structure of the

social stream usually evolves fairly rapidly with time [39]. In such cases, the appropriate links to recommend may vary *significantly* depending on the recent temporal patterns of interaction, and information flow between users. In such networks, the relevant information flow patterns provide interesting patterns of influence in the network [26]. We argue that such patterns of influence are important, since a user who is influenced by another is more likely to link to the latter. Therefore, a natural approach for link prediction [40] in social streams is to find important paths of information flow between nodes, in real-time, in order to make useful predictions in a time-sensitive way. In many cases, specific content in the network may be more significant in finding relevant links between users, and therefore it may be useful to find *topically-relevant* links between nodes. This is also useful for a target-marketing application for recommending nodes in a commercial network, or in a general information network, where nodes may be topically relevant objects (e.g. golf club). For example, when the content corresponding to the topic "golf clubs" is passed around in the network, it is very relevant for link recommendations between users that are interested in golf.

In this thesis, in Chapter 4, we present an information flow-based link prediction algorithm. We show that such an approach can be very effective in finding relevant, real-time and temporally sensitive links. The approach can work with the limited network information that is available in real scenarios, and can adjust to *content-specific* topics for the link prediction process. An important aspect of this link prediction approach is that it is *activity-centered*, in that it preferentially utilizes the more active and recent flow paths in the network in order to make link predictions. The formation of links is an activity in itself, which is closely correlated with other forms of information flow activity in the network and hence our information flow based approach is a natural choice for link prediction in social streams. Our experimental results will demonstrate the effectiveness of such an approach.

1.2.3 Event Detection

Online event detection is an important problem in the context of temporal and streaming text data. It has been mostly studied in the area of *topic detection and tracking* [41, 42, 43, 44, 45, 46, 47]. This problem is also closely related to stream clustering, and attempts to determine new *topical trends in the text stream* and their significant evolution. The idea is that important and newsworthy events in real life are often captured in the form of *temporal bursts of closely related messages* in a social stream. The problem can be proposed in both the supervised and

unsupervised scenarios. In the *unsupervised case*, it is assumed that no training data is available in order to direct the event detection process for the stream. In the *supervised case*, prior data about events is available in order to guide the event detection process.

The events do not occur in vacuum, but rather occurs as temporal bursts in certain network locality. The propagation of content in a local neighborhood can be extracted from information flow and used for effective online clustering of the content information. In Chapter 5, we will study the problems of clustering and event detection in *social streams* in which each text message is associated with at least a pair of actors in the social network. We use the term social network rather loosely, as it could refer to the messages in an online chat messenger service, or it could even refer to an email network in which messages are sent between pairs of actors and such information flow drives the underlying process of online clustering. A number of interesting issues arise in such social networks, because they are dynamic, and are associated with *network structure* in the stream. Specifically, each actor is a node in the social network, and each message sent in the social network is the text content associated with an edge in the social network. Clearly, multiple messages can be sent between the same pair of actors over time. In this case, we would like to use *the topical content of the documents, their temporal distribution, and the network structure of the information flow in order to detect interesting events and their evolution*. Clearly, messages which are sent between a tightly knit group of actors may be more indicative of a particular event of social interest, than a set of messages which are more diffusely related from a structural point of view. Such messages are structurally well connected, when the social network is viewed as a graph, with the edges corresponding to the messages sent between entities.

Clearly, the local cascade structure of a content, encapsulates the local neighborhood structure, and plays a key role in detecting significant events. For example, a major event which is specific to a particular university may correspond to messages primarily within the students and faculty members of that university, whereas a more global event such as the *Mideast Unrest* may correspond to content related messages with a larger locality of the social network. In the former case, the communications are likely to occur between the more closely connected group of entities, while in the latter case, the messages may be more global, with a bias towards a locality of the social network corresponding to the mid east. The ability to detect such different events with different levels of locality and scales is extremely challenging in massive and large scale social networks.

1.3 Related Work

The problem of information flows has traditionally been studied as epidemic spread in computer networks [48, 49, 50, 51, 52, 53]. Much of this work studies patterns of information flows and conditions under which such flows become epidemics. Some recent research [53] studies the information propagation problem in context of similar models for computer virus and epidemic spreading [50, 51]. The work in [48, 54, 53] studies the information flows in the context of social networks and other kinds of computer networks. A method for mining the network value of customers in the context of a viral marketing model was proposed in [3]. None of these methods study the problem of mining information flow patterns from the underlying network structure using the content propagation.

There are several papers [29, 52, 31, 38, 36, 37] that discuss the problem of information diffusion in the network and uses the diffusion model to find the top influencers. These approaches use a fixed and static view of the network in which the edge transmission probabilities are assumed a-priori, and do not focus on characterizing the cascading behavior from the flow of information. In that sense, these methods are too focused on the specific structure of the network, and pre-defined model of edge interactions, rather than the *dynamic interactions in a network during any period*. In fact, the process of modeling edge propagation probabilities and mining influencers are not separate problems, and should be treated in an integrated way, based on the actual behavior of the underlying social stream. From the perspective of influence analysis, it is not only important to identify the key influence points, but also the important information flow patterns in the underlying network.

Some recent work has designed methods for analyzing cascading behavior in large blog graphs [16]. Though this work does not use the actual content of the interactions for modeling the cascading behavior in any particular scenario, and only provides a general study of the typical kinds of shapes and power-laws exhibited during cascades. Some work has also been done on studying cascading behavior of time-series data [55], though these methods are also designed to work with fixed models of network structure and interactions, rather than *purely data-driven methods for influence analysis*. There are other related work that treats the influencer mining problem in a supervised classification setting [56, 11]. Here a set of network features are constructed and the problem is posed as a binary classification problem, where the ground truth is based on the number of user retweets. There are numerous papers that computes influencers

based on the notion of centrality [57, 58], completely ignoring the content characteristics. Our approach addresses these problems by finding the information flow patterns using the actual content propagated in the underlying social network. The proposed approach, for this reason, can be used to find flow patterns, and influencers, specific to a topic or network community.

The approach discussed in [59] to find influencers in an network, is designed for constructing *action* graphs, based on user-specific actions. On the other hand, the focus of our approach is to determine the information flows patterns through the network, in addition to determining the most relevant epicenters of these information flows. This is a more general *flow-centric* framework for influence propagation. Furthermore, by changing the lexicon for tracking the flows, it is possible to determine the *topical* influencers. As our approach works with minimal content-centric data structures, it can be updated in an online fashion, and used to perform the analysis at any point during the progression of the social stream.

1.4 Contributions and Outline

This thesis is structured in two parts: algorithms and applications. Today information flows occur in a variety of networks of different sizes and the volume and velocity of content propagated in these networks spans. For instance, the network could be a small club of few hundred people where users frequently exchanged information, versus a large social network like Twitter, where billions of messages are sent everyday. In the small set of users the dictionary of words used may be very limited, unlike the continuously growing dictionary of words in sites like Twitter. The volume, velocity and variety of such data brings to us different needs in terms of algorithmic design.

In case of small or medium size networks, when the data fits in a single compute memory, we design a sequential approach. We refer to this approach as content-centric and discuss it in detail in Chapter 2. This approach can be run at regular intervals for newly arriving data and recomputed from scratch, especially because the data is reasonably small. When the data size outgrows a single computer, both in terms of the network size and content volume, we need a distributed approach. The distributed approach should be able to scale in terms of multiple machines and take advantage of multiple cores in each machine. We present such an approach in Chapter 3. This approach can be run at frequent intervals that is particularly suited for large scale data sets, using vertex-centric computational models, such as GraphLab [60] or Pregel.

The distributed approach takes advantage of the network structure and parallelize the flow mining process at a vertex-level. Hence, we refer to this principle as "network-centric" approach. Depsite, the massive scalability the approach is not incremental in nature. Whenever a new data point arrives, the entire computation has be to repeated from scratch. This becomes extremely critical for social networks, as the only observable aspect of today's social networks are the continuous stream of activities in the social stream. To accommodate such stream and incremental computation, we propose an online algorithm that keeps track of cascades in an incremental fashion. Depending on available hardware limitations, the approach can also maintain an approximate version of the cascades, yet capturing its most important recent dynamics. We present this online algorithm in Chapter 4.

In terms of algorithmic contributions, we formalize the notion of information flow mining in Chapter 2 and establish the computational complexity of the problem as #P-Complete, in Chapter 3. We discuss a sequential, distributed and streaming algorithm for mining information flow patterns in small, large and streaming data sets respectively. Moreover, we show that our idea of using network-centric models for information flows can be extended to popular frequent and sequence mining problems, that had largely remained inapplicable for massive data sets. We establish bounds on approximating cascades, especially for large volume of social streams and show its effectiveness in link prediction and influence analysis tasks.

This thesis covers three main applications: (1) Influence analysis, (2) Link prediction, and (3) Event detection. These three applications are central to any social and collaboration network [1]. Using sequential and distributed algorithms we show that the influential users obtained using our approach is far superior than traditional and popular baselines [38, 31, 29]. This is covered in Chapters 2 and 3. As link prediction is more real-time in nature (compared to influence analysis), we show the effectiveness of information flow analysis in link-prediction in Chapter 4, using our online algorithm. We also show that information flows can be used in online clustering of content to detect temporal bursts of content propagation in specific network localities. We outperform several well-known baselines and show the effectiveness of our approach using precision and recall analysis. We also detect important events, related to civil protests and earthquake disasters, up to 6 hours before the actual breakout of the news, using our approach. We discuss this application in Chapter 5. We also provide an outline of other applications and interesting extensions of our work in Chapter 6.

At a high-level, this thesis formalizes a very important problem of information flow mining;

establishes its complexity class; develops a sequential, distributed and streaming algorithm for different data sets that is suited for different data sets based on their volume, velocity and variety. Through out this thesis, different applications are used to illustrate the effectiveness of the discovered information flows and its relevance to network analysis.

Chapter 2

Content-centric Information Flow Mining

2.1 Introduction

Social networks such as *Twitter* and *Facebook* have tremendous amount of content being posted on them on a daily basis. A lot of this content (or key parts of them) is often re-posted by different users, and this leads to cascades of information flows. Examples of such cascades are as follows:

- When a user posts a news-article or other interesting item on their wall in *Facebook*, this is replicated by the friends of this user on their own wall.
- When a user tweets a message (or a link) in *Twitter*, the followers of that user re-tweet the message (or the link) to their followers.
- Sometimes, a user may not re-tweet the same message, but may be sufficiently influenced by the message to tweet a different message containing the same topic (which is reflected by its *hash-tag* in *Twitter*). In such a case, the topic of the message has been propagated and not the exact same message.

In the context of many networks, such as *Twitter*, the information flow cascades provide useful insights into the underlying information flow patterns in the social networks, and should therefore be studied for mining and analysis. This is because repeated tweets through the social

network can be tracked as discrete events, and the sequence of such events can be analyzed in conjunction with their linkage relationships. In effect, we study this problem of mining *information flow patterns* in networks using such discrete events tracked through their content propagation.

Furthermore, such patterns provide useful insights about the influence of different entities in the underlying network. Correspondingly, we show that these patterns are useful in order to determine the *frequent epicenters* of the information cascades in the underlying network. The problem of finding the epicenters of cascade patterns is indirectly related to that of finding influential nodes [29, 31, 61, 62, 32] in a network, though the latter problem does not directly track content interactions in order to determine influential nodes.

Most of the work in information flow analysis have been closely associated with influence cascade models [29, 30, 31], such as linear threshold or independent cascade models [29], as they model the underlying process of information diffusion. The main drawback of these models is the use of static edge propagation probabilities, that do not directly capture the dynamic nature of the content flows. Also, the work on actually modeling these transition probabilities is very sparse and almost never uses the *actual* content behavior in the underlying network. Some recent work on estimating influence probabilities [32] uses a number of application-dependent assumptions about the weights of the different edges in the social network. The results of this paper suggest that such static modeling is quite questionable, because the influence behavior of an edge depends on many factors including the actual content of the interaction, the time period, and other factors, *which can only be tracked using the underlying content stream of the social network.* Our proposed approach addresses these problems by tracking the dynamic content flows in the underlying network structure to find information flows and as a result finds the information epicenters or influencers.

2.1.1 Contributions and Organization

Our goal in this paper is to introduce the problem of information flow mining to track the dynamic content flows in the network. We develop an efficient algorithm called *InFlowMine* to discover the information flow patterns using content propagation on the underlying network structure. As the problem of information flow and influence mining are tightly coupled, we also discuss this as a potential application of information flow patterns. We propose a Network Influencer Discovery algorithm called *NDIF*, to discover influencers using the mined information

flows patterns. Our framework can easily be tailored to content- and topic-specific influencer discovery. Finally, we empirically verify the effectiveness and efficiency of our approach against existing methods. Our method runs faster up to an order of magnitude compared to the baseline. We verify the efficiency using the influencer mining application and show that the influencers discovered are much better compared to several influencer mining algorithms.

This paper is organized as follows. The remainder of this section discusses the related work. In section 2 we introduce the problem of information flow mining and in section 3 we discuss an approach to address this problem. In section 4 we explain an application to find the information epicenters using the discovered flow patterns. Section 5 discusses the experimental results. Section 6 discusses flow-based ego networks, which is another interesting application of discovered information flows. We discuss further applications in Section 7. The conclusions and summary are given in Section 8.

2.1.2 Related Work

The problem of information flows has traditionally been studied as epidemic spread in computer networks [48, 49, 50, 51, 52, 53]. Much of this work studies patterns of information flows and conditions under which such flows become epidemics. Some recent research [53] studies the information propagation problem in context of similar models for computer virus and epidemic spreading [50, 51]. The work in [48, 54, 53] studies the information flows in the context of social networks and other kinds of computer networks. A method for mining the network value of customers in the context of a viral marketing model was proposed in [3]. None of these methods study the problem of mining information flow patterns from the underlying network structure using the content propagation.

There are several papers [29, 32, 31, 36, 37] that discuss the problem of information diffusion in the network and uses the diffusion model to find the top influencers. These approaches use a fixed and static view of the network in which the edge transmission probabilities are assumed a-priori, and do not focus on characterizing the cascading behavior from the flow of information. In that sense, these methods are too focused on the specific structure of the network, and pre-defined model of edge interactions, rather than the *dynamic interactions in a network during any period.* In fact, the process of modeling edge propagation probabilities and mining influencers are not separate problems, and should be treated in an integrated way, based on the actual behavior of the underlying social stream. From the perspective of influence analysis, it is

not only important to identify the key influence points, but also the important information flow patterns in the underlying network.

Some recent work has designed methods for analyzing cascading behavior in large blog graphs [16]. Though this work does not use the actual content of the interactions for modeling the cascading behavior in any particular scenario, and only provides a general study of the typical kinds of shapes and power-laws exhibited during cascades. Some work has also been done on studying cascading behavior of time-series data [55], though these methods are also designed to work with fixed models of network structure and interactions, rather than *purely data-driven methods for influence analysis*. There are other related work that treats the influencer mining problem in a supervised classification setting [56, 11]. Here a set of network features are constructed and the problem is posed as a binary classification problem, where the ground truth is based on the number of user retweets. There are numerous papers that computes influencers based on the notion of centrality [57, 58], completely ignoring the content characteristics. Our approach addresses these problems by finding the information flow patterns using the actual content propagated in the underlying social network. The proposed approach, for this reason, can be used to find flow patterns, and influencers, specific to a topic or network community.

The approach discussed in [59] to find influencers in an network, is designed for constructing *action* graphs, based on user-specific actions. On the other hand, the focus of our approach is to determine the information flows patterns through the network, in addition to determining the most relevant epicenters of these information flows. This is a more general *flow-centric* framework for influence propagation. Furthermore, by changing the lexicon for tracking the flows, it is possible to determine the *topical* influencers. As our approach works with minimal content-centric data structures, it can be updated in an online fashion, and used to perform the analysis at any point during the progression of the social stream.

2.2 Problem Definition

In this section, we introduce the necessary notations, and definitions to formally define the problem of information flow mining.

We assume that we have a network $G = (N, A)$, containing the node set N and the edge set A. For ease in exposition, we assume that the edge set A is undirected, though this assumption can be easily relaxed without loss of generality. Each actor $i \in N$ performs a number of

content-based actions such as sending tweets, or posting wall posts etc. Since, 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, denoted by $\mathcal{T} = T_1 \ldots T_n \ldots$. We refer to each T_i as a *content token*. Each such content-token in the stream is associated with the following meta-data:

- The content-token T_i represents the i-th component of the content created by an actor in the social stream. This could correspond to the actual text 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 T_i could also simply be one of a set of a topical dictionary of keywords (contained in the tweet), or it could be the hash-tag from the tweet. We note that since this content may be copied, re-posted, re-tweeted, or may generally influence the future content posted by the different actors (in the form of appropriate keywords or hash-tags), the value of the different strings T_i (over different values of the index i) will often be the same. It is precisely this propagation of the content over the network, which we wish to track. In general, for a given application, it is first pre-decided what kind of content to track for interesting information flow patterns (e.g. URLs posted, full tweets or keywords from a specific topic), and then all future analysis is applied to this set of content-tokens. An interesting feature of our approach is that it can be easily applied to an arbitrary subset of content-tokens in order to facilitate topic-specific influence in the network.

- It is assumed that the content T_i is created by the actor a_i. The value $a_i \in N$ is an index drawn from the actor set N, and it represents the actor who is responsible for the origination of that particular piece of content.

The entire social stream is denoted by $\mathcal{T}$. We note that in many social networks such as *Twitter* and blog networks, the stream of content can be tracked in an automated way. In many cases, the re-posting of the content is a direct result of a particular actor being influenced by specific neighbors, and the cascade effect of this influence behavior, which can be monitored in terms of the URLs posted, the keywords (or hash-tag) in the posts, or the exact content of the tweets. Therefore, it is interesting to determine the frequent information flows, *in which consecutive participants share a neighbor relationship*, and the content of the (tracked subset of the) post is the same.

We define the information flow patterns in terms of *actors* rather than the *content*, as the same sequence of actors may be involved in multiple such content interactions over different portions of the stream. The purpose of our approach is to determine such frequent information flows, where certain content flow may occur frequently in a specific path of the network. In this paper, we assume that flow patterns are essentially sequential paths in the network, however, resulting flow cascades can be obtained by overlaying multiple such frequent flow patterns. Correspondingly, we define the model for determining the information flow patterns in networks. We start off by defining a *valid* flow of information, based on the network relationship of different actors:

Definition 1 (Valid Flow Path)**.** *A valid flow path in the network $G = (N, A)$ is an ordered set of nodes $i_1 \ldots i_k \in N$, such that all nodes $i_1 \ldots i_k$ are distinct from one another, and for each $r \in \{1 \ldots k-1\}$, an edge exists between i_r and i_{r+1} in A.*

We note that the purpose of defining a valid flow path is to focus only on those flows which are governed by the *network relationships* between actors, rather than arbitrary links. This is because of the underlying assumption is that network cascades are caused by social interactions, copying behavior, and more subtle influences between neighbor nodes.

For the ease of analysis, we consider only the *first* posting of a particular piece of content by an actor in the social stream. In effect, this implies, the actors not influenced repeatedly by their own posts. In many cases, repeated postings which are exactly identical are more likely to be spam [63], rather than postings of specific interest, and the issue of *first posting* is usually more critical. Therefore, it makes sense not to include such repeated posts caused by the same content from the same actor. For a given actor $j \in N$, who posts the content $U \in \mathcal{T}$, the index of the first position of the content U in the social stream $\mathcal{T} = T_1 \ldots T_i \ldots$ is denoted by $F(j, U)$. Thus, the content $T_{F(j,U)} \in \mathcal{T}$ is the same as U. In the event that the content U is *not posted* by actor j, we have $F(j, U) = -1$.

Next, we define the concept of *information flow frequency*. This is essentially defined as the number of times that an ordered set of actors posts the same content-token in that order.

Definition 2 (Information Flow Frequency)**.** *The information flow frequency of actors $i_1 \ldots i_k$ is defined as the number r of* unique content-tokens *$U_1 \ldots U_r$, such that for each U_j, we have:*

- *Each actor $i_1 \ldots i_k$ has posted U_j. Thus, we have:*

$$F(i_r, U_j) > 0 \; \forall r, j \tag{2.1}$$

- *Each content U_j was posted by the actors in the order $i_1 \ldots i_k$. Thus, for each U_j we have:*

$$F(i_1, U_j) < \ldots < F(i_r, U_j) < \ldots < F(i_k, U_j) \tag{2.2}$$

The above definition can be generalized in order to determine the *frequent flow path* for a specified frequency. This is essentially a sequence of actors who share a neighbor relationship in the network, and post the same set of content-tokens in a specific order, possibly as a result of being influenced by the behavior of their neighboring actors in the network.

Definition 3 (Frequent Flow Path)**.** *A sequence of actors $i_1 \ldots i_k$ is said to be a frequent flow path at frequency f, if the following two conditions are satisfied:*

- *The sequence of actors $i_1 \ldots i_k$ is valid flow path.*
- *The information flow frequency of the actor sequence $i_1 \ldots i_k$ is at least f.*

The frequency f expressed here is absolute and can be easily converted to relative frequency by appropriate normalization. The frequency, in essence, indicates the strength of the flow path, as together with the *valid flow path* constraint it ensures the flow obeys the structure of the network. The problem of information flow mining is formally defined as follows.

Problem 1 (Information Flow Mining)**.** *Given a graph G, a stream of content propagations $\mathcal{T} = T_1, \ldots, T_i, \ldots$ and a user specified frequency f, the problem of information flow mining is to find the set of all frequent flow paths S in the underlying graph G using the unique content tokens $U_1, U_2, \ldots, U_m$ generated from the content stream $\mathcal{T}$.*

2.3 Information Flow Mining Algorithm

In this section, we discuss our algorithm for mining information flows from the social stream $\mathcal{T}$.

A simple method for doing so would be to generate all possible information flows in a brute-force manner and test if they satisfy the frequent flow path conditions. However, this kind of approach is extremely slow as it completely ignores network relationship and frequency of interest. A relevant observation is that the vast majority of these flow patterns are not valid if they do not satisfy the *flow path constraint*, and therefore it is wasteful to use such an approach

from the perspective of computational efficiency. Therefore, we need a method which uses the network structure directly during the flow mining process.

We propose a level-wise approach to efficiently perform the flow mining process. Our level-wise approach expands the flow paths in the network closely in coordination with the knowledge about the network structure. The k-th iteration of this approach generates an information flow path of length k, which satisfies both the network and frequency constraints. In order to achieve this goal, we maintain a linked list associated with each node in the network. At the end of the k-th iteration, this list essentially contains all the frequent flow paths of length k (i.e. containing k nodes) with at least frequency f, *which end at node* i. We denote this *frequent flow path list* at each node i at the end of iteration k by $\mathcal{F}_i^k$.

The algorithm proceeds as follows. In the first iteration, we generate all the singleton actor nodes, which have at least a frequency of f in the social stream $\mathcal{T}$. Thus, in this case, each node i contains a list $\mathcal{F}_i^1$ of either 0 or 1 element, depending upon whether or not that actor has a frequency of at least f in the social stream. In the next step, we examine all neighbors of a given node in order to generate all paths of length 2. Specifically, we concatenate all frequent neighbors of node i at the end of each path in $\mathcal{F}_i^1$, in order to generate *potential* 2*-length paths* denoted by $\mathcal{P}_i^2$, which have i as the *penultimate node*. From these candidate paths, we determine those for which the frequency is at least f. It is important that the frequent flow paths from $\mathcal{P}_i^2$ cannot be added to $\mathcal{F}_i^2$, since the node i is now the penultimate node for the corresponding flow path, where $\mathcal{F}_i^2$ may contain only paths in which i is the final node. This means that $\mathcal{F}_i^2$ is not a subset of $\mathcal{P}_i^2$, but a subset of the union of the potential lists at its neighbor nodes. Therefore, the frequent paths of $\mathcal{P}_i^2$ are added to the lists $\mathcal{F}_i^2$ corresponding to the final nodes in these paths. This process is repeated until iteration k for node i when there are no more potential paths in $\mathcal{P}_i^{k+1}$.

In order to speed up the process of flow mining, a pruning technique is used. It uses the fact that the frequent flow path satisfies antimonotonicity property, i.e. all k subsets of a $(k+1)$-frequent flow paths are also frequent. We further note that we only care about those k length path, which are *contiguous* in the network; therefore, we only have to check the k length path after dropping[1] the *first* element in a $(k+1)$-candidate. Therefore, we can prune any $(k+1)$-candidate in $\mathcal{P}_i^{k+1}$, if this path of length k is not a frequent flow path. Let us consider a path $Q \in \mathcal{P}_i^{k+1}$, which ends in node r. Then, we prune the flow path Q from the potential list

[1] The path after dropping the *last* element is already known to be in its corresponding frequent path list.

Algorithm *InFlowMine*(SocialStream: $\mathcal{T}$
frequency: f);
begin
for each node i
$\mathcal{F}_i^1 = \{\}$;
if node i has at least f posts in $\mathcal{T}$ **then** $\mathcal{F}_i^1 = \{i\}$;
end
$k = 1$;
while $\cup_i \mathcal{F}_i^k$ is not empty **do**
for each i generate $\mathcal{P}_i^{k+1}$ from $\mathcal{F}_i^k$ by by appending each
frequent neighbor node of i to the paths in $\mathcal{F}_i^k$, and keeping
only potential paths with all distinct nodes;
for each i prune any path from $\mathcal{F}_i^{k+1}$, for which at least
one of its path of length k is not in $\mathcal{F}_r^k$ (for some r);
for each i generate the path in $\mathcal{F}_i^{k+1}$ which
are valid frequent flow paths;
Reorganize the frequent potential paths $\cup_i \mathcal{P}_i^{k+1}$ based
on the last nodes of these paths, and generate the
corresponding path lists $\mathcal{F}_i^{k+1}$ for each node i;
$k = k + 1$;
end
return($\mathcal{S}(f) = \cup_k \cup_i \mathcal{F}_i^k$)
end

Figure 2.1: Algorithm for Information Flow Mining.

if any of its path of length k (and also ending with r), are not found in $\mathcal{F}_r^k$. The process is repeated iteratively to generate frequent flows of longer and longer lengths. The process finally terminates, when in a given iteration, the potential paths $\mathcal{P}_i^{k+1}$ are empty. All the frequent flow paths satisfying the network constraints are returned at the end of the algorithm. The discovered frequent flow paths is denoted by $\mathcal{S}(s) = \cup_k \cup_i \mathcal{F}_i^k$. The overall algorithm is illustrated in Figure 2.1.

Example: We will continue our running example from Section 2.2 in order to understand the algorithm $SequentialCascades$ listed in Figure 2.1. The inputs to the algorithm are the social stream $\mathcal{T}$ and support $s = 2$. In the beginning, all the users in social network are frequent cascades of size 1, and the corresponding rudimentary paths of length 1, starting at the different nodes are $L_P^1 = \{P\}, L_C^1 = \{C\}, L_K^1 = \{K\}, L_V^1 = \{V\}, L_N^1 = \{N\}$. Next, candidate sets of size 2 will be generated by appending different neighbors from the social network. The resulting candidate set $\mathcal{C}_i^2$ is listed below:

$$\mathcal{C}_P^2 = \{\{P, C\}, \{P, K\}, \{P, V\}, \{P, N\}\}$$
$$\mathcal{C}_C^2 = \{\{C, P\}, \{C, K\}\}$$
$$\mathcal{C}_K^2 = \{\{K, P\}, \{K, C\}\}$$
$$\mathcal{C}_V^2 = \{\{V, P\}, \{V, N\}\}$$
$$\mathcal{C}_N^2 = \{\{N, V\}, \{N, P\}\}$$

These candidates may be pruned by examining the sequences obtained by dropping the first element, though there is no further elimination in this step from $\mathcal{C}_i^2$. Next the supports of the candidates are counted, and those below the required level of 2 are eliminated. In this case, there is a drastic 50% reduction in candidates. The resulting 2-cascades $\mathcal{L}_i^2$ specific to different nodes are listed below.

$$\mathcal{L}_P^2 = \{\{K, P\}, \{C, P\}\}$$
$$\mathcal{L}_C^2 = \{\phi\}$$
$$\mathcal{L}_K^2 = \{\{C, K\}\}$$
$$\mathcal{L}_V^2 = \{\{P, V\}\}$$
$$\mathcal{L}_N^2 = \{\{P, N\}, \{V, N\}\}$$

This step can be continued in general from k-cascades to $(k + 1)$-candidates, and then further pruning and support counting.

Now, we will find candidate sets of size 3 using $\mathcal{L}_i^2$ by appending frequent neighbors, and more importantly they should not already exist in the list $\mathcal{L}_i^2$ previously. For example, in the list

$\{K, P\}$ we will not append K to the end of the list as it already exist in the list. After the first and second for loop, the candidate sets are as follows:

$$\mathcal{C}_K^3 = \{\{C, K, P\}\}$$
$$\mathcal{C}_V^3 = \{\{P, V, N\}\}$$

The set $\{C, K, P\}$ is eliminated during the support counting process as it has a support of only 1 message. We will describe efficient ways of counting the support for the candidate sequences later in this section. Now, the only remaining candidate is $\{P, V, N\}$ with support of 2, and it passes on the next round as $\mathcal{L}_N^3 = \{P, V, N\}$. When $k + 1 = 4$, there are no neighbors to append to $\mathcal{L}_N^3$ and the candidate set $\mathcal{C}_i^4 = \{\phi\}$ for all users i. Now the algorithm stops as there is no further candidates to process.

We can show that by continuing this process of counting, the final set of frequent cascades generated are $\mathcal{S}(s) = \{\{P\}, \{K\}, \{C\}, \{V\}, \{N\}, \{K, P\}, \{C, P\}, \{C, K\}, \{P, V\}, \{P, N\}, \{V, N\}, \{P, V, N\}\}$.

2.3.1 Online Re-organization of Social Stream Content via Hashing

The efficiency of the algorithm, listed in Figure 2.1, depends on efficiently keeping track of the frequency of the information flow paths. An important point to remember is that this process requires determination of content-influence behavior between different posts. As mentioned earlier, this influence is detected in terms of *content-token* propagation. In order to simplify the string matching and frequency tracking process, we use a hash table to track the sequences of actors for each distinct content-token in the stream. Thus, for each *distinct* content-token $U \in \mathcal{T}$, we map it into the hash table index $h(U)$ (typically with the use of a standard hashing technique such as a linear hash function with linear probing for collision resolution). As mentioned earlier, the content-token U may be a URL string, tweet message string, hash-tag, keyword, or another text content string depending upon the underlying scenario, and the particular content that we have pre-decided to track in the social stream. Then, for each distinct value of U in the social stream, we determine all actors j for which $F(j, U) > 0$. This is the set $P(U)$.

$$P(U) = \{j : j \in N, F(j, U) > 0\} \tag{2.3}$$

For each distinct value of the content U in the social stream, its hash table slot $h(U)$ points to a linked list of actors in $P(U)$. Furthermore, the actors in $P(U)$ are organized by order of their values of $F(j, U)$. Thus, the set $P(U)$ is converted into the *ordered list* $O(U)$, based on the first time of occurrence of the content U for a particular actor. The entire set of lists in the hash table is denoted by $\mathcal{O} = \cup_{\text{All distinct content } U} O(U)$. After this re-organization process is over, we have a set of ordered lists pointed to by the hash table, which contains the list of actors who have posted that content, in order of their (first) occurrence in the social stream. Because of the use of a hashing approach, this conversion can be performed efficiently, and in online fashion with the social stream. Specifically, for each incoming post of content U by actor j, we determine if the actor j occurs in the list pointed to by $h(U)$. If this is the case, then the list is not modified. Otherwise, the actor j is appended to the end of the list. Thus, at any point in the social stream, this data structure is always available for efficient tracking of flow path frequency and maintained in an online fashion.

As the number of valid potentials for the paths of length two is very high for some tokens, we implemented a specialized tracking method for this case to improve the efficiency further. The reason for this is that the some of the content-tokens U have a large number of actors who posted them, which corresponds to a large value of $|O(U)|$. On the other hand, for most of the tokens it was relatively small. So, depending upon the length of the list $O(U)$, we perform the frequency tracking in the following way:

- We maintain a global data structure containing counts of all pairs of actor sequences. If the actor sequence for the unique content is less than $\sqrt{|\cup_i \mathcal{P}_i^2|}$, we increment the count of all pairs of content-tokens, which occur in this list $O(U)$. This can be achieved by using a doubly nested loop on the ordered list, and this requires $O(|\cup_i \mathcal{P}_i^2|)$ time per list.

- For some lists which are very long, we explicitly check the membership of each flow path of length 2 in $\cup_i \mathcal{P}_i^2$ with respect each unique content U_k, in order to increment the support counts in the global data structure. This approach also requires $O(|\cup_i \mathcal{P}_i^2|)$ time per unique content token.

The reason for using this approach is to optimize the frequency tracking process for longer and shorter flow paths, so that the two approaches require the same amount of time at the threshold value, but are optimized within their respective list sizes.

2.3.2 Topic or Content Specific Flow Mining

An important point to remember is that the information flow mining problem can be made to focus on specific topics or content by calibrating *the kind of content which is tracked for flow paths*. This cannot be easily achieved by cascade and influence analysis techniques which do not directly use the content in the mining process. For example, let us consider the case where a cosmetics manufacturer wishes to determine the content flow patterns corresponding to the topic of *cosmetic products*. The first step is to create a vocabulary D of all content-tokens, such as popular hash-tags, URLs or other strings related to this topic. Then, the occurrence of these content-tokens in the network posts are treated as the strings T_i for content analysis. Specifically, for each post which contains one or more content-tokens from D, we treat each post as a sequence of strings from that user containing all the contained content-tokens from D in that post. We further note that this kind of filtering *greatly reduces* the amount of content (and the number of valid flow paths) which need to be tracked. This can greatly speed up the algorithm *InFlowMine* described in Figure 2.1, while simultaneously achieving the goal of topic-targeting.

2.4 Influence Mining Application

The information flow patterns provide useful insights about the influence of different entities in the network. In this section we describe the use of such information flows in finding the frequent epicenters or influencers of content propagation.

The influence of a node is generally characterized by the number of nodes that forward or propagate its content further in the network [29]. This is precisely captured by the set of all information flow paths discovered in the previous section. There are two main factors that affect the influence of a node in the context of a flow path: (1) the position of the node in a flow path and (2) the number of frequent flow paths in which it occurs. As the node consistently occurs earlier and in more flow paths it is considered as a promising influencer.

We now, more formally, define the problem of influence mining in the context of information flow paths. For any frequent flow path $P = \{i_1 \dots i_k\}$, discovered in the last section, any particular node i_r can influence all the actors, which occur after i_r in P (including itself). This corresponds to the set $\{i_r, i_{r+1} \dots i_k\}$. Correspondingly, we define, the influence set of a node-flow pair.

Definition 4 (Node-Flow Influence Set)**.** *The influence set $Q(i, P)$ for a node-flow pair (i, P) is defined as the set of all nodes in flow path P, which occur after the occurrence of i in P, when P contains i. In the event that node i is not contained in P, this set is empty.*

Intuitively the above definition is the set of all nodes which are *frequently* influenced by a given node i through the frequent flow path P. The definition above can be generalized to the case where we are using the entire set of flow paths derived at frequency f, rather than a single flow path P. First, we define the influence set of a given node i at the frequency f. This is essentially the union of all the node-flow path sets for i for all frequent flow paths P in $\mathcal{S}(f)$.

Definition 5 (Influence of a Node)**.** *The node influence $I(i, f)$ of node i at frequency f is defined to be the size of the union of all the node-flow influence sets for each frequent flow path $P \in \mathcal{S}(f)$. Therefore, we have:*

$$I(i, f) = | \cup_{P \in \mathcal{S}(f)} Q(i, P)| \tag{2.4}$$

Thus, the above definition determines all the nodes, which are influenced by node i by all frequent flow path in set $\mathcal{S}(f)$. This definition can of course be naturally generalized to a set of nodes, rather than a single node, by computing the union of the influence sets of different nodes.

Definition 6 (Influence of a Node Set)**.** *The influence $I(V; f)$ of a node set V given the frequency f is the size of the union of the influence sets of the nodes in V. Therefore, we have:*

$$I(V; f) = | \cup_{i \in V} \cup_{P \in \mathcal{S}(f)} Q(i, P)| \tag{2.5}$$

For notational simplicity, going forward, we denote $I(V; f)$ as $I(V)$ as f is given. Ideally, we would like to determine a node set V (of pre-defined size) in a network, for which the influence set $I(V; f)$ is as large as possible. Therefore, the influence maximization problem may be stated as follows:

Problem 2 (Influence Maximization)**.** *For a given frequency f, determine the node set V with at most k elements, for which $I(V; f)$ is maximum.*

The influence of a given set of nodes V is more likely to be maximized, when we pick nodes from different parts of the network, each of which has a large amount of influence in its locality.

Algorithm *NDIF*(Social Stream: $\mathcal{T}$, frequency f,
Number of Nodes: k);
begin
Determine all frequent flow paths at frequency f
from stream $\mathcal{S}$ (Section 3, Algorithm 2.1)
Construct influence sets $I(i; f)$ for each node i;
$V = \{\}$
while $(|V| \leq k)$ **do**
Add the node i to V, such that
$I(V \cup \{i\}; f) - I(V; f)$ is as large as possible;
end
return(V);
end

Figure 2.2: Influence Maximization using Information Flows.

2.4.1 Sub-modular Maximization

The problem of determining the largest influence set is closely related to that of sub-modular function maximization. A function $g(S)$ is defined as a monotonically non-decreasing sub-modular function over its argument (which is the set S), such that the additional gain from adding an element to the set S reduces by further adding elements to the set. In other words, for two sets V_1, V_2, such that $V_1 \supseteq V_2$. we have:

$$g(V_1) \geq g(V_2) \tag{2.6}$$

$$g(V_1 \cup \{i\}) - g(V_1) \leq g(V_2 \cup \{i\}) - g(V_2) \tag{2.7}$$

Let us now formally show that our influence function $I(V)$ is sub-modular. Each path $p_j \in \mathcal{S}(\{)$ is an ordered set of actors indexed from $(1 \ldots m_j)$, where m_j is the number of actors in p_j. For each path p_j we construct a corresponding non-influenced path r_j^V for some top-k influencer's set V. Then,

$$r_j^V = p_j \setminus \texttt{inf}\left((p_1, \ldots, p_{j-1}), V\right), \tag{2.8}$$

$$\texttt{inf}\left(\zeta = (p_1, \ldots, p_{j-1}), V\right) = \cup_{i \in V} \cup_{p \in \zeta} Q(i, p). \tag{2.9}$$

In other words, r_j^V in (2.8) is a set of the actors that are not yet influenced by actor set V in any of the previous paths from p_1 to p_{j-1} and $\texttt{inf}(...)$ denotes the set of influenced actors due to V in $(p_1, \ldots, p_{j-1})$. Given a path (i.e. an ordered set) $p = (x_1 x_2 \ldots x_k)$, $\gamma(x, p)$ denotes the index of x in p. If x is absent in p, $\gamma(i, p) = 0$. We denote the set of all non-influenced actors using $R^V = (r_1^V, \ldots, r_k^V)$, for the corresponding set of frequent information flows S. In order to show $I(V)$ is sub-modular, we need to establish (2.6) and (2.7).

Lemma 1. *The function $I(V)$ is monotonically increasing, i.e. $I(V \cup x) \geq I(V), x \in N$.*

Proof. Let $I(V \cup x) = I(V) + Y$. In order to compute the additional influence Y due to x, let us partition the set R^V in to R_x^V and R_{-x}^V, which denotes the non-influenced paths (after using set V) where x is present and absent in R respectively. For each path $r_j^V \in R_x^V$, the number of elements from the index of x in r_j^V to the end of the path, is the set of newly influenced nodes, denoted by $|r_j^V| - \gamma(x, r_j^V)$. If $R_x^V = \phi$ then equality holds $I(V \cup x) = I(V)$. When $R_x^V \neq \phi, x$ is present in one or more paths r_j^V for some j. Let q_j^V denotes the set of newly influenced nodes for each path r_j^V, $q_j^V = r_j^V(\gamma(x, r_j^V), \ldots, |r_j^V|)$. Then, $Y = \left|(\cup_{j=1}^k q_j^V) \setminus \texttt{inf}(S(f), V)\right|$. As $|q_j^V| \geq 0 \Leftrightarrow Y \geq 0$, $I(V \cup x) \geq I(V)$. □

Corollary 1. $I(V_1) \geq I(V_2), V_1 \supseteq V_2$.

Proof. This directly follows from the Lemma 1. With out loss of generality, let us assume $V_a = V_1 \setminus V_2 = \{x_1, x_2, \ldots, x_m\}$. Then from Lemma 1, $I(V_2) \leq I(V_2 \cup x_1) \leq I(V_2 \cup x_2) \leq \ldots \leq I(V_2 \cup V_a) = I(V_1)$. □

Theorem 1. *The influence function $I(V)$ is sub-modular.*

Proof. We can establish (2.6) from Corollary 1. It remains to show $I(V_1 \cup x) - I(V_1) \leq I(V_2 \cup x) - I(V_2), V_1 \supseteq V_2$ to establish $I(V)$ is sub-modular. Let $I(V_1 \cup x) - I(V_1) = \left|\cup_{j=1}^k q_j^{V_1}\right|$, where $q_j^{V_1} = r_j^V(\gamma(x, r_j^{V_1}), \ldots, |r_j^{V_1}|)$. Similarly, $I(V_2 \cup x) - I(V_2) = \left|\cup_{j=1}^k q_j^{V_2}\right|$. As,

$$\begin{aligned} \left(\cup_{j=1}^k r_j^{V_1}\right) &\subseteq \left(\cup_{j=1}^k r_j^{V_2}\right) \\ \Leftrightarrow \left(\cup_{j=1}^k q_j^{V_1}\right) &\subseteq \left(\cup_{j=1}^k q_j^{V_2}\right) \\ \Leftrightarrow \left|\cup_{j=1}^k q_j^{V_1}\right| &\leq \left|\cup_{j=1}^k q_j^{V_2}\right| \end{aligned} \tag{2.10}$$

Therefore, $I(V_1 \cup x) - I(V_1) \leq I(V_2 \cup x) - I(V_2), V_1 \supseteq V_2$ and hence $I(V)$ is sub-modular. □

The main idea behind sub-modularity, of influence function $I(V)$, is that the additional gain of adding an element to a larger influence set can be no greater than that of adding that element to its subset (i.e. diminishing returns as the set V gets larger). The sub-modularity immediately suggests that the greedy algorithm of starting with an empty set, and continually adding the element with the largest *incremental* addition on the function $I(V)$ provides an effective solution both in theory and practice [29]. The overall framework for influence maximization and the pseudo-code for the corresponding algorithm is illustrated in Figure 2.2. We refer to the algorithm as *NDIF*, which corresponds to Network Discovery of Influencers using Flows.

Based on the results for sub-modular function maximization, the greedy algorithm also yields a tight approximation bound [64] for influence maximization algorithm *NDIF*.

Theorem 2. *[64] The greedy algorithm for influence set maximization yields an approximation bound of* $(e-1)/e$*, where* e *is the base of the natural logarithm.*

Our approach is fairly easy to adopt for any specific goals of influence maximization. For instance, it is possible to target influence analysis in specific topics or areas of expertise, by focusing only on propagation of specific content-tokens. Similarly, to target specific actors L in the social network, by examining the size of $I(V; f) \cap L$ during the application of the influence maximization algorithm.

2.5 Experimental Results

In this section, we evaluate the efficiency of our approach, *InFlowMine*, in terms of its running time. Then we evaluate the effectiveness of the discovered information flow patterns by comparing the top-k influencers found by our *NDIF* algorithm versus other popular baselines. The results were tested on a 64-bit *Windows* machine, manufactured by Lenovo (model X220T). The machine was running *Windows OS 7*, an Intel i7-2620 2.7Ghz processor, 8GB RAM, and hard disk of 320GB.

2.5.1 Evaluation Measures

Now we describe the measures used for evaluating the flow mining algorithm and the influence mining application.

Evaluation for *InFlowMine*: The proposed *InFlowMine* algorithm addresses a new problem, and hence we have tested its run-time efficiency with respect to a closest baseline, a modified version of a sequence mining algorithm (as we treat each path as a sequence of actors). In order to use any sequence mining baseline, we need to first construct a transaction database D from the given social stream $\mathcal{T}$. For this purpose, we create a unique list of tokens appearing in $\mathcal{T}$ and for each of those keywords we construct an ordered set of actors who propagated these messages sorted by the time of propagation. We call this ordered set for each token as a transaction, denoted by D_i. Each transaction D_i contains only the actors first occurrence, inline with our previous assumption. Given this transaction database D, we can run sequence mining procedure to find all the sequences. However, the sequence mining algorithms do not adhere to the network structure. For this reason, we prune the sequence found by the sequence mining algorithm by applying the *valid flow path* constraint. Note that the absolute frequency used in our model is similar to the notion of support in the sequence mining setting. For consistency with the baseline, we will use the term support to refer the frequency of our approach going forward. The final outcome of the sequence mining approach after pruning is same as ours. Thus, our running time comparison is fair compared against the baseline.

Evaluation for *NDIF*: Now, we describe how we measured the effectiveness of our flow mining procedure using the *NDIF* algorithm. For this purpose we define a measure called *Influence Score*. We compute the *Influence Score* in terms of how the content in the token set R has spread, for a particular set of influencers K. This is achieved as follows. For each transaction D_i we first mark the seed influencers in K present in D_i as influenced. Then we find each of their neighbors who appear after them in ordered set D_i. We do this, because influence behavior is always assumed to be causal in nature and the ordering in D_i captures this causality. We proceed this iteratively until there are no more neighbors who can be influenced in the transaction D_i. Now, we count the number of influenced nodes for the transaction D_i. We do this for all transactions in D and average out the total influenced nodes per transaction and refer to it as *Influence Score*. This measure is quite meaningful as it captures the temporal ordering of influence, via the network structure and the use content tokens (as each row of D represents the temporal ordering of the actors for unique content token). The temporal ordering here has a certain notion of causality, where if b propagates a content token after a has started it, and if a is a friend of b (in the network), then it is highly likely that b's action was influenced by that of a.

2.5.2 Baselines

As our problem definition is new, we use the sequence mining approach as the closest possible baseline. As the sequence mining approaches require a transaction database, we first translated the social stream data $\mathcal{T}$ to transaction database D. Remember, the sequence mining approach *does not directly address our problem*. Hence, the output of sequence mining has to be further processed to satisfy the *flow path constraints* to construct all valid frequent flow paths. We consider one of the most popular sequence mining algorithm *PrefixSpan* [65] as our baseline. We show that our approach extremely efficient in terms of finding flow patterns compared to prefix-span. The efficiency of such an approach is particularly important, because in the case of larger networks, this can restrict our ability to determine important propagation patterns at low frequencies.

The choice of baseline algorithm for comparing against the influence analysis part is very straight-forward. We use the most popular and standard baselines used in several other recent influence analysis work [29, 32, 31, 35]. The first baseline is the *PMIA* algorithm discussed in [31]. This a Prefix excluded extension of the Maximum Influence Arborescence (MIA) model. The model takes a network structure with defined edge probabilities. We calculated the probabilities using the weighed cascade model proposed in [29]. Our next baseline algorithm is *DegreeDiscountIC*, which is the degree discount heuristic of [32] developed for the uniform IC model. The *SP1M* is the shortest path heuristics algorithm of [35] with lazy forward optimization defined in [30].

2.5.3 Data Sets

We have used a variety of data sets to illustrate the effectiveness and efficiency of our approach. The first data set is Twitter, the most popular and commonly used online social network. We use this data set to demonstrate the practical feasibility of our running our approach for social networks. Also, this data set illustrates the importance of using content and time, in an integrated fashion, for finding context specific influencers in social streams, unlike other approaches that only use an edge based probability and a propagation model. Our next data set is the DataBase List of Publications (DBLP). We use this data set to show several important case studies and techniques (such as flow-based ego nets) that we could not share otherwise in complete detail, in Twitter data set, due to privacy concerns. Also, the popularity of certain authors in the DBLP

data set, makes the case study interesting and intuitive. The third data set is US Patent Office (USPTO) inventions. We use this data set as another supporting evidence to our analysis. In addition to showing the effectiveness and efficiency, we highlight several interesting scenarios in this data set, such as the discovery of a Nobel prize winner as a leading influencers using our algorithm, while the US Patent office did not even list him as a prolific inventor at that time. We describe these data sets in detail below.

Twitter Data Set: We extracted the tweets from March 25th to May 1st 2014 (i.e. 37 days) and filtered the tweets containing the following set of hash tags (related to the *twitter blocked news* in turkey after the corruption allegations): "#turkey, #twitterblockedinturkey, #direntwitter, #twitterisblockedinturkey, #25martkorkusu, #twittericinsokagacikiyoruz, #youtubeblockedinturkey, #erdogan, #1mayis". The resulting data set contained 1,919,294 (≈ 1.9 million) tweets. Our data set contained the time stamp and the screen name of each posting. We extracted all the unique hashtags from these postings and treated them as the messages (i.e. U_i), and ordered the sender ids based on the time of posting to create the required input data. Our database contained 131,574 unique hashtags and 293,364 unique users. We constructed the network structure using the mention or retweet activities between users using the past 10 months of twitter data. The network constructed this way contained 1,958,776,515 (≈ 2 billion) edges and 120,775,144 (≈ 120 million) nodes[2] . This was the underlying network structure used for the algorithm.

DBLP Data Set: We downloaded the publicly available DBLP XML data set [3] , and extracted the month, year, title and author details for each of the published document. We cleaned the DBLP data set to remove documents with missing meta-information such as title, author or date. After cleaning, the DBLP data set had 1,008,883 distinct authors and 1,810,117 documents. The dates on the documents were used in order to create a social stream of content, where the keywords in the titles formed the content-tokens. Temporal ties between documents were resolved using their title string. This data set consisted of 198,760 distinct keywords. We also constructed the DBLP co-authorship network, which contained 1,008,883 nodes and 3,383,570 edges.

[2] Extracting the follower network to these number of users is not feasible, as the Twitter API's are highly rate-limited and hence we use the mentions and retweets to reconstruct the network.

[3] `http://dblp.uni-trier.de/xml/`

US Patent Data Set: The United States (US) patent database is publicly available for access from US Patent Office (USPTO)[4] . We downloaded the following set of attributes for all patents granted from June 21, 1977 to December 28, 1999: *Patent Number, Granted Date, Title, Inventors, Assignee, Legal Representative, and Application Number*. After cleaning the data set of documents containing missing meta-information, a total of 1,866,443 (1.9 Million) patents remained in our patent database. We used the patent titles to generate the content tokens. The co-authorship network for the US Patent database consisted of 1,358,116 nodes and 1,018,378 edges. The US patent database was extremely *sparse*, as compared to *DBLP*, and the average degree of a node was only 0.75.

2.5.4 Efficiency Results

As a baseline, we used a modification of the well-known *PrefixSpan* algorithm[5] for frequent sequence mining [65], in which we first find all frequent sequences (irrespective of network structure) at the required support level, and then remove those which do not satisfy the network validity constraints. This is possible only after the appropriate construction of the transaction database as described in Section 2.5.1. The results for *Twitter*, *DBLP* and the *US Patent Database* are illustrated in Figure 2.3. The support level is shown on the X-axis, and the running time on the Y-axis. The actual running times of the $PrefixSpan$ and $InFlowMine$ algorithms are also illustrated in Tables 2.1 and 2.2 respectively.

It is evident that the running times of our approach were significantly lower than the baseline method, and this advantage increased with lowered frequency. This can easily be seen in terms of the scalability with support level, in which our approach scaled linearly with the support, whereas the *PrefixSpan* method scaled exponentially. At the lower frequency levels the *InFlowMine* method is *faster by an order of magnitude compared to the baseline*. The advantage of our method is primarily a result of flow path validity pruning, while in the process of discovering the patterns. This can easily be seen from the columns $\mathcal{P}2$ and $\mathcal{P}3$ of Tables 2.1 and 2.2, which list the number of potential paths of lengths two and three before and after network pruning for each of the data sets. It is evident, that only a very small fraction of the potential paths survive flow path validity constraint, and this significantly contributes to the efficiency of our approach.

[4] `http://uspto.org/`

[5] `http://www.cs.uiuc.edu/~hanj/software/prefixspan.htm`

	Run Time in Secs.		W/O Net. Pruning			W/ Net. Pruning		
Support Level	**Prefix-Span**	**Our Algo.**	$\mathcal{P}1$	$\mathcal{P}2$	$\mathcal{P}3$	$\mathcal{P}1$	$\mathcal{P}2$	$\mathcal{P}3$
0.1	0.78	20	1008883	0	0	1008883	0	0
0.05	0.94	20	1008883	0	0	1008883	0	0
0.01	0.83	23	1008883	0	0	1008883	0	0
0.008	1.25	21	1008883	6	0	1008883	6	0
0.005	10.37	21	1008883	5112	0	1008883	920	0
0.003	52.76	22	1008883	226100	0	1008883	8144	0
0.0025	107.25	26	1008883	858402	0	1008883	15771	0
0.002	190.37	35	1008883	3734556	0	1008883	34181	0
0.0015	415.23	70	1008883	20926050	4060	1008883	82509	355
0.001	2373.38	447	1008883	153351072	630352	1008883	234368	22870
0.0008	5461.77	1123	1008883	365823002	4498694	1008883	361843	79410
0.0007	11580.93	1984	1008883	596751612	13001695	1008883	456795	147361

Table 2.1: Running Time Comparisons and Pruning Effectiveness for DBLP data set.

2.5.5 Case Study on Discovered Information Flow Patterns

We discuss some case studies of interesting flow patterns discovered by the *InFlowMine* algorithm. We present both the cases where we used the full set of content-tokens, as well as content-tokens specific to a particular areas. In order to study flows, which are specific to particular areas, we picked a small set of well known conferences from the *Data Mining*, and *Networking* area, and determined those tokens, which were present in those conferences at a very high frequency with respect to the remaining data set. Specifically, we used the ratio of the relative word occurrence in a particular field with respect to the entire data set, and used the top 1000 and 1200 words respectively as the content-tokens for the data mining and networking areas.

For the overall DBLP dataset (which was not context-specific), we sometimes obtained flow patterns which represented the content propagation influence due to organizational hierarchies. An examples of such a flow is *Hongjiang Zhang*, *Wei-Ying Ma*, and *Lei Zhang* in that order. This flow corresponds to a managing director, assistant managing director and lead researcher respectively within *Microsoft Research*. In many cases, the flow did not necessarily represent organizational hierarchies, but may represent geographical or topical influence. An example of such a flow is *Luca Benini*, *Alberto Macii*, *Enrico Macii*, and *Massimo Poncino*, who are authors from different universities in Italy.

	Running Time in Secs.		W/O Net. Pruning			W/ Net. Pruning		
Support Level	**Prefix-Span**	**Our Algo.**	$\mathcal{P}1$	$\mathcal{P}2$	$\mathcal{P}3$	$\mathcal{P}1$	$\mathcal{P}2$	$\mathcal{P}3$
0.1	0.498	17	1358116	0	0	1358116	0	0
0.05	0.558	17	1358116	0	0	1358116	0	0
0.01	0.883	17	1358116	2	0	1358116	1	0
0.008	0.905	17	1358116	2	0	1358116	1	0
0.005	1.515	18	1358116	72	0	1358116	10	0
0.003	5.217	18	1358116	7656	0	1358116	170	0
0.0025	9.292	19	1358116	46440	0	1358116	503	0
0.002	18.264	22	1358116	219492	0	1358116	1318	0
0.0015	42.11	24	1358116	1704330	2	1358116	4480	1
0.001	134.051	27	1358116	22264242	135	1358116	19860	34
0.0008	230.296	32	1358116	86871720	2955	1358116	42367	150
0.0007	357.809	41	1358116	173751942	11174	1358116	61027	489

Table 2.2: Running Time Comparisons and Pruning Effectiveness USPTO data set

When we used content-tokens from the *Data Mining* category, one of the discovered flow pattern was *Thomas S. Huang*, *Shuicheng Yan*, and *Lei Zhang*. This is because Thomas S. Huang has 42 publications in common with Shuicheng Yan and 12 publications in common between Shuicheng Yan and Lei Zhang. Similarly, for networking area, we obtained the frequent flow as *Donald F. Towsley*, *James F. Kurose*, and *Yong Liu*, where *Donald F. Towsley* and *James F. Kurose* had 136 publications in common. These flow patterns clearly illustrate the qualitative insights which can be gained with the use of the *InFlowMine* algorithm in capturing propagation of co-authorship influence in these networks.

2.5.6 Effectiveness Results

We also tested the effectiveness of InFlowMine algorithm with respect to the influencers found using the discovered information flows. We do this by evaluating the influence score measure for NDIF algorithm varying the number of influencers as shown in Figure 2.4. For DBLP data set we do not show the *SP1M* baseline as it never completed. In general, we note that all methods perform very similarly when the number of influencers is too small, because in these cases the problem becomes trivial. However, in the "interesting" range of values on the X-axis, our approach performed significantly better. This was true for all data sets, though the degree of difference is greater in *Twitter* compared to *DBLP* and *USPTO*. In the case of *DBLP*, with less

Overall Influencers	Data Mining Influencers	Networking Influencers
Thomas S. Huang	Thomas S. Huang	Bo Li
Wei Wang	Jiawei Han	Donald F. Towsley
Jie Yang	Wen Gao	Sajal K. Das
Wen Gao	Qiang Yang	Mario Gerla
Wei Liu	Michael I. Jordan	Jiannong Cao

Table 2.3: The table shows the overall and context-specific influencers for the DBLP data set

than 100 influencers our algorithm is able to propagate almost all the relevant content-tokens to the different nodes as possible. On the other hand, the best baselines such as *PMIA* and *DegreeDiscountIC* do not even come close at a seed of 100 influencers. In Twitter, the baselines perform poorly, as the edge propagation model fails to capture the *data-driven* content and temporal characteristics. Furthermore, the influence does not seem to necessarily propagate in the shortest paths. The shortest path based propagation model (SP1M) performs poorly, as compared to the other two baselines. The main difference between DBLP and Twitter data set is the *content concentration* at a user-node. Each user is more or less focused on the topic of discussion in DBLP, while they discuss arbitrary topics in Twitter. Hence, the use of content along with the network structure makes a significant difference in finding the influencers, which is not very well captured by the edge probabilities and the propagation model. This illustrates the power of using information flows, an integrated approach for combining content, time and network structure, to discover influencers.

2.5.7 Case-Study of NDIF Algorithm

In this section, we study the content-centric capabilities of the *NDIF* algorithm for both the *DBLP* and the *US Patent database*. We generated content-tokens from the titles of the papers and patents. For content-specific analysis we used the same set of content-tokens generated for the content-specific analysis of the InFlowMine algorithm.

For context-specific flow mining, the *InFlowMine* algorithm was applied with the *same network being used in all cases*, but with a different subset of content-tokens. This provides the flexibility, in that it is possible to perform context-specific mining only through the process of identification of context-specific tokens. After applying the *InFlowMine* algorithm, the *NDIF* algorithm was applied to the generated flow patterns in order to mine the influencers. The results

Overall Influencers	Data Mining Influencers	Networking Influencers
Dieter Freitag*	Colin Strutt	Claude Galand
Akira Suzuki	Istvan Bartho	Seitoku Kubo
Wilhelm Brandes*	H. Jim Fulford Jr.	David E. Rasmussen
Manfred H. Vock	Rex L. James	Ronald L. Mahany
Akira Takahashi*	Leopold Flohe'	Richard G. Bratt

Table 2.4: The table shows the overall and context-specific influencers for the USPTO data set.

for the *DBLP* and the *US Patent data set* are illustrated in Tables 2.3 and 2.4 respectively.

First, we examine the results for the *DBLP* data set. Table 2.3 has three columns corresponding to the overall, data mining and networking influencers. In the overall column, the most prolific and sociable authors in different fields were reported. We note that the top influencers from the data mining field are quite different from the overall results. In the data mining area, authors such as *Thomas S. Huang* and *Jiawei Han* are at the top of the list. This seems to be quite a logical choice. On the other hand, in the field of networking, authors such as *Bo Li* (a multi-author case corresponding to 28 authors in the field of networking), and *Donald Towsley* showed up at the top of the list. It should be noted that authors such as *Bo Li*, (which actually corresponds to many authors), also appeared because we did not perform entity disambiguation on the data set. From an algorithmic perspective, it is correct to report such "authors" as influential, because the combined authorship is extremely prolific, and also very "sociable" because of the diverse connections resulting from multiple authors. It is also evident from Table 2.3, that many of the top authors are influential figures in their respective fields of research. Thus, our approach is able to effectively find influential authors in different topics.

The results from the *US Patent Database* are also illustrated in Table 2.4. We note that the US Patent Office publishes a list of its prolific inventors [6] . We found that there were a number of intersections between the list of prolific inventors and the most influential authors. We have marked these authors with an asterisk in the table. We note that prolific authors need not always be influential and vice-versa. However, our approach was able to determine such influential authors quite effectively. For example, the author *Akira Suzuki*, who is listed among the overall influencers in the table is not a prolific author according to the *US Patent Office*. On the other hand, *Akira Suzuki* is a Nobel laureate in chemistry from 2010, who first published the *Suzuki*

[6] `http://www.uspto.gov/web/offices/ac/ido/oeip/taf/inv_prol.pdf`

Reaction. Thus, our approach was able to discover the influential authors, even when they were not prolific during the time for which the data set was extracted.

The results for the content-centric analysis were obtained using the same tokens, as used for the *DBLP* data set. One salient observation is that the influencers in the patent database are very different from the *DBLP* data. This is essentially because most of the authors were from private companies who often did not publish their results in the open literature. Nevertheless, the approach was still able to find some very interesting content-centric influencers from the data. For example, *Ponani Gopalakrishnan* made significant contributions to field of speech recognition and conversational interfaces and was also a *Master Inventor* within IBM for his significant patents. This suggests that the approach is able to determine significant influencers both within the *DBLP* and *US Patent* data sets.

2.5.8 Efficiency of NDIF Algorithm

Our earlier analysis showed that the *InFlowMine* algorithm is extremely efficient in computing information flow patterns compared to the well established *PrefixSpan* algorithm. Next, we analyze the efficiency of the *NDIF* algorithm in comparison with the other baselines. The running time of the *NDIF* algorithm also includes the running time of the *InFlowMine* algorithm.

The results for the different methods are illustrated in Table 2.5. It is evident that some of the methods are so slow, that they can sometimes become hard to use, especially if multiple runs are needed at different times in the social stream scenario. For example, the *PMIA* algorithm was slower by a factor of 10 compared to our *NDIF* algorithm. The *SP1M* algorithm was much slower and never completed on *DBLP* data set and continued running for more than a day. Hence we report > 86400 seconds for SP1M for *DBLP* dataset. In addition to being slower, our results in the previous subsection showed that these algorithms also performed poorly in quantitative influence measures. On the other hand, the running time of *NDIF* was quite modest, and it can easily be applied at any point in the social stream, as long as the online hash tables described are continuously maintained. While *NDIF* is not quite as fast as *DegreeDiscountIC* or *PageRank*, the latter methods performed too poorly to be practical in these scenarios. Therefore, the *NDIF* algorithm provides the best tradeoff between effectiveness and efficiency, and unprecedented flexibility in terms of content-centric analysis.

Method name	**DBLP Dataset** (in secs.)	**Patent Dataset** (in secs.)	**Twitter Dataset** (in secs.)
NDIF	472.20	49.11	12.48
PMIA	1965.59	331.12	620.64
DegreeDiscountIC	21.81	4.31	6.33
SP1M	>86400	34027.20	16540.40

Table 2.5: Running time comparison of NDIF with other baselines.

2.6 Flow-based Egonets

An *ego network* consists of a focal node (called "Ego"), the immediate neighbors of the focal node (called "Alter"), and their relationships. Ego networks are used to study social relationships for several years in sociology. In similar lines, we propose flow-based ego networks, which are effective in studying the implicit campaigning structure used for *information dissemination* and *information sharing*. A flow-based ego network consists of a focal-node ("Ego") and all the nodes that carry-forward the information originating from the focal node ("Propagandist"). A flow-based ego network is directed in nature and the direction of the edge denotes the direction of influence of information in (any) underlying relationship network. The relationship could be a co-authorship relationship in DBLP or a mention relationship in Twitter.

The flow-based ego networks can be constructed by extracting the information flow patterns originating at a particular user (i.e. the focal node). Then overlay these flow patterns one over the other to construct a directed sub-graph of the original network. This sub-graph (and not a necessarily directed acyclic graph) shows the information sharing and campaigning behavior of this particular user (focal node). There are two key differences between the focal node in a regular ego-network and a flow-based ego-network: (1) all immediate neighbors of the focal node *do not necessarily* appear in the flow-based ego network, and (2) the minimum number of hops to reach an propagandist (i.e. an alter) can be more than one in flow-based ego network, while it is exactly one in the other case. These differences make the flow-based ego networks an interesting tool to analyze long-range, and limited-neighborhood communications around the focal node.

Let us illustrate this concept, using a flow-based ego-network constructed for some popular scientists, for different research areas in the DBLP data set. For the field of data mining we extracted the ego network of "Jiawei Han" and for networking domain we extracted the ego-network for "Sajal K Das". The flow-based ego networks for these focal nodes are shown in Figure 2.5. The node sizes are proportional to the PageRank centrality of the node in the network. We also extracted the communities by using a modularity cut algorithm and the community of each node is highlighted by its color.

We make several important observations that are found consistently across leading authors in different fields. All focal nodes have a few propagandists in their ego-network. These propagandists are in-turn leaders in their own sub-communities. For instance, consider Figure 2.5, there are a handful of main (prominent) propagandists for both (a) and (b). In Figure 2.5(a), the most prominent (main) propagandists are Jian Pei, Xifeng Yan, Jeffery Xu Yu, Anthony K. H. Tung, and Haixun Wang. Similarly in Figure 2.5(b), they are Jiannong Cao, Minyi Guo, and Azzedine Boukerche. The degree of influence by these propagandists and the number of such (main) propagandists vary between the ego networks. Moreover, each of these main propagandists have their own communities, where they are influential leaders. For instance, the community influenced by Anthony K. H. Tung is highlighted in yellow color in Figure2.5(a). One can use these flow-based ego networks to understand the proportion of direct and indirect influence. Consider Figure 2.5(a), 32.44% of nodes receive information directly from Jiawei Han, 14.5% of the nodes from Jian Pei, 14.12% from Jeffrey Xu Yu, 10.31% from Haixun Wang, 7.25% from Xifeng Yan. In summary, 78.62% (i.e. around 80%) of nodes receive information from five central nodes in this network.

In summary, flow-based ego network provides an unprecedented view of the information dissemination in ego-networks, using the notion of propagandists and their influential communities. Understanding such information can help focal nodes to improve specific areas of collaboration and target more on influential and productive collaborations.

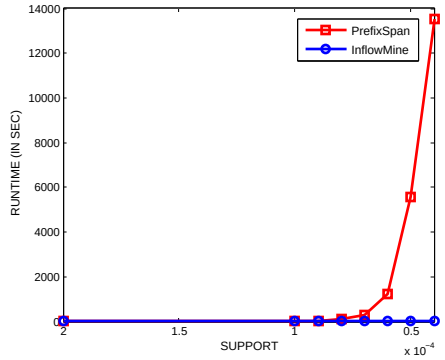
2.6.1 Centrality of Nodes in Flow

The discovered flow structures, in general, select authors with high centrality as leaders and lower centrality as their followers. We compute the average pagerank values for authors in first six positions in DBLP and USPTO network. In DBLP, the centrality scores decrease gradually

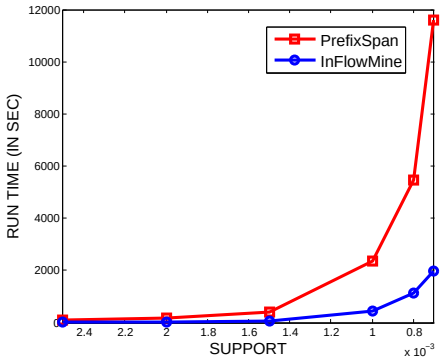
from (1) 1.521e-05, (2) 1.349e-05, (3) 1.271e-05, (4) 0.880e-05, (5) 0.359e-05, and (6) 0.252e-05. The position of nodes in the flow are mentioned in the corresponding brackets. Similarly, for USPTO, (1) 4.401e-06, (2) 3.630e-06, (3) 2.290e-06, (4) 1.611e-06, and (5) 1.546e-06. This clearly shows on average the network centrality of authors in a flow path reduces gradually from high to low. For USPTO, the maximum length of flow identified was five and hence we measured average pagerank until fifth position.

2.7 Conclusions and Summary

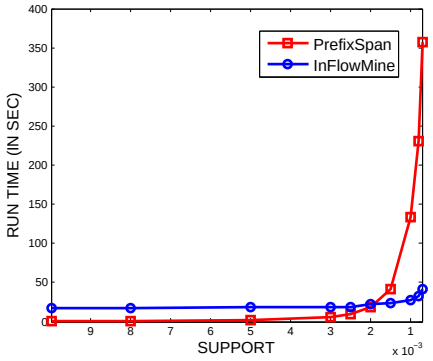
The availability of increasingly larger amounts of content in online networks makes it possible to perform information flow pattern discovery than ever before. In this paper, we introduced the information flow mining problem based on the propagation of content in the network structure. Then, we have proposed an approach to mine the flow patterns, following specific flow validity constraints. Our experimental results demonstrate that our proposed flow mining approach is extremely efficient and is well suited for several application areas, such as influence mining. We showed the effectiveness of the discovered patterns in an influence mining application. In addition, we showed how information flows can be used in studying information dissemination behavior of users, using the concept of flow-based ego-networks. Moreover, we discussed several case studies in publicly available data sets (such as DBLP) to demonstrate the effectiveness of the discovered information flows.



(a) Twitter data set



(b) DBLP data set



(c) USPTO data set

Figure 2.3: A comparison showing the running time of PrefixSpan versus our approach *InFlowMine* for Twitter, DBLP and USPTO data sets. The PrefixSpan run times are increasing exponentially as the support level is reducing, while approach is consistent at a wide range of support levels.

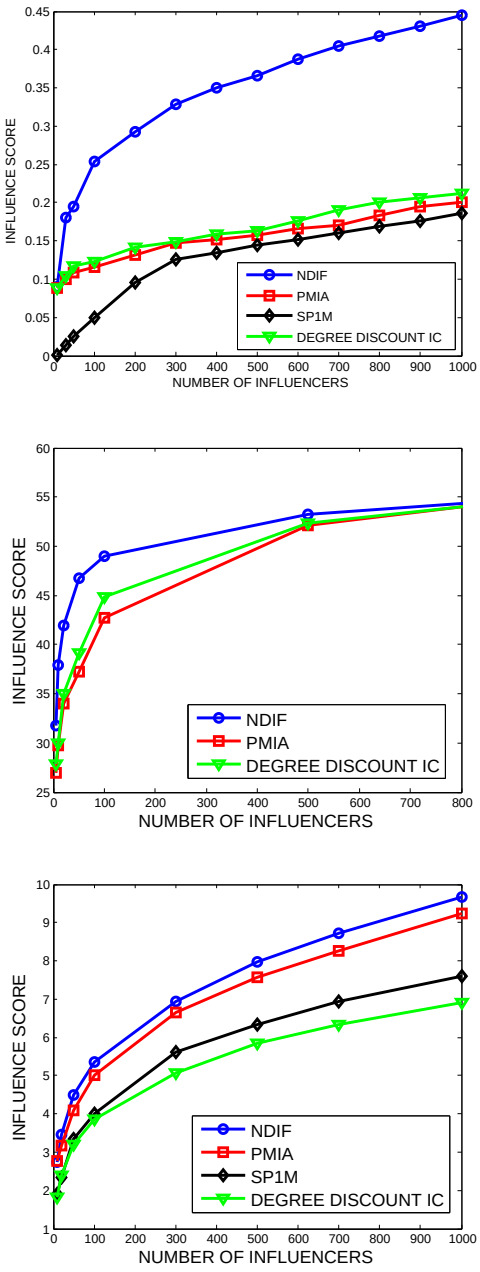
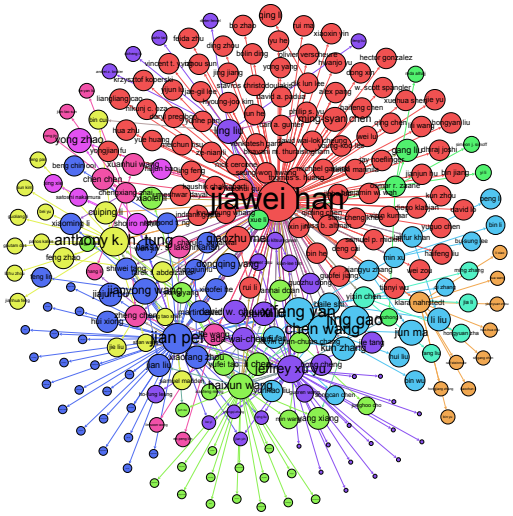
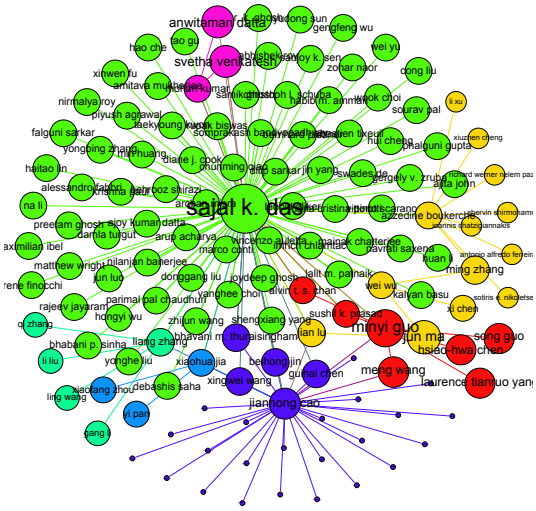


Figure 2.4: The plot shows efficiency of our proposed method for Twitter, DBLP, USPTO networks in capturing the maximum influence score for various number of (top-k) influencers. In social networks, such as Twitter, our approach is significantly better than baselines. While in collaboration networks such as DBLP co-authorship graphs, our approach achieves maximum influence much more quickly than the baselines.



(a) Flow-based ego network for Jiawei Han



(b) Flow-based ego network for Sajal K. Das

Figure 2.5: Flow-based ego-network shown for two focal nodes, Jiawei Han from data mining area (left) and Sajal K. Das from networking area (right). The node size and labels are proportional to the (pagerank) centrality of the node in this network. The node color denotes the community and community structure was identified using modularity-cut algorithm. Each ego-network has one focal node and a handful of alters that acts as main propagandist in the ego network. Each of the propagandist are in-turn leaders in their respective communities, and have their own set followers. The community structure along with the centrality score highlights the hierarchical relationship in the ego-network. Note that the figure can be zoomed in for clarity using any PDF viewer (to read the individual names more clearly).

Chapter 3

Distributed Network-centric Analysis of Flows

3.1 Introduction

The problem of finding dominant content flow trends in networks is an important problem in the context of online social and collaboration networks. In social networks, such as *Twitter* and *Facebook*, every user posts messages, photos and comments to exchange information with their neighbors in the network. The daily volume of content propagation in these networks is in the order of hundreds of millions of posts per day[1] . These posts typically propagate as short phrases [15], topics [34], *hashtags* [39], or *URLs* [24] in specific patterns on the underlying friend or follower network. Some of these posts may go *viral* and reach millions of users within a few hours. The massive reach of these flows may result in significant influence in online user behavior [11]. Therefore, it is desirable to understand such viral information flows for online marketing, advertisement and a variety of other applications. There are several recent works that attempt to understand these viral information flows in terms of memes [8, 15], cascades [14], and events [39].

The existing literature on understanding information flows using cascades [8, 15, 14] and memes [8, 15] analyze a stream or a corpus of text documents where there is no explicit network structure used for communication. For instance, the work in [14] determines the cascade patterns from the temporal sequence of blog posts across multiple blogs. Here, the term cascade,

[1] http://www.digitalbuzzblog.com/facebook-statistics-facts-figures-for-2010/

refers to a phenomenon in which a topic is adopted by a blog and further propagated through the creation of hyperlinks. Note that there is no underlying network structure between the bloggers. On the other hand, online social networks use an explicit network structure, such as follower or friend network, to propagate the information. Therefore, understanding the patterns of propagation in a network structure is likely to yield superior insights than observing patterns from general population. In several other related works, the flow of information is typically analyzed in the absence of the underlying network structure [13, 66].

Another disadvantage of existing approaches [24] is that they ignore the life-span of influence due to information flows, which is very relevant in the social context. For example, a message posted on a *Facebook* wall may not even be available on the first page after a day elapses. If a receiver of that message re-posts the same message after a week, then it is less likely that the user was influenced by the original post sent to his wall. As the life span is not considered, the existing methods produce a large number of cascades as opposed to more active and meaningful ones.

Most of the existing approaches for mining information flows [13, 66, 8, 15, 14] cannot handle large amounts of data, as their processing is centralized in a single server. We propose a distributed approach using vertex-centric computational models [67]. In these models, each vertex is a separate computational unit (available in a core or a machine), and it result in a high level of parallelism. As we show in our experiments, our approach is *three orders of magnitude* faster than existing state-of-the-art approaches. To the best of our knowledge, our approach is the first work in this area and we are not aware of any distributed or parallel information flow mining algorithms.

In this paper, we propose an efficient information FLOWExtractorR algorithm, called *FLOWER*, to discover these information flow patterns. We establish the complexity class of this problem, by showing the counting problem of all maximal information flow patterns is #P-complete. In order to scale up to large networks, we propose a parallel version called *pFLOWER* that runs on vertex-centric graph computational models [67]. In the experimental section, we show that our parallel method *pFLOWER* is faster than the state-of-the art algorithms by up to *three orders of magnitude*. We also study the effectiveness of the discovered information flows in the context of an influence analysis application. Our approach consistently outperforms the popular baselines in terms of precision, recall and F_1 measure.

The paper is organized as follows. The remainder of this section discusses related work.

In the next section, we introduce the preliminaries for the problem of flow mining in networks. Then we describe the flow mining algorithm and propose a parallel version using a vertex-centric computation model. Finally, we demonstrate the efficiency and effectiveness of our approach using multiple real-life data sets.

3.1.1 Related Work

The problem of analyzing information flow has been studied using content influence cascades [15, 14, 34]. Most of these work analyze the blogosphere, and there is *no explicit network* structure over which users exchange information in the blogosphere. However, in general, for social [39] and biological networks [68], there is an explicit network over which the flow of information occurs. Using this structure is important, because it enhances the discoverability of the relevant information flow patterns. In addition, information cascades have a limited life span, and therefore the use of the information life-time of the cascade helps in finding more active cascades caused by intrinsic social network influence rather than external sources [69]. In some recent works [70, 71] an instance of the Independent-Cascade (IC) model is built using the log of past propagation in the network. The aim of these techniques are to sparsify the network for scalability, while our intent is to extract the dominant information flow patterns. There are several other works that use only a network structure to compute social centrality of the users [57].

Content propagation in online media is usually tracked as short and distinct phrases, referred to as memes [15, 34]. Memes tend to have a broader stable vocabulary, that mimic a slowly evolving genetic signature over time. The key idea of this work [15] is to understand how the short phrases evolve over time while the several words of the phrase are intact during the entire period of propagation. However, there is no notion of tracking content flows in a network, while it does track how content evolves over time. There are several other papers that tend to capture such bursty topic behavior over time, based on different notions of topic identification [72]. A more detailed survey of evolution of content in network structures can be found in [58].

In a recent experimental study [8], the diffusion of stories in social networks, such as *Twitter* and *Digg*, are analyzed using the evolution of the number of fan votes in general population and in a network structure. More specifically, this work confirms the importance of using network structure in such studies. There are other recent works that study the distribution of URL cascades [24] in *Twitter* and propose a prediction model to predict the number of mentions of an URL after its posting. However, none of these related work track the dominant content flow

information in a network structure over time.

3.2 Information Flow Mining Model

We define the information flow mining problem and related information flow properties in this section.

Let $G = (V, E)$ be the relationship network containing the node set V and the edge set E. Each actor $a_i \in V$ performs a number of *content-based actions* such as sending tweets or posting wall posts. We denote a content posted as U_j, where j is index of the message, and the time of its posting as t_k. The time points are ordered using their index $k = 1 \ldots T$, such that $t_k < t_{k+1}$. A message U_j can be propagated by different nodes at the same time in different parts of the network, and not all nodes may necessarily propagate all messages. Also, all nodes need not propagate a message at every time point. Consider Figure 3.1, where an example network G and a table of different messages propagated are shown. The node C does not propagate message U_4, and none of the nodes propagate a message at time point t_4.

Definition 7 (Flow Path)**.** *A flow path s in the network $G = (V, E)$ is a sequence of distinct actors $\langle a_1, \ldots, a_k \rangle$, where each actor $a_i \in V$ propagated the same content U_j at least once.*

Each information flow path s is a sequence of distinct *actors* $a_1, \ldots, a_k$, who are involved in multiple content interactions over a period of observation. Note that there are no cycles in a single flow path as it contains a distinct set of actors. The purpose of our approach is to determine such frequent information flow paths, where certain content flows may occur frequently in specific paths of the network. In this paper, we assume that flow patterns are sequential paths in the network, and a general cascade can be constructed by overlaying multiple such sequential paths. In order to distinguish the interesting flow patterns, we define several flow properties. These flow properties ensure the interestingness in terms of *network structure*, *causality*, *frequency* and *life-time*.

Property 1 (Network Structure)**.** *A flow path $s = \langle a_1, \ldots, a_k \rangle$ satisfies the network structure property in the network $G = (V, E)$, if for each $r \in \{1 \ldots k-1\}$, an edge exists between a_r and a_{r+1} in E.*

By Property 1, we consider only the information flows that adhere to the network structure, which also has the effect of focussing on relevant patterns. As in social networks, similar nodes

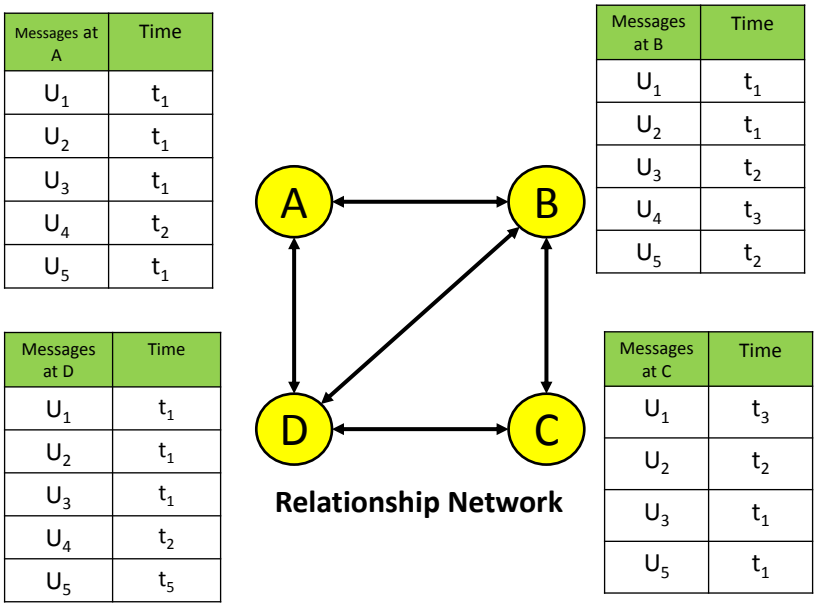


Figure 3.1: An illustrative example showing different messages $U_1, \ldots, U_5$ propagated at different times $t_1, \ldots, t_5$ from different actors (A, B, C, and D) in a small example network.

are related to each other by a neighboring relationship, such as a friend or a follower, and the content recommendation in a social network is often guided by such relationships.

Property 2 (Causality)**.** *A flow path $s = \langle a_1, \ldots, a_k \rangle$ satisfies the causality property in the network $G = (V, E)$, if the actors in the flow path propagate a message U_j at time points $t_1 \ldots t_k$ where $t_m < t_{m+1}, \forall m = 1 \ldots k-1$.*

The causality[2] property defines the interestingness of a flow pattern in terms of the time of propagation. An actor a_i broadcasts a message at time t_k and the neighbor a_j broadcasts the same message at time t_{k+1}, then a valid flow must only consider the path from a_i to a_j and not the other way around, as a_i may have caused a_j to transmit the message.

Property 3 (Frequency)**.** *A flow path $s = \langle a_1, \ldots, a_k \rangle$ satisfies the frequency property in the network $G = (V, E)$, if at least f distinct messages $U_1 \ldots U_f$ are propagated over the flow path s.*

A sequence of actors who share a neighbor relationship in the network and post a particular set of content-tokens in a temporal order is indicative of a signal of influence along the flow path. This influence signal is especially strong when actors behave in a similar way multiple times over many (possibly different) pieces of content. The frequency property captures the strength of such influences in a flow path through a pre-specified frequency level. The frequency parameter f is the count of the number of such repeated flows.

Property 4 (Life-time)**.** *A flow path $s = \langle a_1, \ldots, a_k \rangle$ has a* valid life-time period *τ, if the message propagated U_j at time points $t_1 \ldots t_k$ is such that $t_i - t_{i-1} \leq \tau, i = 2 \ldots k$.*

A post on a *Facebook* wall is not available forever for further propagation due to new incoming posts [8], or due to limited user attention span [73]. The notion of life-time is designed to model such real-life situations. All the aforementioned definitions can be generalized to multiple messages.

The problem of information flow mining is to extract all valid flow paths s that satisfy these flow properties. In the following problem definition, we denote the set of all messages U_j and the corresponding time stamps t_j sent by each actor a_i as T_i. The size of T_i is denoted as m_i. For example, in Figure 3.1, message table T_C of actor C contains four messages and time-stamp pairs: $T_C = \{(U_1, t_3), (U_2, t_2), (U_3, t_1), (U_5, t_1)\}$.

[2] The notion of "causality" in this paper is only based on temporal ordering, and no explicit mechanisms of cause and effect are assumed. Clearly, such temporal orderings might also occur by chance.

Problem 3 (Information Flow Mining)**.** *Given a graph $G = (V, E)$, a set of m_i messages propagated by each actor a_i, and their corresponding time-stamps denoted by $T_i = \{(U_j, t_j)_{j=1...m_i}\}$, the problem of information flow mining is to extract the set of all valid flow paths $F = \{s_1, \ldots, s_p\}$ that satisfy the network, causality, frequency, and life-time properties (Properties 1-4).*

Our approach provides a generic framework to analyze information flows in a variety of domains such as social networks, fMRI brain networks, or Internet networks. The messages propagated in these networks correspond to user posts, molecular interactions or data packets, respectively. With appropriate functions to compare and track similar signals across multiple nodes, our approach can be easily generalized to other domains [68]. However, in this paper, we restrict our attention to propagation of textual content as information signals over discrete time points.

3.3 Information Flow Mining Algorithm

A major challenge in information flow mining is that of incorporating the impact of network structure directly into the flow mining process. The key issue here is that a set of users who propagate the same message at approximately the same time period provide us with very little knowledge about their actual path through the network. Typically, when a message is popular, it might be independently propagated by users in many different regions of the network. Therefore, how does one "connect up" the propagation of the different users over the entire network? It is here that the linkage information between the users comes in handy. A message is assumed to have been propagated from one user to the other, only if two neighboring users propagate the same message within the life-time constraint. Therefore, an efficient algorithm for mining the information flow patterns needs to integrate the sequences of user posts, the network structure, and the temporal aspects in a holistic way, to extract the relevant flow patterns over the network structure.

One way of mining the patterns is to extract the flow paths in a *content-centric* fashion. Consider a message U_j sent by a set of actors $a_1, a_2, ..., a_k$. These actors might have sent these messages at different time points. One can order the actors in increasing temporal order and extract all subsequences of actors that appear in at least f such messages. We can then eliminate all flow paths that do not have a valid edge in the network. In this approach, we first use the content to create the flow paths and finally we apply the network validity property. The main

disadvantage of this approach is that the number of possible subsequences of actors in general population is very large. Therefore, it is prudent to use the network structure to eliminate such unnecessary candidates directly *during* the mining process. This idea is the key ingredient of the *network-centric approach*. In this approach, the message table T_i for the actor a_i is sent to each of its neighbor a_j iteratively. Each neighbor checks for validity and lifetime constraints by comparing the table T_i and T_j. If there are at least f messages that survive after the validation, the message is sent to the neighbors of a_j and so on. The advantage of this approach is that the sparsity of the network reduces the number of candidate flow paths dramatically [26]. An added advantage of the network-centric approach is that it can be easily parallelized using vertex-centric computational models. This aspect will be addressed in Section 3.4.

The pseudocode for our algorithm is shown in Algorithm 1. We refer to our approach as *FLOWER*, which stands for FLOW ExtractOR. The algorithm first extract the actors that have at least f distinct messages in their respective message tables (lines 2-4). Then, the information flow paths originating at each actor a_i are extracted by calling a recursive procedure *FLOWPROP*. This procedure extracts the messages that support causality and life-time property, compared to the incoming message table T_{seq}. If the number of messages in new supporting message table is at least f, then all its neighbors are iteratively explored (lines 4-5). If there are no neighbors for the actor a_i, the message is added to set F. This *if* condition ensures that the flow paths added to F are maximal in nature; in other words, a flow path is added only when none of its sub-sequence flow paths are already in F. One can ignore lines 6 and 7 in FLOWPROP to extract all paths that are not only maximal.

3.3.1 Complexity Analysis

In this section, we show that the counting version of the information flow mining problem is #P-complete. For this purpose, we reduce the maximal frequent sequence mining problem that is #P-complete [74], in polynomial time to the problem of mining information flow patterns (Problem 3). The notion of maximal information flow patterns is to retain only the longest frequent flow paths in the set F of Algorithm 1, beyond which the flow path does not satisfy one of the frequency, network, or lifetime constraints.

Theorem 3. *[74] Let $\mathcal{D}$ be a database of sequences with m transactions. The problem of*

Algorithm 1: ***FLOWER***

Input: $G = (V, E)$: Relationship network; T_i: message table for actor a_i; f: frequency; and τ: life-time;

Output: F: Set of flows satisfying all the properties

```
1 Initialize V' and S to empty set
2 for each a_i ∈ V do
3     if number of distinct messages in T_i ≥ f then
4         Add a_i to V' and F
5 for each a_i ∈ V' do
6     FLOWPROP({φ}, T_i, a_i, T_i, τ, f, V', F)
7 return F;
```

counting the number of maximal f-frequent subsequences in $\mathcal{D}$, where $1 \leq f \leq m$ is #P-complete.

Let T_i be the message table of the actor a_i in network $G = (V, E)$. We define a function Q that converts the database $\mathcal{D}$ to individual user message-tables $T_1, \ldots, T_{|V|}$.

Definition 8 (Function Q)**.** *Let $Q : \mathcal{D} \rightarrow \{T_1, \ldots, T_n\}$, where T_i is the message table of actor a_i. The ith transaction maps to a unique message id U_i. For each item k in the ith transaction of $\mathcal{D}$, a corresponding set $(U_i, \gamma(k, i))$ is added to T_k by Q, where $\gamma(k, i)$ denotes the first occurrence of actor k in the ith transaction of $\mathcal{D}$.*

The running time of function Q for a database D with m transactions and n items is $O(mn)$. Let $S_{\mathcal{D}}(f)$ be the set of maximal frequent sequences for support f for database $\mathcal{D}$. Let $F(f, G, \tau)$ be the set of all maximal frequent flows discovered for the information flow mining problem, as described in Problem 3, for support f, graph G and lifetime τ.

Lemma 2. *All maximal frequent sequences of set $S_{\mathcal{D}}(f)$ are present in $F(f, G, \tau)$, when G is complete and $\tau = 0$.*

Proof: Consider a path P that is a valid maximal frequent flow for $\tau = 0$ and the underlying graph G is complete. When $\tau = 0$ all valid paths in the graph G satisfy the lifetime constraint, because all lifetimes are non-negative. Also, given a complete graph any permutation of n nodes in the graph is a valid path. As the path P is frequent, there are at least f messages flowing

Algorithm 2: FLOWPROP

Input: seq: current flow path; T_{seq}: message table supporting the current flow path; a_i: current actor; T_i: message table for actor a_i; τ: life-time; f: frequency; V': frequent actors set; F: frequent flows discovered;

```
T_new = Extract messages that satisfy causality and life-time property from T_i and T_seq
if number of messages in T_new ≥ f then
    Γ_i = Get neighbors of a_i not in current flow path seq and in V'
    for each a_j ∈ Γ_i do
        FLOWPROP(⟨seq ∪ {a_i}⟩, T_new, a_j, T_j, τ, f, V', F)
    if Γ_i is empty then
        Add ⟨seq, {a_i}⟩ to F
```

along path P. Hence, the actors in that path must be appearing in that order in at least f such transactions in database $\mathcal{D}$, as per the function Q. Thus, the path P must be a frequent sequence in the database $\mathcal{D}$. So every f-frequent flow path P that is a valid for $\tau = 0$ and for a complete graph G is present in $S_{\mathcal{D}}(f)$. Because the path P is maximal, there cannot exist a longer path in set F that contains some actor a_r after P. If this is the case, per function Q, there cannot also exist a minimum of f transactions where P followed a_r is present. ■

From Lemma 2, it is evident that set $F(f, G, \tau)$ can be extracted from set $S_{\mathcal{D}}(f)$ by pruning the frequent sequences with lifetime lower than τ. For sparser graphs, several paths are invalid and hence several sequences are removed. Because the pruning process results in a much smaller set F compared to the original set $S_{\mathcal{D}}$, the complexity of mining maximal frequent sequences acts as an upper bound on mining sets of maximal frequent information flows.

Theorem 4. *The problem of counting all maximal flow patterns in the information flow mining problem is #P-complete.*

Proof: The maximal sequence mining problem can be reduced to an equivalent maximal information flow mining problem in two steps: (a) converting the database $\mathcal{D}$ using function Q (see Definition 8) into actor level message tables and (b) create a complete graph G. The computational complexity of function Q is $O(mn)$ and the complete graph creation is polynomial in n and the total time required is $O(mn + n^2)$. When $n >> m$, the total complexity is polynomial in n and when $m >> n$ it is polynomial in m. In either case, the maximal frequent

sequence mining problem can be reduced to maximal information flow mining in polynomial time in the size of the sequence database $\mathcal{D}$ and hence it is #P-complete. ■

3.4 Accelerating FLOWER

There are several computational challenges associated with the flow mining problem, which can affect the performance of the *FLOWER* algorithm presented in Section 3.3. While *FLOWER* is designed to be inherently efficient because of careful network-centric pruning, the problem itself can sometimes be fundamentally intractable for large networks. For example, in a completely connected graph, traversing every possible actor sequence from every source vertex has $O(|V|!)$ complexity. This is, of course, not true in most real networks, where the linkage structure is sparse and not all actors send the same set of messages at the same time. Nevertheless, it is still possible to envision scenarios, where the *FLOWER* approach might be undesirably slow for certain parameter settings (such as low values of f and high values of τ). Because these challenges are *inherent* to the problem at hand, it is natural to explore whether parallelization can be used to accelerate *FLOWER*.

There are several ways to parallelize the *FLOWER* algorithm. In Algorithm 1, line 6 executes the subroutine *Flowprop* for each vertex v. This can be executed in parallel, as each call is independent of the other. Similarly, each of the flow paths in the recursion tree from root to leaf is independent of one another and is therefore easy to parallelize. In these approaches, the parallelism is performed at a path-level, where each path can be treated as a independent computational unit. In path-wise parallelization, however, care must be taken to reduce redundant computations at the parent nodes, as they form common prefixes in different flow paths.

The highest level of parallelism, however, can be achieved if we can parallelize at the vertex level, where each vertex can be treated as a separate computational unit that can be executed in parallel. This is typical in vertex-centric computational models, such as GraphLab [60] or Pregel. As seen earlier, our sequential approach propagates messages between the neighboring vertices, and it naturally fits into this framework. We discuss a brief overview of the vertex-centric computational models in the next couple of paragraphs.

In vertex-centric computational models, any *vertex* needs to perform three main operations: *Gather*, *Apply*, and *Scatter*. The *Gather* operation receives messages through the incoming

edges of a vertex, the *Apply* operation processes the incoming messages and the *Scatter* operation distributes the processed messages to the neighbors via outgoing edges. Due to these three operations, vertex-centric computational abstractions are popularly referred to as the GAS framework.

The main problem with the GAS framework is in scenarios, where they deal with natural graphs having power-law degree distribution. Such graphs have very few nodes with extremely high degree and the remaining nodes have very small degree [67]. Hence balanced distribution of computational load, storage and communication is extremely challenging in this framework and to address this issue new frameworks, such as PowerGraph [67], have been developed. For a more detailed review of the PowerGraph, please refer to [67].

Algorithm 3: Scatter

Input: $icontext\_type$: context, $vertex\_type$: current_vertex, $edge\_type$: edge

```
if edge.destination_node does not have f words in its message table then
    return
for each sequence_to_send in current_vertex do
    if sequence_to_send in current_vertex has the destination vertex Id then
        continue
    if edge.destination_node satisfies all properties (1)-(4) then
        add the destination id to sequences_to_send and copy it to edge data
```

Algorithm 4: Apply

Input: $icontext\_type$: context, $vertex\_type$: current_vertex, $gather\_type$: incoming_object

```
for each sequence in the incoming_object.sequences do
    Add sequence to the current_vertex saved_sequences list
    Add sequence in to sequences_to_send array for scatter method to pick up
if (context.iteration <= max_iterations) && (number of sequences_to_send in current_vertex
> 0) then
    schedule current_vertex for next iteration
```

In the distributed version of our algorithm, each vertex contains three pieces of meta-data: its message table $(U_j, t_l)_{j=1...m}$, frequent flows ending at that vertex, and messages to forward to neighbors in the next iteration. Each edge acts as a channel that carries the message from

Algorithm 5: Gather

Input: $icontext\_type$: context, $vertex\_type$: current_vertex, $edge\_type$: edge

Output: $gather\_type$: gathered_obj

```
1 return received sequences from edge data
```

Algorithm 6: Gather Operator+=

Input: $gather\_type$: that

Output: $gather\_type$: ret_obj

```
1 if this.incoming_sequences has no sequence then
2     copy that.incoming_sequences to this.incoming_sequences
3     return this
4 else if that.incoming_sequences has no sequence then
5     return that
6 else
7     for each seq in that.incoming_sequences do
8         this.incoming_sequences.push_back(seq)
9     return this
```

a source to a destination vertex. The message carried by each edge has a set of flow objects, where each flow object contains a flow sequence and a set of word messages that support the sequence. Note that we do not need to carry any temporal information along the edges, because it can be reconstructed at each vertex based on the set of words that support the flow sequence. This approach provides significant savings in time and space. We also optimize the computation by initializing only the vertices that have message table of length at least f.

During the *Scatter* phase, each edge is invoked to scatter a message from source to the destination vertex. Each source vertex does an advanced lookup of the destination vertex message table, to verify the possibility of extending the flow by adding the destination node. If any of the properties (1)-(4) fail, then the message is not scattered to the destination. The pseudocode for the *Scatter* subroutine is listed in Algorithm 3.

In the *Gather* operation, each vertex is invoked to gather the messages from the incoming edges. This step eventually appends all the incoming flows, one after the other, from different edges into a single incoming flow object containing several flow sequences and corresponding

word signals. In GraphLab the *operator+=* appends all the sequences from each edge and the *Gather* function merely copies the reference of data from each edge and passes it to the operator. The pseudocode for the *Gather* subroutine and *operator+=* are provided in Algorithm 5 and Algorithm 6, respectively.

Each vertex during the apply phase saves the incoming sequences (from the *Gather* operation) in its own frequent sequence table. As the *Scatter* phase does advance lookup and scatters only valid sequences, the apply phase can save these sequences with no additional validations. Each vertex then schedules itself for the next iteration, as there could be potential extensions of recently added frequent flow sequences. Also, if one is interested in sequences of length not more than L, then the vertex can stop scheduling, if the current iteration number is greater than L. The pseudocode for the *Apply* routine is listed in Algorithm 4.

The main advantage of the GraphLab framework is that it is a unified framework for multi-core and distributed computation. Graphlab can use multiple cores on a single multi-core server, and if that is not sufficient it can scale to multiple servers. There is no additional coding or algorithmic changes required to switch from one infrastructure to another. We refer to our parallel version of FLOWER as *pFLOWER*.

3.5 Experimental Results

We evaluate the efficiency of our algorithm in terms of runtime of the algorithm. The implementation of the algorithm was done using C++ and the runtime was evaluated on a Linux server with Ubuntu 10.04 OS, 24GB RAM, 24 cores with each running 2.67GHz Intel Xeon processor. We used Graphlab version 2.1 [67] for our parallel *pFLOWER* evaluation.

3.5.1 Data Sets

We used two data sets: the *DataBase List of Publications* (DBLP) and the *US Patent Office* (USPTO) database. These data sets are described below in detail. We are interested in extracting the information flow patterns in the co-authorship network of both these data sets. In DBLP, for instance, the “mining” keyword may propagate across a sequence of authors forming an information flow path. We use all the words in the abstract of the papers and patents to generate the messages. The time-stamp of the document was used to generate the message time-stamp. The co-authorship network was used as the underlying network of communication.

DBLP Data Set: We downloaded the publicly available DBLP data set [3] , and extracted the year of publication, abstract and authors for each of the published documents. We removed entities with multiple identities using the data available in the DBLP website [4] . The cleaned data set had 444,406 authors and 1,572,277 papers. We stemmed the words, removed stop words, and stripped off punctuations in the abstract. The resulting dictionary was 600,718 words. All the publications were between the time-period of 1945 to 2011. We used publication abstracts to generate the content tokens. The network was constructed using the co-authorship relationship with 444,406 authors (nodes) and 1,280,168 edges.

US Patent Data Set: The United States (US) patent database is publicly available for access from the US Patent Office (USPTO)[5] . We downloaded the following set of attributes for all patents granted from June 21, 1977 to December 28, 1999: *Patent Number*, *Granted Date, Abstract, Inventors, Assignee, Legal Representative,* and *Application Number*. After cleaning the data set of documents containing missing meta-information, a total of 1,813,616 patents remained in the patent database. We used the patent abstract to generate the content tokens. The co-authorship network for the US Patent database contained 1,310,057 nodes and 2,444,474 edges.

3.5.2 Evaluation Approach

We measured the efficiency in terms of the running time of the algorithm. We evaluate the scalability of the distributed approach by varying the number of cores used for *pFLOWER*. We used the *PrefixSpan*[6] [65] sequence mining algorithm followed by post-processing of the output sequences to apply the network and life-time properties. The input to *PrefixSpan* is a set of transactions, where each transaction corresponds to a message U_j and the temporal order of the actors a_i who propagated that message (as ordered singleton itemsets). The output of *PrefixSpan* is a set of author sequences (corresponding to information flow paths), except that they do not satisfy the network validity and lifetime constraints. This is checked explicitly by using a constant time look-up table for each author and message pair. The resulting output of *PrefixSpan*, after post-processing yields the same output as our algorithm. Therefore, the running times of the methods can be meaningfully compared. We also compared the running time

[3] http://arnetminer.org/citation
[4] http://dblp.uni-trier.de/xml/
[5] http://uspto.org/
[6] http://www.cs.uiuc.edu/homes/hanj/software/prefixspan.htm

DBLP		USPTO	
f	Runtime (secs.)	f	Runtime (secs.)
480	36185.52	470	20905.63
540	16284.14	530	12311.91

Table 3.1: The running time (seconds) for the *PrefixSpan* baseline for DBLP and USPTO data sets. For an f value smaller than 450, *PrefixSpan* ran for more than a day (>86400 seconds) and did not complete.

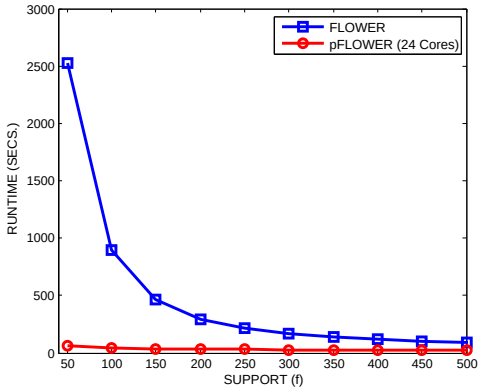
of our sequential version of the algorithm (*FLOWER*) against our parallel version (*pFLOWER*).

3.5.3 Results

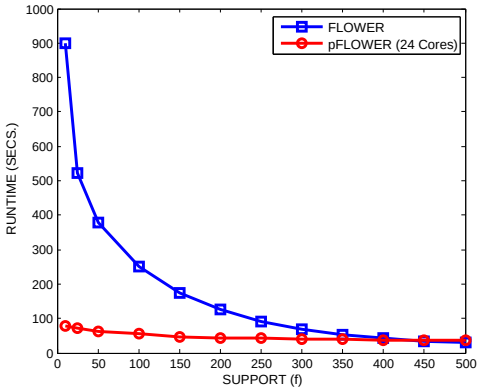
We compared the running time of *PrefixSpan*, *FLOWER* and *pFLOWER*. However, in Figure 3.2, we could not plot *PrefixSpan* running times as they were extremely large. Therefore, we list the running times of *PrefixSpan* separately in Table 3.1. Furthermore, we are unable to show the results for several values of $f \leq 450$, because *PrefixSpan* did not complete within a day. On the other hand, as evident from Figure 3.2, both *FLOWER* and *pFLOWER* completed in less than a couple of minutes over most parameter settings.

We compared the running times of *FLOWER* and *pFLOWER* algorithm, in Figures 3.2(a) and (b). As the number of words (f) required for the frequency property (Property 3) decreases, the number of possible flow paths increases exponentially. The *FLOWER* approach explores each of these paths sequentially, resulting in an exponential complexity with path length. On the other hand, the parallel algorithm *pFLOWER* scales extremely well at very low f values and the running time remains extremely small throughout the entire range of f values. The *pFLOWER* algorithm performs up to *three orders of magnitude faster* than PrefixSpan, and *two orders of magnitude faster* than *FLOWER* at low f values. These observations are consistent in both DBLP and USPTO data sets as shown in Figures 3.2(a) and (b), respectively. These observations also highlight the importance of a *network-centric approach* for computing information flow paths.

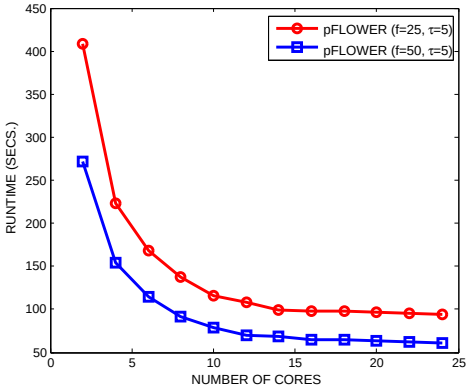
We evaluated the scalability of the *pFLOWER* algorithm in terms of the number of cores in Figures 3.2(c) and (d). The figure shows that the running time is roughly inversely proportional to the number of cores used for computation. In other words, linear speed-up is achieved in



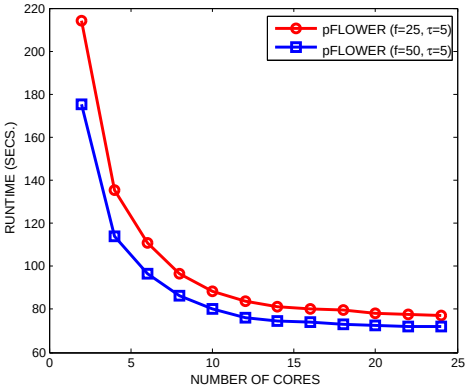
(a) DBLP runtime plot



(b) USPTO runtime plot



(c) DBLP scalability plot



(d) USPTO scalability plot

Figure 3.2: The two plots in the *top row* show the running time measurements for the DBLP and USPTO dataset by varying f. The two plots in the *bottom row* show the scalability analysis for the DBLP and USPTO data sets by varying the number of cores.

terms of the number of cores. It also demonstrates the efficiency of vertex-centric computational models in scaling up scenarios where sequential approaches are computationally infeasible. In this case, 14 cores were sufficient to complete the flow mining algorithm in less than a minute for low values of f, whereas straightforward sequential approaches do not terminate in reasonable running times (see Table 3.1). Thus, the proposed approach can be used to find information flows in networks with large number of activities and interactions, which may otherwise be computationally intractable using a single core.

3.6 Influence Analysis: An Application

Information flow patterns are sequences of actors who propagate at least f messages repeatedly preserving the temporal order in each propagation. These flow patterns denote the flow of influence along the network paths. The nature of influence depends on the nature of underlying network relationship or interactions. In DBLP and USPTO data set, we considered the co-authorship network and the nature of influence in these data sets are through co-authorship interactions. For instance, a flow path $\langle a, b, c\rangle$ denotes a word w used by author a, followed by b, and then c. When a used the word w because a and b have a co-authorship relationship, b may have been influenced by the word w through a and propagated it further to its neighbors. Similarly, c may have been influenced from b and propagated the word w to its neighbors. In a sense, for the example sequence, a is the leader and b and c are its followers. Similarly, b is the leader of the sub-sequence $\langle b, c\rangle$ with c as its follower. For each actor, we can compute the total number of followers (in this way) across all the flow patterns and we refer to this as the (co-authorship) influence score of that actor in the (co-authorship) network. The actor with the highest influence score in this DBLP or USPTO co-authorship network denotes the most influential co-author.

One might argue that using centrality measures or popular influence mining algorithms (such as PMIA [31], DegreeDiscountIC [38]) in a static co-authorship network are sufficient to measure the influence. We evaluate this hypothesis by comparing the influential co-authors found using the popular influence analysis algorithms such as degree-centrality, PageRank, PMIA [31] and DegreeDiscountIC [38] against the influencers found using the flow patterns. As the notion of influence has *no absolute ground truth* (similar to intelligence or trust), we use the author *citation counts as a proxy* for author influence. Here, we assume that an author has

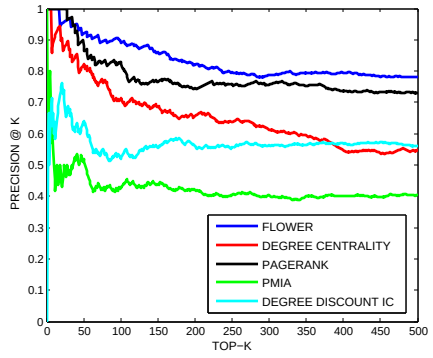
very high citation count if the author has considerable influence in the area. We computed the precision-at-K (P@K), precision-recall, and the F_1 score for the top-500 influencers found by each method (compared against the ground truth).

3.6.1 Evaluation Baselines

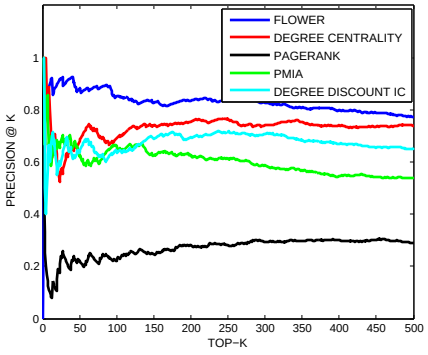
Let us now describe the baselines we used for evaluating our hypothesis. PMIA [31] is the prefix excluded extension of Maximum Influence Arborescense model. We used the weighted cascade model proposed in [29] to compute the edge probabilities for this approach. The degree-centrality approach uses the maximum total out-degree and DegreeDiscountIC [38] heuristic developed for the uniform IC [29] model with propagation probability $p = 0.01$. For PageRank, the restart probability was set to 0.15 and the stopping criterion, which is based on the L_1 norm difference between two successive iterations, was set to 10^{-7}. We use *FLOWER* to denote the influencers found using the information flow-based approach.

3.6.2 Evaluation Results

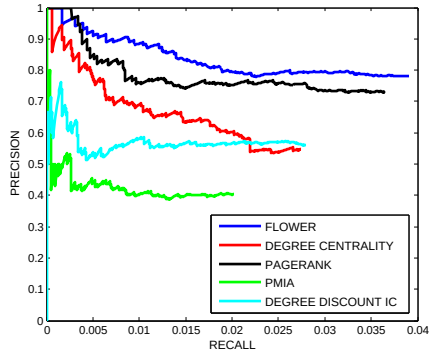
Our evaluation results are shown in Figure 3.3. The figure clearly shows that the order of baselines are not consistent in both data sets. The first- or second-order centrality measures might work in some data sets, while the information diffusion based method might work in others. But the flow-based techniques (like FLOWER) capture the lead authors whose ideas propagate dominantly and later gets picked up by other highly cited authors, resulting in high precision and recall compared to baselines. Moreover, our approach works consistently well in both data sets. In Figures 3.3(a) and (d), the precision gradually reduces as the top-K increases. This is because the number of authors in the ground truth reduces significantly as K increases. However, our method does not suddenly drop unlike the baseline methods, such as PMIA. Our approach is very stable and decreases gradually. As evident from Figures 3.3(b) and (e), the precision and recall of our methods are considerably better than the baselines. In terms of the F_1 measure (see Figures 3.3(c) and (f)), our approach performs better than baselines over all values of K.



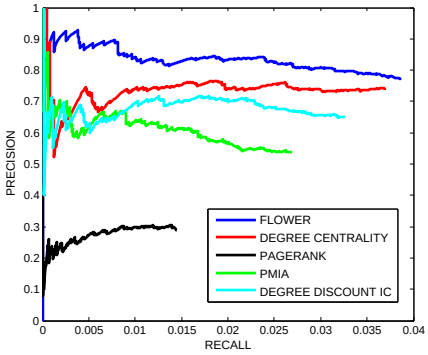
(a) DBLP P@K



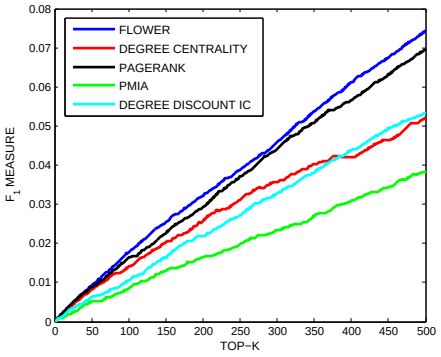
(d) USPTO P@K



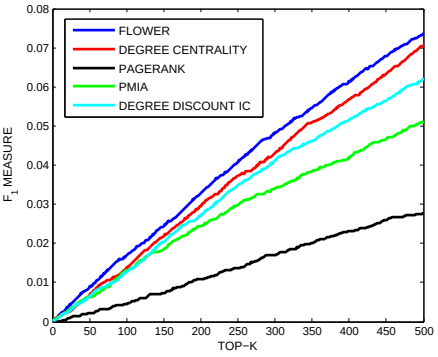
(b) DBLP P-R Curve



(e) USPTO P-R Curve



(c) DBLP F_1 measure



(f) USPTO F_1 measure

Figure 3.3: The P@K, P-R and F_1 measure plots for DBLP and USPTO data sets.

3.7 Conclusions

In this paper, we proposed an information flow mining problem with several desired properties. We developed a sequential version of the algorithm and established that the computational complexity of this problem is #P-complete. In order to scale for large networks, we described a parallel algorithm using vertex-centric computational models. Our parallel algorithm provides *three orders of magnitude* scale up over the state-of-the-art and with an increasing advantage with greater number of cores. Finally, we showed the effectiveness of the discovered flow patterns using an influence analysis application.

Chapter 4

Real-time Information Flow Analysis

4.1 Introduction

The problem of finding influential actors is important in various domains such as viral marketing [33] and political campaigns. The problem was formally defined by Kempe et al. [29] as an optimization problem over all possible subsets of nodes with cardinality k. Subsequently, a significant amount of work [29, 35, 37, 14, 38, 31, 36] has been done on this area. All these approaches are static, in a sense that any incremental changes to the network structure or edge probabilities require the re-computation of the influencers from scratch, and does not take advantage of previous computations. This inflexibility poses a major problem in the social stream setting, where the only observable aspect is a *continuous stream of social activity*, and one needs to be able to discover various types of influencers in this setting.

Another major disadvantage of existing methods is the inability to query the influencers in a *context-specific* fashion. For example, the top influencers for the search term "*Egypt Unrest*" would be very different from that of the search term "*PGA Golf Tour*". Many of the existing methods [14, 38, 31, 29] decouple the problem of influence analysis from the content-centric edge probability estimation [75]. Therefore, finding influencers for a *context-specific* query requires re-estimation of edge probabilities, specific to that query.

The influencers in a social stream are *time-sensitive* and may rapidly evolve, 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: (a) influencer, (b) influenced, (c) context, and (d) time. The last of these components is often (implicitly) assumed to be the current time, though our framework is flexible enough to provide responses to historical queries as well. The influence score $\mathcal{I}(\cdot, \cdot, \cdot, \cdot)$ is expressed as a function of function of four components. In the following, we provide a list of the key queries enabled by our method, though other variations of these queries are also possible.

- $\mathcal{I}(S_1, S_2, C, t)$: This quantity represents the aggregate influence score of actor set S_1 on actor set S_2 with respect to content C at time t. The content C may be specified with the use of either keywords (as in a search engine), or as a document. In practice, rather than the influence scores, the top influence pairs drawn from $S_1 \times S_2$ will be returned. This is, in fact, the most generic form of the query and the other queries are specializations of this query as discussed below. In most cases, it is these special forms that are of interest in application-specific scenarios.

- In a special case of the aforementioned query, either S_1 or S_2 can be set to a "don't care", denoted by "*". Thus, this query returns the global influence on a set of actors, or a set of actors on the remaining population with respect to content C. Note that when the content C is also set to "*", the resulting query is very similar to traditional influence analysis models.

- In a special case of the first query, the either S_1 or S_2 or both are set to atomic elements (i.e. individual actors). Note that this query can be used as a subroutine to determine the top context-sensitive influencers.

- In the event that the content C is set to "*" then the top *context-free* influencers will be determined. This query is quite similar to traditional influence analysis models, except

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

that it allows the ability to constrain specific subsets of actors to be influencers or as those that are influenced.

These queries can provide significant business insights into the content, network, and temporal dynamics of the social stream. For instance, the query $\mathcal{I}(*, \text{“David”}, \text{“Egypt unrest”}, t)$ can be used to obtain the list of influencers who influence “David” on the topic “Egypt unrest”. Such queries can be extremely useful in context-specific subscription of content (rather than blind following of popular users). Similarly, finding the most influential user across all topics for “David” can be very helpful in link recommendation.

In addition, several existing problems can be formulated using this querying framework to take advantage of our approach. For example, the traditional influence maximization problem [29] is that of maximizing the influence of node set X (of size k) for the entire network, for any content, across all time points, i.e. $\max_X \mathcal{I}(X, *, *, *)$. Similarly, the influence maximization problem, can be formulated for specific context and time as, $\max_X \mathcal{I}(X, *, C, t)$, where C and t are given. We will revisit these queries later in this paper.

4.1.1 Contributions and Organization

In this paper, we propose an online approach in which we process the incoming social stream objects to create an incrementally 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 [38, 31, 29, 35]. We establish the relationship of our approach to the well-established Katz similarity. It is also suited to applications where the hardware available for execution is limited. In such case, we provide an approximation technique (based on the available hardware) and show that our approximation close enough to actual value. We establish a tight lower bound on the accuracy for this purpose. 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.

4.1.2 Related Work

The problem of tracking influencers in a network is often treated as an influence maximization problem [29, 35, 37, 14, 38, 31, 36]. 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) [29] 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. This is a separate problem in its own right [75]. 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 [76, 29]. Related work on information flow mining and cascades may be found in [48, 49, 14, 14, 77]. The problem of information flow mining and influence analysis has been related in [59, 26, 78]. There are a few topic-specific influence analysis models [79, 80, 81, 82], but either they are not online or cannot be queried in a context- and time-sensitive fashion.

There is a surge of recent literature on social streams. 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 [39, 83], topic-based browsing [84], and influence analysis [26]. There have been some attempts to compute influencers in an online setting [85, 86], but these approaches are not context-specific and cannot be queried. There are other interesting works on topic-specific influence maximization [87]. 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 [40], a popular centrality measure.

4.2 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 $i \in V$ 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, denoted by $\mathcal{K}(t) = K_1 \ldots K_t$. We refer to each keyword $K_i \in \mathcal{K}$ as a *content token*. Each such content-token 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 text 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), or it could be the hash-tag from the tweet. Because this content may be copied, re-posted, re-tweeted, or may generally influence the future content posted by the different actors (in the form of appropriate keywords or hash-tags), the value of the different strings K_i (over different values of the index i) will often be the same. It is precisely this propagation of the content over the network that needs to be tracked. There are several types of contents that can be tracked, such as URLs, full tweets, hashtags, etc. All future analysis is applied to this set of content-tokens. 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 4.3.3).

- It is assumed that the content K_i is created by the actor a_i. The value $a_i \in V(t)$ 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 b_i from the actor set $V(t)$, 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 entire social stream is denoted by $\mathcal{S}$. In many social networks such as *Twitter* and blog networks, the stream of content can be tracked in an automated way. In the case of social networks such as *Twitter* subscription-based access is often available for the tweet-stream content. The propagation of content is usually closely related to the underlying network structure, and it often implies patterns of influence in the underlying network.

4.2.1 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. The main goal of this model is to set up an incrementally updatable data structure which implicitly stores all the information required for the querying process. While this data structure of flow paths will be set up in the next section, we will show in this section how flow paths are used for the purpose of query processing. The main 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 i is influential on actor 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 i to j through a flow path and then to j's neighbors. 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 j and 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 $j = i_1 \ldots i_r = k$ leading from j to k, such that each transmission between successive nodes occurs in that order. We formally define this concept as a *valid flow*:

Definition 9 (Valid Flow)**.** *A valid flow $\mathcal{F}$ of keyword K_i from j to k is an ordered set of nodes $j = i_1 \ldots i_s = k \in N$, such that the following conditions are satisfied:*

1. *An edge exists between nodes i_r and i_{r+1} in the base network.*
2. *The node i_{r+1} transmits a message containing the keyword K_i, after a message is transmitted from node i_r containing the same keyword.*

Thus, a flow from node j to node k, of the keyword K_i is indicative of the influence that node k has over node j. 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 j. We refer to this as the flow duration:

Definition 10 (Flow Duration)**.** *Consider a valid flow $\mathcal{F}$ of keyword K_i from j to k. Let t_c be the current time, and t_o be the original time at which the keyword was transmitted from node j. Then, the flow duration from node j to node 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 11 (Decayed Flow Weight)**.** *Let δt be the flow duration of a particular flow at a current time, and λ be the decay factor. Then, the decayed flow weight is given by $2^{-(\lambda(t_c - t_o))}$.*

Therefore, the flow has a half-life of $1/\lambda$, since the weight of a flow reduces by a factor of 2 every $1/\lambda$ time units. The decayed flow weight can be used in order to define the aggregate path flow.

Definition 12 (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 $\mathcal{P} = j = i_1 \ldots i_r = k$, 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 13 (Aggregate Pairwise Flow)**.** *The aggregate pairwise flow $\mathcal{V}(j, k, K_i, t_c)$ at time t_c for keyword K_i from node j to node k is equal to sum of the value of the flows on the paths from node j to node k. Therefore, is $\mathcal{S}^{jk}$ be the set of paths from node j to node k, we have:*

$$\mathcal{V}(j, k, K_i, t_c) = \sum_{\mathcal{P} \in \mathcal{S}_{jk}} \mathcal{A}(\mathcal{P}, K_i, t_c) \tag{4.1}$$

We now formally introduce the atomic influence function $\mathcal{I}(j, k, \mathcal{Q}, t)$, which is the influence exerted by node j on 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 j to k can be very different from the influence in the other direction.

Definition 14 (Atomic Influence Function)**.** *The atomic influence value $\mathcal{I}(j, k, \mathcal{Q}, t)$ for a node j to influence k, is defined as the sum of the aggregate pairwise flows over all keywords $K_i \in \mathcal{Q}$ in the data:*

$$\mathcal{I}(j, k, \mathcal{Q}, t_c) = \sum_{K_i \in \mathcal{Q}} \mathcal{V}(j, k, K_i, t) \tag{4.2}$$

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 $\mathcal{I}$, and how this function can be used in various queries. 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.

4.2.2 General Flow-based Query Processing

The aforementioned computation of the atomic influence function for calculating influences 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, $I(*, k, w, t) = \sum_{j \in V} I(j, k, w, t)$. In this section, we use j and k as the running indices for the identities of the influencer and influenced users, respectively. The three main applications of the atomic influence function are as follows:

Influencer Search Queries ($\alpha\,(j, Q, t)$): A typical form of querying is similar to using a search engine. Given a contextual query, $Q = \langle w_1, w_2, ..., w_k \rangle$, a relevant set of influencers are computed for Q at the current time t, using the atomic influence function $\alpha(j, Q, t) = I(j, *, Q, t)/I(*, *, Q, t)$. We normalize the scores across all users for the query, before comparing them with each other. We find the top-k j nodes that have the highest influence scores (using $\alpha(j, Q, t)$) and arrange them in the descending order. A user can also query the influencers, in a similar way, for an earlier time $t' < t$. We evaluate the effectiveness of influencer search queries in detail using multiple data sets in Section 4.4.

Link Prediction Queries ($\beta\,(j, k, t)$): Users in social network tend to connect to the users who influence them highly. Hence, one can use this query to find the top-m users who has the highest influence on user k at current time t. This is very similar to link prediction problem. Here, the atomic influence function can be used as $\beta(j, k, t) = I(j, k, *, t)/I(*, k, *, t)$ to find the influence of nodes j on k across all $w \in \mathcal{K}$.

Concept-specific temporal queries ($\gamma\,(Q, [t_1, t_2])$): These queries can be used to find the evolution of influence of a 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 $\gamma(Q, [t_1, t_2]) = I(*, *, Q, t)/I(*, *, Q, *), t \in [t1, t2]$. We demonstrate the use of this query in Section 4.4.

4.2.3 Relationship with Katz Measure

The aforementioned atomic influence function is closely related to the Katz measure [40]. 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

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

path. Specifically, if $\mathcal{P}_{ij}$ be the set of paths between nodes i and j, then the Katz measure $K(i,j)$ is defined as follows:

$$K(i,j) = \sum_{P \in \mathcal{P}_{ij}} \beta^{|P|} \tag{4.3}$$

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 in static networks, the close analogy between the two ways of measuring node importance lends 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 4.3.2). 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 bit later in Section 4.3.1.

4.2.4 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 all 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.

4.3 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 objects 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 $\mathcal{K}(t)$ contains a single keyword K. 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 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 flows in the tree 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 15 (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 is equal to the flow weight corresponding to the path.*

Algorithm *UpdateFlowPaths*(Originating Node: 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 i;
Create singleton node i in tree $\mathcal{T}$ as child of root
node if it does not already exist;
$\mathcal{C} = \{i\}$; { Candidate paths for expansion }
Update weight of singleton path containing only
node i in tree $\mathcal{T}$ by 1;
while $\mathcal{C}$ is not empty **do**
begin
Delete the first path P from $\mathcal{C}$ and
denote the first node of P by j;
for each $k \notin P$ in $V(t)$ with an incoming edge to j
if prefix of path $k \oplus P$ exists in T and k has
propagated keyword K prior to j
if the complete path $k \oplus P$ exists in T
Increment weight of last node of path $k \oplus P$
by 1 in T;
else
Create last node of P as child for prefix
of path $k \oplus P$ in T with weight as 1;
Add $k \oplus P$ to $\mathcal{C}$;
endfor
end
end

Figure 4.1: Updating the Flow Paths for Single Keyword Simplification

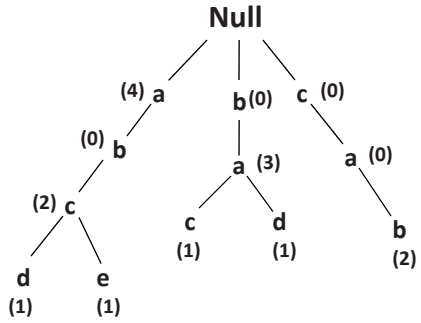


Figure 4.2: An illustrative example of a flow-path tree.

In order to illustrate an example of a flow-path tree, consider a set of paths $\{abc(2),\, ba(3),\, a(4), bac(2), cab(2),\, abcd(1),\, abce(1),\, bad(1)\}$, along with the relevant flow weights. Note that singleton nodes are allowed in the form of degenerate "paths". The relevant flow-path tree with the corresponding weights is illustrated in Fig. 4.2. Note that in this case, the flow tree is drawn for an arbitrary set of paths and flow weights, and therefore ancestors may have smaller weights than their descendents. In practice however, flow weights are often such that ancestors have higher weights than descendents. This is because any prefix of a valid flow is also valid flow in of itself, and will therefore be counted in the flow weights. We have however presented an arbitrary example above for greater generality.

The overall framework for the influencer tracking algorithm (with a single keyword) is illustrated in Fig. 4.1. 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 i. The input to the algorithm is the network $G(t)$, the originating node 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 i, at which the keyword K has appeared in the past. For example, if node j is incoming into node i, and has propagated the keyword K before i, then the path ji is added to the tree $\mathcal{T}$ if non-existent, and the count is increased by 1. Then, the path ji is added to the candidate list $\mathcal{C}$ for further backtracking. Note that all paths in $\mathcal{C}$ will always end with the node 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 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 k, a new path $k \oplus P$ is created by appending k to the beginning of the path P. Then, it is checked whether the *prefix of* $k \oplus P$ [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 $k \oplus P$, if such a child node does not already exist. The count of this child node is incremented by 1. The path $k \oplus P$ 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.

4.3.1 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 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,

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

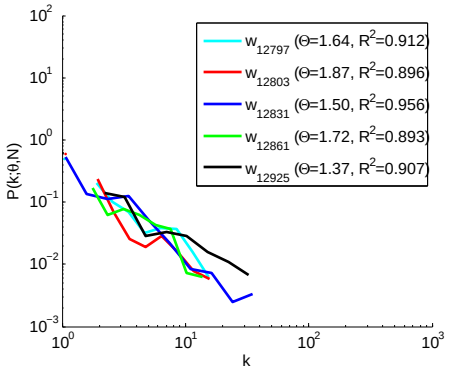


Figure 4.3: The flow weights of a flow-path tree follow Zipf distribution. The estimate of the Zipf parameter θ and corresponding R^2 for several trees are shown in the legend.

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 distribution of flow weights for several flow-path trees, in real data sets used for our experiments. In Fig. 4.3, we perform a linear fit on the log-log plot of rank and frequency of flow weights for a few flow-path trees and show that the flow weights follow a Zipf distribution with very high R^2. Now, we will show that the reduction in flow weight, drawn from a Zipf distribution, is typically less than $\log(\alpha \cdot N)/\log(N)$.

Theorem 5. *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:*

$$F(N, \alpha) \geq log(n)/log(N) = 1 - log(1/\alpha)/log(N) \tag{4.4}$$

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

$$C(n) = \sum_{i=1}^{n} 1/i^{\theta} \tag{4.5}$$

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

$$F(N, \alpha) \geq \frac{\sum_{i=1}^{n} 1/i}{\sum_{i=1}^{N} 1/i} \tag{4.6}$$

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.

4.3.2 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 us 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 3 (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 j. Then, if no new message has been transmitted at destination node j 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:*

$$\mathcal{A}(\mathcal{P}, K_i, t_2) = \mathcal{A}(\mathcal{P}, K_i, t_1) \cdot 2^{-\lambda(t_2 - t_1)} \tag{4.7}$$

The key pre-condition in the aforementioned rule is that no message should have been transmitted at the destination node j 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 11, 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. 4.1 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. 4.1, 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.

4.3.3 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.

While the algorithm is designed to create a flow-path tree for each keyword because its natural connection to query-processing, it is sometimes helpful to track latent topics. One reason is the ever-growing vocabulary in social streams [88] which makes it more difficult to build the index in terms of keywords. In such cases, one can compute the posterior of each topic for the query keyword, and compute the influence as a linear combination of the corresponding topics. The atomic influence function for the generalized list of latent topics ($\mathcal{Z}$) is shown in (4.8).

$$I(j, k, \mathcal{Q}, t_c) = \sum_{z \in \mathcal{Z}} \sum_{K_i \in \mathcal{Q}} p(z|K_i, t_c) \mathcal{V}(j, k, z, t_c) \tag{4.8}$$

This way one can track the most relevant topics for each post, rather than explicitly tracking all the content tokens in the post. The proposed approach is general enough to handle individual content tokens, n-gram phrases, and latent topics. We refer to our algorithm, for the generalized full keyword sets, as **QUIS** which is an acronym for **QU**erying **I**nfluencers in social **S**treams.

4.4 Influence Analysis 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, followed by 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.

4.4.1 Data sets

We evaluate the effectiveness and efficiency in two different data sets: a social and a collaboration data set. For the social data set we used Twitter, and for the collaboration data set we used DBLP.

Twitter Data Set: We used tweets from 25^{th} March to 1^{st} May 2014 (i.e., 37 days) (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, #youtubeblockedinturkey, #erdogan, #1mayis". The resulting data set contained 1,919,294 ($\approx$ 1.9 million) tweets. Each posting extracted contained the time stamp, screen name and its content. We extracted all the unique hash-tags from these postings and treated them as the keywords propagated. This entire stream contained 131,574 unique hash-tags and 293,364 unique users. We constructed the network structure using the mention or retweet activities between users using the past 10 months of *Twitter* data. The resulting network contained 1,958,776,515 ($\approx$ 2 billion) edges and 120,775,144 ($\approx$ 120 million) nodes [4] . This activity network was used as the underlying relationship network in the algorithm.

[4] Extracting the follower network for such a large number of users is not feasible, because the *Twitter* APIs are highly rate-limited. Therefore, we use the mention and retweet activities to represent the relationship network.

DBLP Data Set: The DBLP data set is publicly available and can be downloaded from arnetminer.org [5] [89]. This data set contains a stream of documents (2,084,055 papers), temporally ordered based on the year of publication (until 2011). We also considered each abstract published as a message that was posted by the authors of the paper at the same time (i.e., year of publication). 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. Our keyword set was constructed using uni-, bi- and tri-grams of the words appearing in the 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 phrases (uni- to tri-grams) for each topic and created a model based on tracking 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.

4.4.2 Baselines

We used some of the most popular baselines used in several other recent influence analysis works [29, 38, 31]. The first baseline is the *PMIA* algorithm discussed in [31]. The model uses a network structure with defined edge probabilities. We calculated the probabilities using the weighed cascade model proposed in [29]. Our next baseline algorithm is *DegreeDiscountIC*, which is the degree discount heuristic of [38] developed for the uniform IC model. The restart probability was set to 0.15. The stopping criteria was based on the difference in $L1$-norm values between successive iterations, and was set to 10^{-7}. We also used the degree and weighted-degree as additional baselines.

[5] `http://arnetminer.org/citation` (version 6.0)

4.4.3 Evaluation Measures

We evaluated the effectiveness of our approach in terms of the quality of the influencers returned for 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 (4.9) and (4.10), respectively, to compare against the baselines. The list of top-k influencers returned by each approach are 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 users that are relevant that are successfully retrieved.

$$precision = \frac{|\texttt{relevant} \cap \texttt{retrieved}|}{|\texttt{retrieved}|}, \tag{4.9}$$

$$recall = \frac{|\texttt{relevant} \cap \texttt{retrieved}|}{|\texttt{relevant}|}. \tag{4.10}$$

The ground truth used for the *DBLP* data set were 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.

4.4.4 Effectiveness Results

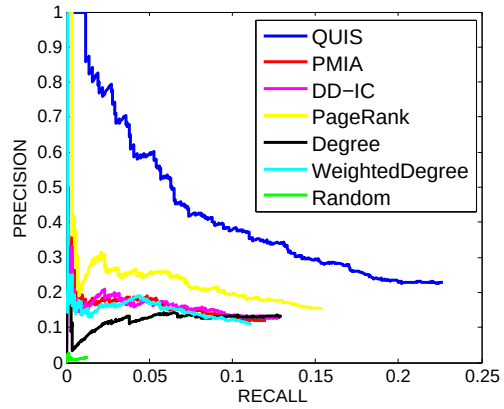
We evaluated the quality of the influencers discovered by each approach in comparison with the ground truth. For our approach, we constructed the list of top-1000 influencers using *"Influencer Search Queries"* as described in Section 4.2.2. The precision and recall values were computed by sweeping through 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, #twitterblockedinturkey, #direntwitter, #twitterisblockedinturkey, #25martkorkusu, #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.

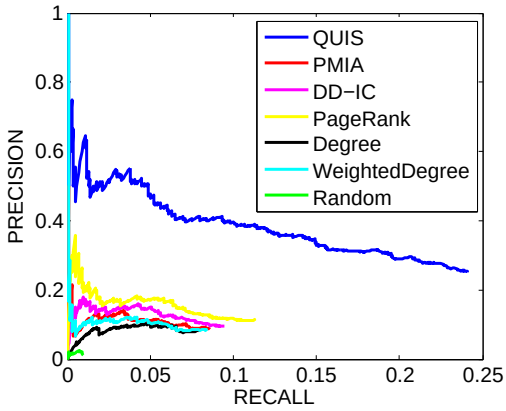
The precision-recall (PR) curves for the data mining and networking areas, at the beginning of year 2011 and 2005, are shown in Fig. 4.4(a) and (b), respectively. The precision-recall plot for the *Twitter* data set computed at the beginning of day May 1, 2014 is shown in Fig. 4.4(c). Due to limited space, we show the area under the curve (a point-wise value) for PR plots for 2005 and 2011 for six different fields in Table 4.1 and 4.2 respectively. The influencers of Turkey incident was evaluated on May 1 and April 13, 2014. As per the AUC values, our approach performs the best in context-specific influence 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 [38, 31] looks only at pairwise interactions. Also, abstract all the query information in to a single probability score between a pair of nodes is too unstable, especially when the words chosen for the query becomes overlapping 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 the overall influence for the query. This may also be the reason for the inconsistency in the performance of baselines. Among the baselines, PageRank does very well in DBLP, while PMIA does well in Twitter.

4.4.5 Sensitivity Analysis

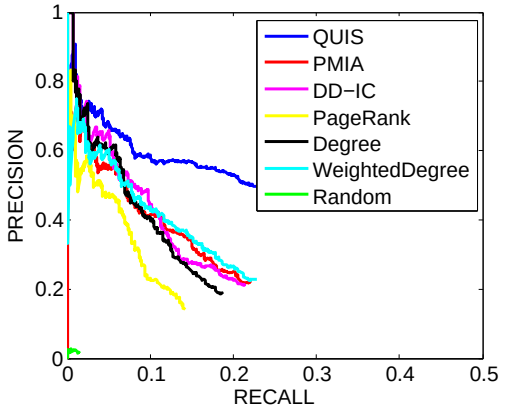
The value of α denotes the pruning parameter and ranges from 0 to 1. The higher the value of α is, the lesser the amount of pruning per tree. In order to measure the impact of pruning on the effectiveness of the analysis, we computed the value $\Delta_x(\alpha) = (I(x,*,*,t) - I_\alpha(x,*,*,t))/I(x,*,*,t)$. Here $I(x,*,*,t)$ denotes the total influence of user x with no pruning, and $I_\alpha(x,*,*,t)$ is the total influence computed after pruning with parameter α. Similarly, the quantity $\Delta_w(\alpha) = (I(*,*,w,t) - I_\alpha(*,*,w,t))/I(*,*,w,t)$ computes the fractional loss



(a) Data mining query (2011)



(b) Network query (2005)



(c) Twitter Turkey query (1st May 2014)

Figure 4.4: 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.

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

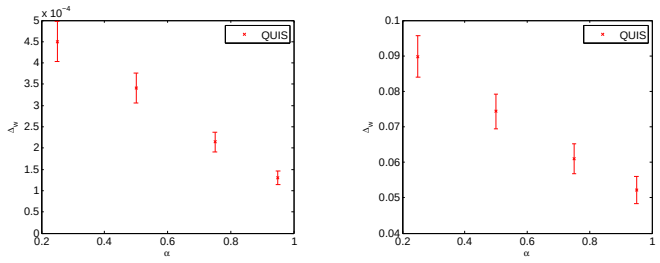
Table 4.1: AUC computed for PR plots for six different computer science fields in 2005

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

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



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

Figure 4.5: 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.

	DBLP		Twitter	
α	mean(Δ_x)	deviation(Δ_x)	mean(Δ_x)	deviation(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 4.4: Mean and deviation of Δ_x for DBLP and Twitter data sets computed at 2011 and May 1, 2014 respectively.

of influence scores per word w. We reported the mean and deviation of the Δ_w in Fig. 4.5, and those of Δ_x in Table 4.4. As discussed earlier, by retaining even a small fraction $\alpha = 0.25$ of the nodes provides an excellent estimate of the total influence in both data sets.

4.4.6 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 show 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 2002, the most influential individuals were Jiawei Han, Yiwen Yin and Jian Pei. The temporal aspects of influence queries can be used to understand

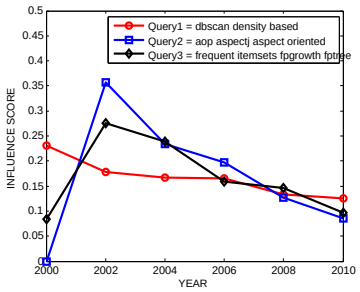


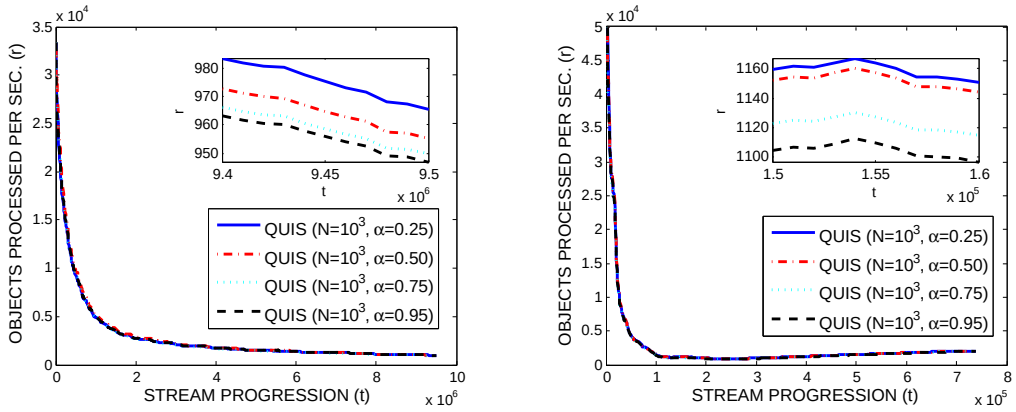
Figure 4.6: Growth and decay of influence for various concept-based queries.

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. 4.6. 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 4.2.2.

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.

4.4.7 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. 4.7. 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.



(a) DBLP (b) Twitter

Figure 4.7: Number of objects processed per second with stream progression. The inset shows the computational reduction level due to pruning.

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 querying 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 4.5, are in seconds and computed for DBLP in the beginning of 2011, and for the Twitter data sets on May 1, 2014.

	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 4.5: Running time (seconds) for various methods.

4.5 Link Prediction

An important aspect of many networks is that they are often available in the form of social stream of content interactions between the users. Many networks cannot be easily collected for analysis, and in some cases such as an interaction network, the underlying "social network" is transient without a base static network. Even in the case of more formal social networks such as *Twitter*, the *social stream is the only observable aspect of the dynamics in the social network,* at least from a practical perspective. Furthermore, the implicit network and content structure of the social stream usually evolves fairly rapidly with time [39] In such cases, the appropriate links to recommend may vary *significantly* depending on the recent temporal patterns of interaction, and information flow between users. In such networks, the relevant information flow patterns provide interesting patterns of influence in the network [26]. We argue that such patterns of influence are important, since a user who is influenced by another is more likely to link to the latter. Therefore, a natural approach for link prediction [40] in social streams is to find important paths of information flow between nodes, in real-time, in order to make useful predictions in a time-sensitive way. In many cases, specific content in the network may be more significant in finding relevant links between users, and therefore it may be useful to find *topically-relevant* links between nodes. This is also useful for a target-marketing application for recommending nodes in a commercial network, or in a general information network, where nodes may be topically relevant objects (e.g. golf club). For example, when the content corresponding to the topic "golf clubs" is passed around in the network, it is very relevant for link recommendations between users that are interested in golf.

The problem of link prediction was first proposed in [40]. This work proposed a number of structural measures for effectively solving the link prediction problem. The work in [34] enhanced the neighborhood measures with weights on nodes. A number of models have also shown how content can be included in the link prediction process [90, 91, 92, 93, 94]. A number of methods [90, 95, 96] have also been designed for link prediction in heterogeneous networks. The link prediction problem may also be viewed as a supervised learning problem in which the prediction of a link corresponds to a binary label, which represents the presence or absence of the link [97, 98, 99, 100]. A number of recent techniques [101, 102] use innovative techniques such as subgraph counting for the link prediction problem. A survey on link prediction models may be found in [103]. However, none of these techniques have been designed for the social stream scenario, in which only the stream of social content is observed for the link prediction process.

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 [90, 29]. Related work on information flow mining and cascades may be found in [48, 49, 30, 16]. The problem of information flow mining and influence analysis has been related in [59, 26]. Because information flows are indicative of interest levels of different entities for each other, it is natural to predict links on the basis of information flows.

There is a surge of recent literature on social streams, 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 [39, 83], topic-based browsing [84], and influence analysis [26]. It has been shown that the use of such streams often provides novel insights which is not possible with traditional models of social network analysis. This paper studies an influence-based framework for link prediction from social streams. As we will see later, the approach is related to a flow-based version of the Katz measure [40] for link prediction.

We now present an information flow-based link prediction algorithm. We show that such an approach can be very effective in finding relevant, real-time and temporally sensitive links. The approach can work with the limited network information that is available in real scenarios, and can adjust to *content-specific* topics for the link prediction process. An important aspect of this link prediction approach is that it is *activity-centered*, in that it preferentially utilizes the more active and recent flow paths in the network in order to make link predictions. The formation

of links is an activity in itself, which is closely correlated with other forms of information flow activity in the network and hence our information flow based approach is a natural choice for link prediction in social streams. Our experimental results will demonstrate the effectiveness of such an approach.

The entire social stream is denoted by $\mathcal{T}$. In many social networks such as *Twitter* and blog networks, the stream of content can be tracked in an automated way. In the case of social networks such as *Twitter* subscription-based access is often available for the tweet stream content. The propagation of content is usually closely related to the underlying network structure, and it often implies patterns of influence in the underlying network. Therefore, the core-principle of our link prediction is as follows:

A directed link from node A to node B should be recommended, if the former is likely to receive content flows from the latter.

The direction of the link is important, in that the directed links should point *towards* more influential nodes. This is an issue in some social networks such as *Twitter*, in which links are asymmetric with follower-based relationships. The approach trivially generalizes to undirected networks, by using the average influence in both directions.

In most information network entities such as the web, the directed links point towards more influential entities. One advantage of this approach is that link recommendations can be made *topical*, by restricting the analyzed content flows to specific topics or keywords. This can be extremely useful in many commercial applications, where link recommendations need to be product- or topic-specific.

Definition 16 (Influence-based Link Prediction)**.** *The influence-based link prediction value $\mathcal{L}(j, k, t_c)$ for a directed link from node k to node j, is defined as the sum of the aggregate pairwise flows over all keywords $\mathcal{K}$ in the data:*

$$\mathcal{L}(j, k, t_c) = \sum_{K_i \in \mathcal{K}} \mathcal{I}(j, k, K_i, t_c) \tag{4.11}$$

It is important to note that the direction of the links is important, since the prediction of the links is in a direction *opposite* to the flow. This is because the links should point in a direction which is indicative of the greater influence of the source node, as implied by the flow. This is

consistent with all other indirect measures of influence such as page-rank, which are commonly used in web-centered applications. In cases, where the network is undirected, the prediction values can averaged in both directions.

4.6 Link Prediction Results

In this section we analyze the effectiveness and efficiency of our link prediction algorithm with respect to popular baselines. We describe our data sets, evaluation measures, and experimental results.

4.6.1 Data sets

We have used two data sets corresponding to *DBLP* and the *US Patent Database* respectively. These data sets are described in detail below.

DBLP Data Set: We used the DataBase List of Publications (DBLP) data set [6] , and extracted the year, title, abstract and author details for each of the published documents from 1945 until 2005. We cleaned the DBLP data set to remove documents with missing meta-information such as the title, author or date. After cleaning, the DBLP data set had 444,576 distinct authors and 529,102 documents. The dates on the documents were used in order to create a social stream of content, where the keywords in the abstract formed the content-tokens. Each content token was stemmed and stop words were removed. The number of distinct content tokens after processing was 586,564. Ties between documents were resolved using the lexicographic ordering of their title string. Each document was further converted to a social stream considering every pair of authors in each paper and the content tokens generated by them. As the co-author relationship in DBLP publications are bi-directional, we created edges in both directions and added to the social stream. Each social stream object includes a pair of authors representing the "from" and "to" nodes, year of publication and content tokens extracted from the abstract. The total number of social stream objects generated for the 60 year period was 980,281 ($\approx$ 1 million).

US Patent Data Set: The United States (US) patent database is publicly available for access from US Patent Office (USPTO)[7] . We downloaded the following set of attributes for all

[6] `http://dblp.uni-trier.de/xml/`
[7] `http://uspto.org/`

patents granted from June 21, 1977 to December 28, 1999: *patent number, granted year, title, inventors, assignee, legal representative, and application number*. After cleaning the data set of documents containing missing meta-information, a total of 1,861,000 patents remained in our patent database. We used the patent abstracts to generate the content tokens. We created pairwise edges for constructing the social stream. The year in which the patent was granted was used to create the temporal ordering of the social stream. The words from patent abstract, after stemming and removing stop words, were used as content tokens for edge in the social stream. As the patent co-authorship is bi-directional we created edges in both direction in the social stream. The total number of distinct authors in this data set was 959,451 and the number of social stream objects generated for the 22 year period was 5,132,534 ($\approx$ 5.1 million).

4.6.2 Baselines

We used three different baselines to cover both supervised and unsupervised categories of link prediction methods. In the supervised category, we used the squared [100] and logistic loss classifiers [97] and in unsupervised we use the most popular Jaccards Coefficient measure [40]. Both the supervised classifiers are feature-based. The hyper-parameters of the baseline models are chosen using a hold out validation data set using the first 5 years of the social stream data. This data was not used in the evaluation process. Here below we describe the baselines in detail.

- **LASSO:** This baseline uses LASSO regression, where the loss function is a squared loss and the regularization term is L_1. We used an Alternating Direction Method of Multipliers (ADMM) to compute the regression coefficients for this baseline. The augmented Lagrangian parameter was set to 1.5 and the maximum iterations were set to 100. We constructed eight features for this baseline [40]: common neighbors, Adar-Adamic score, Jaccard's coefficient, preferential attachment, in- and out-degree for "from" node, in- and out-degree for "to" node.

- **LOGISTIC:** We use a logistic loss function with a L_1 regularization term for this baseline. The same set of features constructed for lasso were used for this baseline as well. We used ADMM to estimate the model parameters. The augmented Lagrangian parameters and the maximum iterations were set to the same value as lasso baseline.

- **JACCARD:** The Jaccard's coefficient (J) is an unsupervised measure, computed for a pair of nodes x and y, is shown in (4.12). The set of neighbors of a node x was denoted

by $N(x)$. This was computed for all edges available in the test data set. The top-k recommendations, based on the descending order of the J value, were used for each user in the precision and recall computation.

$$J = \frac{|N(x) \cap N(y)|}{|N(x) \cup N(y)|} \tag{4.12}$$

4.6.3 Evaluation Measures

We evaluated the effectiveness and efficiency of our approach with respect to other baselines. We measured the effectiveness of our approach in terms of precision and recall. We used the notion of precision and recall as it is typically used in information retrieval scenarios. We consider each *test* author as a query and the recommendations for that author as the retrieved result. The actual links formed for that author corresponds to the relevant results. Given the relevant ground truth (G) and retrieved recommendation (R) sets for each user (u) we can compute the precision and recall as in (4.13) and (4.14). In our results, we report the average precision and recall computed across all users in the test data set.

$$Precision_u = \frac{|R_u \cap G_u|}{|R_u|} \tag{4.13}$$

$$Recall_u = \frac{|R_u \cap G_u|}{|G_u|} \tag{4.14}$$

The efficiency was measured in terms of running time in seconds, at regular intervals along the progression of the stream. Because the running time varied across different intervals, we also measured the average running time in seconds across all intervals. We also measured the number of objects processed per second, at several points in the stream.

4.6.4 Experimental Setup

We evaluated the average precision and recall on the social stream at regular intervals. We first trained our model until a bootstrap point. Then, for every successive year, we predicted the links that would form in the next year. The bootstrap point for the DBLP data set was set to 1990, and for USPTO, it was set to 1985. Test data set for a particular year was used as training data for subsequent years in streaming fashion. We measured the precision and recall for all users who formed a *new* co-authorship relationship in the test period. Given a test user for recommendation, our method will recommend top-k users based on the influence score stored

in the content trees. The baseline methods are supervised or unsupervised binary classifiers and require positive and negative edge examples. To address this, we sample 1000 negative edges for each user that did not form in that year and previously, in their two hop neighborhood. If there are not enough nodes in the two hop neighborhood, we randomly sample from the universe of all nodes that have appeared in the stream so far. Note that our approach implicitly considers every node that has occurred in the stream as a possible candidate. However, the baseline classifiers have to choose only from a small sample of edges that include both positive and negative edge samples.

4.6.5 Experimental Results

In this section, we organize our results in to effectiveness and efficiency categories and discuss in that order.

Effectiveness Results

The effectiveness was measured in terms of precision and recall averaged across all users in the test data set. Figure 4.8(a) and 4.9(a) shows the precision-recall (P-R) values measured for top-20 recommendations and averaged across all one year intervals in the entire stream for DBLP and USPTO respectively. In the DBLP data set, our approach (FLIPS) does much better both in terms of precision and recall. Despite choosing a low number of candidate edges for each test user, classifiers such as logistic and lasso perform poorly. As our target variable is binary, the use of logistic loss is more appropriate as it penalizes large errors to an asymptotic constant. On the other hand, lasso penalizes large errors quadratically. This difference clearly shows up in terms of their classification performance. The lasso baseline knocks out several features that are not relevant due to L_1 regularization. In our experiments, we found Jaccard's Coefficient (JC) was most promising feature across several runs. This is also one of the reasons for lasso to perform quite closely compared to Jaccard baseline. The use of other features, in addition to JC, has a minor positive effect in lasso. The significant performance improvement of our approach is mainly due to the use of multiple propagation flow paths of arbitrary lengths across two nodes. Furthermore, *flow activity-centered* link prediction often provides insights about future activity such as link formation, that cannot be provided by more static measures.

The precision-recall results for the USPTO data set are illustrated in Figure 4.9(a). The order

of performance of these baselines is quite similar to DBLP, except they are well differentiated. The reason for this is that the co-authorship network structure of inventors are closed (within companies), and in smaller communities, unlike more open academic collaborations. As a specific example of this phenomenon, the USPTO co-authorship network had 417,993 weakly connected components while DBLP had only 104,299 components. In smaller communities, it is easier to determine the recommendations than larger ones.

We measured the precision for the top-20 query results for each method across the entire stream. The results are shown in Figures 4.8(b) and 4.9(b) for the DBLP and USPTO data sets respectively. Our method performs quite well in terms of precision and recall across the entire stream. All methods show a slight improvement in precision and recall values as the stream progresses, due to the availability of a richer underlying network structure. The order of baseline performance in precision and recall curves in USPTO are similar to that of DBLP.

Efficiency Results

We measured the efficiency in terms of the running time in seconds, by computing the total time taken by each method for training and testing. We show the running time with progression of stream to emphasize the efficiency of incremental updates rather than batch processing. In Figure 4.8(d) and 4.9(d) we compute the running time for each method to train its model using all data available until that point in the stream and make new recommendations for the test period. It is evident that all baseline models perform quite closely until the number of training and test samples become quite large, as the number of publications collected by these data sets expands. This is especially true after year 2000 in Figure 4.8(d), and after 1990 in 4.9(d). The overall running times of the USPTO data set is quite large compared to DBLP. The average number of edges per year in DBLP is 49,470, while in USPTO it is 223,153. This is an extra factor of 5 in the latter case. As the stream progresses, the data available to each method increases and hence the running time increases as well. However, the rate of increase is much larger for the logistic method, as compared to other methods. This is because the coefficient update step in the logistic method is quite expensive compared to lasso. This is one of the reasons that the squared error loss function is sometimes practically preferred over the logistic function [100]. The number of samples affects the overall running time as well. In the case of the DBLP data set, the rate of increase of number of edges along the stream was quadratic, while in the USPTO data set it was linear. This linear and quadratic behavior is clearly reflected in the

performance of all methods across the entire stream in USPTO and DBLP data sets respectively.

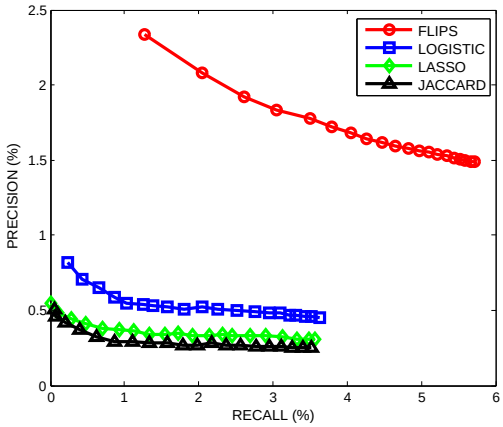
Figures 4.8(e) and 4.9(k) illustrate the performance in terms of the number of objects processed per second over the life of the data stream for the two data sets. In this case, we have illustrated our approach over various content tree hash table sizes (m). As the content tree hash table size increases, the amount of content compression reduces drastically. This results in more work for the algorithm. However, the benefit decreases as the stream progresses, due to the increase in the number of flow path updates and corresponding influence scores in each content tree. Furthermore, the performance drop is initially quite rapid, as the number of new content tokens and nodes appear much faster, than at later stages of the stream. The curves for DBLP and USPTO are shown in Figures 4.8(e) and 4.9(e). These results are quite similar, except that the number of objects processed varies by a constant factor across the data sets due to the difference in the volume of the stream.

The average running time is measured for each method as a point-value comparison across the baselines and our approach. We measured the running time for all methods at multiple stream points and averaged them. This is shown in Figures 4.8(f) and 4.9(f) The Jaccard's coefficient computation is a fastest as it is unsupervised, and no training is involved. The logistic method is quite time consuming on the average across the entire stream, compared to other methods. Lasso is quite efficient and can be easily parallelized. Therefore, it is definitely a good choice, if the underlying network has many small connected components, as in the case of USPTO. Our method performs well in any kind of network structure and performs well with respect to the various baselines.

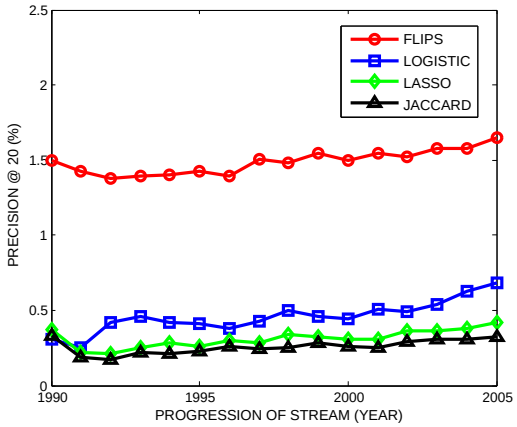
4.7 Conclusions and Summary

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 influence 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.

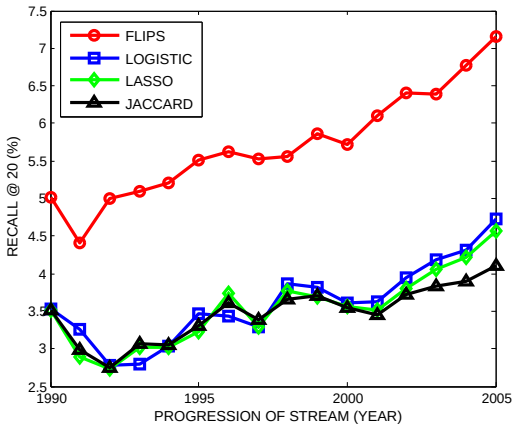
Link prediction is an important problem in online social and collaboration networks. In most of the social networks, the social stream is the only observable aspect of the dynamics. Hence,



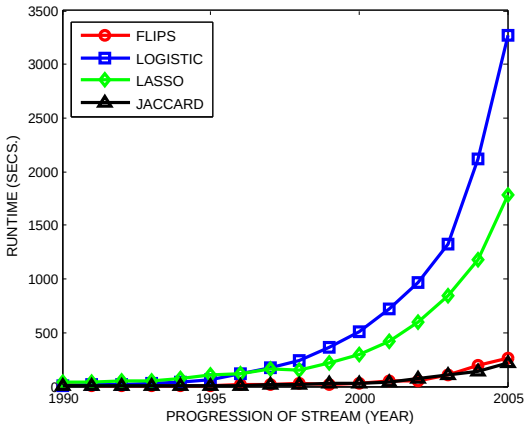
(a) DBLP P-R Curve



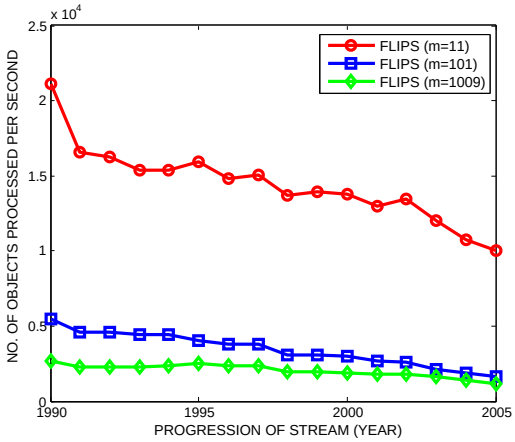
(b) DBLP P@K Vs. Stream



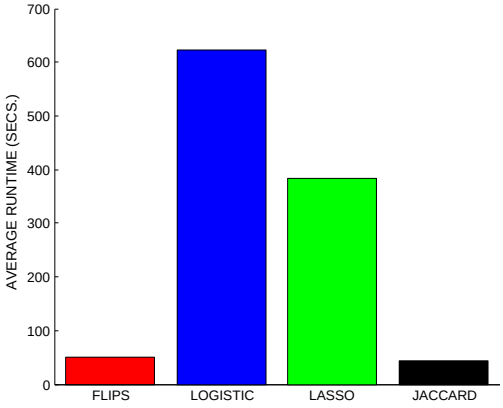
(c) DBLP R@K Vs. Stream



(d) DBLP Runtime Vs. Stream

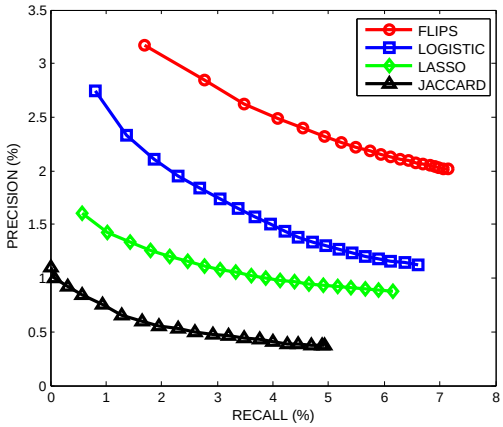


(e) DBLP Objects Processed/Sec Vs. Stream

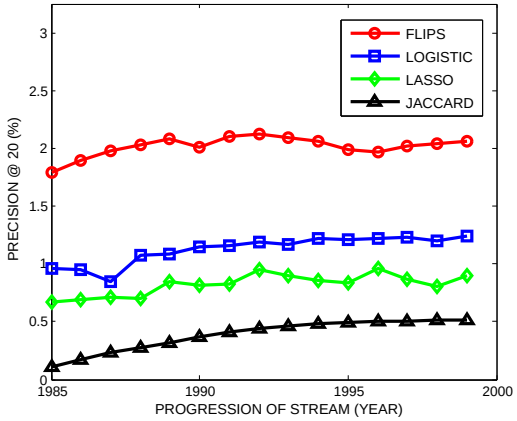


(f) DBLP Average running time

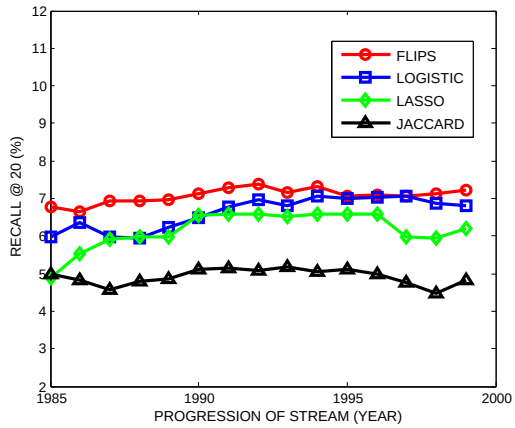
Figure 4.8: Link prediction efficiency and effectiveness results for DBLP data set.



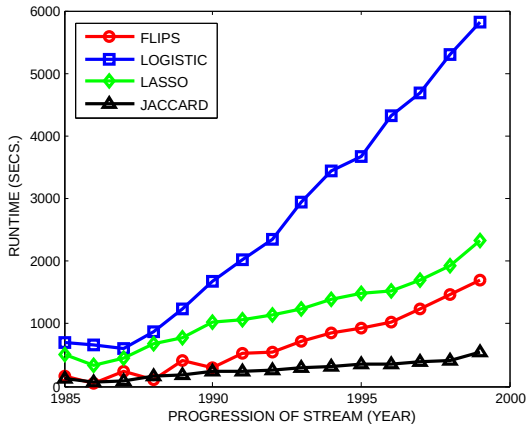
(a) USPTO P-R Curve



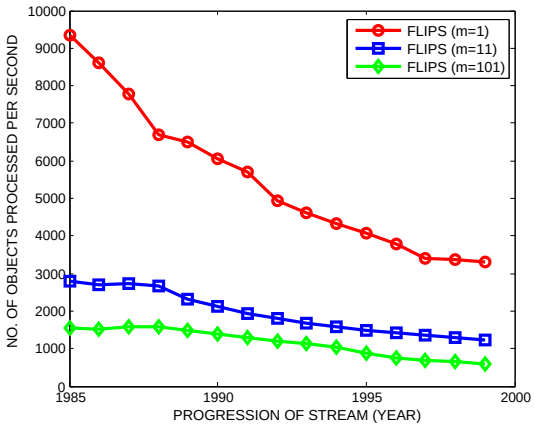
(b) USPTO P@K Vs. Stream



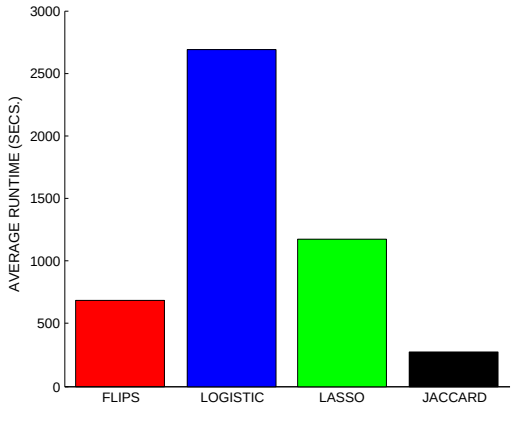
(c) USPTO R@K Vs. Stream



(d) USPTO Runtime Vs. Stream



(e) USPTO Objects Processed/Sec Vs. Stream



(f) USPTO Average running time

Figure 4.9: Link prediction efficiency and effectiveness results for USPTO data set.

it is important to develop link recommendation algorithms for such streaming scenarios. Most of the existing algorithms for link prediction use two or three hop path features for effective link prediction. While some of these features may work in some applications, no single feature can work in all applications. More importantly, these methods are agnostic to the underlying network activity, which is crucial in making predictions about future network activity, such as the formation of links. Therefore, it is important to develop a flexible framework that can use different kinds of activity paths in the network, while making recommendations. The flow of content in a network provides the important activity-centered insights, that are useful for making predictions.

In this chapter, we developed a link prediction algorithm for social streams. We used information flow paths to leverage flow paths of influence for link prediction. We evaluate our algorithm against popular baselines, and show the effectiveness in terms of precision and recall measures. We also show the efficiency of our approach in streaming scenarios. We anticipate that out scheme will enable broader classes of influence functions, which will be helpful in various social network applications.

Chapter 5

Event Detection using Information Flows

5.1 Introduction

Much of the text data in social scenarios arises in the context of streaming applications, in which the text arrives as a continuous and massive stream of text segments [104]. Such applications present a special challenge for mining algorithms, because of the fact that it is often necessary to *process the data in a single pass* and one cannot store all the data on disk for re-processing.

The online event detection problem is closely related to that of *topic detection and tracking* [41, 42, 43, 44, 45, 72, 46, 47]. This problem is also closely related to stream clustering, and attempts to determine new *topical trends in the text stream* and their significant evolution. The idea is that important and newsworthy events in real life are often captured in the form of *temporal bursts of closely related messages* in a social stream. The problem can be proposed in both the supervised and unsupervised scenarios. In the *unsupervised case*, it is assumed that no training data is available in order to direct the event detection process for the stream. In the *supervised case*, prior data about events is available in order to guide the event detection process.

In this paper, we will study the problems of clustering and event detection in *social streams* in which each text message is associated with at least a pair of actors in the social network. We use the term social network rather loosely, as it could refer to the messages in an online chat

messenger service, or it could even refer to an email network in which messages are sent between pairs of actors. A number of interesting issues arise in such social networks, because they are dynamic, and are associated with *network structure* in the stream. Specifically, each actor is a node in the social network, and each message sent in the social network is the text content associated with an edge in the social network. Clearly, multiple messages can be sent between the same pair of actors over time. In this case, we would like to use *the topical content of the documents, their temporal distribution, and the graphical structure of the dynamic network of interactions in order to detect interesting events and their evolution*. Clearly, messages which are sent between a tightly knit group of actors may be more indicative of a particular event of social interest, than a set of messages which are more diffusely related from a structural point of view. Such messages are structurally well connected, when the social network is viewed as a graph, with the edges corresponding to the messages sent between entities. This is related to the problem of *community detection* [105], in which we try to find structurally connected regions of the social network. At the same time, the *content and topics* of the documents should also play a strong role in the event detection process. Thus, both network locality and structure need to be leveraged in a *dynamic streaming scenario* for the event detection process.

Therefore, the key challenges for event detection in social streams are as follows: **(a)** The ability to use both the content and the (graphical) structure of the interactions for event detection. **(b)** The ability to use temporal information in the event detection process. For example, a new trend of closely related text documents from a structural and content point of view, which have not been encountered earlier may correspond to a new event in the stream. **(c)** The ability to handle *very large and massive* volumes of text documents under the *one-pass constraint of streaming scenarios*.

This paper is organized as follows. The remainder of this section presents related work. In section 2, we present the model for event mining in social streams. The algorithms for supervised and unsupervised event detection are presented in section 3. In section 4, we present the experimental results. Section 5 presents the conclusions and summary.

5.1.1 Related Work

The problem of determining events in streams is closely related to the problem of stream clustering [104]. This method has been studied extensively by the text mining community in the context of the topic detection and tracking problem [41, 42, 43, 44, 45, 72, 46, 47]. This is

also related to the problem of clustering and topic modeling [106, 72, 107, 108, 109, 110] in dynamic text streams.

However, in social networks, there is a rich amount of structure available in determining the key events in the network. For example, an event corresponding to *Mideast Unrest* may often correspond to text streams exchanged between members who are closely linked to one another based on geographical proximity. While the use of linkages in order to determine clusters and patterns has been widely studied by the social networking community [111, 112, 50, 113], these methods are typically designed for static networks. Some clustering methods have recently also been designed for dynamic networks [111, 112], though they do not use the content of the underlying network for the mining process. On the other hand, some recent methods for pattern discovery in networks use both content and structure [113, 114], though these methods are not defined for the problem of event detection in the *temporal scenario*. A method in [115] is designed to measure the diffusion and spread characteristics of known and popular events, but is not designed for *new* event *discovery*. In this paper, we design a method which can use the content, structural and temporal information in a holistic way in order to detect relevant clusters and events in social streams.

5.2 Event Mining in Social Streams: The Model

In this section, we introduce the notations and definitions and the model for event mining in social streams. Such social streams are assumed to consist of *content-based interactions between structurally connected entities* in the data. Therefore, we will propose a number of notations and definitions for this purpose. We assume that the structure of the social network is denoted by the graph $G = (N, A)$. The node set is denoted by N and edge set is denoted by A. The social stream corresponds to the interactions between the different actors in the node N, and each interaction is an edge drawn from the linkage set A. Therefore, the graph G provides information about the universe of interactions in the social network. We next define the concept of a social stream which is overlaid on this social network structure.

Definition 17 (Social Stream)**.** *A social stream is a continuous and temporal sequence of objects $S_1 \ldots S_r \ldots$, such that each object S_i corresponds to a content-based interaction between social entities, and contains explicit content information and linkage information between entities as follows:*

- *The object S_i contains a text document T_i which corresponds to the content of the interaction of an entity in the social network with one or more other entities.*

- *The object S_i contains the origination node $q_i \in N$ which is the sender of the message T_i to other nodes.*

- *The object S_i contains a* set of one or more *receiver nodes $R_i \subseteq N$, which correspond to all recipients of the message T_i from node q_i. Thus, the message T_i is sent from the origination node q_i to each node $r \in R_i$. It is assumed that each edge (q_i, r) belongs to the set A.*

Thus, the object S_i is represented by the tuple (q_i, R_i, T_i).

We note that the above definition of a social stream captures a number of different natural scenarios in different kinds of social networks:

- In the *Twitter* social network, the document T_i is the tweet content, the node q_i is the tweeting actor, and the set R_i is the recipient set.

- Email and chat interaction networks may also be considered social networks with an exactly similar interpretation to the above.

- In many social networks, a posting on the wall of one actor to another corresponds to an edge with the document T_i corresponding to the content of the posting.

Such a social stream typically contains rich information about the trends which may lead to changes both in the content and the structural locality of the network in which the interactions may occur. We begin with describing an unsupervised technique for event detection, which continuously characterizes the incoming interactions in the form of clusters, and leverages them in order to report events in the data stream. We formally define the social stream clustering problem.

Definition 18 (Social Stream Clustering)**.** *A social stream $S_1 \ldots S_r \ldots$ is continuously partitioned into k* current *clusters $\mathcal{C}_1 \ldots \mathcal{C}_k$, such that:*

- *Each object S_i belongs to at most one of the current clusters $\mathcal{C}_r$.*

- *The objects are assigned to the different clusters with the use of a similarity function which captures both the content of the interchanged messages, and the dynamic social network structure implied by the different messages.*

We will use both the content and linkage information in order to create the clusters. Since the clusters are created dynamically, they may change considerably over time, as the stream evolves, and new points are added to the clusters. Furthermore, in some cases, an incoming object may be different enough from the current clusters. In that case, it may be put into a cluster of its own, and one of the current clusters may be removed from the set $\mathcal{C}_1 \ldots \mathcal{C}_k$. Such an event may be an interesting one, especially if the newly created cluster starts a new pattern of activity in which more stream objects are subsequently added. At the same time, in some cases, the events may not be entirely new, but may correspond to significant changes in the patterns of the arriving objects in terms of their relative distribution to clusters. Therefore, we define two kinds of events, which are referred to as *novel events* and *evolution events* in order to describe these different scenarios.

Definition 19 (Novel event)**.** *The arrival of a data point S_i is said to be a novel event if it is placed as a single point within a newly created cluster $\mathcal{C}_i$.*

We denote the creation time for cluster $\mathcal{C}_i$ by $t(\mathcal{C}_i)$. The event in this case is the story or topic underlying the data point S_i and not the data point itself. We will discuss the process for creation and maintenance of a cluster in a social stream slightly later. Next, we define the concept of an *evolution event*. An evolution event is defined with respect to specific time horizon and represents a change in the relative activity for that particular cluster. We first define the concept of fractional cluster presence.

Definition 20 (Fractional Cluster Presence)**.** *The fractional cluster presence for cluster $\mathcal{C}_i$ in the time period (t_1, t_2) is the fraction of records from the social stream arriving during time period (t_1, t_2), which belong to the cluster $\mathcal{C}_i$. This fractional presence is denoted by $F(t_1, t_2, \mathcal{C}_i)$.*

The occurrence of a new event typically affects the relative presence of the data points in the different clusters, or it may result in a novel event. For example, the *Mideast Unrest* event may result in either the creation of a new cluster or the significant addition of new data points to the clusters most closely related to this topic. This is because it is often possible for previously existing clusters to match closely with a sudden burst of objects related to a particular topic.

This sudden burst is characterized by a change in fractional presence of data points in clusters. Formally, we define such an event as an *Evolution Event*. In order to determine such evolution events, we determine the higher rate at which data points points have arrived to this cluster in the previous time window of length H, as compared to the that even before it. A parameter α is used in order to measure this evolution rate.

Definition 21 (Evolution Event)**.** *An evolution event over horizon H at current time t_c is said to have occurred at threshold α for cluster $\mathcal{C}_i$, if the ratio of the relative presence of points in cluster $\mathcal{C}_i$ over the horizon $(t_c - H, t_c)$ to that before time $t_c - H$ is greater than the threshold α. In other words, we have:*

$$\frac{F(t_c - H, t_c, \mathcal{C}_i)}{F(t(\mathcal{C}_i), t_c - H, \mathcal{C}_i)} \geq \alpha \tag{5.1}$$

Furthermore, it is assumed that $t_c - 2 \cdot H \geq t(\mathcal{C}_i)$.

We assume that the value of $t_c - 2 \cdot H$ is larger than the cluster creation time $t(\mathcal{C}_i)$ in order to define the afore-mentioned evolution ratio in a stable way. This ensures that at least H units of time are used in the computation of the denominator of this ratio.

5.3 Social Stream Clustering

The design of an effective online clustering algorithm is the key to the event detection process. Therefore, we will first focus on the problem of online clustering. Then, we will discuss how to leverage this clustering for event detection. We assume that the clustering algorithm uses the number of clusters k as input, and maintains the structural and content information in the underlying clusters in the form of node and word frequencies in the cluster. The clusters are denoted by $\mathcal{C}_1 \ldots \mathcal{C}_k$. We assume that the set of nodes associated with the cluster $\mathcal{C}_i$ is denoted by V_i, and the set of words associated with it is denoted by W_i. The set V_i is referred to as the node summary, whereas the set W_i is referred to as the word summary. As we will see later, this summary characterization can be used for assignment of incoming social stream objects to clusters. The set V_i contains the nodes $j_{i1}, j_{i2} \ldots j_{is}$, together with node frequencies denoted by $\nu_{i1} \ldots \nu_{is}$. The word set W_i contains the word identifiers $l_{i1}, l_{i2}, \ldots l_{is}$ together with word frequencies denoted by $\phi_{i1}, \phi_{i2} \ldots \phi_{is}$. One challenge with maintaining node summaries in large social networks is that the number of nodes can be extremely large (in the hundreds of millions), as a result of which the summary may be too large to use efficiently, especially in the

Algorithm *SocialStreamClustering*(NumClusters: k);
begin
Initialize clusters $\mathcal{C}_1 \ldots \mathcal{C}_r$ to null;
Initialize i, μ, σ, M_0, M_1, M_2 to 0;
repeat
$i = i + 1$;
Receive next social stream object S_i;
for each cluster $\mathcal{C}_l$ compute $Sim(S_i, \mathcal{C}_l)$;
Let r be the index of cluster $\mathcal{C}_r$ with largest similarity to S_i;
if $(Sim(S_i, \mathcal{C}_r) < \mu - 3 \cdot \sigma)$ **then** replace most stale
cluster with a new cluster containing the single point S_i;
else add S_i to $\mathcal{C}_r$ and update statistics $\psi_r(\mathcal{C}_r)$ of cluster $\mathcal{C}_r$;
Update M_0, M_1, M_2 additively;
$\mu = M_1/M_0$;
$\sigma = \sqrt{M_2/M_0 - \mu^2}$;
until(*end_of_stream*);
end

Figure 5.1: Social Stream Clustering for Event Detection

online context. Later, we will discuss how to use sketch-based methods in order to compute the similarities more effectively. First, we will discuss a more straightforward method for online maintenance of the clusters with the direct use of the cluster-summary. Now, we will formally define the concept of cluster-summary.

Definition 22. *The cluster-summary $\psi_i(\mathcal{C}_i)$ of cluster $\mathcal{C}_i$ is defined as follows:*

- *It contains the node-summary, which is a set of nodes $V_i = \{j_{i1}, j_{i2} \ldots j_{is_i}\}$ together with their frequencies $\eta_i = \nu_{i1} \ldots \nu_{is_i}$. The node set V_i is assumed to contain s_i nodes.*
- *It contains the content-summary, which is a set of word identifiers $W_i = \{l_{i1}, l_{i2}, \ldots l_{iu_i}\}$ together with their corresponding word frequencies $\Phi_i = \phi_{i1}, \phi_{i2} \ldots \phi_{iu_i}$. The content-summary W_i is assumed to contain u_i words.*

The overall summary is $\psi_i(\mathcal{C}_i) = (V_i, \eta_i, W_i, \Phi_i)$.

We design an online partition-based clustering methodology, in which a set of clusters $\mathcal{C}_1 \ldots \mathcal{C}_k$ are maintained together with their cluster summaries $\psi_1(\mathcal{C}_1) \ldots \psi_k(\mathcal{C}_k)$. As new social stream objects arrive, the clusters are continuously updated. At the same time, the changes

in the underlying clusters are continuously tracked and used in order to raise alarms for new events. First, we will discuss how the clusters are maintained. We will discuss the process of constructing event alarms from cluster statistics slightly later.

In each iteration, we compute the similarity of the incoming social stream object S_i to each cluster summary $\psi_i(\mathcal{C}_i)$. The details of the similarity computation will be provided later. The incoming stream object is then assigned to its closest cluster, unless the closest similarity value is significantly lower than that attained for the stream objects encountered so far. In order to determine if the closest similarity value is significantly lower than that attained by previous stream objects, we maintain the mean μ and standard deviation σ of all closest similarity values of incoming stream objects to cluster summaries. The similarity value is said to be significantly below the threshold, if it is less than $\mu - 3 \cdot \sigma$. We will describe the process of dynamically maintaining μ and σ at a later stage of the paper.

If the similarity value of the closest cluster is above the threshold of $\mu - 3 \cdot \sigma$, then we can assign the incoming stream object S_i to its closest cluster centroid. Once the stream object S_i is assigned to its closest cluster centroid $\mathcal{C}_r$, we update the corresponding cluster summary $\psi_r(\mathcal{C}_r)$. Specifically, any new nodes in S_i, which are not already included in V_r are added to V_r, and the frequency of the nodes of S_i which are included in V_r are incremented by 1. We note that the nodes in S_i correspond to both the source node q_i and the destination nodes R_i. In other words, the set $R_i \cup \{q_i\}$ is used to update the set V_r and its member frequencies. The same approach is applied to the words in W_r with the use of the words in the social stream object S_i. The only difference in this case is that the frequencies of the words are not incremented only by 1, but by their frequency of presence in the underlying document. On the other hand, if the similarity of S_i to $\mathcal{C}_r$ is greater than $\mu - 3 \cdot \sigma$, then we create a singleton cluster containing only the object S_i and corresponding cluster summary statistics. This cluster replaces the most stale cluster from the current collection $\mathcal{C}_1 \ldots \mathcal{C}_k$. The most stale cluster is defined as the one which was updated the least recently. In the event that a null cluster exists (which has never been updated), it is automatically considered the most stale cluster. Any ties are broken randomly.

It remains to explain how the similarity of the stream object S_i is to computed to the cluster $\mathcal{C}_r$. In order to compute the similarity, we need to compute both the structural $SimS(S_i, \mathcal{C}_r)$ and the content components $SimC(S_i, \mathcal{C}_r)$ of the similarity value. The content-components of the similarity is straightforward, and is simply the tf-idf based [116] similarity between the content T_i (belonging to social stream object S_i) and the content W_r. The information in this scenario

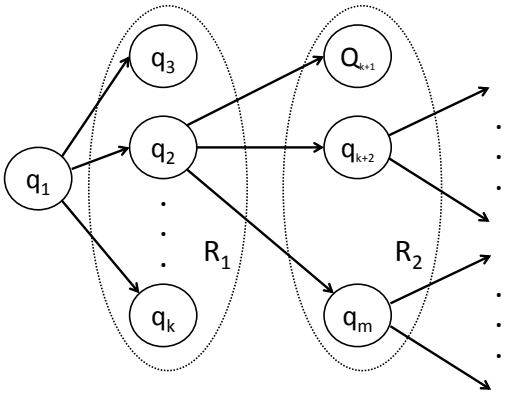


Figure 5.2: The flow of information from q_i to neighbors R_i is shown in this figure.

flows from q_i to R_i and denotes the corresponding information flow pattern at a vertex-level, where the vertex is q_i. This is shown in Figure 5.2. The information from q_1 is first propagated to set of neighbors in R_1, and from q_2 to R_2 and so on. We use the local information flow pattern to compute the structural similarity.

The structural similarity between the nodes V_r and the nodes $R_i \cup \{q_i\}$ in the social stream as follows. Let $B(S_i) = (b_1, b_2, \ldots b_{s_r})$ be the bit-vector representation of $R_i \cup \{q_i\}$, which has a bit for each node in V_r, and in the same order as the frequency vector $\eta = (\nu_{r1}, \nu_{r2}, \ldots \nu_{rs_r})$ of V_r. The bit value is 1, if the corresponding node is included in $R_i \cup \{q_i\}$ and otherwise it is 0. The structural similarity between the object S_i and the frequency-weighted node set of cluster $\mathcal{C}_r$ is defined as follows:

$$SimS(S_i, \mathcal{C}_r) = \frac{\sum_{t=1}^{s_r} b_t \cdot \nu_{rt}}{\sqrt{||R_i \cup \{q_i\}||} \cdot (\sum_{t=1}^{s_r} \nu_{rt})} \tag{5.2}$$

Note that we are using the L_1-norm of the node-frequency vector in the denominator as opposed to the normal use of the L_2-norm in order to penalize the creation of clusters which are too large. This will result in more balanced clusters.

The overall similarity $Sim(S_i, \mathcal{C}_r)$ is computed as a linear combination of the structural and content-based similarity values.

$$Sim(S_i, \mathcal{C}_r) = \lambda \cdot SimS(S_i, \mathcal{C}_r) + (1 - \lambda) \cdot SimC(S_i, \mathcal{C}_r) \tag{5.3}$$

The parameter λ is the balancing parameter, and lies in the range $(0, 1)$. This parameter is specified by the user.

It remains to explain how we maintain the mean and standard deviation of the (closest) similarity values of the incoming objects to the clusters. For this purpose, we maintain the zeroth, first and second order moments M_0, M_1 and M_2 of the closest similarity values continuously. These values can be easily maintained in the stream scenario, because they can be *additively* maintained over the data stream. The mean μ and standard deviation σ can be expressed in terms of these moments as follows:

$$\mu = M_1/M_0, \quad \sigma = \sqrt{M_2/M_0 - (M_1/M_0)^2}$$

The overall algorithm for cluster maintenance is illustrated in Figure 5.1. The main challenge of this algorithm is that the node based statistics can be rather large and the corresponding similarity computations cumbersome. For example, the number of nodes in V_r may be of the order of tens of millions and this can make the algorithm extremely slow. Therefore, we will design a sketch-based technique in order to speed up the computations.

5.3.1 Sketch-based Speedup

In this section, we discuss the sketch-based technique for maintaining node statistics. Sketch based techniques [117] are a natural method for compressing the counting information in the underlying data so that the broad characteristics of the dominant counts can be maintained in a space-efficient way. In this paper, we will apply the count-min sketch [117] for maintaining node counts in the underlying clusters. In the count-min sketch, a hashing approach is utilized in order to keep track of the node counts in the underlying data stream. We use $w = \lceil \ln(1/\delta) \rceil$ pairwise independent hash functions, each of which map onto uniformly random integers in the range $h = [0, e/\epsilon]$, where e is the base of the natural logarithm. The data structure itself consists of a two dimensional array with $w \cdot h$ cells with a length of h and width of w. Each hash function corresponds to one of w 1-dimensional arrays with h cells each. In standard applications of the count-min sketch, the hash functions are used in order to update the counts of the different cells in this 2-dimensional data structure. For example, consider a 1-dimensional data stream with elements drawn from a massive set of domain values. For example, in our application, this domain of values corresponds to the different node identifiers in the social network. When a new element of the data stream is received, we apply each of the w hash functions to map onto a number in $[0 \ldots h-1]$. The count of each of the set of w cells is incremented by 1. In order to *estimate* the count of an item, we determine the set of w cells to which each of the w

hash-functions map, and compute the minimum value among all these cells. Let c_t be the true value of the count being estimated. We note that the estimated count is at least equal to c_t, since we are dealing with non-negative counts only, and there may be an over-estimation because of collisions among hash cells. As it turns out, a probabilistic upper bound to the estimate may also be determined. It has been shown in [117], that for a data stream with T arrivals, the estimate is at most $c_t + \epsilon \cdot T$ with probability at least $1 - \delta$.

In order to use the count-min sketch for improving the node-count estimation process, we maintain a sketch table for each cluster in the data. The sketch table is used for the purpose of maintaining the frequency counts of the nodes in the incoming data stream. Specifically, the sketch table for the jth cluster is denoted by U_j. If desired, we can use the same set of w hash functions for the different clusters. The main condition is that the set of w hash functions should be independent of one another. For each incoming object S_i, we update the sketch table for the cluster to which it is assigned on the basis of the similarity measure. We apply the w different hash functions to the (string representation of the identifier of the) nodes in $R_i \cup \{q_i\}$, and add 1 to the counts of the corresponding cells. Thus, for the incoming object R_i, we need to apply each of hash functions to the different $|R_i| + 1$ different nodes, and update the corresponding cells. This corresponds to an application of $(|R_i| + 1) \cdot w$ hash function instantiations and corresponding cell updates.

The sketch-based structure can also be used to effectively *estimate* the similarity value $SimS(S_i, \mathcal{C}_r)$. We note that this similarity computation needs to be performed for each cluster $\mathcal{C}_r$, and its corresponding sketch table U_r in order to determine the closest cluster to the incoming object based on the composite similarity measure. The denominator of $SimS(S_i, \mathcal{C}_r)$ can be exactly estimated, because the object S_i is known, and therefore the value of $\sqrt{|R_i \cup \{q_i\}|}$ can also be known exactly. The value of $\sum_{t=1}^{s_r} \nu_{rt}$ can also be known exactly, because it is simply equal to the sum of all the values in the sketch table cells in U_r for any one of the w hash functions. Thus, this value may be obtained exactly by summing up the h cells for any particular[1] one of the hash functions. On the other hand, the numerator needs to estimated approximately. Note that the numerator is essentially defined by the sum of the estimated values of the frequencies of the nodes included in $R_i \cup \{q_i\}$.

The frequency of each such node can be estimated in a manner discussed in [117]. Specifically, for each node included in $R_i \cup \{q_i\}$, the corresponding cluster-specific frequency of the

[1] The sum of the h cells would be the same, no matter which hash function is picked.

node can be obtained by applying the hash function to the identifier of each node. We note that the value of the corresponding hash cell will always be an overestimate because of collisions between the different node identifiers to the same hash cell. Therefore, the minimum of these values across the w different hash functions will also be an over-estimate, though it will be much tighter and more robust because of the use of different hash functions. We sum up these estimated frequency values over the different nodes in $R_i \cup \{q_i\}$. This is essentially the estimate of the numerator. This estimation of the numerator can be used in conjunction with our exact knowledge about the different denominator values in order to create an estimate of $SimS(S_i, \mathcal{C}_r)$. Let $EstSimS(S_i, \mathcal{C}_r)$ represent the *estimated* similarity of S_i to $\mathcal{C}_r$ with the use of the sketch-based approach. Then, we can show the following result:

Lemma 4. *If a sketch-table with length h and width w is used, then for some small value $\epsilon > \sqrt{|R_i|+1}/h$, the estimated value of the similarity $EstSimS(S_i, \mathcal{C}_r)$ is bounded to the following range with probability at least $1 - \left(\frac{\sqrt{|R_i|+1}}{h \cdot \epsilon}\right)^w$:*

$$SimS(S_i, \mathcal{C}_r) \leq EstSimS(S_i, \mathcal{C}_r) \leq SimS(S_i, \mathcal{C}_r) + \epsilon \tag{5.4}$$

Proof. As discussed earlier in Equation 5.2, the structural similarity is given by the following equation:

$$SimS(S_i, \mathcal{C}_r) = \frac{\sum_{t=1}^{s_r} b_t \cdot \nu_{rt}}{\sqrt{||R_i \cup \{q_i\}||} \cdot (\sum_{t=1}^{s_r} \nu_{rt})} \tag{5.5}$$

We note that the numerator is (over)-estimated approximately with the use of the sketch-based process, whereas the denominator can be known exactly. It is evident that $SimS(S_i, \mathcal{C}_r) \leq EstSimS(S_i, \mathcal{C}_r)$ because of the over-estimation of the numerator. It remains to show that $EstSimS(S_i, \mathcal{C}_r) \leq SimS(S_i, \mathcal{C}_r) + \epsilon$ with probability at least $1 - \left(\frac{\sqrt{|R_i|+1}}{h \cdot \epsilon}\right)^w$.

Let $SimSN(S_i, \mathcal{C}_r)$ and $EstSimSN(S_i, \mathcal{C}_r)$ be the estimation of the numerator with the sketch-based approach. Then, since the denominator can be computed exactly, we have:

$$EstSimS(S_i, \mathcal{C}_r) = \frac{EstSimSN(S_i, \mathcal{C}_r)}{\sqrt{|R_i|+1} \cdot (\sum_{t=1}^{s_r} \nu_{rt})} \tag{5.6}$$

Furthermore, we have:

$$SimS(S_i, \mathcal{C}_r) = \frac{SimSN(S_i, \mathcal{C}_r)}{\sqrt{|R_i|+1} \cdot (\sum_{t=1}^{s_r} \nu_{rt})} \tag{5.7}$$

Therefore, in order to prove the result in the lemma we need to prove bounds on the approximation in the numerator. Specifically, we **need to prove that** the following holds true with probability at least $1 - \left(\frac{\sqrt{|R_i|+1}}{h \cdot \epsilon}\right)^w$.

$$EstSimSN(S_i, \mathcal{C}_r) \leq SimSN(S_i, \mathcal{C}_r) + \epsilon \cdot \sqrt{|R_i| + 1} \cdot (\sum_{t=1}^{s_r} \nu_{rt}) \tag{5.8}$$

Let us abbreviate one of the terms on the right hand side of the above equation by $B = \epsilon \cdot \sqrt{|R_i| + 1} \cdot (\sum_{t=1}^{s_r} \nu_{rt})$. In other words, we need to show the above bound on the probability for the condition that:

$$EstSimSN(S_i, \mathcal{C}_r) - SimSN(S_i, \mathcal{C}_r) \leq B \tag{5.9}$$

The expected value of the estimation of $SimSN(S_i, \mathcal{C}_r) - SimSN(S_i, \mathcal{C}_r)$ *with the use of any particular hash function* is $(|R_i| + 1) \cdot (\sum_{t=1}^{s_r} \nu_{rt})/h$. This is because, we need to sum up the errors over $|R_i| + 1$ different frequency estimations, and the expected number of collisions for any of these cell-based estimations is $(\sum_{t=1}^{s_r} \nu_{rt})/h$. Then, we can use the Markov inequality to show that the probability that the condition in Equation 5.9 is violated *with the use of* 1 *hash function* with probability at most $\frac{(|R_i|+1) \cdot (\sum_{t=1}^{s_r} \nu_{rt})}{B \cdot h}$. We can generalize the probability of this to w independent hash functions to at most $\left(\frac{(|R_i|+1) \cdot (\sum_{t=1}^{s_r} \nu_{rt})}{B \cdot h}\right)^w$. Therefore, the condition in Equation 5.9 holds with probability at least $1 - \left(\frac{(|R_i|+1) \cdot (\sum_{t=1}^{s_r} \nu_{rt})}{B \cdot h}\right)^w$. By substituting the value of B in the above equation, we get the desired result. □

The above result suggests that the value of the similarity can be estimated quite accurately with the use of modest memory requirements of a sketch table. For example, consider a tweet with $|R| \approx 100$, and a similarity estimation bound of $\epsilon = 0.001$. If we use a sketch table with $h = 200,000$ and $w = 5$ (typical values), it will require a storage of only 1 million cells, which is in the megabyte order. The similarity estimate lies within $\epsilon = 0.001$ of the true value with probability at least $1 - (1/20)^5 > 1 - 10^{-6}$. In practice, since these theoretical bounds are quite loose, the estimates are much better.

5.3.2 Event Detection with Clustering

The clustering method discussed above can be used directly in order to perform the event detection. As discussed before, the event detection algorithm uses a time horizon H as the input

which is used for the event detection process. In order to perform the event detection, we monitor the ratio $\frac{F(t_c-H,t_c,\mathcal{C}_i)}{F(t(\mathcal{C}_i),t_c-H,\mathcal{C}_i)}$ for each cluster $\mathcal{C}_i$ continuously over time, and trigger an alarm whenever this ratio exceeds the threshold of α. This suggests a significant change in the underlying social stream, which is detected by a significant change in the ratios of stream objects being assigned to the different clusters.

Supervised Event Detection

The afore-mentioned approach discusses the case of *unsupervised* event detection in which we only try to find significant events in the social network without any particular regard to their nature. In practice, one may want to detect *known events* which have been encountered earlier in the stream. This is the case of *supervised event detection*. In this case, we assume that we have access to the past history of the stream in which the event $\mathcal{E}$ has been known to have occurred. In addition, we have information about at least a subset of the social stream tweets which are relevant to this particular event. This is the ground truth which can be leveraged for more accurate event detection.

In order to perform supervised event detection, we need to make some changes to the clustering portion of the algorithm as well. One major change is that we do not allow replacement of old clusters or creation of new clusters when a new incoming point does not naturally fit in any cluster. Rather, it is always assigned to its closest cluster, with any ties broken randomly. This is done in order to provide stability to the clustering characteristics, and is essential for characterizing the events in a consistent way over time with respect to the clusters in the underlying data.

The *relative distribution of event-specific stream objects* to clusters is used as a signature which is specific to the event. These can be used in order to perform the detection in real time. The assumption in the supervised case is that the training data about the social stream objects which are related to the event are available in the historical training data. The signature of the event $\mathcal{E}$ is defined as follows.

Definition 23 (Event Signature)**.** *The event signature of a social stream is a k-dimensional vector $V(\mathcal{E})$ containing the (average) relative distribution of event-specific stream objects to clusters. In other words, the ith component of $V(\mathcal{E})$ is the fraction of event-specific (training) stream objects which are assigned cluster i.*

Clearly, the event signature provides a useful characterization of the relative topical distribution during an event of significance. For example, during a period of mideast unrest (the event $\mathcal{E}$), some clusters are likely to be much more active than others, and this can be captured in the vector $V(\mathcal{E})$, as long as ground truth is available to do so. The event signatures can be compared to *horizon signatures*, which are essentially defined in the same way as the event signatures, except that they are defined over the more recent time horizon $(t_c - H, t_c)$ of length H.

Definition 24 (Horizon Signature)**.** *The horizon signature over the last time period* $(t_c - H, t_c)$ *is a* k*-dimensional vector containing the relative distribution of social stream objects to clusters which have arrived in the period* $(t_c - H, t_c)$.

In order to perform the supervised event detection, we simply compute the dot product of the horizon signature with the known event signature (which was computed by the ground truth), and output an alarm level which is equal to this value. The tradeoff between false positives and false negatives is determined by the threshold chosen to decide when such an event has really occurred.

5.4 Experimental Results

In this section, we will study the clustering and event detection algorithms for effectiveness and efficiency. We used two real data sets to evaluate the effectiveness of our approach.

5.4.1 Data Sets

Our algorithm was tested on the following two data sets:

Twitter Social Stream: This is a stream of tweets which was crawled from the *Twitter* social network. Each social object contains the network structure and content in a tweet. The stream contained a total of 1,628,779 tweets, which were distributed over a total of 59,192,401 nodes. The nodes include the sender and the receivers either in the form of direct mentions from senders or their followers in the case of broadcast messages. On the average, each stream object contained about 84 nodes per tweet.

Enron Email Stream: The well known Enron email data set[2] was converted to a stream with

[2] `http://www.cs.cmu.edu/~enron/`

the use of the time stamp information in the emails. Each object contained the text of the email, and the network structure corresponding to the sender and receiver(s). In this sense, the network structure of an email is very similar to a tweet with a single sender and multiple receivers. The *Enron* email data stream contained a total of 517,432 emails. We eliminated emails that did not have valid sender and receiver email identifiers. We also filtered out the calender invites, duplicate emails, and the email history at the bottom of each email. The total emails after the filtering process were 349911, which were distributed over a total of 29,083 individuals. On the average, each email contained 3.62 receivers.

5.4.2 Evaluation Measures

We tested both the clustering and event detection methods for effectiveness. For the case of efficiency, we tested only the clustering method, because the majority of the time for event detection was spent in the clustering process. In order to test the effectiveness, we used a number of class labels which were associated with the objects in the social stream. For the case of the *Twitter* stream, these class labels were the hash tags which were associated with the tweet. The most frequent hash tags in the stream often correspond to events in the stream such as *Uganda protests* or the *Japan Earthquake*, and represent meaningful characteristics of the underlying objects which ought to be clustered together. These hash tags also often represent the meaningful events in the stream. We note that not every tweet may contain hash tags, and therefore, the hash tags were only associated with a subset of the tweets. For the case of the *Enron* email stream, the class labels were defined by the most frequently occurring tokens in the subject line. These tokens were as follows:

meeting, agreement, gas, energy, power, report, update, request, conference, letter, deal, credit, california, trading, contract, project, presentation, houston, announcement

All emails which contained any of the above tokens in the subject line were tagged with the corresponding class label. Thus, for both data streams, a subset of the stream objects were tagged with a class label. Clearly, clusters of high quality would tend to put objects with similar tags in the same cluster. Therefore, we computed the dominant class purity of each cluster. For each cluster, we determined the tag with the highest presence, and computed the fraction of the (tagged) cluster objects, which belonged to that label. This value was averaged over the different clusters in a weighted way, where the weight of a cluster was proportional to the number of (tagged) objects in it.

We also tested the efficiency of the social stream clustering process. In order to test efficiency, we computed the number of social stream objects processed per unit of time, and we presented this number for the different algorithms.

We also also tested the effectiveness of the event detection algorithm. For the case of the unsupervised algorithm, we will present a case study, which illustrates the interesting events found by the technique. This provides an intuitive idea about the effectiveness of the technique. For the supervised algorithm, we first need to generate the ground truth for the supervised algorithm. For this purpose, we used the hash tags in the *Twitter Stream* in order to generate a 0-1 bit stream corresponding to when the events actually occurred. We used the two hash tags corresponding to *#japan* and *#uganda* in order to generate the events in our *Twitter* data stream. Specifically, at each time stamp, we looked at the past window of length h and counted the number of occurrences of a particular hash tag in that window. If the number of occurrences of the hash tag was at least 3, then we generated a bit of 1 at that time stamp to indicate that the event has indeed occurred.

We note that our supervised event detection algorithm generates a *continuous* alarm level. It is possible to use a threshold t on this continuous alarm level in order to generate a 0-1 bit stream corresponding to the *algorithmic prediction* of when the event has occurred. By using different thresholds t on this alarm level, we can obtain different tradeoffs between precision and recall. Let $S_F(t)$ be the set of time stamps at which an alarm is generated with the use of threshold value of t on the real alarm level. Let S_G be the ground truth set of time stamps at which the event truly occurs. Then, $Precision(t)$ and $Recall(t)$ can be computed as follows:

$$Precision(t) = \frac{|S_F(t) \cap S_G|}{|S_F(t)|} \tag{5.10}$$

$$Recall(t) = \frac{|S_F(t) \cap S_G|}{|S_G|} \tag{5.11}$$

$$\tag{5.12}$$

We presented the tradeoff between the precision and recall by varying the value of t in order to generate a plot between the two.

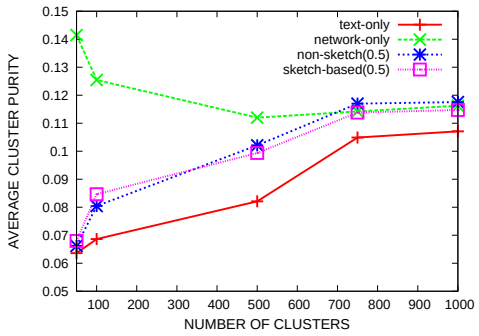
5.4.3 Algorithm Variations and Baseline

We tested different algorithm settings to determine the effect of content and network structure on accuracy and efficiency. We note that by setting λ to the extreme values of 0 and 1, we can

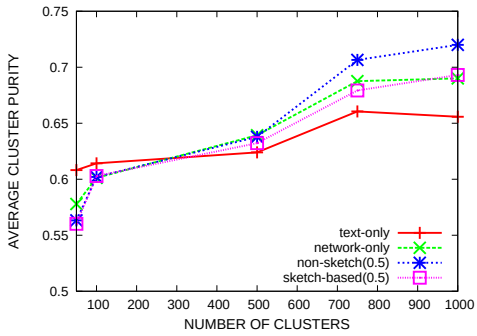
test how well the algorithm performs, when we use either the network only, or the text only for the clustering process. We also tested a combination scheme in which the value of λ was set to 0.5. This kind of scheme provides equal weight to the text and content in the clustering and event detection process. For the combination scheme, we tested it both with and without the sketch in order to estimate the effects of sketch use on the scheme in terms of accuracy and efficiency. We note that the text-only variation can be considered quite similar to the stream text-clustering approach discussed in [118]. Therefore, this variation also serves as a good baseline, because the use of pure text content is the only natural alternative for this problem at this juncture.

5.4.4 Effectiveness Results for Clustering

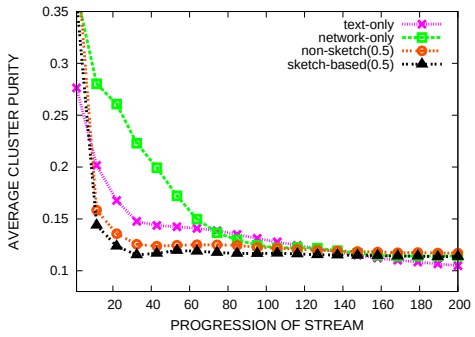
First, we will present the effectiveness results of the clustering algorithm in terms of the cluster purity. We tested the effectiveness of the approach with increasing number of clusters. The results for the *Twitter* and *Enron* streams are illustrated in Figure 5.3(a) and (b) respectively. The sketch table length h was set to 262,213, whereas the sketch table width w was set to 2 for the *Twitter* social stream, and these values were set to 16,369 and 2 for the *Enron* data stream. In each case, we have illustrated the number of clusters on the X-axis, and the cluster purity on the Y-axis. A very interesting trend was observed for both the data sets in terms of the relative performance of the different algorithms. In all cases, the algorithm which used only text performed the worst among all the algorithms. The trends between the purely network-based approach and combined approach were dependent upon the level of granularity at which the clustering was performed. For both data streams, we found that when a small number of clusters were used, the network-based approach was superior. When a larger number of clusters were used, the combination methods outperformed the purely network-based approach. This is because the network locality information is more than sufficient to effectively partition the clusters, when the granularity is relatively coarse. In such cases, the addition of text does not improve the quality of the underlying clusters, and can even be detrimental to clustering quality. However, when the number of clusters increases, the combination approach tends to perform better, because the granularity of the clusters is much higher, and therefore more attributes are required to distinguish between the different clusters. This is particularly evident in the case of the *Enron* data stream, in which the gap between the combination-approach and purely network-based approach is rather large. In all cases, we found that the use of sketches lead to



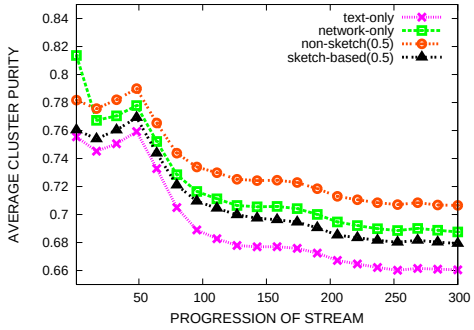
(a) Purity with Number of Clusters (*Twitter*)



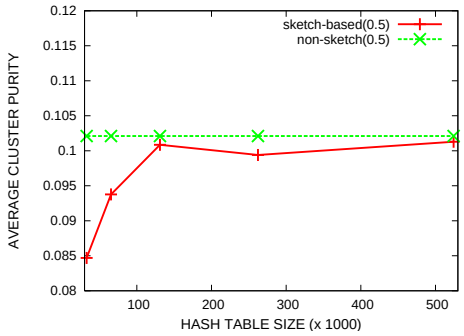
(b) Purity with Number of Clusters (*Enron*)



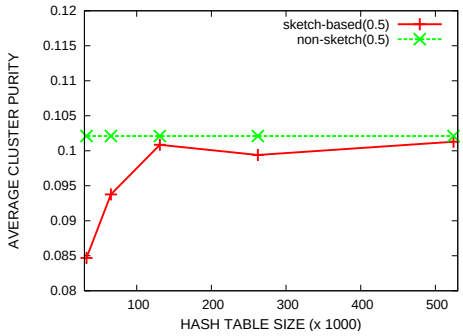
(c) Purity with Stream Progression (*Twitter*)



(d) Purity with Stream Progression (*Enron*)



(e) Hash Table Length Sensitivity (*Twitter*)



(f) Hash Table Length Sensitivity (*Twitter*)

Figure 5.3: Effectiveness Results for Clustering

some loss of accuracy. However, this loss of accuracy is not very significant, especially when we consider the fact that the sketch-based approach was significantly faster. It is also important to note that the pure text-based approach (which is our baseline) performed the worst in all scenarios. These results also seem to suggest that the network information in the social stream provides much more powerful information for clustering as compared to the text-information. This is not particularly surprising for sources such as the *Twitter* data stream in which the text is rather noisy and often contains non-standard acronyms or other text which are hard to use in a meaningful way. However, we found it somewhat interesting and surprising that these trends were also true for a source such as the *Enron* data stream, in which the text was relatively clean and was usually quite informative for the underlying class labels.

We have also illustrated the cluster purity with progression of the stream. This provides a *dynamic idea* of how the clustering process performed during the progression of the stream itself. The results for the *Twitter* and *Enron* data streams are illustrated in Figures 5.3(c) and (d) respectively. The number of clusters was fixed to 750. For the *Twitter* data stream the sketch table length was fixed to 262,213 and for *Enron* it was set to 16,369. The sketch table width was fixed to 2 in both cases. The X-axis illustrates the progression of the stream for every 1000 objects processed, and the Y-axis illustrates the cluster purity in the last time window of 1 hour. The relative trends between the different methods are similar to our earlier observations, though the main observation is that the cluster purities generally reduce with progression of the stream. The reason for this is that the number of class labels in the stream generally increase over the progression of the stream, as new classes, tags and events are encountered. As a result, the cluster purity also generally reduces with stream progression.

Finally, we also tested the sensitivity of the approach with sketch-table length and width. The sensitivity results with increasing sketch-table length for the *Twitter* data stream is illustrated in Figures 5.3(e). The sketch table length is illustrated on the X-axis, whereas the cluster purity is illustrated on the Y-axis. The number of clusters was set to 500 in this case and sketch table width was set to 2. We have also shown the results for the method which does not use the sketch in the same figure in order to provide a baseline for the relative effectiveness of the incorporation of the sketch structure. It is clear that the cluster-purity increases with increasing sketch-table length. This is because a larger sketch-table length reduces the number of collisions in the sketch table, and it therefore improves the overall accuracy. We also note that when the sketch-table length is increased sufficiently, the accuracy of the approach based on

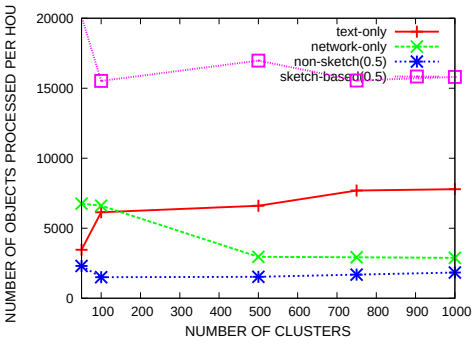
the sketch-table approaches that of the method which does not use the sketch table. In such cases, the collisions reduce sufficiently to the point that there of reduction in accuracy because of sketch=table use.

We also tested the sensitivity of the approach with increasing sketch-table width. The sensitivity results with increasing sketch-table width for the *Twitter* data stream is illustrated in Figures 5.3(f). The number of clusters was set to 500 and sketch-table length was set to 262,213. While the effectiveness results improve with increasing sketch-table width, it is evident that the purity results are not quite as sensitive to sketch-table width, as they were to the sketch-table length. This is because an increase in the number of hash functions provides additional robustness, but it does not drastically reduce the number of collisions between the different items.

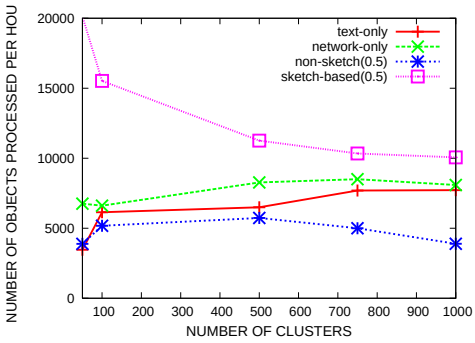
5.4.5 Efficiency Results for Clustering

We also tested the efficiency of the clustering approach with increasing number of clusters. The parameter settings were the same as those of the effectiveness results. The efficiency results for the *Twitter* and *Enron* data streams are illustrated in Figures 5.4(a) and (b) respectively. The X-axis denotes the number of clusters, whereas the Y-axis denotes the number of stream objects processed every hour. It is evident that the network-based approach was much slower than the text-based approach. This is because of the large number of distinct nodes which need to be processed by a text-based approach. However, the sketch-based approach is significantly faster in both data streams, because of the fact that the number of operations in the similarity computation are reduced by the sketch representation. Another interesting observation is that the processing rate did not necessarily reduce with increasing number of clusters. Much of this was also because the a larger number of clusters resulted in faster similarity computations between incoming objects and the underlying clusters. This is because a larger number of clusters resulted in sparser clusters with fewer number of objects in each cluster.

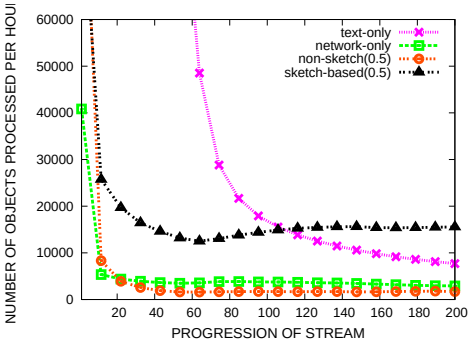
We also tested the efficiency with stream progression and present the results in Figure 5.4(c) and (d) respectively. The stream progression is illustrated on the X-axis, and the processing rate is illustrated on the Y-axis. It is evident that the processing rate reduces with stream progression for all the different methods. This is because the clusters contain a larger number of objects with progression of the stream. This increases the complexity of the similarity computations because the number of attributes in each cluster (in terms of the number of text words or nodes) increases as well. However, this slow-down levels out after a certain point as the number of attributes in



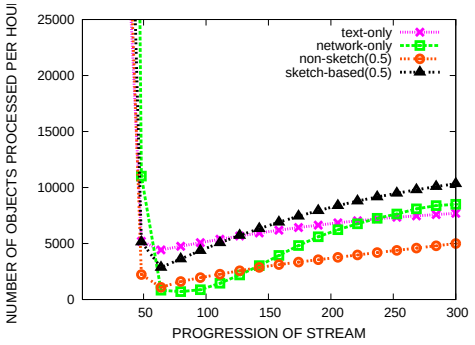
(a) Efficiency with Number of Clusters (Twitter)



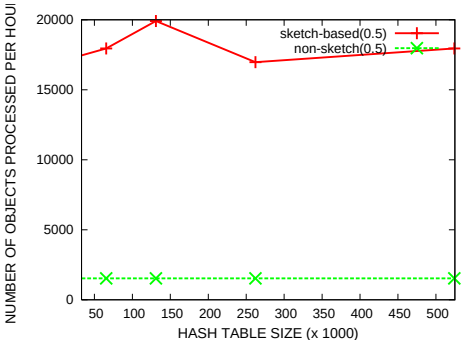
(b) Efficiency with Number of Clusters (Enron)



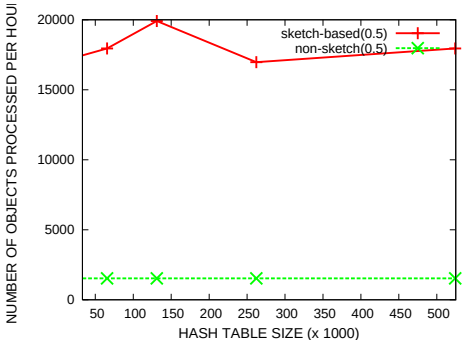
(c) Efficiency with Stream Progression (Twitter)



(d) Efficiency with Stream Progression (Enron)



(e) Hash Table Length Sentivity (Twitter)



(f) Hash Table Length Sentivity (Enron)

Figure 5.4: Efficiency Results for Clustering

each cluster stabilizes.

We also tested the sensitivity of the approach with hash-table length and width. The sensitivity results with hash-table length and width for the *Twitter* data streams are illustrated in Figures 5.4(e), and (f). It is evident from the figures that the running time is not very sensitive to the hash table length, and most of the variations are really random variations. On the other hand, the hash table width affects the number of hash functions which need to be computed. Therefore, the running time generally reduces with increasing hash table width. These results seem to suggest that a greater hash table width is not particularly useful because it does not affect the purity very much, but it affects the efficiency substantially. Therefore additional memory should be used for increasing hash table length rather than width.

5.4.6 Unsupervised Event Detection Case Study

In this section, we provided a case study of the unsupervised event detection problem. Both evolutionary and novel events were detected with the use of this approach. Typically, such events were associated with a particular news, country, language or mood.

The first event was related to the Japan nuclear crisis. In particular, the relevant segment of the social stream corresponds to the case where the Japanese Prime Minister requested the Chubu electric company to shut down the Hamaoka nuclear plant. This event generated considerable chatter in the social stream, in which the underlying actors were discussing the positives and negatives of this directive. The corresponding portion of the social stream contained structural nodes which were geographically biased towards Japan, and generally created clusters of their own on the basis of the underlying network and content information. The frequent text content in these clusters included the following words:

nuclear, hamaoka, plant, concerns, dilemma, gaswat, hairline, halt, shut, heavy, japan, neglect, operation (1.152)

One interesting aspect of our algorithm was that it was able to detect events, for which the content was in a foreign language. The reason for this is that the event detection algorithm does not use any methods which are specific to English, and the use of network structure is blind to the use of a specific language. For example, the minister for finance in Indonesia issued an order on May 9, 2011 to buy 7% of the shares in PT Newmont Nusa Tenggara company. This triggered discussion threads in twitter which were captured because of their related content and network structure. The frequent text-content which was associated in the cluster most related to

this event was the following:

keputusan, menkeu, beli, newmont, saham, wapres, didukung, challenge, minskade, metro, jak, kimiad, menos (0.9964)

We note that the entire text content in this event consists of foreign language words.

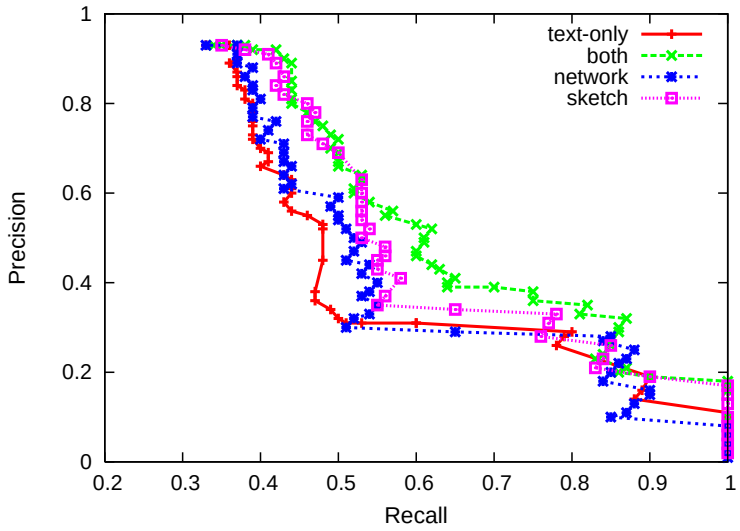
Our event detection algorithm was also capable of connecting related events. This is because the actors and events are often overlapping in such cases. This overlap could result in the placement of closely related events in the same cluster. An example of this were the protests on May 9, 2011 on the Gay bill death penalty in Uganda. Many of these same actors were also discussing issues related to the Kentucky anti-gay marriage law, even though the two events were geographically quite disparate. As a result, these two related events were reported as a single composite event, since they reflected issues which were discussed in relation to one another by the participants. The corresponding content words were as follows:

protest, power, sign, identity, much, please, political, citizens, petition, stop, kill, gays, bill, kentucky, progress, support (0.6490)

Thus, our unsupervised event detection approach was capable of discovering interesting and novel events in the underlying social stream. Such inference can be very useful in diagnosing important social moods which are related to real events.

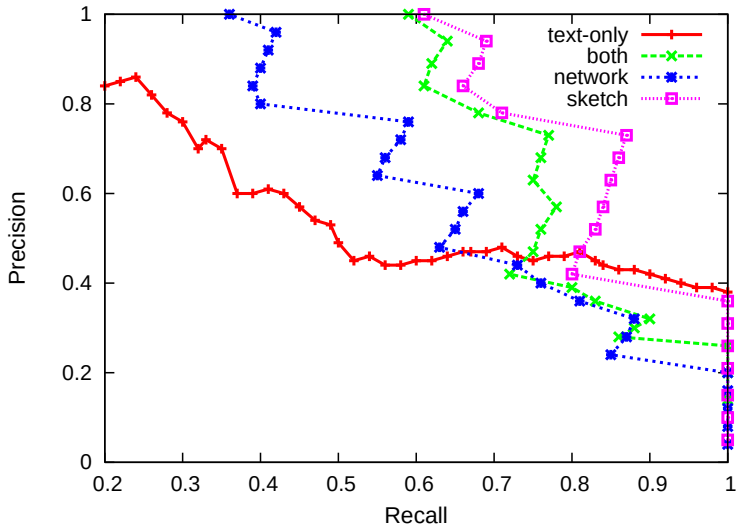
5.4.7 Supervised Event Detection

We also tested the supervised event detection method on the *Twitter* stream. To create the supervised events, we set the horizon to 5 minutes and determined the periods in which events corresponding to the *Japan Nuclear Crisis* and *Uganda Protest* occurred in the data stream. In each case, we set the number of clusters to 750 in order to detect the underlying events. We illustrated the tradeoff between the precision and recall on Figures 5.5(a) and (b) respectively. The recall is illustrated on the X-axis, whereas the precision is illustrated on the Y-axis. In each case, we have implemented the event detection algorithm with different variations of the algorithm. It is clear that the use of only text did not provide as accurate an event detection than all the methods which used network structure in the event detection process. In particular, the method which used both the network structure and the text content provided the most accurate results. The use of sketches degraded the accuracy to some extent, but the approach was still more accurate than that with the use of pure text. It is also evident that the absolute values of precision and recall were fairly high. For example, in the case of *Japan Nuclear event*, a



(a) Supervised Event Detection

(Japan Nuclear Crisis)



(b) Supervised Event Detection

(Uganda Protests)

Figure 5.5: Supervised Event Detection

precision of 0.525 was obtained at a recall point of approximately 0.62, when both network and text were used. On the other hand, the scheme which used only either text or network achieved a precision of about 0.3 at a recall point of around 0.6.

In the case of *Uganda Protests*, the differences between the different methods were even sharper. Both combination methods were extremely accurate, and maintained a precision of 1 at a recall point of around 0.6. This essentially means that the top reported alarms (in terms of alarm magnitude) were all correct up to the point where 60% of the correct alarm points were detected. On the other hand, the text-based baseline provided a precision of only about 0.5 at the same recall point. This means that about 50% of the reported alarms were incorrect at the same recall point. The purely network-based method was also not significantly better in this case. Thus, the use of a combination of network and text greatly improved the accuracy of the event detection algorithm. These results seem to suggest that our approach of combining network and text content for clustering and event detection can provide useful and accurate results in a wide variety of scenarios.

5.5 Conclusions and Summary

In this paper, we proposed new methods for clustering and event-detection in social streams using information flows. We show that the use of content- and flow-based clustering and event detection has a number of fundamental advantages which cannot be easily handled by pure text-based methods, which are the current state-of-the-art. Our results suggest that social streams can be used as a valuable resource to monitor and detect relevant and interesting events in the social stream. We present experimental results on a number of real social streams, which illustrate the effectiveness of our approach.

Chapter 6

Conclusions and Future Work

Social and collaboration networks are rich in content, due to exchange of large volumes of tweets, messages, and other textual information. It is imperative for all social network applications to take advantage of this rich content in order to improve the overall effectiveness of the underlying clustering, classification or mining approach. Content is useful, however, content alone does not complete the story of network analysis. The temporal dynamics of the content activities, occurring due to varying activity levels and content life-time for each user, plays a vital role in completing the story of network analysis. In this context, information flow acts as an abstraction of content, network and temporal dynamics of the social and collaboration network activities in an unified fashion. In this thesis, we focused on the problem of extracting such information flows, especially their algorithms and applications.

We first introduced the abstraction of information flows and the problem of information flow mining in Chapter 2. We then established its complexity class as #P-complete by providing a polynomial time reduction algorithm – to reduce a traditional counting version of a maximal sequence mining problem to information flow mining problem. This was discussed in Chapter 3. We propose an algorithm for extracting information flow patterns from a static dataset using a content-centric approach, where we developed an efficient network pruning strategy along with pattern extraction. In Chapter 3, we demonstrated the effectiveness of flows extracted in the context of influence mining application. We also showed, in Chapter 2, how flow-based ego-networks can be useful in identifying key *information propagandists* of a focal node.

The size of social networks is extremely large and correspondingly the volume of content generated every day is hundreds of millions of messages. Designing scalable methods for

such large social media and network was another important goal of this thesis. For example, there are several popular algorithms for influence analysis and link prediction, but they are often computationally intractable for large datasets with millions of nodes and trillions of edges. We designed scalable algorithms that process the data in an online fashion using a streaming algorithm for more real-time applications and for batch modes we use a distributed computational framework, such as GraphLab. The distributed and real-time algorithms were discussed in Chapter 3 and 4 respectively. Scalability always comes at a cost of approximation, either by randomization or pruning. We used techniques from data stream analysis and provided the necessary optimality bounds for all our approximation in streaming algorithms. In event detection, our algorithm discovers events by maintaining temporally evolving cluster approximations, with one-pass constraint over the streaming social media content. This was discussed in detail in Chapter 5. On the other hand, influence scores are often computed at periodic intervals in a batch-mode. For this application, we designed a distributed approach that computes the influence scores parallelized at vertex-level using vertex-centric computational models, such as GraphLab.

Overall, this thesis presents a set of information flow mining algorithms, of high practical value, for analyzing large scale social and collaboration networks, using content, network and temporal information in a unified way. We developed a sequential, distributed and streaming algorithm for different data sets based on their volume, velocity and variety. We used relevant applications, along with various algorithms, to illustrate the effectiveness of the discovered information flows and its relevance to network analysis.

Here are a list some more applications that might be of interest to the readers, where the extracted information flows can be extremely helpful.

Community Boundaries: In several real-life instances, a particular idea captures the attention of a community and spreads like wild fire with in the community. The spread of these ideas outside the communication boundary is not so fast or wide-range as with-in the communities. One can make use of these properties to identify the information flow-based community structures in large networks. The flow-based ego networks discovered using our approach naturally captures the communication boundaries. It would be interesting to analyze and build community detection algorithms using these boundaries of communication.

Graph Sparsification: There are several graph sparsification techniques for sampling the underlying graph with specific properties, such as degree distribution. The approach proposed

in this paper samples the graph using the dominant directions of content flows. Understanding the properties of this backbone structure and using this for scalable graph sampling purposes is extremely useful for development and testing of large social network applications.

Content Recommendation: Recommending the relevant posts to the social stream of an user is an important problem in online social networks. As several information flow patterns are extracted using our approach, one can cluster them, identify key characteristics and recommend content across similar flow channels to expedite the flow of information in networks. In essence, rather than designing content recommendation at a node level one could design recommendation engines for similar content channels.

References

[1] Stanley Wasserman. *Social network analysis: Methods and applications*, volume 8. Cambridge university press, 1994.

[2] Jure Leskovec, Lada A Adamic, and Bernardo A Huberman. The dynamics of viral marketing. *ACM Transactions on the Web (TWEB)*, 1(1):5, 2007.

[3] Pedro Domingos and Matt Richardson. Mining the network value of customers. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 57–66. ACM, 2001.

[4] `http://www.uspto.gov`.

[5] `http://www.informatik.uni-trier.de/~ley/db/`.

[6] `http://researchgate.com/`.

[7] `http://arnetminer.org/`.

[8] Kristina Lerman and Rumi Ghosh. Information Contagion: An Empirical Study of the Spread of News on Digg and Twitter Social Networks. In *ICWSM*, 2010.

[9] Eytan Bakshy, Jake M Hofman, Winter A Mason, and Duncan J Watts. Everyone's an influencer: quantifying influence on twitter. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 65–74. ACM, 2011.

[10] Ting-Kai Huang, Md Sazzadur Rahman, Harsha V Madhyastha, Michalis Faloutsos, and Bruno Ribeiro. An analysis of socware cascades in online social networks. In *Proceedings of the 22nd international conference on World Wide Web*, pages 619–630. International World Wide Web Conferences Steering Committee, 2013.

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

[12] Nathan Eagle, Alex Sandy Pentland, and David Lazer. Inferring friendship network structure by using mobile phone data. *Proceedings of the National Academy of Sciences*, 106(36):15274–15278, 2009.

[13] Manuel Gomez-Rodriguez, Jure Leskovec, and Andreas Krause. Inferring networks of diffusion and influence. In *KDD*, pages 1019–1028, 2010.

[14] Jure Leskovec, Mary McGlohon, Christos Faloutsos, Natalie Glance, and Matthew Hurst. Cascading Behavior in Large Blog Graphs. In *SDM*, 2007.

[15] Jure Leskovec, Lars Backstrom, and Jon M Kleinberg. Meme-tracking and the dynamics of the news cycle. In *KDD*, pages 497–506, 2009.

[16] Jure Leskovec, Mary McGlohon, Christos Faloutsos, Natalie S Glance, and Matthew Hurst. Patterns of Cascading Behavior in Large Blog Graphs. In *SDM*, volume 7, pages 551–556. SIAM, 2007.

[17] `https://www.cs.cmu.edu/~./enron/`.

[18] Giuliano Andrea Pagani and Marco Aiello. The power grid as a complex network: a survey. *arXiv preprint arXiv:1105.3338*, 2011.

[19] Noel M Tichy, Michael L Tushman, and Charles Fombrun. Social network analysis for organizations. *Academy of management review*, 4(4):507–519, 1979.

[20] Duncan J Watts. *Six degrees: The science of a connected age*. WW Norton & Company, 2004.

[21] Duncan J Watts and Steven H Strogatz. Collective dynamics of small-worldnetworks. *nature*, 393(6684):440–442, 1998.

[22] Paul Erdős and Alfréd Rényi. On the strength of connectedness of a random graph. *Acta Mathematica Hungarica*, 12(1):261–267, 1961.

[23] Justin Cheng, Lada Adamic, P Alex Dow, Jon Michael Kleinberg, and Jure Leskovec. Can cascades be predicted? In *Proceedings of the 23rd international conference on World wide web*, pages 925–936. International World Wide Web Conferences Steering Committee, 2014.

[24] Wojciech Galuba, Karl Aberer, Dipanjan Chakraborty, Zoran Despotovic, and Wolfgang Kellerer. Outtweeting the twitterers-predicting information cascades in microblogs. In *WOSN)*, 2010.

[25] Michael R Heithaus, Alejandro Frid, Aaron J Wirsing, and Boris Worm. Predicting ecological consequences of marine top predator declines. *Trends in Ecology & Evolution*, 23(4):202–210, 2008.

[26] Karthik Subbian, Charu C Aggarwal, and Jaideep Srivastava. Content-centric Flow Mining for Influence Analysis in Social Streams. In *CIKM*, 2013.

[27] Jianshu Weng and Bu-Sung Lee. Event detection in twitter. *ICWSM*, 11:401–408, 2011.

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

[29] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 137–146. ACM, 2003.

[30] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. Cost-effective outbreak detection in networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 420–429. ACM, 2007.

[31] Wei Chen, Chi Wang, and Yajun Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *KDD*, pages 1029–1038. ACM, 2010.

[32] Chi Wang, Jie Tang, Jimeng Sun, and Jiawei Han. Dynamic social influence analysis through time-dependent factor graphs. In *Advances in Social Networks Analysis and Mining (ASONAM), 2011 International Conference on*, pages 239–246. IEEE, 2011.

[33] Smriti Bhagat, Amit Goyal, and Laks V S Lakshmanan. Maximizing product adoption in social networks. In *WSDM*, pages 603–612. ACM, 2012.

[34] Eytan Adar and Lada Adamic. Tracking Information Epidemics in Blogspace. In *Web Intelligence*, pages 207–214, 2005.

[35] Masahiro Kimura and Kazumi Saito. Tractable models for information diffusion in social networks. In *Knowledge Discovery in Databases: PKDD 2006*, pages 259–271. Springer, 2006.

[36] Amit Goyal, Wei Lu, and Laks V S Lakshmanan. Celf++: optimizing the greedy algorithm for influence maximization in social networks. In *Proceedings of the 20th international conference companion on World wide web*, pages 47–48. ACM, 2011.

[37] Amit Goyal, Wei Lu, and Laks V S Lakshmanan. Simpath: An efficient algorithm for influence maximization under the linear threshold model. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on*, pages 211–220. IEEE, 2011.

[38] Wei Chen, Yajun Wang, and Siyu Yang. Efficient influence maximization in social networks. In *KDD*, pages 199–208. ACM, 2009.

[39] Charu Aggarwal and Karthik Subbian. Event Detection in Social Streams. In *SDM*, pages 624–635, 2012.

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

[41] James Allan, Ron Papka, and Victor Lavrenko. On-line new event detection and tracking. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 37–45. ACM, 1998.

[42] James Allan, Victor Lavrenko, and Hubert Jin. First story detection in TDT is hard. In *Proceedings of the ninth international conference on Information and knowledge management*, pages 374–381. ACM, 2000.

[43] Thorsten Brants, Francine Chen, and Ayman Farahat. A system for new event detection. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 330–337. ACM, 2003.

[44] April Kontostathis, Leon M Galitsky, William M Pottenger, Soma Roy, and Daniel J Phelps. A survey of emerging trend detection in textual data mining. In *Survey of Text Mining*, pages 185–224. Springer, 2004.

[45] Qiaozhu Mei and ChengXiang Zhai. Discovering evolutionary theme patterns from text: an exploration of temporal text mining. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 198–207. ACM, 2005.

[46] Yiming Yang, Tom Pierce, and Jaime Carbonell. A study of retrospective and on-line event detection. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 28–36. ACM, 1998.

[47] Yiming Yang, Jian Zhang, Jaime Carbonell, and Chun Jin. Topic-conditioned novelty detection. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 688–693. ACM, 2002.

[48] Deepayan Chakrabarti, Yang Wang, Chenxi Wang, Jurij Leskovec, and Christos Faloutsos. Epidemic thresholds in real networks. *ACM Transactions on Information and System Security (TISSEC)*, 10(4):1, 2008.

[49] Jon Kleinberg. The flow of on-line information in global networks. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 1–2. ACM, 2010.

[50] Mark E J Newman, Stephanie Forrest, and Justin Balthrop. Email networks and the spread of computer viruses. *Physical Review E*, 66(3):35101, 2002.

[51] Yang Wang, Deepayan Chakrabarti, Chenxi Wang, and Christos Faloutsos. Epidemic spreading in real networks: An eigenvalue viewpoint. In *Reliable Distributed Systems, 2003. Proceedings. 22nd International Symposium on*, pages 25–34. IEEE, 2003.

[52] Chenxi Wang, John C Knight, and Matthew C Elder. On computer viral infection and the effect of immunization. In *Computer Security Applications, 2000. ACSAC'00. 16th Annual Conference*, pages 246–256. IEEE, 2000.

[53] Fang Wu, Bernardo A Huberman, Lada A Adamic, and Joshua R Tyler. Information flow in social groups. *Physica A: Statistical Mechanics and its Applications*, 337(1):327–335, 2004.

[54] Mark E J Newman. Spread of epidemic disease on networks. *Physical review E*, 66(1):16128, 2002.

[55] Xiaoxiao Shi, Wei Fan, Jianping Zhang, and Philip Yu. Discovering shakers from evolving entities via cascading graph inference. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1001–1009. ACM, 2011.

[56] Prem Melville, Karthik Subbian, C Perlich, R Lawrence, and E Meliksetian. A predictive perspective on measures of influence in networks. In *Proceedings of the Workshop on Information in Networks*, 2010.

[57] Karthik Subbian, Dhruv Sharma, Zhen Wen, and Jaideep Srivastava. Social capital: the power of influencers in networks. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 1243–1244. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

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

[59] Amit Goyal, Francesco Bonchi, and Laks V S Lakshmanan. A data-based approach to social influence maximization. *Proceedings of the VLDB Endowment*, 5(1):73–84, 2011.

[60] Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M. Hellerstein. Graphlab: A new framework for parallel machine learning. *arXiv:1006.4990*, 2010.

[61] Lu Liu, Jie Tang, Jiawei Han, and Shiqiang Yang. Learning influence from heterogeneous social networks. *Data Mining and Knowledge Discovery*, 25(3):511–544, 2012.

[62] Nitin Agarwal, Huan Liu, Lei Tang, and Philip S Yu. Identifying the influential bloggers in a community. In *Proceedings of the 2008 international conference on web search and data mining*, pages 207–218. ACM, 2008.

[63] Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna. Detecting spammers on social networks. In *Proceedings of the 26th Annual Computer Security Applications Conference*, pages 1–9. ACM, 2010.

[64] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functionsI. *Mathematical Programming*, 14(1):265–294, 1978.

[65] Jian Pei, Helen Pinto, Qiming Chen, Jiawei Han, Behzad Mortazavi-Asl, Umeshwar Dayal, and Mei-Chun Hsu. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, page 215. IEEE Computer Society, 2001.

[66] Manuel Gomez-Rodriguez, Jure Leskovec, and Bernhard Scholkopf. Structure and dynamics of information pathways in online media. In *WSDM*, pages 23–32, 2013.

[67] Joseph E. Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. PowerGraph: Distributed graph-parallel computation on natural graphs. In *USENIX*, 2012.

[68] Boris Kholodenko, Michael B Yaffe, and Walter Kolch. Computational approaches for analyzing information flow in biological networks. *Science Signaling*, 5(220), 2012.

[69] Seth Myers, Chenguang Zhu, and Jure Leskovec. Information diffusion and external influence in networks. In *KDD*, pages 33–41, 2012.

[70] Michael Mathioudakis, Francesco Bonchi, Carlos Castillo, Aristides Gionis, and Antti Ukkonen. Sparsification of influence networks. In *KDD*, pages 529–537, 2011.

[71] Francesco Bonchi, Gianmarco De Francisci Morales, Aristides Gionis, and Antti Ukkonen. Activity preserving graph simplification. *Data Mining and Knowledge Discovery*, 27(3):321–343, 2013.

[72] Xuanhui Wang, ChengXiang Zhai, Xiao Hu, Richard Sproat, Xuanhui Wang, ChengXiang Zhai, Xiao Hu, and Richard Sproat. Mining correlated bursty topic patterns from coordinated text streams. In *KDD*, pages 784–793. ACM, 2007.

[73] Lilian Weng, Alessandro Flammini, Alessandro Vespignani, and Filippo Menczer. Competition among memes in a world with limited attention. *Scientific Reports*, 2, 2012.

[74] Guizhen Yang. The complexity of mining maximal frequent itemsets and maximal frequent patterns. In *SIGKDD*, pages 344–353, 2004.

[75] Amit Goyal, Francesco Bonchi, and Laks V S Lakshmanan. Learning influence probabilities in social networks. In *WSDM*, pages 241–250. ACM, 2010.

[76] Charu C Aggarwal, Arijit Khan, and Xifeng Yan. On Flow Authority Discovery in Social Networks. In *SDM*, pages 522–533. SIAM, 2011.

[77] B Aditya Prakash, Deepayan Chakrabarti, Nicholas C Valler, Michalis Faloutsos, and Christos Faloutsos. Threshold conditions for arbitrary cascade models on arbitrary networks. *Knowledge and information systems*, 33(3):549–575, 2012.

[78] Nitin Agarwal, Huan Liu, Lei Tang, and S Yu Philip. Modeling blogger influence in a community. *Social Network Analysis and Mining*, 2(2):139–162, 2012.

[79] Nicola Barbieri, Francesco Bonchi, and Giuseppe Manco. Topic-aware social influence propagation models. *Knowledge and information systems*, 37(3):555–584, 2013.

[80] Bin Bi, Yuanyuan Tian, Yannis Sismanis, Andrey Balmin, and Junghoo Cho. Scalable topic-specific influence analysis on microblogs. In *WSDM*, pages 513–522. ACM, 2014.

[81] Lu Liu, Jie Tang, Jiawei Han, Meng Jiang, and Shiqiang Yang. Mining topic-level influence in heterogeneous networks. In *CIKM*, pages 199–208. ACM, 2010.

[82] Jie Tang, Jimeng Sun, Chi Wang, and Zi Yang. Social influence analysis in large-scale networks. In *KDD*, pages 807–816. ACM, 2009.

[83] Alan Ritter, Oren Etzioni, Sam Clark, et al. Open domain event extraction from twitter. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1104–1112. ACM, 2012.

[84] Michael S Bernstein, Bongwon Suh, Lichan Hong, Jilin Chen, Sanjay Kairam, and Ed H Chi. Eddi: interactive topic-based browsing of social status streams. In *User Interface Software and Technology*, pages 303–312, 2010.

[85] Konstantin Kutzkov, Albert Bifet, Francesco Bonchi, and Aristides Gionis. Strip: stream learning of influence probabilities. In *KDD*, pages 275–283. ACM, 2013.

[86] Honglei Zhuang, Yihan Sun, Jie Tang, Jialin Zhang, and Xiaoming Sun. Influence maximization in dynamic social networks. In *ICDM*, pages 1313–1318. IEEE, 2013.

[87] Cigdem Aslay, Nicola Barbieri, Francesco Bonchi, and Ricardo A Baeza-Yates. Online Topic-aware Influence Maximization Queries. In *EDBT*, pages 295–306, 2014.

[88] Daniel Ramage, Susan T Dumais, and Daniel J Liebling. Characterizing Microblogs with Topic Models. In *ICWSM*, 2010.

[89] Jie Tang, Duo Zhang, and Limin Yao. Social Network Extraction of Academic Researchers. In *ICDM'07*, pages 292–301, 2007.

[90] Charu C Aggarwal, Yan Xie, and S Yu Philip. On dynamic link inference in heterogeneous networks. In *SDM*, pages 415–426. SIAM, 2012.

[91] Lise Getoor, Nir Friedman, Daphne Koller, and Benjamin Taskar. Learning probabilistic models of relational structure. In *ICML*, volume 1, pages 170–177, 2001.

[92] Lisa Getoor, Nir Friedman, Daphne Koller, and Benjamin Taskar. Learning probabilistic models of link structure. *The Journal of Machine Learning Research*, 3:679–707, 2003.

[93] Ben Taskar, Ming-Fai Wong, Pieter Abbeel, and Daphne Koller. Link prediction in relational data. In *Advances in neural information processing systems*, page None, 2003.

[94] Yang Yang, Nitesh V Chawla, Yizhou Sun, and Jiawei Han. Predicting links in multi-relational and heterogeneous networks. In *ICDM*, volume 12, pages 755–764, 2012.

[95] Yizhou Sun, Rick Barber, Manish Gupta, Charu C Aggarwal, and Jiawei Han. Co-author relationship prediction in heterogeneous bibliographic networks. In *Advances in Social Networks Analysis and Mining (ASONAM), 2011 International Conference on*, pages 121–128. IEEE, 2011.

[96] Yizhou Sun, Jiawei Han, Charu C Aggarwal, and Nitesh V Chawla. When will it happen?: relationship prediction in heterogeneous information networks. In *Proceedings of*

the fifth ACM international conference on Web search and data mining, pages 663–672. ACM, 2012.

[97] Mohammad Al Hasan, Vineet Chaoji, Saeed Salem, and Mohammed Zaki. Link prediction using supervised learning. In *SDM06: Workshop on Link Analysis, Counter-terrorism and Security*, 2006.

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

[99] Hisashi Kashima and Naoki Abe. A parameterized probabilistic model of network evolution for supervised link prediction. In *Data Mining, 2006. ICDM'06. Sixth International Conference on*, pages 340–349. IEEE, 2006.

[100] Zhengdong Lu, Berkant Savas, Wei Tang, and Inderjit S Dhillon. Supervised link prediction using multiple sources. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 923–928. IEEE, 2010.

[101] Ryan N Lichtenwalter and Nitesh V Chawla. Lpmade: Link prediction made easy. *The Journal of Machine Learning Research*, 12:2489–2492, 2011.

[102] Ryan N Lichtenwalter and Nitesh V Chawla. Vertex collocation profiles: subgraph counting for link analysis and prediction. In *Proceedings of the 21st international conference on World Wide Web*, pages 1019–1028. ACM, 2012.

[103] Lise Getoor and Christopher P Diehl. Link mining: a survey. *ACM SIGKDD Explorations Newsletter*, 7(2):3–12, 2005.

[104] Charu C Aggarwal. *Data streams: models and algorithms*, volume 31. Springer, 2007.

[105] Charu C Aggarwal. *An introduction to social network data analytics*. Springer, 2011.

[106] D Blei and J Lafferty. Dynamic Topic Models. In *Proceedings of the 23rd International Conference on Machine Learning*, 2006.

[107] Qi He, Kuiyu Chang, Ee-Peng Lim, and Jun Zhang. Bursty Feature Representation for Clustering Text Streams. In *SDM*, pages 491–496. SIAM, 2007.

[108] Yu-Bao Liu, Jia-Rong Cai, Jian Yin, and Ada Wai-Chee Fu. Clustering text data streams. *Journal of computer science and technology*, 23(1):112–128, 2008.

[109] Arun C Surendran and Suvrit Sra. Incremental aspect models for mining document streams. In *Knowledge Discovery in Databases: PKDD 2006*, pages 633–640. Springer, 2006.

[110] Shi Zhong. Efficient streaming text clustering. *Neural Networks*, 18(5):790–798, 2005.

[111] Deepayan Chakrabarti, Ravi Kumar, and Andrew Tomkins. Evolutionary clustering. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 554–560. ACM, 2006.

[112] Yun Chi, Xiaodan Song, Dengyong Zhou, Koji Hino, and Belle L Tseng. Evolutionary spectral clustering by incorporating temporal smoothness. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 153–162. ACM, 2007.

[113] Tianbao Yang, Rong Jin, Yun Chi, and Shenghuo Zhu. Combining link and content for community detection: a discriminative approach. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 927–936. ACM, 2009.

[114] Yang Zhou, Hong Cheng, and Jeffrey Xu Yu. Graph clustering based on structural/attribute similarities. *Proceedings of the VLDB Endowment*, 2(1):718–729, 2009.

[115] Cindy Xide Lin, Bo Zhao, Qiaozhu Mei, and Jiawei Han. PET: a statistical model for popular events tracking in social communities. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 929–938. ACM, 2010.

[116] Gerard Salton and Michael J McGill. *Introduction to modern information retrieval*. McGraw Hill Publications, 1983.

[117] Graham Cormode and S Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[118] Charu C Aggarwal and Philip Yu. A Framework for Clustering Massive Text and Categorical Data Streams. In *SDM*, pages 479–483. SIAM, 2006.