Scalable Information Flow Mining in Networks

Karthik Subbian[1], Chidananda Sridhar[1], Charu C. Aggarwal[2], and Jaideep Srivastava[1]

[1] University of Minnesota, Minneapolis, MN, USA
{karthik,sridh050,srivasta}@cs.umn.edu
[2] IBM T. J. Watson Research Center, Yorktown Heights, NY, USA
charu@us.ibm.com

Abstract. The problem of understanding user activities and their patterns of communication is extremely important in social and collaboration networks. This can be achieved by tracking the dominant content flow trends and their interactions between users in the network. Our approach tracks all possible paths of information flow using its network structure, content propagated and the time of propagation. We also show that the complexity class of this problem is #P-complete. Because most social networks have many activities and interactions, it is inevitable the proposed method will be computationally intensive. Therefore, we propose an efficient method for mining information flow patterns, especially in large networks, using distributed vertex-centric computational models. We use the Gather-Apply-Scatter (GAS) paradigm to implement our approach. We experimentally show that our approach achieves over *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.

Keywords: Information Flow Mining, Vertex-centric models, Influence Analysis Network-centric approach, Scalable Influence Analysis.

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 [12], topics [1], *hashtags* [2], or *URLs* [7] 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 [21]. 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 [11,12], cascades [13], and events [2].

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

The existing literature on understanding information flows using cascades [11,12,13] and memes [11,12] analyze a stream or a corpus of text documents where there is no explicit network structure used for communication. For instance, the work in [13] determines the cascade patterns from the temporal sequence of blog posts across multiple blogs. Here, the term cascade, 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 [18,19].

Another disadvantage of existing approaches [7] 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 [18,19,11,12,13] 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 [8]. 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 [8]. 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.

1.1 Related Work

The problem of analyzing information flow has been studied using content influence cascades [12,13,1]. 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 [2] and biological networks [10], 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 [16]. In some recent works [15,4] 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 [22].

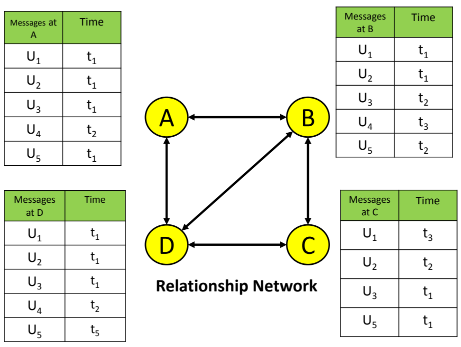
Content propagation in online media is usually tracked as short and distinct phrases, referred to as memes [12,1]. Memes tend to have a broader stable vocabulary, that mimic a slowly evolving genetic signature over time. The key idea of this work [12] is to understand how the short phrases evolve over 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 [23]. A more detailed survey of evolution of content in network structures can be found in [3].

In a recent experimental study [11], 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 [7] 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.

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 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.



Messages at A	Time
U_1	t_1
U_2	t_1
U_3	t_1
U_4	t_2
U_5	t_1

Messages at B	Time
U_1	t_1
U_2	t_1
U_3	t_2
U_4	t_3
U_5	t_2

Messages at D	Time
U_1	t_1
U_2	t_1
U_3	t_1
U_4	t_2
U_5	t_5

Messages at C	Time
U_1	t_3
U_2	t_2
U_3	t_1
U_5	t_1

Fig. 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

Definition 1 (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 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

[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.

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 [11], or due to limited user attention span [24]. 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 1, message table T_C of actor C contains four messages and time-stamp pairs: $T_C = \{(U_1, t_3), (U_2, t_2), (U_3, t_1), (U_5, t_1)\}$.

Problem 1 (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 $T_i = \{(U_j, t_j)_{j=1 \ldots m_i}\}$, 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 [10]. However, in this paper, we restrict our attention to propagation of textual content as information signals over discrete time points.

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 [20]. 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 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.

3.1 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 [25], in polynomial time to the problem of mining information flow patterns (Problem 1). The notion of maximal information flow patterns is

Algorithm 1. ***FLOWER***

```
Input: $G = (V, E)$: Relationship network; $T_i$: message table for actor $a_i$; $f$: frequency;
       and $\tau$: life-time;
Output: $F$: Set of flows satisfying all the properties
1 Initialize $V'$ and $S$ to empty set
2 for each $a_i \in V$ do
3     if number of distinct messages in $T_i \geq f$ then
4         Add $a_i$ to $V'$ and $F$
5 for each $a_i \in V'$ do
6     FLOWPROP($\{\phi\}, T_i, a_i, T_i, \tau, f, V', F$)
7 return $F$;
```

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 $a_i$; $\tau$: life-time; $f$: frequency; $V'$:
       frequent actors set; $F$: frequent flows discovered;
1 $T_{new}$ = Extract messages that satisfy causality and life-time property from $T_i$ and $T_{seq}$
2 if number of messages in $T_{new} \geq f$ then
3     $\Gamma_i$ = Get neighbors of $a_i$ not in current flow path $seq$ and in $V'$
4     for each $a_j \in \Gamma_i$ do
5         FLOWPROP($\langle seq \cup \{a_i\}\rangle, T_{new}, a_j, T_j, \tau, f, V', F$)
6     if $\Gamma_i$ is empty then
7         Add $\langle seq, \{a_i\}\rangle$ to $F$
```

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 1. *[25] Let $\mathcal{D}$ be a database of sequences with m transactions. The problem of 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 2 (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 1, for support f, graph G and lifetime τ.

Lemma 1. *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 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 1, 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 2. *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 2) 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. ■

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. 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 [14] 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 [8]. 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 [8], have been developed. For a more detailed review of the PowerGraph, please refer to [8].

Algorithm 3. Scatter

```
Input: $icontext\_type$: context, $vertex\_type$: current_vertex, $edge\_type$: edge
1 if edge.destination_node does not have $f$ words in its message table then
2     return
3 for each sequence_to_send in current_vertex do
4     if sequence_to_send in current_vertex has the destination vertex Id then
5         continue
6     if edge.destination_node satisfies all properties (1)-(4) then
7         add the destination id to sequences_to_send and copy it to edge data
```

In the distributed version of our algorithm, each vertex contains three pieces of metadata: 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 a source to a destination vertex. The message carried by each

Algorithm 4. Apply

```
Input: $icontext\_type$: context, $vertex\_type$: current_vertex, $gather\_type$:
        incoming_object
1 for each sequence in the incoming_object.sequences do
2     Add $sequence$ to the current_vertex saved_sequences list
3     Add $sequence$ in to sequences_to_send array for scatter method to pick up
4 if (context.iteration $<=$ max_iterations) && (number of sequences_to_send in
  current_vertex $> 0$) then
5     schedule current_vertex for next iteration
```

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
```

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 $\underline{p}$arallel version of FLOWER as *pFLOWER*.

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 [8] for our parallel *pFLOWER* evaluation.

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

[3] `http://arnetminer.org/citation`
[4] `http://dblp.uni-trier.de/xml/`

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.

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

5.3 Results

We compared the running time of *PrefixSpan*, *FLOWER* and *pFLOWER*. However, in Figure 2, we could not plot *PrefixSpan* running times as they were extremely large. Therefore, we list the running times of *PrefixSpan* separately in Table 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 2, both *FLOWER* and *pFLOWER* completed in less than a couple of minutes over most parameter settings.

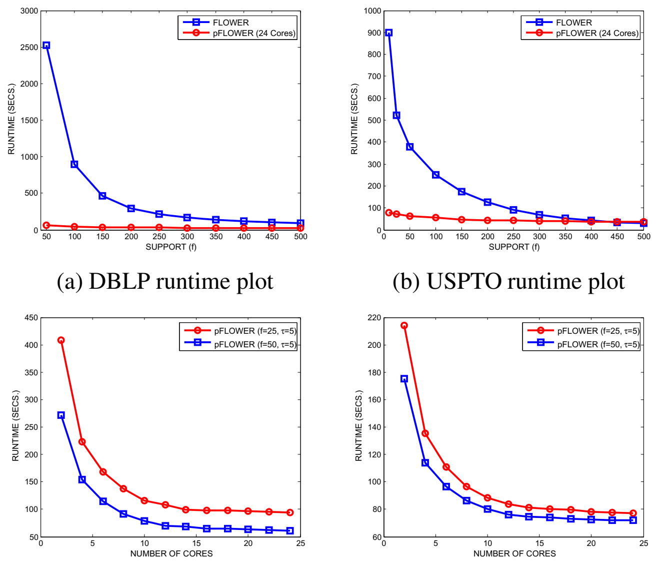
We compared the running times of *FLOWER* and *pFLOWER* algorithm, in Figures 2(a) and (b). As the number of words (f) required for the frequency property (Property 3) decreases, the number of possible flow paths increases exponentially. The *FLOWER* approach explores each of these paths sequentially, resulting in an exponential complexity with path length. On the other hand, the parallel algorithm *pFLOWER*

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

[6] `http://www.cs.uiuc.edu/homes/hanj/software/prefixspan.htm`

Table 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.

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



(a) DBLP runtime plot (b) USPTO runtime plot

(c) DBLP scalability plot (d) USPTO scalability plot

Fig. 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.

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 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 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 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 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.

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.

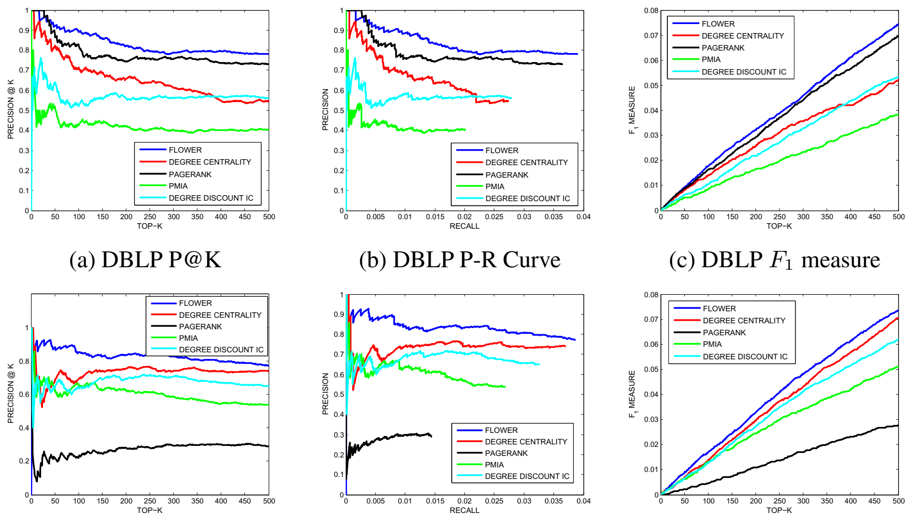
One might argue that using centrality measures or popular influence mining algorithms (such as PMIA [5], DegreeDiscountIC [6]) in a static co-authorship network are sufficient to measure the influence. We evaluate this hypothesis by comparing the influential co-authors found using the popular influence analysis algorithms such as degree-centrality, PageRank, PMIA [5] and DegreeDiscountIC [6] 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).

6.1 Evaluation Baselines

Let us now describe the baselines we used for evaluating our hypothesis. PMIA [5] is the prefix excluded extension of Maximum Influence Arborescense model. We used the weighted cascade model proposed in [9] to compute the edge probabilities for this approach. The degree-centrality approach uses the maximum total out-degree and DegreeDiscountIC [6] heuristic developed for the uniform IC [9] 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.

6.2 Evaluation Results

Our evaluation results are shown in Figure 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(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(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(c) and (f)), our approach performs better than baselines over all values of K.



(a) DBLP P@K (b) DBLP P-R Curve (c) DBLP F_1 measure

(d) USPTO P@K (e) USPTO P-R Curve (f) USPTO F_1 measure

Fig. 3. The P@K, P-R and F_1 measure plots for DBLP and USPTO data sets

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.

Acknowledgments. This research was sponsored by the Defense Advanced Research Project Agency (DARPA) agreement number W911NF-12-C-0028 and Army Research Laboratory (ARL) cooperative agreement number W911NF-09-2-0053.

References

1. Adar, E., Adamic, L.: Tracking information epidemics in blogspace. In: Web Intelligence, pp. 207–214 (2005)
2. Aggarwal, C., Subbian, K.: Event detection in social streams. In: SDM, pp. 624–635 (2012)
3. Aggarwal, C., Subbian, K.: Evolutionary network analysis: A survey. ACM Comput. Surv. 47(1), 10 (2014)
4. Bonchi, F., De Francisci Morales, G., Gionis, A., Ukkonen, A.: Activity preserving graph simplification. Data Mining and Knowledge Discovery 27(3), 321–343 (2013)
5. Chen, W., Wang, C., Wang, Y.: Scalable influence maximization for prevalent viral marketing in large-scale social networks. In: KDD, pp. 1029–1038 (2010)
6. Chen, W., Wang, Y., Yang, S.: Efficient influence maximization in social networks. In: KDD, pp. 199–208 (2009)
7. Galuba, W., Aberer, K., Chakraborty, D., Despotovic, Z., Kellerer, W.: Outtweeting the twitterers-predicting information cascades in microblogs. In: WOSN (2010)
8. Gonzalez, J., Low, Y., Gu, H., Bickson, D., Guestrin, C.: Powergraph: Distributed graph-parallel computation on natural graphs. In: USENIX (2012)
9. Kempe, D., Kleinberg, J.M., Tardos, E.: Maximizing the spread of influence through a social network. In: KDD, pp. 137–146 (2003)
10. Kim, Y.A., Przytycki, J.H., Wuchty, S., Przytycka, T.M.: Modeling information flow in biological networks. Physical Biology 8(3), 035012 (2011)
11. Lerman, K., Ghosh, R.: Information contagion: An empirical study of the spread of news on digg and twitter social networks. In: ICWSM (2010)
12. Leskovec, J., Backstrom, L., Kleinberg, J.M.: Meme-tracking and the dynamics of the news cycle. In: KDD, pp. 497–506 (2009)
13. Leskovec, J., McGlohon, M., Faloutsos, C., Glance, N.S., Hurst, M.: Cascading behavior in large blog graphs. In: SDM (2007)
14. Low, Y., Gonzalez, J., Kyrola, A., Bickson, D., Guestrin, C., Hellerstein, J.M.: Graphlab: A new framework for parallel machine learning. arXiv:1006.4990 (2010)
15. Mathioudakis, M., Bonchi, F., Castillo, C., Gionis, A., Ukkonen, A.: Sparsification of influence networks. In: KDD, pp. 529–537 (2011)
16. Myers, S.A., Zhu, C., Leskovec, J.: Information diffusion and external influence in networks. In: KDD, pp. 33–41 (2012)
17. Pei, J., Pinto, H., Chen, Q., Han, J., Mortazavi-Asl, B., Dayal, U., Hsu, M.-C.: Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In: ICDE, pp. 215–215 (2001)
18. Rodriguez, M.G., Leskovec, J., Krause, A.: Inferring networks of diffusion and influence. In: KDD, pp. 1019–1028 (2010)
19. Rodriguez, M.G., Leskovec, J., Schölkopf, B.: Structure and dynamics of information pathways in online media. In: WSDM, pp. 23–32 (2013)

20. Subbian, K., Aggarwal, C., Srivastava, J.: Content-centric flow mining for influence analysis in social streams. In: CIKM (2013)
21. Subbian, K., Melville, P.: Supervised rank aggregation for predicting influencers in twitter. In: SocialCom, pp. 661–665 (2011)
22. Subbian, K., Sharma, D., Wen, Z., Srivastava, J.: Social capital: the power of influencers in networks. In: AAMAS, pp. 1243–1244 (2013)
23. Wang, X., Zhai, C., Hu, X., Sproat, R.: Mining correlated bursty topic patterns from coordinated text streams. In: KDD, pp. 784–793 (2007)
24. Weng, L., Flammini, A., Vespignani, A., Menczer, F.: Competition among memes in a world with limited attention. Scientific Reports 2 (2012)
25. Yang, G.: The complexity of mining maximal frequent itemsets and maximal frequent patterns. In: SIGKDD, pp. 344–353 (2004)