Discovering Leaders from Community Actions

Amit Goyal
University of British Columbia
Vancouver, BC, Canada
goyal@cs.ubc.ca

Francesco Bonchi
Yahoo! Research
Barcelona, Spain
bonchi@yahoo-inc.com

Laks V. S. Lakshmanan
University of British Columbia
Vancouver, BC, Canada
laks@cs.ubc.ca

ABSTRACT

We introduce a novel frequent pattern mining approach to discover leaders and tribes in social networks. In particular, we consider social networks where users perform actions. Actions may be as simple as tagging resources (urls) as in del.icio.us, rating songs as in Yahoo! Music, or movies as in Yahoo! Movies, or users buying gadgets such as cameras, handhelds, etc. and blogging a review on the gadgets. The assumption is that actions performed by a user can be seen by their network friends. Users seeing their friends' actions are sometimes tempted to perform those actions. We are interested in the problem of studying the propagation of such "influence", and on this basis, identifying which users are leaders when it comes to setting the trend for performing various actions. We consider alternative definitions of leaders based on frequent patterns and develop algorithms for their efficient discovery. Our definitions are based on observing the way influence propagates in a time window, as the window is moved in time. Given a social graph and a table of user actions, our algorithms can discover leaders of various flavors by making one pass over the actions table. We run detailed experiments to evaluate the utility and scalability of our algorithms on real-life data. The results of our experiments confirm on the one hand, the efficiency of the proposed algorithm, and on the other hand, the effectiveness and relevance of the overall framework. To the best of our knowledge, this the first frequent pattern based approach to social network mining.

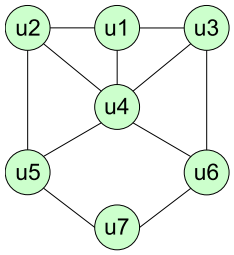
Categories and Subject Descriptors H.2.8 [Database Management]: Database Applications - *Data Mining*
General Terms: Algorithms
Keywords: Social networks, Frequent pattern discovery, Influence, Leadership, Tribes, Viral marketing.

1. INTRODUCTION

Consider a social network where users perform various actions. As an example, in del.icio.us (`http://del.icio.us/`), users bookmark urls and annotate them with various tags. Here, tagging a url with a tag such as *Ocean Kayaking* is an action. As another example, a user in Yahoo! Movie (`http://movies.yahoo.com/`) may decide to rate the movie *There will be blood*, which is also an action. As a third example, a user in facebook (`http://www.facebook.com/`) may decide to buy a new camera and decide to write a review on it in her personal blog. In each of these examples, users may choose to let their network friends see their actions. Seeing actions performed by their friends may make users curious and may sometimes tempt some fraction of the users to perform those actions themselves, some of the time. Informally, we can think of this as an influence (for performing certain actions) propagating from users to their network friends, potentially recursively. If such influence patterns repeat with some statistical significance, that can be of interest to companies, say for targeted advertising. E.g., if we know that there are a small number of "leaders" who set the trend for various actions, then targeting them for adoption of new products or technology could be profitable to the companies.

Figure 1(a)-(b) shows an example of a social graph and a log of actions performed by users. By linking the actions log in Figure 1(b) to the social graph in Figure 1(a), we can trace the propagation of influence for performing actions. The result, for actions a and b, is shown in Figure 1(c)-(d). Observe that the graphs in Figure 1(c)-(d) are directed and labelled even though the original social graph is undirected and unlabelled. Direction is dictated by the order in which actions were performed. If x performed action a before y and there is a social tie between x and y in Figure 1(a), then Figure 1(c) contains a directed edge from x to y; while the label on the edge is given by the time elapsed between the two actions.

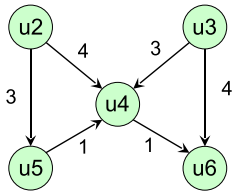
Given a social graph together with a log of user actions, we can determine the propagation of influence for performing each of the actions. How can we leverage this for determining who are the leaders in setting the trend for performing actions? It turns out there are a number of options for defining leadership. First off, intuitively, we want to think of user u as a *leader* w.r.t. an action a provided u performed a and within a chosen time bound after u performed a, a sufficient number of other users performed a. Furthermore, we require that these other users are reachable from u thus capturing the role social ties may have played. Finally, we may want to extract leadership w.r.t. performing actions in general or performing a class of actions. For example, in Figure 1, if we choose the time bound to be 5 units, number of users to be influenced by a leader to be *three*, then user $u2$ is a leader w.r.t. both actions a and b, since users $u4, u5, u6$



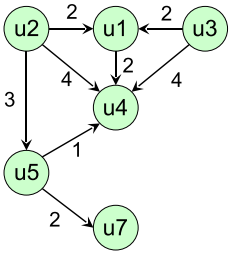
(a)

User	Action	Time
u3	b	13
u2	b	13
u2	a	15
u1	b	15
u3	a	16
u5	b	16
u4	b	17
u5	a	18
u7	b	18
u4	a	19
u6	a	20

(b)



(c)



(d)

Figure 1: (a) Example social graph; (b) A log of actions; (c) Propagation of action a and (d) of action b.

are influenced by u2 in Figure 1(c), while u1, u4, u5, u7 are influenced by u2 in Figure 1(d). Notice that u3 is not a leader w.r.t. either action. If user u5 had performed b after u4, then u3 would be regarded a leader w.r.t. b, in addition to u2. A stronger notion of leadership might be based on requiring that w.r.t. each of a class of actions of interest, the set of influenced users must be the same. To distinguish from the notion of leader illustrated above, we refer to this notion as *tribe leader*, meaning the user leads a fixed set of users (tribe) w.r.t. a set of actions. To appreciate the difference between leader and tribe leader, consider that we require *two* users to be influenced by a leader and these two users must be the same for both actions a and b. Then 2 is a tribe leader for the set of actions {a, b} since users 4 and 5 are influenced by 2 w.r.t. both a and b. Clearly, tribe leaders are leaders but not vice versa.

The notion of leader admits several candidates. However, the influence emanating from some of them may be "subsumed" by others. As an example, in Figure 1(c), both 2 and 5 are leaders w.r.t. a when the required number of influenced users is two. However, it is clear that user 5's influence is subsumed by that of user 2, as 5 is one of those influenced by 2. Thus, it is interesting to ask whether we can discover "genuine" leaders, i.e., leaders whose influence is not subsumed by others. Notice that a tribe leader may be genuine or not just as a leader may be genuine or not.

In this paper, we make the following contributions:

- We formally define the problem of extracting leaders from a database consisting of a social graph plus a table representing the log of user actions. We define various notions of leader, tribe leader, and their confident and genuine variants (Section 3).
- We develop efficient algorithms for extracting leaders of various flavors from an input data consisting of a social graph and a table of user actions. Our algorithms make one pass over the actions table. This is significant since we expect the table to be very large (Section 4).
- We demonstrate the utility and scalability of our algorithms, via an extensive set of experiments (Section 5) on a real-world dataset obtained by joining data coming from Yahoo! Messenger (the social graph) and Yahoo! Movies (the action log, where the action corresponds to rating a movie).

Influence propagation and leadership have been considered in the context of *viral marketing* [10, 22, 17]. There, the model is based on probabilistic causation: there is a probability with which a user will perform an action if his neighbors have performed it. These works focus on the optimization problem of finding the k best people to target an ad to in order to maximize the influence. The framework is entirely based on the social graph together with edge weights representing the probabilities of influence. Our problem setting is considerably different. The input includes not just a graph (which is not edge-weighted) but an actions table which plays a central role in the definition of leaders. Secondly, our focus is on finding all leaders based on the frequent pattern mining approach, which is different from the approach employed in the above works. To the best of our knowledge, our notions of leaders and our algorithms are novel. Related work appears in the next section. We summarize the paper in Section 6 and discuss future work.

2. RELATED WORK

The study of the spread of influence through a social network, i.e., the extent to which people are likely to be influenced by actions of their friends has a long history in the social sciences. The first investigations focused on the adoption of medical [9] and agricultural innovations [24]. Later marketing researchers have investigated the diffusion process of the "word-of-mouth" effect and its application to the so-called viral marketing [6, 12, 21].

The first work to consider the propagation of influence and the problem of identification of influential users by a data mining perspective is [10, 22]. Suppose we are given a social network together with the estimates of reciprocal influence between individuals in the network, and suppose that we want to push a new product in the market. The idea behind viral marketing is that by targeting the most influential users in the network we can activate a chain-reaction of influence driven by word-of-mouth, in such a way that with a very small marketing cost we can actually reach a very large portion of the network. The mining problem is the following: given such a network with influence estimates, how to select a set of users such that by targeting this set we maximize the spread of influence. The problem is tackled by means of a probabilistic model of interaction, and heuristics are given for choosing users with a large overall effect on the network. More recently [17] attacked the exact same problem as a problem in discrete optimization obtaining provable performance guarantees for approximation algorithms in several preexisting models coming from mathematical sociology. The same group of authors in [16] adopts the *decreasing cascade model* used in sociology and economics and devises a simple greedy algorithm for it. In [13] the price of the item subject to viral marketing is introduced in the model, leading to the problem of *revenue maximization*. Another related problem, namely *outbreak detection* is studied in [19]: how to select nodes in a network in order to detect the spread of a virus as fast as possible? In [1] instead the problem of how to identify influential bloggers is studied.

As discussed in the introduction, even if the general goal is very similar, our problem setting is considerably different, as we have a different input and as we adopt a completely different mining approach, i.e., pattern discovery.

Among the various kinds of patterns that we study, there is also *tribe leaders*, i.e., users that lead a *fixed* set of users for different actions. As a side effect of mining tribe leaders, we also mine tribes which can be considered as small communities. The problem of identifying communities in social networks has been studied extensively in the data mining community, especially focusing on how to model community formation and evolution in time [14, 18, 4, 23, 15, 5, 11]. Again our problem statement is very different in that we focus on actions log, and in terms of the pattern discovery approach that drives our work. [20] considers a particular form of social network, called friendship-event network, made by the usual social ties, plus a series of events in which the various nodes of the network may be *organizers* or *participants*. Thus also in their context there is a kind of actions log, that is the events organization and participation. Our work is different as we consider actions more in general than the organizer-participant relationship. Also the objective is different: [20] studies the notion of *social capital* based on the actor-organizer friendship relationship and the notion of *benefit*, based on event participation.

3. PROBLEM DEFINITION

A *social graph* is an undirected graph $G = (V, E)$ where the nodes are users. There is an undirected edge between users u and v representing a social tie between the users. The tie may be explicit in the form of declared friendship, or it may be derived on the basis of shared interests between users. Our discussion and algorithms are neutral to the source that gives rise to these edges. An *action log* is a relation $\mathtt{Actions}(\mathtt{User}, \mathtt{Action}, \mathtt{Time})$, which contains a tuple (u, t, a) indicating that user u performed action a at time t. It contains such a tuple for every action performed by every user of the system. We will assume that the projection of $\mathtt{Actions}$ on the first column is contained in the set of nodes V of the social graph G. In other words, users in the $\mathtt{Actions}$ table correspond to nodes of the graph. We let $\mathcal{A}$ denote the universe of actions. We next define propagation of actions. In the following, we assume for ease of exposition that a user performs an action at most once. Our algorithms do not assume this. In Section 5, we explain how our algorithms handle the case where users may perform an action multiple times.

Definition 1 (Action propagation). *We say that an action $a \in \mathcal{A}$ propagates from user v_i to v_j iff $(v_i, v_j) \in E$ and $\exists (v_i, a, t_i), (v_j, a, t_j) \in \mathtt{Actions}$ with $t_i < t_j$.*

Notice that there must be a social tie between v_i and v_j, both must have performed the action, one strictly before the other. This leads to a natural notion of a propagation graph, defined next.

Definition 2 (Propagation graph). *For each action a, we define a propagation graph $PG(a) = (V(a), E(a))$, defined as follows. $V(a) = \{v \mid \exists t : (v, a, t) \in \mathtt{Actions}\}$; there is a directed edge $v_i \xrightarrow{\Delta t} v_j$ whenever a propagates from v_i to v_j, with $(v_i, a, t_i), (v_j, a, t_j) \in \mathtt{Actions}$, where $\Delta t = t_j - t_i$.*

The propagation graph consists of users who performed the action, with edges connecting them in the direction of propagation. Observe that the propagation graph is a DAG. Each node can have more than one parent; it is directed, and cycles are impossible due to the time constraint which is the basis for the definition of propagation. Note that the propagation graph can possibly have disconnected components. In other words, the propagation of an action is just a directed instance (a flow) of the undirected graph G, and the log of actions $\mathtt{Actions}(\mathtt{User}, \mathtt{Action}, \mathtt{Time})$ can be seen as a collection of propagations. Influence can propagate transitively. Thus, a definition of leader w.r.t. setting the trend for performing an action should take this into account. To aid in the definition of leaders, we define the notion of an influence graph next. The elapsed time along a (directed) path in a propagation graph is the sum of edge labels along the path. E.g., in Figure 1(c), the elapsed time on the path $u2 \rightarrow u4 \rightarrow u6$ is $4 + 1 = 5$.

Definition 3 (User influence graph). *Given action a, a user u, and a maximum propagation time threshold π, we define the influence graph of the user u, denoted $\mathrm{Inf}_\pi(u, a)$ as the subgraph $PG(a)$ rooted at u, such that it consists of those nodes of $PG(a)$ which are reachable from u in $PG(a)$ and such that every path from u to any other node in $\mathrm{Inf}_\pi(u, a)$ has an elapsed time at most π.*

In Figure 2 we show an example of propagation graph $PG(a)$, and two user influence graphs.[1] The propagation time threshold π allows us to set limits on how long after an action is performed, we regard another user performing that action as influenced by the previous user. Given a threshold ψ, we say that user u acted as a *leader w.r.t. action* a whenever the size (in number of nodes) of $\mathrm{Inf}_\pi(u, a)$ is at least ψ. The threshold ψ ensures sufficiently many users are influenced by the given user in the context of action a. Note that the two constraints defined by the thresholds π and ψ are conflicting constraints. In fact, π limits the influence graph rooted at u to be not grown after a certain elapsed time, while the second constraint requires to have an influence graph larger than ψ.

A user to be identified as a leader must act as such sufficiently often, i.e., for a number of actions larger than a given action threshold σ. This may be seen as the *minimum frequency* constraint in pattern discovery and association rule mining [2].

Problem 1 (Leaders). *Given a set of actions $I \subseteq A$, and three thresholds π, ψ and σ, a user $v \in V$ is a* leader *iff:*

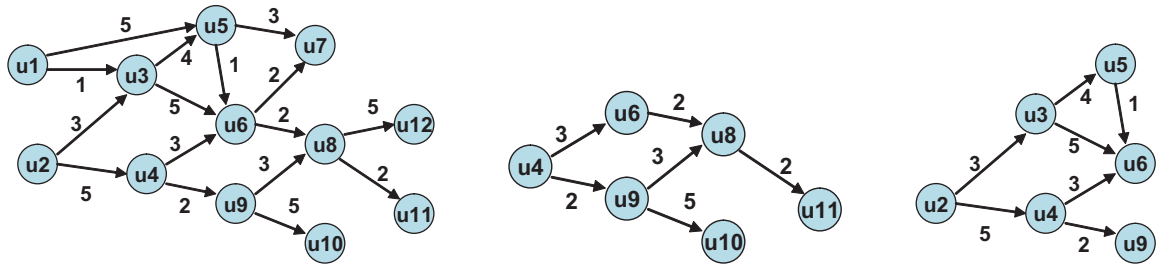
$$\exists S \subseteq I, |S| \geq \sigma : \forall a \in S \,.\, \mathtt{size}(\mathrm{Inf}_\pi(v, a)) \geq \psi$$

The intuition behind the above definition is that for sufficiently many actions ($\geq \sigma$) at least ψ other users are influenced to perform the action within π time units from when the given user v performed it. In this case, we regard v as a leader. Notice the definition does not force the set of influenced users for different actions to be the same.

Problem 2 (Tribe-leaders). *Given a set of actions $I \subseteq A$, and three thresholds π, ψ and σ, a user $v \in V$ is a* tribe leader *iff:*

$$\exists S \subseteq I, |S| \geq \sigma, \exists U \subset V, |U| \geq \psi \,:\, \forall a \in S \,.\, U \subseteq \mathrm{Inf}_\pi(v, a).$$

[1] W.r.t. some underlying social graph and action log, not shown.



(a) (b) (c)

Figure 2: The propagation graph of an action $\mathtt{PG}(a)$ in fig.(a), $\mathtt{Inf}_8(u4, a)$ in fig.(b), $\mathtt{Inf}_8(u2, a)$ in fig.(c).

The notions leader and tribe leader were illustrated in Section 1. Clearly, every tribe leader is a leader, but not vice versa.

3.1 Additional constraints

In addition to using an absolute lower bound on the number of actions in which a user acts as a leader, we could apply a "confidence threshold" (similarly to the classical measure of confidence in association rules [2]), requiring that a user is a leader for a large fraction of the actions he performs.

More precisely, for a user $v \in V$, let $P(v) = \{a \in \mathcal{A} \mid v \text{ performed } a\}$ and $L(v) = \{a \in \mathcal{A} \mid v \text{ is a leader w.r.t. } a\}$. Then the *leadership confidence* of v is the ratio $\mathtt{conf}(v) = |L(v)|/|P(v)|$. We define:

PROBLEM 3 (LEADERS WITH CONFIDENCE MEASURE). *Given a set of actions $I \subseteq \mathcal{A}$, and a confidence threshold $0 < \varphi \leq 1$, a user v is said to be a* confidence leader *if it is a leader and $\mathtt{conf}(v) \geq \varphi$.*

It may happen that one user acts as a leader according to the given definitions, but in concrete he is always a follower of the same leader. In some sense he benefits from the influence of a true leader, so that he also may seem a leader. To avoid this kind of "fake" leaders, we propose the following.

Given the usual three thresholds π, ψ and σ, for a user v, let $\mathtt{gen}(v)$ denote the ratio

$$\frac{|\{a \in L(v) \mid \nexists u \in V : u \text{ is leader for } a \wedge v \in \mathtt{Inf}_\pi(u, a)\}|}{|L(v)|}$$

i.e., the fraction of actions led by v for which v's leadership is genuine, in that it is not a consequence of v being present in the influence graph of some other leader w.r.t. that action. We call $\mathtt{gen}(v)$ the genuineness score of v. The notion of genuine leaders is defined next.

PROBLEM 4 (GENUINE LEADERS). *Given a set of actions $I \subseteq \mathcal{A}$, and a threshold $0 < \gamma \leq 1$, we define a leader v to be a* genuine leader *provided the genuineness score of v is above the threshold, i.e., $\mathtt{gen}(v) \geq \gamma$.*

Both confidence and genuineness constraints can be applied to tribe leaders as well. Note that when we focus on a single action a, the notions of leader for a and tribe leader for a coincide. Thus we use the exact same definitions. So a genuine tribe leader is a tribe leader whose genuineness score is above a threshold, and similarly for confidence. Our goal is to efficiently extract leaders and tribe leaders, possibly with each of the other criteria (confidence and genuineness) or even with both. This leads to eight distinct problems: leaders and tribe leaders without additional constraints, with one of the two, or with both additional constraints.

We provided the basic pattern discovery framework for mining leaders given the social graph and an action log. However, it should be noted that it is not mandatory for the final analyst to provide all the thresholds: she can query our system asking, e.g., the top-k users w.r.t. average number of followers per action given π, or the top-k tribe leaders w.r.t. confidence. In fact, our algorithms by scanning the actions log, produce a summary of the influence for which only the π threshold is needed. The various kinds of leaders then can be selected from this summary by a simple post-processing, as described in the next section.

4. ALGORITHMS

Any algorithm for extracting leaders must scan the action log table and traverse the graph. Our assumption is that the graph is large (in the order of hundreds of thousands to millions of nodes) and the action log contains millions of tuples. While the graph is large, given a (maximum propagation) time threshold π, and a specific time interval with length π, we assume the subgraph corresponding to users performing actions within the interval is small. This is confirmed by the real data sets we used for our experiments, where π was set to several weeks. Thus, our focus is on minimizing the scans of the action log table.

Our algorithms are able to extract the patterns (leaders and tribe leaders) in no more than one scan of the action log table Actions. The key insight behind this is as follows. We assume that the action log is stored in chronological order of actions. Given the threshold π, we scan the Actions table by means of a window of width π. Thus at any position of the window, a subset of tuples in Actions falls in the window. Let W denote the current position of the window. The position W induces a subgraph of the social graph G: $G_W = (V_W, E_W)$, where V_W is the set of users who performed some action within the window W and an edge is present in E_W iff it is present in E and both its endpoints are in V_W. We call G_W the subgraph of G visible from the window.

By sliding the window chronologically backwards we can compute an *influence matrix* $IM_\pi(U, A)$, where U is the number of users and A is the number of actions. The entry $IM_\pi(u, a)$ is the number of users/nodes, influenced by u w.r.t. action a within time π. This number includes u. So, a user u performed action a iff $IM_\pi(u, a) > 0$. Then leaders can be computed from the IM_π easily. When it comes to compute tribe leaders, influence matrix is inadequate: for tribe leaders, we need to check that a fixed set of $\geq \psi$ users were influenced by the leader on sufficiently many actions. To address this problem, we propose the *influence cube*. The influence cube is a User $\times$ Action $\times$ User cube with cells containing boolean entries: $IC_\pi(u, a, v) = 1$ if user v was

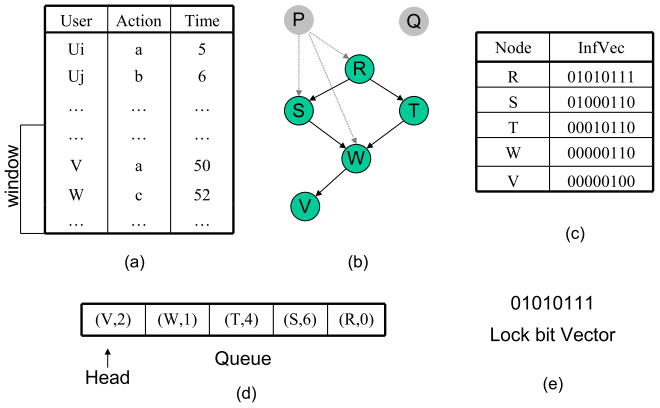


Figure 3: (a) Action log; (b) Propagation graph visible in current window for a action; (c) Influence Vector for the current nodes; (d) Queue for the current action; (e) Lock Bit Vector for the current action.

influenced by user u w.r.t. action a, w.r.t. an underlying time threshold π. From this, we can determine whether u is a tribe leader, as described later in Section 4.3.

4.1 Computing Influence Matrix

We now describe the major steps in detail and discuss their efficient implementation. To illustrate our algorithms, we will use the running example shown in Figure 3. Figure 3(a) shows a schematic action log and shows the window at the bottom-most position, i.e., the most recent actions fall in the window. Figure 3(b) shows a possible subgraph of G visible from the window. Nodes in light grey and their incident edges are not yet visible.

Algorithm 1 gives the algorithm at a high level. The steps are explained in detail next. We start by positioning the window at the bottom of the action log, thus seeing only most recent actions (step 1). We visit the nodes of G_W by exploring G and computing necessary state at the visited nodes, e.g., the nodes influenced by a given node (steps 2-3). We move the window backward in time. Clearly, as the window moves existing action tuples may fall off the window (view) and other tuples may enter the window. This in turn causes parts of the visible graph G_W to disappear and new parts to appear. Our algorithms make use of two key operations – *update* and *propagate*. For simplicity of exposition, let us focus on one action a, although our algorithms simultaneously keep track of the influence propagation information for several actions. When a node of G_W disappears as the window moves, we need to update the state of existing nodes. When a new node appears in the window, we need to propagate the state from existing nodes to the new node so that its state can be computed efficiently (steps 6-8).

By repeated application of update and propagate as the window is moved backward in time, we can compute the *influence matrix* $IM_\pi(U, A)$.

Next we describe the update and propagate procedures in detail. We start with the representation of node (user) state. We use a bit vector to track which users are influenced by a given user. Since the graph is huge and only a relatively small number of nodes are visible from any window, we would like to optimize the number of bits. However, as the window moves, the bits need to be *dynamically allocated*. Moreover, when we adapt the algorithm for computing tribe leaders, we would want to know for any user and action, not just the number of users influenced but *which ones were influenced*. We solve this problem of dynamic allocation of bits for user state, by maintaining a *Lock Map* for each position of the window. The lock map keeps track of which bit is allocated to which user in the current window. As shown in Figure 3(d), we keep the lock map as entries in a queue for efficient manipulation. E.g., it shows user R is assigned the right-most bit (0-th bit), user S is assigned the 6-th bit (from right), etc. To facilitate quick allocation of "free" bits, we keep a *Lock Bit Vector* in which set bits correspond to bits or locks owned by nodes and 0 bits correspond to "free" bit positions. E.g., bit positions $0, 1, 2, 4, 6$ are owned by nodes R, W, V, T, S. When the window moves, the lock map entries and the Lock Bit Vector are kept updated as will be explained shortly. We represent the state of a node as an *influence vector* (IV). There is one influence vector per action performed by the user in the current window. E.g., the influence vector of user S is 01000110. This says the users influenced by S correspond to bit positions $1, 2, 6$. Positions $1, 2$ correspond to users W, V, which are the users influenced by S.

Algorithm 1 Compute Influence Matrix

Input: Graph G; Action log Actions; Threshold π.
Output: Influence matrix $IM_\pi(U, A)$.
1: Position a window of size π at the end of table Actions.
2: Discover the visible subgraph G_W of G on the fly from the tuples in the window.
3: Compute the state of every node by starting from the most recent tuples and working backward up the graph.
4: Fill in the cells $IM_\pi(u, a)$ whenever we have the state for node u w.r.t. action a computed.
5: **while** top of Actions not reached **do**
6: Move the window from the most recent tuple in the window to the next earlier tuple.
7: For every tuple that drops off the window, **update** the state of every other node.
8: For every new tuple that appears in the window, compute the state of the corresponding node by **propagating** the state from its children in the visible graph.
9: Update the IM_π entries as needed.

Algorithm 2 Propagate

Input: A tuple τ in the form of $< u, a, t >$.
1: Issue the first free lock to u. Let its index be i.
2: Initialize the IV for u with only i-th bit set to 1.
3: **for** each element v of Queue(a) **do**
4: **if** there is an edge from u and v **then**
5: IV(u) = IV(u) OR IV(v).
6: Add the node for u at the tail of Queue(a)

Algorithm 2 describes the procedure Propagate. When a new tuple is inserted, we issue a free lock to its associated node (step 1). Searching a free lock is as simple as finding the first 0 bit in the Lock Bit Vector (from the right). The IV for the new element u is initialized with its own index set to 1 (step 2). We then traverse the queue and search for a child of u. If there is an edge from u to the current element in the queue, then the IV of u is updated as the bit OR of IV of u and the IV of the current element. This way, we can compute the IV of every new node in the window (steps 3-5). When the time window moves up, a node may drop off the window. In such a case, we need to update the states of the remaining nodes. To do this, we traverse the queue and reset the lock index bit in IV of each element. Then the

lock is released by resetting the bit in the Lock Bit Vector. Procedure Update describes the algorithm.

Algorithm 3 Update

Input: A tuple $\tau =< u, a, t >$ that has become invisible in the window.

1: i = lock index of u
2: **for** each element v in Queue(a) **do**
3: reset the index i in IV(v).
4: Delete the element u from the Queue(a).
5: Release the lock.

When the window is at its bottom-most position, we compute the state of every user by bit OR operations. Initialize the IV (for a given action) for every node to be the bit vector where only the bit owned by the node is set. In our running example, the IV of W is obtained as the bit OR of IV of V and the current IV of W. This way, we can compute the IV of every node in the window by bit OR'ing it with the IV of its children. Suppose the window now moves such that node V drops off the window and nodes P and Q enter the window (along with their edges linking to nodes in the window). Notice the new edges can only be directed from nodes that just entered the window to nodes that are present in the window. First, we update the state of nodes R, S, T, W which are present in the old and new positions of the window. This is done by simply zeroing the bit corresponding to V, i.e., bit 2 (from the right) in all their states. Next, by examining the Lock Bit Vector, we allocate the first "free" bit to each new node – bit 3 to P and bit 5 to Q. Then, in order to compute the state of new nodes P and Q, we propagate the states from their children to those nodes. Node Q has no children in the window so its IV is just 100000. For node P, we can compute its IV as a bit OR with its default IV and the current IV of R, S, W.

4.2 Computing Leaders

Once the influence matrix IM_π is available, we can compute leaders as follows. Given a user u and thresholds ψ, σ, let $L(u) = \{a \mid IM_\pi(u, a) > \psi\}$. Then u is a leader iff the count $|L(u)| \geq \sigma$. Thus, leaders are computed by scanning the rows of the influence matrix. By the same token, confidence leaders are evaluated as follows. Given a confidence threshold φ, a user is a confidence leader iff $|\{a \mid IM_\pi(u, a) > \psi|/|\{a \mid IM_\pi(u, a) > 0\}| \geq \varphi$. The denominator corresponds to the number of actions performed by u. Recall, $IM_\pi(u, a)$ keeps track of the number of users influenced by u, *including* u, for performing action a within the window π.

The genuineness score is computed by maintaining a fake counter for every user: $fake(u)$ is incremented whenever u is found to be a leader for some action a and in the influence graph $Inf_\pi(v, a)$ of some other leader w.r.t. a. Then the numerator in the definition of $gen(u)$ is simply $|L(u)| - fake(u)$.

4.3 Computing Tribe Leaders

As discussed previously influence matrix is inadequate for computing tribe leaders, we need the *influence cube*, i.e. a User × Action × User cube with cells containing boolean entries: $IC_\pi(u, a, v) = 1$ if user v was influenced by user u w.r.t. action a, w.r.t. an underlying time threshold π. From this, we can determine whether u is a tribe leader by means of frequent itemsets mining. In fact, we can view each array $IC_\pi(u, a, *)$ (where '$*$' denotes "don't care") as a transaction of items where items correspond to candidate influenced users. More precisely, for a given user u, we have a transaction corresponding to every action. Then we can see a tribe as an itemset, and finding tribe leaders as finding frequent itemsets larger than a given threshold ψ on the minimum acceptable tribe size, where we ensure the itemset does not include u when we are trying to determine if u is a tribe leader.

LEMMA 1. *A user u is a tribe leader iff there is at least one itemset of size ψ (not containing u) that has a frequency of σ or more in the transaction database $IC_\pi(u, *, *)$.*

In our implementation, we do not explicitly compute the influence cube. The reason is that real data sets tend to be quite sparse w.r.t. users performing actions. This is a common phenomenon found in del.icio.us (for tagging actions) and Yahoo! Movies (w.r.t. rating movies) and similar social collaborative sites. Thus, a direct implementation of the cube will be inefficient w.r.t. space. We make the observation that when Algorithm 1 computes the influence matrix, it does so using influence vectors. Recall that for each user and each action we have an influence vector, for every position of the time window. Also, for every position, we have a lock map which "decodes" the bit positions in the influence vectors into corresponding node id's.

Consider a user u and action a. Let t be the time u performed a. Consider the position of the time window when its beginning corresponds to time t (and end to $t + \pi$). The influence vector of u w.r.t. a for this position tells us exactly which are the various bit positions corresponding to users influenced by user u on action a. Thus, the *combination of influence vector of u for a for window position t and the lock map* provides a compact representation of the transaction, i.e., row $IC_\pi(u, a, *)$ in the influence cube.

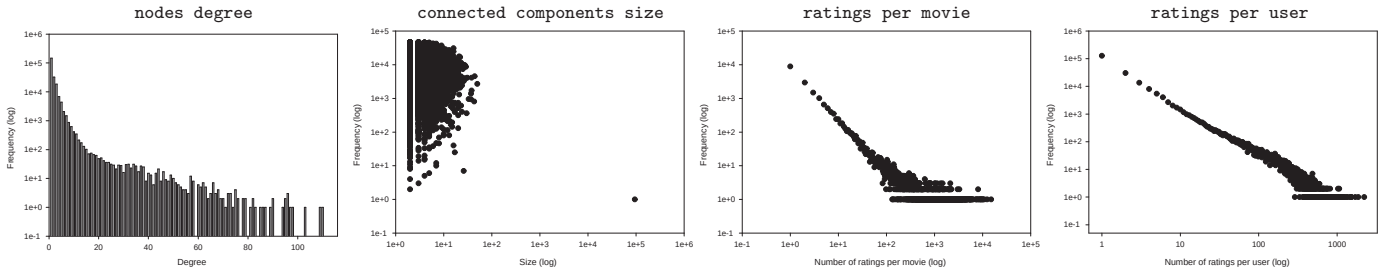
Algorithm 4 Tribe Leaders

Input: Graph G; Action log Actions; Thresholds π, ψ, σ.

1: Position a time window of size π at the end of table Actions and move it backward in time just as in Algorithm 1.
2: Compute the influence vector for every node and every action as before.
3: When the IV of user u w.r.t. action a for window position t is available, where u performed a at t, update the influence cube row $IC_\pi(u, a, *)$ using the IV and the current lock map.
4: When the influence cube is computed, compute σ-frequent itemsets of size $\geq \psi$ (not containing u) by means of ExAMiner [7].
5: Determine the tribe leaders and tribe led by them using the frequent itemsets computed above, using Lemma 1.

We then compute frequent itemsets using a special algorithm optimized for our needs. Notice that we are *not* interested in all frequent itemsets, but only in those ones whose size is ψ or more. This problem has been deeply studied in the literature [8, 7] and a large variety of optimizations have been developed in this context. Here we adopt the ExAMiner implementation [7]. The idea is that a transaction whose size is $< \psi$ can be discarded since none of its subsets will have size $\geq \psi$ either, and therefore the transaction cannot support any valid itemsets. This data reduction, in turn reduces the support of other frequent and invalid itemsets thus reducing the search space and creating new room for removal of other transactions. This has a recursive rippling effect that reduces the computation considerably.

The overall approach for computing tribe leaders is summarized in Algorithm 4.



(a) (b) (c) (d)

Figure 4: The dataset used in the experimentation: (a) nodes degree distribution, (b) distribution of the size of the connected components, (c) distribution of ratings per movie, and (d) distribution of ratings per user.

4.4 Complexity Analysis

Suppose the maximum number of nodes visible in any position of the time window is n. We expect n to be a small fraction of the total number of nodes in the graph G. At any stage, we need n bits to be used for both the Lock Bit Vector and for each of the influence vectors. The key atomic operations in our algorithm are bit operations (AND and OR) and entering or removing a lock map entry into the queue. Each update operation might involve the deletion of $O(n)$ nodes from the queue. Updating the Lock Bit Vector as nodes get deleted costs at most $O(n^2)$ *bit level operations.*[2] Similarly, when a node gets deleted, updating the influence vector of other nodes can cost $O(n)$ bit AND operations, which corresponds to $O(n^2)$ bit level operations. Suppose on an average when the window moves k nodes drop off the window and k nodes enter the window, where $k < n$ and k is $O(n)$. Then the worst case cost for updating all existing nodes in the window when k nodes drop off is $O(kn^2)$. Let T be the number of tuples in the Actions table. Then the number of distinct positions of the window is $O(T/k)$. Let A be maximum number of actions performed by a user. Since we need to maintain influence vectors separately for each action, the total number of bit level operations for update is $O(TAn^2)$.

By a similar analysis, we can show that propagate, which involves the atomic steps of finding a "free" bit in the Lock Bit Vector, entering a node in the queue, finding children of a node among nodes in the window, and modifying the influence vector of a node by bit OR'ing with influence vectors of its children cost $O(TAn^2)$ bit level operations. A node is inserted into the queue or deleted from the queue at most once per tuple in the action log table. We thus have:

LEMMA 2. *The influence matrix can be computed using* $O(TAn^2)$ *bit level operations, and using* $O(T)$ *insert and delete operations on the queue. Here,* T, A, n *are parameters as explained above.*

The complexity of computing tribe leaders is determined by: (a) computing the influence cube and (b) finding the existence of frequent itemsets. Influence cube can be computed with the same number of bit operations as influence matrix (Lemma 2). Finding frequent itemsets is well-known to be of exponential complexity. But it's a thoroughly studied problem and the use of several algorithmic tricks makes it quite efficient in practice.

5. EXPERIMENTAL EVALUATION

In this section, we describe the comprehensive empirical evaluation we conducted in order to assess both the efficiency of our algorithms and the relevance and usefulness of the overall proposal. All the experiments are performed on a 3 GHz Intel Pentium 4 CPU with 2 Gb main memory, running Suse Linux 10.1 with kernel 2.6.16.54. The algorithms were implemented in C++ with STL and compiled with gnu g++ compiler version 4.1.0. We also exploited the bitstring handling library proposed in [3].

[2]As opposed to word level.

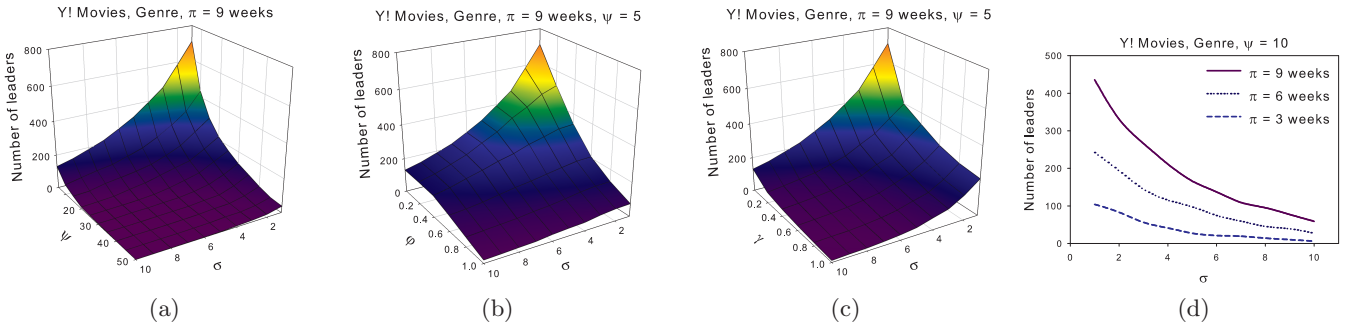
5.1 Dataset used

In our problem setting, we need both the social network and the actions database. Using the Yahoo! Movies data we have the actions (ratings of movies) but we do not have any social network information. Thus for producing the social network we "crossed-over" the Yahoo! Movies ratings with a subgraph of the Yahoo! Instant Messenger graph, as described in the following.

For the sake of privacy, all the user data in this paper were provided to us after anonymization. We started from a subgraph of the Yahoo! Instant Messenger friends-relationship graph, containing the most active users (approx. 110 M nodes), and through common user identifiers, we projected this graph on the subset of users that have also rated a movie in Yahoo! Movies (or more precisely, that have a rating in the set of 21 million ratings provided to us). We then removed isolated nodes, i.e., nodes that were not connected to other nodes, after this projection. Accordingly, we then selected only the ratings belonging to the users present in the resulting subgraph. This data preparation made the graph shrink to a smaller graph containing approximately 217,494 nodes, and 212,373 edges. Accordingly the Yahoo! Movies data reduced from more than 21 millions ratings, to 1.8 millions ratings.

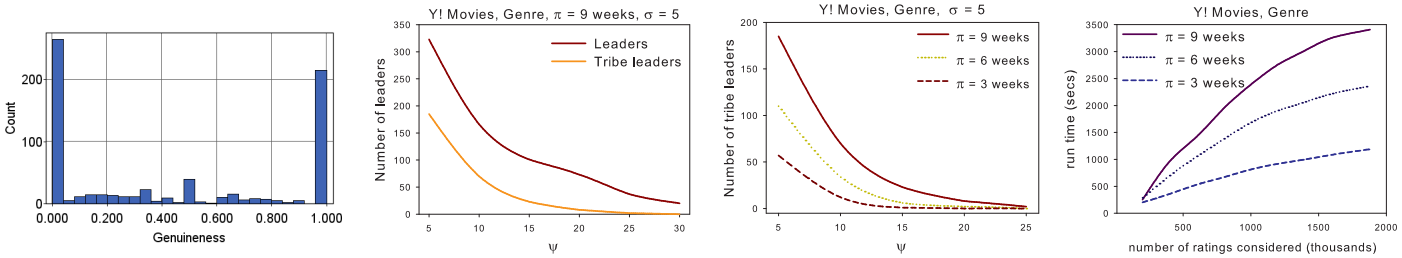
In Figure 4 we report some statistics about the dataset we generated. The social graph is quite disconnected: it contains 46,650 connected components, one of which has 94,032 nodes (see the bottom-right corner of Figure 4(b)), while all the others have less than 50 nodes. In Figure 4(a) the distribution of nodes degree is reported. It should be noted that our graph exhibits a quite low density w.r.t. what is expected from a social network graph: this is natural as our graph is not a social network in its whole, but it is just a projection of a social network on a subset of its nodes, i.e., the set of users which performed actions.

Moreover, due to the sparsity of the Yahoo! graph that we constructed, in the following, we report experiments conducted not at the granularity of the movie, but going a step up in the hierarchy, namely at the granularity of *genres*. When we roll up to the level of genre, we have an average of 7.62 actions per user, and an average of 46093 ratings per genre. Note that when we go up in the granularity we have to face the following situation: the same user may execute the same actions more than once at different timestamps (e.g., rating different movies corresponding to the same genre).



(a) (b) (c) (d)

Figure 5: (a) Number of leaders found on Yahoo! Movies dataset, with $\pi = 9$ weeks, for $\sigma \in [1,10]$ and $\psi \in [5,50]$; (b) number of confidence leaders with $\psi = 5$ and varying confidence threshold; (c) number of genuine leaders with $\psi = 5$ and varying genuineness threshold; (d) number of leaders with varying σ and π.



(a) (b) (c) (d)

Figure 6: (a) Histogram of genuineness of leaders for $\pi = 9$ weeks, (b) comparison between number of leaders and number of tribe leaders with varying ψ, (c) comparison of number of tribe leaders for three different π, (d) and the corresponding run time.

This situation is not only practically relevant but it also theoretically interesting as also discussed later in Section 6. When a user performs an action multiple times, in principle we may use any occurrence of action as a basis for defining propagation of influence. In the experiments we used the first occurrence of a given action, as we believe it is a more reliable indicator of influence propagation. At any rate, measuring influence based on first occurrence is certainly representative of the options available.

5.2 Mining Leaders

In Figure 5 we report the number of leaders found in the Yahoo! Movies dataset for various combinations of the input parameters. All these graphs correspond to a time window of $\pi = 9$ weeks. Figure 5(a) measures the number of leaders as a function of σ and ψ. Recall σ controls the minimum number of actions where leadership is required in order for a user to be declared a leader, while ψ controls the minimum number of influenced users. As expected, as these thresholds go up, the number of leaders drops. E.g., when $\sigma = 5$ and $\psi = 5$, the number of leaders is 323 and it drops to 59 when σ becomes 10 and ψ becomes 10. Figure 5(b)-(c) depict the variation in confidence leaders and genuine leaders as a function of the various parameters. They can all be seen to exhibit similar behavior. In Figure 5(d), we see how the number of leaders varies as a function of π and σ. When π is increased from 3 to 6 weeks at any given σ, the number of leaders nearly doubles. This happens again when π is raised from 6 to 9 weeks. In Figure 6(a) the histogram of genuineness is reported: it shows that the concept of genuineness is almost binary, clearly cutting between leaders that are dominated by other leaders and truly genuine leaders.

5.3 Mining Tribe Leaders

In Figure 6(b) a comparison between number of leaders and number of tribe leaders with varying ψ is reported, while comparison of number of tribe leaders for three different π is reported in Figure 6(c). Obviously for larger ψ and smaller π we extract a lower number of leaders. In Figure 6(d), we report the run time for extracting the influence vectors needed to mine tribe leaders. Recall that extracting the influence vector only depends on the time threshold π and not on the other two thresholds σ and ψ. The plot only reports the run time needed for the construction of the influence vectors: after this step, one frequent itemsets extraction is needed for each leader to see if it is also a tribe leader. The run time for this second phase is not reported for two reason: firstly this is based on pre-existing results and it is not contribution of this paper; secondly, we found this execution time was always very small and negligible w.r.t. the time needed to construct the influence vectors. Note that as expected run time is larger for larger time threshold π, and it is linear or sublinear w.r.t. the size of the actions log (reported on the x axis).

5.4 Qualitative evaluation

Table 1 reports the top 10 tribe leaders w.r.t. size of their largest tribe. Note that this is not by any means an indicator that these are the *best* tribe leaders. We could have shown the top 10 by genuineness, or by confidence times genuineness, but this way we would have missed the possibility of discussing leaders with very low genuineness or confidence. Thus we choose to inspect those ones with the largest tribes and to see their level of confidence and genuineness. We use

$\pi = 3\,weeks$, $\sigma = 5$

rank	user_id	tribesize	#actions	conf.	genu.
1	198671	15	15	0.58	0.4
2	199726	13	8	1	1
3	130514	12	17	0.89	0.94
4	206280	12	11	0.92	1
5	68241	11	12	0.92	0
6	22830	11	19	0.86	1
7	20045	11	9	0.5	0.22
8	170467	10	7	1	0.29
9	75923	10	9	0.75	1
10	81903	10	11	0.65	0.64

(a)

$\pi = 9\,weeks$, $\sigma = 5$

rank	user_id	tribesize	#actions	conf.	genu.
1	98711	25	11	0.85	0.73
2	31018	25	9	1	1
3	170467	23	8	1	0
4	20045	22	11	0.61	0.55
5	66331	21	13	0.81	0.92
6	27381	21	16	0.57	0.63
7	85363	20	5	0.63	0.8
8	144314	20	19	0.86	0.1
9	153181	19	22	0.76	0.82
10	206280	19	12	1	0.67

(b)

Table 1: Top 10 tribe leaders (those with the largest tribe) for two different values of π.

two very distant values of ψ, and for $\sigma = 5$, i.e., they have to influence the same tribe for at least 5 different actions. The table reports the leader user_id, the size of its largest tribe, the number of actions in which it was a leader (a standard one not a tribe leader), the confidence value, and the genuineness value.

There are many interesting things worth noting in these two tables. The first thing is that large tribe leaders usually exhibit high confidence. Consider that for $\pi = 3\,weeks$ and $\sigma = 5$ the average confidence recorded among the leaders is approximately 0.5. For genuineness the results are different. Among the top 10 tribe leaders w.r.t. size of their largest tribe, we find approximately half showing very high genuineness and half showing very low genuineness indicating that they are dominated by other leaders. In particular users 170467 and 20045 which are present in both tables, user 68241 in Table 1(a), and user 144314 in Table 1(b). We checked the reason for these low values of genuineness and we found out that these tribe leaders were dominated by other tribe leaders still belonging to the top-10. In particular we found that 206280, 75923 and 198671 dominate 68241, 170467 is dominated by 130514, and 20045 is dominated by 130514 and 22830. Another positive fact worth noting, is that there are three tribe leaders in the intersection of the two top-10 lists.

Finally, let us highlight that being a tribe leader is not simply being a strong leaders, but it is an orthogonal concept. We have found many leaders which were acting as leader in many actions and with a large group of followers, but that in the end resulted not to be tribe leaders. On the other hand, we found leaders acting as leaders in a few actions, but that were always followed by the same tribe. Consider for instance user 31018 in Table1(b): it acts as a leader in only 9 actions, but in all these 9 actions it is always followed by a tribe of the same 25 users, and it is a very genuine leader (genuineness score = 1).

The discussion above confirms the relevance and usefulness of the proposed framework. All the concepts of leaders, confidence, genuineness and tribe leaders make sense per se, and they seems to converge on "strong" leaders. Conjoining these various concepts, we can identify a small set of strong tribe leaders that may be targeted by a viral marketing campaign, for instance by giving them free sample of the product, or by sending to them the new album of their favorite rock band before the official release, in an attempt of maximizing the word-of-mouth effect.

6. CONCLUSIONS AND DISCUSSION

We introduced a novel data mining approach to social networks analysis, based on frequent pattern discovery. While frequent pattern mining has been studied a lot in the last decade and it has found many real-world application scenarios, to the best of our knowledge this is the first proposal of a framework based on frequent pattern mining for discovering leaders in social networks.

In particular, we considered social networks where users perform actions such as tagging, buying or rating an item. Given a database consisting of a social graph together with a chronologically ordered log of user actions, we motivated the problem of extracting leaders of various flavors from this database. To that end, we proposed notions of leaders and tribe leaders together with possible additional properties such as confidence and genuineness. We developed efficient algorithms for extracting all types of leaders proposed by us. A notable feature of our algorithms is that all of them make one scan of the action log table. This is accomplished by performing two key operations – *update* and *propagate* – on the fly while sliding a window through the action log table backward in time. Various optimizations were incorporated in the implementation of our algorithms. A key summary data structure employed in our algorithms is called an influence matrix. In the case of tribe leaders, we use an influence cube instead of the matrix.

We conducted extensive experiments on a combination of Yahoo! Instant Messenger dataset forming the social graph with user ids correlated to user ids in Yahoo! Movies dataset. In our experiments, we demonstrated the scalability of our algorithms, and we showed many real interesting patterns of leadership, establishing that the notions of leaders proposed in this paper are useful and interesting. This work complements work in the area of viral marketing [22, 16] as pointed out in Section 2.

One potential issue with using pattern discovery for mining leaders is the effect of popular actions. Actions that are very popular may be performed by many users without there really being any influence. This is similar to the famous beer-diaper problem in association rule mining. One way to combat this problem is to discard the most popular actions from the action log and use the rest of the log for mining leaders. Another approach is to associate a "credit" with each action, which is inversely proportional to its popularity and use the credit to determine leadership.

We are currently collecting other data sets where the social network graph comes together with an action log: we plan to test our algorithms on larger and denser data, both for quantitative and qualitative evaluation. In future, it'd be interesting to investigate whether patterns extracted from this approach could be used to determine the parameters required for the viral marketing problem. Furthermore, as discussed in the experiments section, in order to determine interesting patterns, we needed to roll up from the level of individual movies to a more general level of genre, director, or actor. This is a feature that depends on the dataset. Given classification hierarchies on users, actions, and time, it is interesting to ask whether *leadership cube* corresponding to multi-level generalizations over different dimensions can be computed efficiently. Finally, it is worth noting that by mining tribe leaders as a side effect we also mine tribes, that can be considered as small communities. We plan to investigate the relationships between our tribes and the communities structures studied in the literature [14, 18, 23, 11].

Acknowledgments: We acknowledge Aris Anagnostopoulos and Ravi Kumar for the Yahoo! Instant Messenger graph, Seung-Taek Park and David M. Pennock for the Yahoo! Movie dataset.

7. REFERENCES

[1] N. Agarwal, H. Liu, L. Tang, and P. S. Yu. Identifying the influential bloggers in a community. In *Proceedings of the First International Conference on Web Search and Web Data Mining, (WSDM'08)*.

[2] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In *Proc. of the 1993 ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'93)*.

[3] C. Allison. Bit handling in c++, part 1. *C Users Journal*, 11(12):71–90, 1993.

[4] L. Backstrom, D. P. Huttenlocher, J. M. Kleinberg, and X. Lan. Group formation in large social networks: membership, growth, and evolution. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD'06)*.

[5] L. Backstrom, R. Kumar, C. Marlow, J. Novak, and A. Tomkins. Preferential behavior in online groups. In *Proceedings of the First International Conference on Web Search and Web Data Mining, (WSDM'08)*.

[6] F. Bass. A new product growth model for consumer durables. *Management Science*, 15:215–227, 1969.

[7] F. Bonchi, F. Giannotti, A. Mazzanti, and D. Pedreschi. ExAMiner: Optimized level-wise frequent pattern mining with monotone constraints. In *Proceedings of the Third IEEE International Conference on Data Mining (ICDM'03)*.

[8] C. Bucila, J. Gehrke, D. Kifer, and W. White. DualMiner: A dual-pruning algorithm for itemsets with constraints. In *Proceedings of the 8th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD'02)*.

[9] J. Coleman, H. Menzel, and E. Katz. *Medical Innovations: A Diffusion Study*. Bobbs Merrill, 1966.

[10] P. Domingos and M. Richardson. Mining the network value of customers. In *Proc. of the Seventh ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'01)*.

[11] L. Friedland and D. Jensen. Finding tribes: identifying close-knit individuals from employment patterns. In *Proc. of the 13th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'07)*.

[12] J. Goldenberg, B. Libai, and E. Muller. Talk of the network: A complex systems look at the underlying process of word-of-mouth. *Marketing Letters*, 12(3):211–223, 2001.

[13] J. D. Hartline, V. S. Mirrokni, and M. Sundararajan. Optimal marketing strategies over social networks. In *Proceedings of the 17th International Conference on World Wide Web, (WWW'08)*.

[14] J. E. Hopcroft, O. Khan, B. Kulis, and B. Selman. Natural communities in large linked networks. In *Proc. of the Ninth ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'03)*.

[15] J. Huang, Z. Zhuang, J. Li, and C. L. Giles. Collaboration over time: characterizing and modeling network evolution. In *Proceedings of the First International Conference on Web Search and Web Data Mining, (WSDM'08)*.

[16] D. Kempe, J. M. Kleinberg, and É. Tardos. Influential nodes in a diffusion model for social networks. In *Automata, Languages and Programming, 32nd International Colloquium, (ICALP'05)*.

[17] D. Kempe, J. M. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *Proc. of the Ninth ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'03)*.

[18] R. Kumar, J. Novak, and A. Tomkins. Structure and evolution of online social networks. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD'06)*.

[19] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. S. Glance. Cost-effective outbreak detection in networks. In *Proc. of the 13th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'07)*.

[20] L. Licamele and L. Getoor. Social capital in friendship-event networks. In *Proc. of the Sixth IEEE Int. Conf. on Data Mining (ICDM'06)*.

[21] V. Mahajan, E. Muller, and F. Bass. New product diffusion models in marketing: A review and directions for research. *Journal of Marketing*, 54(1):1–26, 1990.

[22] M. Richardson and P. Domingos. Mining knowledge-sharing sites for viral marketing. In *Proc. of the Eighth ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'02)*.

[23] C. Tantipathananandh, T. Berger-Wolf, and D. Kempe. A framework for community identification in dynamic social networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD'07)*.

[24] T. Valente. *Network Models of the Diffusion of Innovations*. Hampton Press, 1955.