Digital Library for Topic Hierarchies from Information Streams

Krishna Y. Kamath
Texas A&M University
College Station, TX 77843
kykamath@cs.tamu.edu

ABSTRACT

In this project we are interested in developing a digital library for discovering, archiving and browsing topic hierarchies from information streams. In this direction, we propose a framework that enables us to achieve this goal. We also present the details of algorithms that can be used discover topic hierarchies from information streams. We finally give the details of our dataset and evaluation methods we will be using to test the framework.

1. INTRODUCTION

In this project we are interested in developing a framework for discovering, archiving and browsing topic/concept hierarchies from information streams. Discovering concept hierarchies enables us to understand the entities that are *driving* an event in social media. These concept hierarchies can also be used in applications like recommendation engines, search etc. Since events occur at different overlapping time scales, we want to develop techniques that can allow us to determine concept hierarchies at different time scales as well.

In Figure 1 we show an example of events during Middle East revolutions of 2011. In these revolutions we could identify several concept hierarchies that were driving it at different times. Initially, the event was driven by Egypt, followed by happenings in Libya and then Syria. All of these are related to Middle East, but each one was driving the event at different times. Similar to this, at a much smaller time frame, we can see events like super bowl, which are being driven by various happenings around the game.

2. PROBLEM STATEMENT

Before describing the challenges in our problem, we first describe some general properties of algorithms that are designed for social media related information streams [8]:

Property 1: As large amounts of data is generated in data streams at a very high rate, it is unrealistic to keep entire

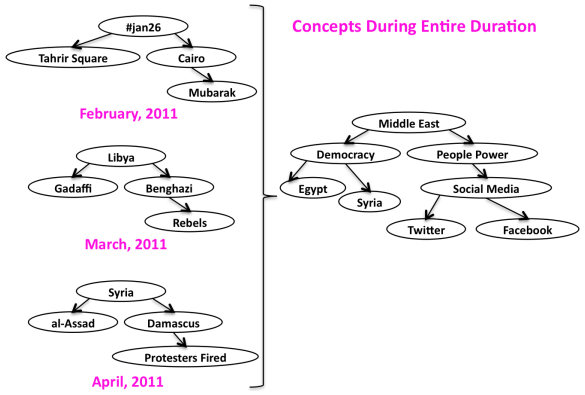

Figure 1: Examples of concept hierarchies during Middle-East revolutions in 2011.

stream in main memory or even secondary storage during analysis. Hence, these algorithms should be single-pass algorithms that use approximation techniques like sampling, randomization, Fourier transforms etc., to summarize information flowing in streams.

Property 2: The results from stream analysis should be available in real-time. Hence, these algorithms should have the ability to process collected information quickly to generate results.

Property 3: Unlike general streams, streams in social media are highly susceptible to concept drift. In addition, the properties of streams also follow temporal and event driven patterns. Hence, algorithms developed for these streams should also be robust against changes occurring in streams.

With these properties in mind, we describe the challenges for topic hierarchy extraction from information streams:

- The first challenge is to identify phrases that can be used to build a concept hierarchy. The simplest approach is to use all the phrases that appear in the stream. But, this approach is not scalable for information streams. Hence, we sample only trending phrases from the stream to discover concept hierarchies (Property 1).

- The next challenge is to identify trending phrases from a stream. One approach, is to think of this problem as a version of frequent items problem, where the item

score decays with time, and select top-k items as trending phrases. But, this approach neither guarantees that the top-k phrases returned are all trending, nor that all trending phrases are returned. In addition, the value of k is hard to set and does not depending on the dynamic stream properties. The details of this are given in Section **??**. Hence, we propose an approach that identifies all the trending phrases in a stream and is flexible to the changing stream properties (Property 3).

- Now to determine the properties of information streams that impacts discovery of trending phrases. In this paper, we identify these properties and show how values for these properties can be calculated empirically for an information stream.

- The next challenge is to make sure the algorithm we discover is efficient (Property 2).. The efficiency of our approach can be split into 2 parts: (i) Efficiency during trend discovery and (ii) Efficiency of concept hierarchy discovered. It is important that our algorithm is efficient and has the ability to generate real-time results (Property 2). We describe our approach to improve the efficiency of our algorithm in Section 5 and show the performance in Section refsec:experiments. We then compare the concept discovery time and quality of concepts discovered by our approach with the approach used in [14] in Section 7

3. RELATED WORK

In the problem of determining aggregates on data streams, which is made up of objects ordered in time, the challenge is to maintain statistics like count, mean, etc., about the objects in the stream. Our problem of finding trending phrases is closely related to a problem in this domain called frequent items (FI) finding problem. In FI problem, a data structure maintains the score of objects seen on the stream, using which we can determine top-k items. To be more accurate, our problem is related to an variant of FI problem called time-decayed FI problem, where recent objects have higher scores than the older ones.

Cormode and Hadjieleftheriou in [3] give details of several algorithms that have been proposed for FI problem. Based on [3], the solutions to FI problem can be split into counter-based algorithms [13, 11, 12], quantile algorithms [9, 15] and sketches [2]. One approach that has been proposed to deal with the time-decayed FI problem is using a sliding window [5, 1, 10]. In these algorithms, the data structure maintains information for only last n items or n items during the past t time units. One of the issues with this approach is the need to maintain extra space for older windows. The performance of the algorithm is dependent on the length of sliding window. To overcome these issues an approach that uses exponentially decayed weights of items has been proposed by Cormode et. al. [4]. They show the performance of their algorithm using decay is close the algorithms that don't use decay.

Many of these algorithms use a data structure of fixed size k, that is updated during processing of the stream. When the data structure is full, an item is removed from it using some criterion and the newly observed item is added. Using a fixed size data structure of size k has the advantage

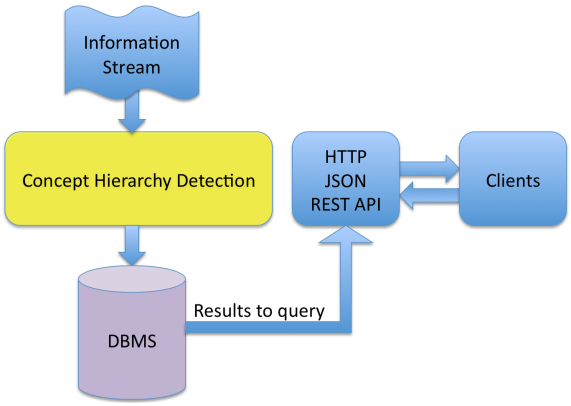

Figure 2: Architecture for Topic Detection from Information Streams.

of not having to deal with problems associated with memory space. But, the dis-advantage of this approach is that the performance of an algorithm is highly dependent on k, which might not be easy to determine. A way to overcome the limitations imposed by a fixed-size data structure is to have an additional pruning step, that remove old elements using some heuristics. In our problem we are not interested in finding top-k elements but determining trending phrases which might be larger or smaller than a constant k, depending on the nature of stream at that time. Hence, we will be using an approach that uses decaying phrase scores with the additional step of pruning.

4. MATERIALS AND METHODS

The architecture for topic detection from information streams is shown in Figure 2. The concept hierarchy detection module takes the information stream as input. The topic hierarchies obtained from the module is saved in a DBMS. We then develop a HTTP JSON REST API that provides an interface for clients to access the topic hierarchies discovered by the module. The API interacts with DBMS to retrieve information requested by clients. We now present the details of concept hierarchy detection module next.

As shown in Figure 3, the problem of concept hierarchy detection can be broken into 4 sub-problems: (i) Trending Phrases Discovery Problem: This problem helps us determine phrases (concepts) that are trending in the stream. (ii) Trending Phrase Graph Problem: Given trending phrases at any-time, this problem helps us to determine relationships between various trending phrases. Representing related trending phrases as graph allows us to group together concepts related to a specific event and hence discover the event. (iii) Graph Union Problem: This problem helps us determine events and driving concepts using trending phrase graphs at different time-scales and at different granularities. (iv) Phrase Topic Hierarchies Problem: This helps us take a trending phrase graph and determine various events in it and the concepts that are driving the event in that graph.

The proposal is organized as follows. In Section 5 we work on the first two problems related to determining trending phrases and developing relationships among the trending phrases. We then describe the next two problems of combining trending phrase graphs at different time-scales and

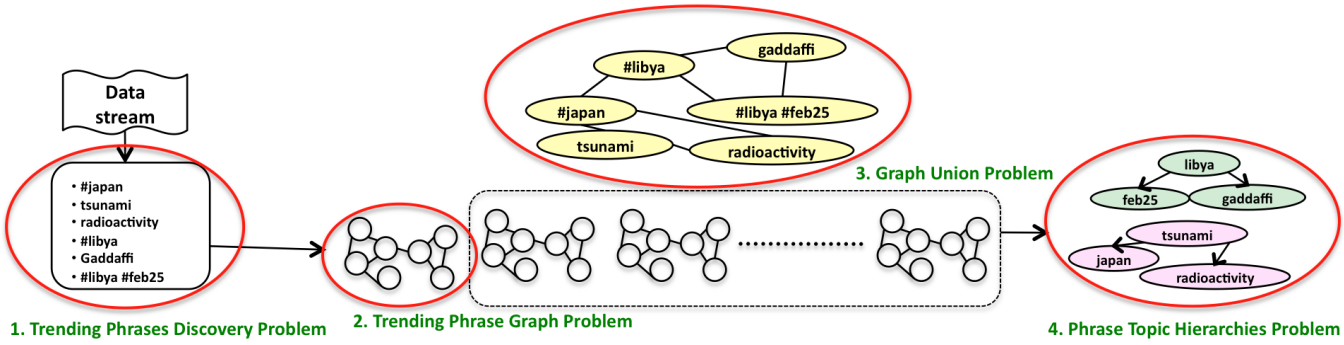

Figure 3: Problem description.

identifying topic hierarchies in them in Section 6. We then describe our dataset and some experiments to evaluate our methods in Section 7.

5. TRENDING PHRASE GRAPH

5.1 Basic Definitions

DEFINITION 5.1. *(**Data Stream**) A data stream $D = \{(d_1, t_1), (d_2, t_1), (d_3, t_2) \dots\}$ is an ordered set of document tuples. The stream D is of infinite size and is ordered in a non-decreasing fashion by time-stamp values of the documents. Note that, there might be multiple documents in the stream that share the same time-stamp.*

Examples of data streams that satisfy this property are tweets, blogs, mobile short messages, etc.

DEFINITION 5.2. *(**Candidate Phrase**) Consider a document $d \in D$, where $|d|$ equal to the number of terms in the document. Let, P_k be a set of phrases of length k. Then, d can be represented using the set of phrases $P = P_1 \cup P_2 \cup \dots \cup P_{|d|}$. Such a phrase $p \in P$ is called a candidate phrase and the set of all the candidate phrases, observed in the data stream till time t, is represented as C_t.*

For example, the document "Packers win Superbowl" can be represented by P = {"packers", "win", "superbowl", "packers win", "win superbowl", "packers win superbowl"}. The phrases "superbowl", "packers win", etc are examples of candidate phrases and $P \subseteq C$.

DEFINITION 5.3. *(**Candidate Phrase Instance**) A single candidate phrase p may appear multiple times in the stream. Each such instance of the phrase p_i is called candidate phrase instance.*

For example, consider the document tuples ("Packers win", t_1) and ("Champions packers ", t_2). In this case, the candidate phrase "packers" has two instances each with time-stamp t_1 and t_2 respectively. Similarly, "win" and "champions" have single instances.

Every candidate phrase instance p_i is associated with a score, that decreases as the phrase ages, and a time-stamp t_i of the document in which the instance was observed. The score $S(p_i, t)$ for p_i at time t is given by:

$$S(p_i, t) = \lambda_c^{t-t_i} \tag{1}$$

where $\lambda_c \in (0, 1)$ is a constant know as ***phrase score decay rate***.

DEFINITION 5.4. *(**Candidate Phrase Scores**) In a stream we can observe several instances of a candidate phrase p. Let, $E(p, t)$ be the set of all the instances of p that have been observed until time t. Then, the candidate phrase score at t, $S(p, t)$ is defined as the sum of all the candidate phrase instances of P.*

$$S(p, t) = \sum_{p_i \in E(p,t)} S(p_i, t) = \sum_{p_i \in E(p,t)} \lambda_c^{t-t_i}$$

The definition of $S(p, t)$ requires us to remember all the previous observed instances $E(p, t)$ for p and its calculation taken $O(|E(p, t)|)$ time. This is in-efficient in real-world system where the number of phrases will be very large. Hence, we prove a proposition that will help us calculate this score efficiently in $O(1)$ time and does not have the requirement of storing $E(p, t)$.

PROPOSITION 5.1. *Given a candidate phrase p with score $S(p, t_l)$ at time t_l, if a phrase instance p_i for p is observed at time t_n, where $t_n > t_l$ then the new score for p at t_n, $S(p, t_n)$ is given as:*

$$S(p, t_n) = \lambda_c^{t_n - t_l} S(p, t_l) + 1$$

PROOF. Let $P = \{p_1, p_2, \dots p_m\}$ be the set all the instance of the candidate phrases that have been observed till t_n. Hence,

$$S(p, t_l) = \sum_{p_i \in P} S(p_i, t_l)$$

Using (1), $\forall p_i \in P$ we can write:

$$S(p_i, t_n) = \lambda_c^{t_n - t_i} = \lambda_c^{(t_n - t_l) + (t_l - t_i)} = \lambda_c^{(t_n - t_l)} \lambda_c^{(t_l - t_i)}$$

$$S(p_i, t_n) = \lambda_c^{(t_n - t_l)} S(p_i, t_l)$$

where t_i is the time-stamp of p_i.

Using this we can write,

$$S(p, t_n) = \sum_{p_i \in P} S(p_i, t_n) + 1 = \sum_{p_i \in P} \lambda_c^{(t_n - t_l)} S(p_i, t_l) + 1$$

$$S(p, t_n) = \lambda_c^{(t_n - t_l)} \sum_{p_i \in P} S(p_i, t_l) + 1 = \lambda_c^{(t_n - t_l)} S(p, t_l) + 1$$

□

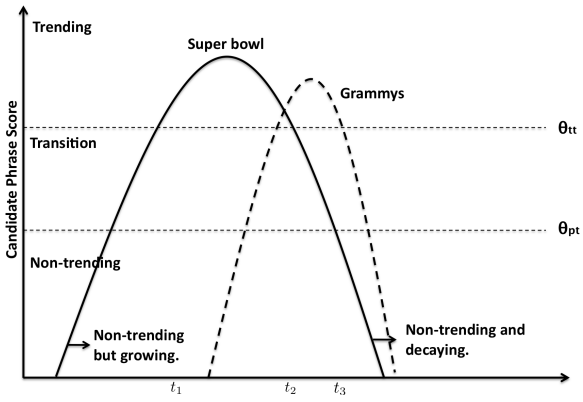

Figure 4: Examples for trending phrases discovery problem.

5.2 Trending Phrases Discovery Problem

PROBLEM 1. *(**Trending Phrases Discovery Problem**) Given an infinite data stream D and the set of candidate phrases C_t observed in D, the trending phrases discovery problem is to identify the subset of phrases $\Gamma_t \subseteq C_t$ that are trending at time t.*

We can solve the trending phrases discovery problem, by defining a method to verify if a candidate phrase p is popular or not. We can use the candidate phrase scores, discussed in the previous section, and ***trending threshold*** θ_{tt} to determine the popularity of p. Hence, for time t we have:

$$\Gamma_t = \{p \mid S(p,t) \geq \theta_{tt} \forall p \in C_t\} \quad (2)$$

Note that the value of θ_{tt} is dependent on the number of candidate phrases in C_t. This is done so that the discovery of trending phrases can adjust to temporal changes in the data stream.

To understand this approach take a look at the example shown in Figure 4. It shows variation in scores for two candidate phrases "super bowl" and "Grammys" with time. To keep the example simple we assume that the phrase rate in the data stream doesn't change and hence value of θ_{tt} is unchanged. So, using (2) we have, Γ_{t_1} = {"super bowl"}, Γ_{t_2} = {"super bowl", "Grammys"} and Γ_{t_3} = {"Grammys'}.

Pruning Candidate Phrases: In real-world applications, the size of candidate phrases C_t becomes very large. This makes the determination of Γ using (2) in-efficient. Hence, we describe a method to determine a set of candidate phrases to prune, C'_t using ***pruning threshold*** θ_{pt} , where $C'_t \subseteq C_t$. Like trending threshold, pruning threshold is also dependent on C_t.

Using the candidate phrase score, the two thresholds θ_{tt} and θ_{pt}, split C_t into 3 sets: trending, transitional and non-trending. An example of this is shown in Figure 4. Though the candidate phrases to prune are in the non-trending set, not all the phrases in this set can be pruned. This is because, the candidate phrases that are in non-trending set can be either be trending up or trending down. Figure 4 shows both the cases for the phrase "Super bowl". The phrases to be pruned are the ones that are trending down, but we cannot determine the direction of the trend just based on the candidate phrase score. Hence, we will the using some heuristic approaches to prune the candidate phrases.

For a candidate phrase $p \in C_t$ that is in the non-trending set we use the last time a phrase instance of p was observed, t_p to decide if it should be pruned. To determine C'_t, we propose two approaches that use t_p: (i) deterministic approach (ii) randomized approach. These approaches also use a parameter ***maximum phrase inactivity time*** T_{pi}.

Using the deterministic approach C' is defined as follows:

$$C'_t = \{p \mid S(p,t) \leq \theta_{pt} \text{ and } (t - t_p) \geq T_{pi} \ \forall p \in C_t\} \quad (3)$$

The randomized approach uses a function $R(p,t)\ \forall p \in C_t$ defined as:

$$R(p,t) = \begin{cases} 0 & \text{if } t - t_p < T_{pi} \\ 1 & \text{if } t - t_p \geq 2T_{pi} \\ coinFlip(\frac{t-t_p}{2T_{pi}}) & \text{Otherwise} \end{cases}$$

where, $coinFlip : [0,1] \rightarrow \{0,1\}$, is a function where the probability of 1 increases as input is closer to 1. Now, C'_t is defined as follows:

$$C'_t = \{p \mid S(p,t) \leq \theta_{pt} \text{ and } R(p,t) = 1 \ \forall p \in C_t\} \quad (4)$$

5.3 Trend Threshold Parameters

In the previous section we described how θ_{tt} and θ_{pt} can be used to determine Γ efficiently. In this section, we describe methods to calculate these threshold parameters using data stream properties.

We first give a lemma that gives us the upper bound on the sum of scores for all the candidate phrase instances that have been observed on the data stream. We will then use the lemma to show the theorem that determines the thresholds. The proofs have been left out of this proposal.

LEMMA 5.1. *Given a data stream D with a phrase rate of η and the set of all candidate phrase instances P_t that have been observed on D during time intervals $0, 1, 2...t$, we have:*

$$\sum_{p_i \in P_t} S(p_i,t) \leq \frac{\eta}{1-\lambda_c} \quad \text{and} \quad \lim_{t\to\infty} \sum_{p_i \in P_t} S(p_i,t) = \frac{\eta}{1-\lambda_c}$$

THEOREM 5.1. *Given a data stream D at a phrase rate η and the set of candidate phrases C_t at time t, the trending threshold θ_{tt} and the pruning threshold θ_{pt} are given as:*

$$\theta_{tt} = \frac{\eta S_t}{|C_t|(1-\lambda_c)} \quad \text{and} \quad \theta_{pt} = \frac{\eta S_p}{|C_t|(1-\lambda_c)}$$

where, $S_t > 1$ and $S_p < S_t$ are data stream specific constants.

The values of S_t and S_p are still unknown and depend on the data stream and the application requirements. We, will describe methods to estimate these values in Section 7 .

5.4 Trending Phrase Graph Problem

PROBLEM 2. *(**Trending Phrase Graph Problem**) Given data stream D and the set of latest trending phrases Γ, construct an un-directed graph $G_{\gamma t}(V,E)$ that describes the relationships between trending phrases at time t, such that $V = \Gamma$ and the edge weight $w(u,v)\ \forall\ u,v \in V$, measures closeness between the phrases u and v.*

We use two features to measure the closeness (edge-weights) between phrases:

- **Co-occurance**: Phrases that appear together more often should be closer to each other than phrases that don't appear together. This feature helps in formation of clusters of related phrases which will be used in topic hierarchy detection.
- **Temporal locality**: Phrases co-occurring in recent time intervals should be closer to each other than the phrases that have co-occurred in previous time intervals. This feature helps us to model dynamics of topic evolution like introduction of new phrases into an existing topic cluster and removal of older phrases, topic split, topic merge etc.

Consider phrases u and v, that last co-occurred at t_l and have edge weight $w_{t_l}(u,v)$. When these phrases co-occur again at time t_n, we calculate the new edge weight as:

$$w_{t_n}(u,v) = w_{t_l}(u,v)\lambda_\gamma^{t_n - t_l} + 1 \quad (5)$$

where $\lambda_\gamma \in [0,1]$ is a constant called phrase graph edge decay rate. To model temporal locality, that we described earlier, we decay the edge weights in phrase graph using (5).

5.5 Trending Phrase Graph Algorithm

In this section, we describe the trending phrase graph algorithm that we have developed based on the analysis in the previous sections. The complete algorithm is given in Algorithm 1.

The algorithm first initializes Γ to an empty set. Γ contains the set of latest trending phrases that are generated every T_γ intervals. The algorithm operates on a single document during every iteration. For every document, it first retrieves the phrases in the document and updates the candidate phrase scores for each phrase. It also, updates the edges in $G_{\gamma t_i}$ for every pair of phrases that are trending. Every T_{cp} time intervals C_{t_i} is pruned as described in Section 5.2. Similarly, every T_γ time intervals it generates the set of latest trending phrases Γ.

Algorithm 1 TrendPhrasesDiscovery

Initiliaize: $\Gamma = \emptyset$
while True **do**
1. Process document d_i: Get the next document d_i, generated at t_i, from the input data stream. Extract phrases P_i from the document.
2. Update C_{t_i}: Update the score of every phrase $p \in P_i$, where t_p is the time the last time p was updated, using:

$$S(p,t_i) = \lambda_c^{t_i - t_p} S(p,t_p) + 1 \qquad [\because \textit{Proposition } 5.1]$$

3. Update $G_{\gamma t_i}$: For every pair of phrases in $\{(u,v) | \forall\ u,v \in P_i \cap \Gamma\}$, if an edge exists between u and v update it using (5). If an edge does not exist add and edge with $w_{t_i}(u,v) = 1$.
4. Prune C_{t_i}: Every T_{cp} time intervals, prune C_{t_i}. Before pruning, for every phrase $p \in C_{t_i}$ update its candidate phrase score using:

$$S(p,t_i) = \lambda_c^{t_i - t_p} S(p,t_p)$$

Now, prune C_{t_i} using either Equation 3 or Equation 4.
5. Update Γ: Every T_γ time intervals, re-generate the set of trending phrases Γ_{t_i} from C_{t_i}. Initialize Γ_{t_i} to $\emptyset$ and for every phrase $p \in C_{t_i}$ update its candidate phrase score using:

$$S(p,t_i) = \lambda_c^{t_i - t_p} S(p,t_p)$$

Now, re-generate Γ_{t_i} using Equation 2. Set $\Gamma = \Gamma_{t_i}$. Also remove phrases from $G_{\gamma t_i}$ that are not present in Γ.
end while

6. PHRASE HIERARCHY RETRIEVAL

6.1 Graph Union Problem

Problem 3. *(**Graph Union Problem**) For a data stream D, given a time interval $I = [t_l, t_u]$ and graph union threshold θ_u, construct an un-directed graph $G_u(V_u, E_u)$ that describes the relationships between trending phrases during the entire interval I. The edge weights $w_u(u,v)$, $\forall\ u,v \in V_u$ measures closeness between u and v. Consider, t_l and t_u to be multiples of a constant δ.*

In the interval I, t_l and t_u are multiples of δ, hence, I can be broken into $n = \left(\lfloor \frac{t_u - t_l}{\delta} \rfloor + 1\right)$ time intervals of equal length. For example, when $\delta = 1$, for the interval $[1,10]$, $n = 10$; and for $[3,7]$, $n = 5$. So, we generate the trend phrase graphs G_γ at regular intervals of length δ and then combine the graphs using some method to construct G_u for I. We can construct G_u using two different approaches: (i) Linear graph union, which has a complexity of $O(n)$, (ii) Logarithmic graph union, which with moderate amount of preprocessing has a complexity of $O(\log_2 n)$

Generic Graph Union Algorithm: We design an algorithm to determine the graph union G_u, given a set of graphs G'_u, from which it should be constructed. This algorithm will be used by both the linear and logarithmic graph union methods, described later in this section. The complete algorithm is given in Algorithm 2. The algorithm takes graph set G'_u, interval I and the threshold θ_u as input and constructs G_u.

In the first step of the algorithm, we combine all the vertices and edges in G'_u to form G_u. For all vertices $(v_1, v_2) \in V_u$, the parallel edges in G_u are then replaced by a single edge (v_1, v_2) and the edge weight $W_u(v_1, v_2, I)$ is set to the sum of parallel edges between v_1 and v_2. If there is a single edge between (v_1, v_2) then $W_u(v_1, v_2, I)$ is set to the weight of that edge. In a way, the value $W_u(v_1, v_2, I)$ is equal to the number of times edge (v_1, v_2) occurs in the interval I.

In the next step, we filter G_u to remove edges that don't meet the graph union threshold criterion θ_u. This threshold specifies the percentage of time intervals in which a edge must have existed. This corresponds to the number of times two trending phrases are related to each other in I.

Linear Graph Union: In this approach, we create the set of graphs G'_u that contains all the G_γ graphs for the interval I.

$$G'_u = \{G_{\gamma t_l}, G_{\gamma t_l + \delta}, G_{\gamma t_l + 2\delta}, \ldots, G_{\gamma t_u}\}$$
$$G'_u = \{G_{\gamma t}\}_{t=t_l}^{t_u}$$

Algorithm 2 GRAPHUNION(G'_u, I, θ_u)

1. Construct $G_u(V_u, E_u)$: Construct the union graph $G_u(V_u, E_u)$, such that:

$$V_u = \cup_{G_{\gamma t_i} \in G'_u} V_{\gamma t_i}$$
$$E_u = \cup_{G_{\gamma t_i} \in G'_u} E_{\gamma t_i}$$

For every pair of connected vertices $(v_1, v_2) \in V_u$, set $W_u(v_1, v_2, I)$ to the sum of weights of all the parallel edges between (v_1, v_2). Replace all the parallel edges with a single edge.

2. Filter $G_u(V_u, E_u)$: Remove any edge $(v_1, v_2) \in E_u$, that meets the condition:

$$W_u(v_1, v_2, I) < \theta_u \left(\lfloor \frac{t_u - t_l}{\delta} \rfloor + 1 \right)$$

For every graph in G'_u, we set all the weight of all edges to 1. With the modified, G'_u, graph union algorithm described in Algorithm 2 can be used to construct G_u. Since, G'_u has a cardinality of n, the algorithm has a complexity of $O(n)$.

Logarithmic Graph Union: Like before, we generate a trend phrase graph G_γ at every interval of length δ. But, in addition to these graphs, we also generate extra graphs called combined trending phrase graphs G^k_γ that are formed by combination of k different G_γ graphs.

Combining information from smaller G_γ graphs enables us to develop efficient algorithms for construction of G_u, with moderate amount of preprocessing during the generation of G_γ. G^k_γ is constructed using graphs from $G^{k'}_{\gamma t}$, which consists of trending phrase graphs for the current interval and the past $k-1$ intervals, $G^{k'}_{\gamma t} = \{G_{\gamma(t-i\delta)}\}_{i=0}^{i=k-1}$. For example, when $\delta = 1$, $G^4_{\gamma 4}$ is formed by combining graphs in $G^{4'}_{\gamma 4} = \{G_{\gamma 4}, G_{\gamma 3}, G_{\gamma 2}, G_{\gamma 1}\}$.

We generate $G^k_{\gamma t}$ using steps similar to that followed in linear graph union. We first set the weight of all edges in all the graphs in $G^{k'}_{\gamma t}$ to 1. We then combine the graphs as decribed in Step 1. of Algorithm 2. But, we don't filter out any edges like we did in that case. Filtering of edges is done only during construction of G_u.

To build G_u given an interval I, we have to first determine G'_u, the set of combined trending phrase graphs using which it will be constructed. For an interval $I = [a, b]$, the corresponding combined trending phrase graph is $G^{b-a+1}_{\gamma b}$. So, we split I into several intervals for which combined trending phrase graphs are pre-built. The algorithm to generate these intervals is trivial and a recursive algorithm to generate them is given in [7]. For example to build G_u for the interval $I = [1, 7]$, we split I into, $[1, 4]$, $[5, 6]$ and $[7, 7]$, which gives $G'_u = \{G^4_{\gamma 4}, G^2_{\gamma 6}, G^1_{\gamma 7}\}$.

Untill now, we have described how to construct G_u using $G^k_{\gamma t}$, and how to generate $G^k_{\gamma t}$. But, we haven't described when to generate $G^k_{\gamma t}$. We now describe this. In logarithmic graph union method, we construct extra graphs at different times. At any time t, where t is even and a multiple of δ, we generate $\log_2 t + 1$ additional graphs. The graphs generated are $\{G^{\frac{t}{2^i}}_{\gamma t}\}_{i=0}^{\log_2 t}$. For example, when $t = 8$, we generated 4 graphs: $G^8_{\gamma 8}$, $G^4_{\gamma 8}$, $G^2_{\gamma 8}$ and $G^1_{\gamma 8}$.

An example, of graphs generated using these two different methods is shown in Figure 5. We see that, for an interval of length 8δ, the linear graph union approach requires processing of 8 graphs, while logarithmic approach requires only 1 graph.

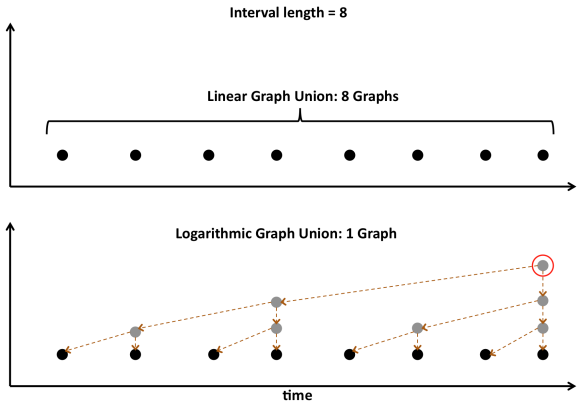

Figure 5: Examples for graph union using linear and logarithmic methods.

We now give a definition of identical graphs and then give a theorem that shows the graphs generated by both these methods are identical.

DEFINITION 6.1. *(**Identical Graphs**) Given an interval I and a threshold θ_u, two graphs G_{u1} and G_{u2} constructed using the graph union methods, described before, are said to be identical iff the following conditions hold:*

$$V_{u1} = V_{u2},\ E_{u1} = E_{u2},\ and$$
$$W_{u1}(v_1, v_2, I) = W_{u2}(v_1, v_2, I)\ \forall\ (v_1, v_2) \in E_{u1}$$

THEOREM 6.1. *Given an interval I and a threshold θ_u, the graph generated by both, linear and logarithmic graph union methods are identical.*

6.2 Phrase Topic Hierarchies Problem

PROBLEM 4. *(**Phrase Topic Hierarchies Problem**) Given an undirected graph $G(V, E)$ of trending phrases, construct phrase topic hierarchies H.*

We divide the task of determining topic hierarchies H using G into two steps. In the first step we cluster G to determine topics. We then use the individual clusters to determine topic hierarchies. We next describe these two steps next.

Graph Clustering: In trending phrases graph G, phrases related to each other are close to each other. Hence, we cluster this graph to determine topic clusters that are trending together. We use the Markov Cluster Algorithm (MCL) [6], to cluster G. MCL is a fast and scalable clustering algorithm that uses simulation of flows in the graph to cluster it. We then use each of these clusters to determine topic hierarchies.

Topic Hierarchies: A topic hierarchy is a directed acyclic graph, which contains phrases as vertices and the relationships between phrases is indicated using directed edges. For every topic cluster in G, we determine the ***root phrase*** and

then construct the hierarchy using breadth first traversal. The vertex to be selected as root phrase should describe the cluster well and hence should be a significant vertex based on some centrality measure. The metric we use to choose root phrase will determine the type of relationship the topic hierarchy describes. We describe two types of topic hierarchies and the metric that is used to generate these hierarchies:

- **Driven By Genrality**: In this method the topic hierarchy is built such that a parent node describes a concept that is more general than the child node. One way to define genrality of a node in a graph is to measure the number of nodes (degree centrality) it is connected to. For example, a node like "Middle-East" will be connected to more phrases in topic cluster about Middle-East than the node "Hosni Mubarak". Hence, we can say "Middle-east" is more general than "Hosni Mubarak".

- **Driven By Popularity**: In Section 5, we assigned scores to phrases, which were used in identifying trending phrases. We can use these scores to build a topic hierarchies such that a parent vertex has a trending score higher than the child vertex. For example, like before, consider the topic cluster about Middle-East, though "Middle-East" is connected to more vertices than "Hosni Mubarak", "Hosni Mubarak" might have a higher trend score than "Middle-East". This shows that "Hosni Mubarak" is in way driving the issues in Middle-East.

In this way, we can use two different metrics to construct phrase topic hierarchies, each giving us a different perspective of information on the real-time web.

7. EVALUATION

In this section, we describe the experimenents that we plan to conduct. The experiments are:

- **Stream Parameters Estimation**: In Section 5, we described an algorithm to extract trending phrases from a data stream. The algorithm uses threshold parameters like trending threshold θ_{tt} and pruning threshold θ_{pt} which are dependent on data stream specific constants like S_t and S_p respectively. The algorithm also uses a parameter called stream parameter maximum phrase inactivity time T_{pi}, to improve its efficiency. We will propose methods to perform stream analysis to determine these parameters.

- **Deterministic Vs Randomized Algorithm**: The trending algorithm can use deterministic or randomized depending on the approach taken to prune the candidate phrase set C_t. The approaches were described in Section 5.2. In this experiment, we will compare these two different approaches.

- **Comparison with Sanderson's Approach**: We will compare our approach to generate topic hierarchies on a data stream with the approach used by Sanderson and Croft [14].

- **Examples of Topic Hierarchies**: We will finally show some examples of the concepts discovered using this approach.

Stream	Tweets[a]	Phrase rate[b]
Trending topic stream	3191.26	7714.04
Domain specific stream	87.54	382.90

[a]Every 5 minutes
[b]Every minute

Table 1: Data stream properties

We first describe details of the data streams that we will be using in our experiments.

7.1 Data Streams For Evaluation

To perform our experiments we will use data collected using Twitter Streaming API [1]. We use two different streams: (i) Trending topic stream, and, (ii) Domain specific stream. To generate the trending topic stream, we get current trending topics using Twitter APIs and use the "track" parameter of the filter method from Streaming API. To generate domain specific stream we first determine prominent users for 4 domains - technology, entertainment, politics and sports, using snowball sampling approach described in [16]. Then for each domain we pick 1250 users and use the "follow" parameter of the filter method from Streaming API to generate domain specific stream. Some statistics from the streams is shown in Table 1. These statistics were calculated on a single day sample of these streams.

8. CONCLUSION

In this project, we proposed a framework for discovering, archiving and browsing topic hierarchies from information streams. We also presented the details of algorithms that can be used discover topic hierarchies from information streams. We finally gave the details of our dataset and evaluation methods we will be using to test the framework.

9. REFERENCES

[1] A. Arasu. Approximate counts and quantiles over sliding windows. In *In PODS*, pages 286–296, 2004.

[2] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming*, ICALP '02, pages 693–703, London, UK, UK, 2002. Springer-Verlag.

[3] G. Cormode and M. Hadjieleftheriou. Methods for finding frequent items in data streams. *The VLDB Journal*, 19:3–20, February 2010.

[4] G. Cormode, F. Korn, and S. Tirthapura. Exponentially decayed aggregates on data streams. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pages 1379 –1381, 2008.

[5] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows (extended abstract), 2002.

[6] S. V. Dongen. Graph clustering by flow simulation. *Mathematics in Computer Science*, 2000.

[7] M. Dubinko, R. Kumar, J. Magnani, J. Novak, P. Raghavan, and A. Tomkins. Visualizing tags over time. In *Proceedings of the 15th international conference on World Wide Web*, WWW '06, pages 193–202, New York, NY, USA, 2006. ACM.

[8] J. Gama. *Knowledge Discovery from Data Streams*. Joao Gama, 2010.

[9] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In *In SIGMOD*, pages 58–66, 2001.

[1]dev.twitter.com

[10] L. K. Lee and H. F. Ting. A simpler and more efficient deterministic scheme for finding frequent items over sliding windows. In *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '06, pages 290–297, New York, NY, USA, 2006. ACM.

[11] G. S. Manku and R. Motwani. Approximate frequency counts over data streams, 2002.

[12] A. Metwally, D. Agrawal, and A. E. Abbadi. Efficient computation of frequent and top-k elements in data streams. In *IN ICDT*, pages 398–412, 2005.

[13] J. Misra and D. Gries. Finding repeated elements. Technical report, Ithaca, NY, USA, 1982.

[14] M. Sanderson and B. Croft. Deriving concept hierarchies from text. In *In Proceedings of the 22nd annual international ACM SIGIR conference on Research and Development in Information Retrieval*, pages 206–213, 1999.

[15] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri. Medians and beyond: New aggregation techniques for sensor networks. pages 239–249. ACM Press, 2004.

[16] S. Wu, J. M. Hofman, D. J. Watts, and W. A. Mason. Who says what to whom on twitter categories and subject descriptors. *Communication*, 2011.