Transient Crowd Discovery on the Real-Time Social Web*

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

James Caverlee
Texas A&M University
College Station, TX 77843
caverlee@cse.tamu.edu

ABSTRACT

In this paper, we study the problem of automatically discovering and tracking *transient crowds* in highly-dynamic social messaging systems like Twitter and Facebook. Unlike the more static and long-lived group-based membership offered on many social networks (e.g., fan of the LA Lakers), a transient crowd is a short-lived ad-hoc collection of users, representing a "hotspot" on the real-time web. Successful detection of these hotspots can positively impact related research directions in online event detection, content personalization, social information discovery, etc. Concretely, we propose to model crowd formation and dispersion through a message-based communication clustering approach over time-evolving graphs that captures the natural conversational nature of social messaging systems. Two of the salient features of the proposed approach are (i) an efficient locality-based clustering approach for identifying crowds of users in near real-time compared to more heavyweight static clustering algorithms; and (ii) a novel crowd tracking and evolution approach for linking crowds across time periods. We find that the locality-based clustering approach results in empirically high-quality clusters relative to static graph clustering techniques at a fraction of the computational cost. Based on a three month snapshot of Twitter consisting of 711,612 users and 61.3 million messages, we show how the proposed approach can successfully identify and track interesting crowds based on the Twitter communication structure and uncover crowd-based topics of interest.

Categories and Subject Descriptors

H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Information networks*

General Terms

Algorithms, Experimentation

*This paper is an extension of our prior work in [8].

WSDM'11, February 9–12, 2011, Hong Kong, China.

Keywords

clustering, social media, community detection, real-time web

1. INTRODUCTION

In much the same way as web search engines provide instant access to the *retrospective web* of previously crawled and indexed content, there is growing excitement over a new generation of applications for monitoring, analyzing, and distilling information from the *prospective web* of real-time content that reflects the current activity of the web's participants. While the space of all possible real-time web updates is large (e.g., small changes to static web pages), there is growing emphasis on a class of interesting and important real-time web updates – updates associated with user-based actions and activities on the social web. The social web is the fastest growing phenomenon on the web, enabling millions of users to generate and share knowledge. Example sites include Facebook, Flickr, and Twitter; similar social systems are increasingly being adopted by governments and enterprises interested in exploiting the emergent collective knowledge ("wisdom of the crowds") embedded in the activities and actions of their users.

As a step towards this vision of a prospective web information platform, this paper focuses on identifying emerging "hotspots" on the real-time social web. In general, a "hotspot" could be defined by the posting and sharing actions of users in social systems, for example triggered by an offline event (e.g., Facebook posts and Tweets in response to a live Presidential debate or a chemical fire at a nearby refinery) or by an online phenomenon (e.g., reaction to Internet memes, online discussion). Detecting these hotspots as they arise in real-time is an important and fundamental building block for enabling new real-time web applications, applications related to identification and dissemination of disaster and emergency-related information, among many other emerging social mining applications.

Towards the goal of identifying online hotspots, we focus in this paper on identifying *crowds* of users through an analysis of their online actions. A single user action – for example, posting a Tweet mentioning a smoke plume at a nearby factory – though perhaps interesting itself, does not convey a strong community or social-based importance to the user action. In contrast, a flurry of activity associated with a "crowd" is a strong indicator of an emergent online phenomenon that may be worth identifying and directing to interested users. Unlike the more static and long-lived group-based membership offered on many social networks (e.g., fan of the LA Lakers), our goal is to support

the discovery of organic and highly-temporal group affiliation, which we refer to as *transient crowds*. A *transient crowd* is a potentially short-lived ad-hoc collection of users bound together by some common thread. For example, transient crowds could be viewed through several overlapping perspectives: (i) *communication-based*, reflecting groups of users who are actively messaging each other, e.g., users coordinating a meeting; (ii) *location-based*, reflecting groups of users who are geographically bounded, e.g., users posting messages from Houston, Texas; and (iii) *interest-based*, reflecting groups of users who share a common interest, e.g., users posting messages about a presidential debate.

Transient crowds are dynamically formed and potentially short-lived. Hence, it is a major challenge to efficiently identify coherent crowds across a potentially vast collection of non-obviously connected user actions. Considering Twitter alone, there are potentially 100s of millions of active users inserting new messages into the system at a high-rate. How can we identify and extract real-time crowds efficiently without sacrificing crowd quality? In addition to identifying a particular crowd at a point-in-time, how can we efficiently and successfully track the crowd over time as users join, crowds merge, and crowds disperse?

We propose to model crowd formation and dispersion through a message-based communication clustering approach over time-evolving graphs. Two of the salient features of the proposed approach are (i) an efficient locality-based clustering approach for identifying crowds of users, and (ii) a novel crowd tracking and evolution approach for linking crowds across time periods. The efficient locality-based clustering is developed on the notions that (i) changes in a small region of a graph should not affect the entire graph; and (ii) that edge weights should reflect temporal and interest locality (e.g., decaying based on communication recency).

2. PROBLEM STATEMENT

We are interested in exploring short-lived group formations on large and growing social messaging systems like Twitter and Facebook. As we have noted, users on these social networks may be grouped along a number of dimensions including content-based (or thematic interest), geographic-based, communication-based, and so on. In this paper, we focus on the specific challenge of uncovering and tracking groups of users – what we refer to as *transient crowds* – according to their communication patterns. Compared to previous works [9, 11] that seek to do fast clustering on an as-needed basis, our approach is to detect and track crowds in real-time (e.g., every minute). This requires both a single-shot fast clustering and cluster evolution to track changes and trends. In addition these previous works use offline algorithms which are not suitable for our requirements.

Historically, direct communication between people has been mostly unobservable or unavailable for large-scale web mining. For example, private email and instant messages between two users are typically not made available for natural reasons. But with the rise of new social messaging systems like Twitter and Facebook, communications between users can be monitored. For example, Twitter supports the public messaging of users through the inclusion of @⟨*username*⟩ in a Twitter post (a "tweet"). So a tweet from the user *nod* can be addressed to the user *kykamath* like so: "@kykamath What do you think about the new iPad?". This type of observable communication is on the rise and a significant portion of all messages posted on Twitter, with estimates placing the percent of all tweets containing the @⟨*username*⟩ at 30% (or about 7 million observable communications per day) [1]. Similar messaging functionality has recently been adopted by Facebook. Based on these observable communication patterns, we study how to efficiently discover and track transient crowds. We now give some definitions before framing the problem.

Definition (Time-Evolving Communication Network) A time-evolving communication network is an undirected graph $G_t(V,\ E)$ graph with $|V| = n$ vertices and $|E| = m$ edges, where each vertex corresponds to a user in the social messaging system and an edge corresponds to a communication between two users. The weight of an edge between vertices u and v at time t is represented by $w_t(u,\ v)$.

The communication network is time evolving because the relationship between users – as indicated by $w_t(u,\ v)$ – changes over time. In practice, the edge weights in a time-evolving communication network could be based on the geographical distance between users, the "semantic" closeness based on an analysis of the content of their messages, or other context-sensitive factors. For concreteness, in this study we focus on purely communication-based properties (the recency and number of messages between the users) for determining the edge weights in the time-evolving communication network.

Definition (Transient Crowd): A transient crowd $C \in K_t$ is a time-sensitive collection of users who form a cluster in G_t, where K_t is the set of all transient crowds in G_t. A transient crowd represents a collection of users who are actively communicating with each other at time t.

Based on these definitions, we can now break our problem into two parts:

(i) *Crowd Discovery Problem*: Discover the set of transient crowds K_t that exist in the communication network $G_t(V,\ E)$ at time t; and

(ii) *Crowd Tracking Problem*: Track the evolution of transient crowds discovered across time periods as they grow, merge, split, and disperse.

2.1 Example

To illustrate the problem of crowd discovery, consider the simple example in Figure 1. At time t=1, users A and B send messages to each other, as do users C and D.[1] The associated communication graph shows an edge between the two pairs, where for simplicity the edge is annotated with the number of messages between the users (2, in both cases). Further, suppose we identify crowds based purely on graph connectivity. So for time t=1, we see there are two crowds discovered {A,B} and {C,D}. For each crowd, we can characterize the semantics of their communication with simple keywords extracted from the content of the tweets: ("oil", "gulf") and ("walcott", "capello"). At time t=2, the communication graph is updated with a new edge (connecting User A and User C), and the existing edges are decayed by one (again, a simplifying assumption for the purposes of this

[1]For simplicity, the example discretizes time so that all messages between users occur in steps. In practice, the proposed algorithm relaxes this assumption and can handle arbitrary message sending times.

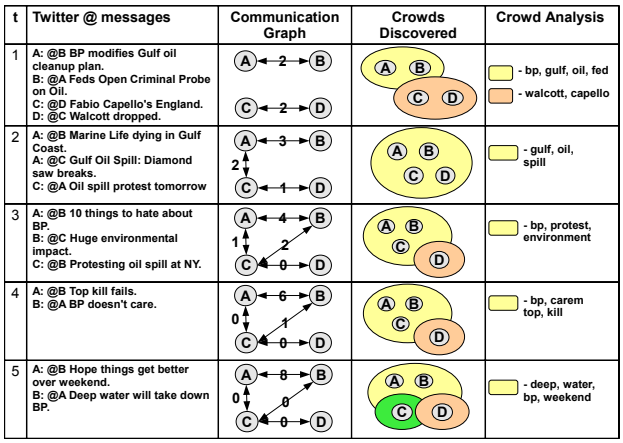


Figure 1: Example of crowd discovery and tracking in Twitter.

example). A single crowd is discovered since all users are connected via edges with non-zero edge weights. At time t=3, User D leaves the main crowd since no messages to or from User D have been observed since time t=1. This process continues until time t=5 when User C also leaves the main crowd due to inactivity. Note that crowds are discovered from communication graph only and not from the content of the messages. As an example of crowd tracking, we can track the evolution of the yellow crowd across time periods, observing the changes it goes through as it grows in size from t=1 to t=2 and then reduces to two users by t=5.

2.2 Challenges

Based on the simple example above, we could imagine directly scaling the basic transient crowd discovery and tracking approach to systems like Facebook and Twitter. For practical crowd discovery and tracking in a large time-evolving communication network, however, we face four key challenges:

- First, systems like Facebook and Twitter are extremely large (on the order of 100s of millions of unique users), placing huge demands on the computational cost of traditional community detection approaches (which can be $O(n^3)$ in the number of users [5]).
- Second, these services support a high-rate of edge addition (new messages) so the discovered crowds may become stale quickly, resulting in the need to re-identify all crowds at regular intervals (again, incurring the high cost of community detection). The bursty nature of user communication demands a crowd discovery approach that can capture these highly-temporal based clusters.
- Third, the strength of association between two users may depend on many factors (e.g., recency of communication), meaning that a crowd discovery approach based on graph clustering should carefully consider edge weights. With no decay at all (meaning that edges are only inserted into the network but never removed), all users will tend towards a single trivial large crowd. Conversely, overly aggressive edge decay may inhibit any crowd formation at all (since edges between users may be removed nearly as soon as they are added).
- Fourth, crowds may evolve at different rates, with some evolving over several minutes, while others taking several days. Since crowds are inherently ad-hoc (without unique community identifiers – e.g., Fans of LA Lakers), the formation, growth and dispersal of crowds must be carefully managed for meaningful crowd analysis.

3. CROWD DISCOVERY AND TRACKING

With these challenges in mind, we propose to discover and track transient crowds through a communication based clustering approach over time-evolving graphs that captures the natural conversational nature of social messaging systems. Two of the salient features of the proposed approach are (i) an efficient locality-based clustering approach for identifying crowds of users in near real-time compared to more heavyweight static clustering algorithms; and (ii) a novel crowd tracking and evolution approach for linking crowds across time periods. In the rest of this section we tackle each of these key areas in turn before evaluating the proposed approach in Section 4 (Experiments).

3.1 Locality in Social Messaging Systems

To support transient crowd discovery in Twitter-like services with 100s of millions of participants, we propose to leverage the inherent locality in social messaging systems. Concretely, we identify two types of locality that are evident in Twitter-like messaging systems: (i) temporal locality and (ii) spatial locality.

Temporal Locality: Transient crowds are intuitively short-lived, since they correspond to actively communicating groups of users. Hence, the composition of a crowd at a point-in-time should be impacted by recent messages as opposed to older messages. As more users interact with the crowd, the crowd should grow reflecting this *temporal locality* and then shrink as users in the crowd become inactive (that is, their last communication with the crowd becomes more distant in time).

Spatial Locality: Intuitively, transient crowds are made up of a very small percentage of users compared to the entire population of the social network. Hence, new messages (corresponding to the addition of edges to the communication network) should have only a local influence on the crowds that exist at any given time. That is, changes in a small region of a graph should not affect the entire graph. In a dataset of 61 million Twitter messages described in Section 4, we have confirmed the existence of this *spatial locality* by finding that only about 1% of users are within two hops, meaning that an edge insertion has only a local effect.

Hence, we can take advantage of both, local changes to the overall communication network (spatial locality) and recent changes to the network (temporal locality), for supporting efficient transient crowd discovery. We next describe how we can use these locality properties in our proposed solution.

3.2 Modeling Temporal Locality

Temporal locality suggests that transient crowds should be composed of users who have communicated with the crowd recently, and that older messages should be treated less significantly. In the motivating example in Figure 1, we implemented temporal locality by reducing the edge weight

by 1 at each time step if no messages are exchanged in a particular time interval, and increasing the edge weight by 1 if messages were exchanged. In the following discussion we explore some more refined approaches for exploiting temporal locality for transient crowd discovery.

Recall that the time-evolving communication network $G_t(V, E)$ has edge weights between vertices u and v at time t represented by $w_t(u,\ v)$. Suppose that the network also stores the latest time any two users communicated $\tau(u,\ v)$ and that we have access to the current time in the system T_{now}.

Fixed window: One approach to model temporal locality is to consider only edges within a fixed time-window β. That is, consider only edges $(u,\ v)$ such that $T_{now} - \tau(u,\ v) < \beta$. In this case, messages sent more than β time units earlier are completely disregarded by the crowd discovery system. A common problem with such a windowing approach is the loss of historical information. In our case this means a possibility that we will miss some significant edges, just because a user didn't communicate in the last β time units. For example, consider 2 users who have constantly exchanged messages over a year, except for the last 1 week. If β is set to 1 week, then the relationship between these 2 users is lost. Hence, using this approach might result in the discovery of imprecise crowds.

Exponential Decay: Alternatively, we propose an edge-weight decay function that gradually fades the impact of older messages relative to newer ones. Concretely, we propose an exponentially decaying impact function based on a decaying coefficient ξ for controlling the rate of decay. The value of ξ determines the type of crowds we identify. Crowds forming slowly can be identified with lower values of ξ while higher value of ξ identifies only crowds forming quickly. Hence, this parameter can be tuned according to the particular application requirements. Since we are interested in transient crowds we will set the values of ξ to relatively higher values.

For edges $(u,\ v) \mid w(u, v) > 0$ we update the new edge weight, conditioned on message exchange, at time t as:

Messages not exchanged:

$$w_t(u,\ v) = w_{t-1}(u,\ v) - \log(T_{now} - \tau(u,\ v)) \ \times \ \xi$$

Messages exchanged:

$$w_t(u,\ v) = w_{t-1}(u,\ v) + 1 - \log(T_{now} - \tau(u,\ v)) \ \times \ \xi$$

To illustrate the impact of this exponential fading, a typical communication graph between two users is shown in Figure 2. The upper plot shows the number of messages exchanged between two users and the bottom plots shows the exponentially decayed edge weights. The middle plot shows the effect of exponential decaying with $\xi = 0.3$ and the bottom plot with $\xi = 1.0$. As expected, we observe that edge weights fall much faster in the bottom plot than in the middle plot.

3.3 Exploiting Spatial Locality

Given the temporal locality-inspired optimization of transient crowd discovery, we now turn to spatial locality. To take advantage of spatial locality, we propose to augment a traditional (expensive) graph clustering algorithm by selectively applying the algorithm to small portions of the entire

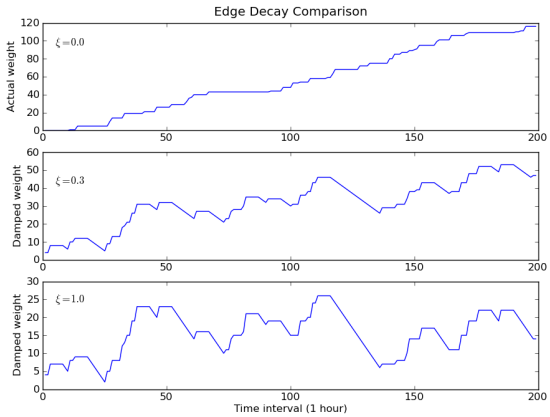


Figure 2: Changes to edge weights with *Exponential Decay.*

communication network, thereby saving the computational cost of running the algorithm over the entire large network.

Let C_{ti} represents the i^{th} crowd in K_t. Users are assigned to one and only one crowd, i.e., $C_{ti} \cap C_{tj} = \phi,\ \forall\ C_{ti}, C_{tj} \in K_t$. To discover K_t, we could apply one of a number of graph clustering algorithms, including MCL [12], multilevel graph clustering [4], etc. For concreteness in this paper, we consider min-cut clustering [5, 7], a popular graph clustering algorithm that has shown good success across real-world datasets like web pages, citation networks, etc. While the following discussion focuses on min-cut clustering (in the interest of providing a baseline for experimental comparison of transient crowd discovery), the general locality principles discussed in this section could be applied to other clustering algorithms.

3.3.1 Preliminaries

To begin our development of locality-based clustering, we first present some preliminaries to describe min-cut clustering.

Minimum cut: The minimum cut of a graph G with respect to vertices s and t, where $s \in S, t \in T$, is defined as partition of V into S and T such that, the total weight of edges connecting the partitions is minimum. This is represented as $c(S, T)$. For an undirected graph G we can define a weighted tree T_G called the minimum-cut tree [7]. We can determine $c(S, T)$ by analyzing the path from s to t in T_G, where the value of $c(S, T)$ is equal to the smallest edge on this path.

Min-cut clustering: The min-cut clustering algorithm [5] clusters a graph G first by adding an artificial sink t. All of the vertices of G are connected to the artificial sink with an edge capacity of α, to form a modified graph G', where α is a parameter guiding the quality guarantees of the resulting clusters. The minimum-cut tree T' for G' is then calculated. The connected components of T' obtained after removing the artificial sink t are clusters in G. Min-cut clustering relies on the special parameter α to ensure the quality of the clusters generated, where:

$$\frac{c(S, V - S)}{|V - S|} \leq \alpha \leq \frac{c(P, Q)}{min(|P|, |Q|)} \qquad (1)$$

with, $P \cap Q = \phi$ and $P \cup Q = S$. By tuning this α parameter, the number and size of the resulting clusters can be varied (from one large cluster with all nodes to a trivial clustering consisting of n singleton nodes).

3.3.2 Locality-Based Clustering

Of course, we could directly apply the min-cut clustering algorithm to the large time-evolving communication network G_t directly. The output would be a set of clusters K_t which we could take to be transient crowds, however, at a considerable expense. Coupled with the need to re-compute clusters as the network evolves, straightforward application of a traditional graph clustering approach is infeasible for efficient transient crowd discovery.

Towards exploiting spatial locality for efficient crowd discovery, we must address two issues: (i) The application of min-cut clustering to a particular subgraph of the entire communication network; and (ii) The determination of which subgraphs of the communication network to select for clustering.

Subgraph clustering: The first challenge is to perform local clustering, given an identified region of the communication network (corresponding to some locally impacted portion of the network). By clustering a local region of the communication network we can begin to reduce the expense of clustering the entire network.

Given a subgraph S (the part of the communication network impacted by edge addition) to cluster, the algorithm first contracts G_t to G'_t. As shown Algorithm 1, this approach then creates a new graph G''_t by adding an artificial sink w_s to G'_t and connecting all the vertices of S to t with edges of capacity α and all the vertices in $(V' - S)$ with edges of capacity of $\alpha|V - S|$ as in [10]. It then determines the minimum-cut tree T''_t for G''_t. The connected components obtained after removing w_s from T''_t are the new clusters (which correspond to transient crowds). In this way, only a small portion of the communication network is impacted, leading to more efficient clustering that clustering the entire network.

Algorithm 1 ClusterSubGraph(S)

1. Contract G_t: Reduce G_t to G'_t by replacing vertices $V - S$ with a new vertex x. All the resulting loops are deleted and parallel edges are replaced with a single edge with weight equal to the sum of the edges.
2. Expand G'_t: Construct a new graph G''_t by adding vertex w_s to $G'_t(V', E')$. Connect w_s to $v, \forall v \in S$ with edge capacity of α and w_s to $v', \forall v' \in (V' - S)$ with edge capacity of $\alpha|V - S|$.
3. Minimum-cut tree: Determine minimum-cut tree T''_t for G''_t. The connected components obtained in T''_t after removing vertices w_s and x from it are the clusters in S.

Subgraph selection: The second challenge is to determine which subgraphs are to be selected for clustering, i.e. how do we select S in Algorithm 1? Selecting too many subgraphs for re-clustering may result in expensive computation, whereas selecting too few may result in poor crowd quality. Following [10], we adopt an approach triggered on *each edge insertion* to identify subgraphs that need to be clustered.

Depending on the position where an edge is inserted and the effect of edge addition on the quality of clustering there are four ways to select clusters for local clustering. The first case is when an edge is added within an existing cluster C_u. In this case there is a probability that this addition might have resulted in subclusters within C_u, that improve clustering quality. Hence, only C_u is selected for clustering (*Case i*). An edge can also be added between 2 clusters. In this case, if the quality of clustering is maintained in spite of this edge addition, then re-clustering is not required (*Case ii*). Otherwise, if the quality of clustering is reduced, then we select both clusters for re-clustering (*Case iv*). If the addition of an edge between 2 clusters results in satisfying the condition for cluster merging, then the 2 clusters are merged (*Case iii*). The pseudocode for subgraph selection is given in *Step 2* of Algorithm 2.

The proof of correctness of these cluster selection methods is given in [10]. We empirically validate the clustering quality in Section 4.

Algorithm 2 Locality Clustering Algorithm

For every new edge (u, v) added to the graph, perform the following 3 steps.
1. Initialization: If the added edge has vertices that have not been observed before add them to vertex set V. Create singleton clusters for the new vertices and add them to cluster set K.
2. Clustering: Let u, v belong to clusters C_u and C_v respectively. For every edge (internal and boundary) in C_u and C_v decay the edge weights as mentioned in Section 3.2. Now depending on the conditions that match perform the corresponding clustering operations:

Case i. If the vertices belong to same cluster then updating the edge weights might have resulted in formation of clusters within this cluster. Check for new clusters using $ClusterSubGraph(C_u)$.

Case ii. If the vertices belong to different clusters and the addition of the edge does not reduce the quality of clustering, then perform no action. The quality of the clustering is maintained if the following inequalities (Equation 1) are satisfied.

$$\frac{c(C_u, V - C_u)}{|V - C_u|} \leq \alpha \text{ and } \frac{c(C_v, V - C_v)}{|V - C_v|} \leq \alpha$$

Case iii. If the vertices belong to different clusters and the addition of the edge satisfies the merging condition $\frac{2c(C_u, C_v)}{|V|} \geq \alpha$, merge the 2 clusters.

Case iv. If the vertices belong to different clusters and the previous 2 conditions are not met then the quality of clustering has reduced. Hence, perform $ClusterSubGraph(C_u \cup C_v)$ to generate clusters that maintain clustering quality.

Time Complexity Analysis: The algorithm to cluster subgraphs uses the relabel-to-front approach of the push-relabel algorithm [6] to calculate the minimum-cut tree. It has a time complexity of $O(l^3)$, where l is the number of vertices in the minimum-cut tree. Let $k = \max_{i=1}^{|K_t|}(|C_{ti}|)$, the size of the largest crowd in K_t. In edge addition algorithm when both the vertices of the edge belong to the same crowd we decay $O(k^2)$ edges and re-cluster $O(k)$ vertices. In this case the time complexity is $O(k^3)$. In case where the quality of crowds is maintained on addition of the edge, the time

complexity is $O(2k^2)$ for damping the edges. During the merge operation, we dampen $O(2k^2)$ edges and re-cluster $O(2k+1)$ vertices, which results in a time complexity of $O(k^3)$. Hence, the time complexity of the algorithm on an edge addition is $O(k^3)$, as compared to the time complexity $O(n^3)$ for the original min-cut clustering algorithm in [5].

To summarize, in this section we have described how we can use the spatial and temporal locality observed in social messaging systems to design an efficient clustering algorithm.

3.4 Crowd Tracking

Finally, we turn to the second of two key challenges for transient crowd discovery and tracking – how to track crowds over time as users join, crowds merge, and crowds disperse. For example, when we discover a new crowd that is discussing an upcoming event (say the World Cup), we need a method to track the users participating in this crowd in consequent intervals. This would give us an ability to analyze crowd dynamics leading up to and after the event.

Recently, there has been some work analyzing communities across times. In [3], the authors look at communities on large social networks like LiveJournal and MySpace. Since the communities are explicitly defined in these networks, the task of determining evolution of graph is trivial. In [2] the authors observe changes clusters undergo between time intervals and consider the changes to be events.

Crowd tracking would be straightforward if each crowd were associated with a unique community identifier (e.g., Fans of LA Lakers). Facebook and Twitter have adopted methods for group affiliation like fanclubs and lists, but these longer-lived affiliations are not available nor appropriate for short-lived transient crowds. Since crowds are inherently ad-hoc we define in this section the problem of crowd tracking and present a graph-based approach to solve it.

Crowd Tracking Graphs: A crowd tracking graph G_c is constructed using the crowds obtained at different time intervals. This graph helps us understand the changes that take place in these crowds between time intervals. It is a directed graph with crowds as vertices and the direction of the edge denoting the parent-child relationship. Node colors are used to indicate the state of crowd evolution. A green node indicates that the crowd has been discovered for the first time and a red node indicates dispersal of the crowd. Intermittent crowds are shown in blue color. To track the evolution of a crowd we start at the green node and follow the edges until we reach the red node. An example of a crowd tracking graph is shown in Figure 3(a). The graph also shows examples of merging and splitting of crowds.

Transient Crowd Tracking Problem: Given a time-evolving communication network $G_t(V,\ E)$ and a set of transient crowds K_t identified at every time interval t, construct a Crowd Tracking Graph G_c.

We propose an algorithm to construct a crowd tracking graph. The algorithm takes the crowd set for the s^{th} interval K_s and the crowd set for previous interval K_{s-1} as input. For every crowd $C_{si} \in K_s$, it determines the parent crowd in K_{s-1}. It then adds a directed edge from the parent to the child crowd. The pseudocode for this algorithm is given in Algorithm 3.

Examples of how parent crowds are selected is shown in Figure 3(b). (I) shows two crowds merging. Here, the parent of the crowd in t_{s+1} is the crowd in t_s that contributed the maximum nodes to the crowd in t_{s+1}. (II) shows a single crowd in t_s being split into two crowds in t_{s+1}. The parent of the two crowds in t_{s+1} is obtained directly. (III) shows a case where three crowds in t_s contribute to three crowds in t_{s+1}. Though crowd A and B contribute two nodes each to D, B is designated as the parent of D since B is older than A.

Algorithm 3 Crowd Tracking Algorithm

```
Let K_s be the crowd set for the current interval and K_{s-1} the previous.
for Every crowd C_si ∈ K_s do
    if C_si is a newly discovered crowd then
        Create a green node in G_c.
    else
        Get the parent crowd C_{s-1j} ∈ K_{s-1} with maximum common users with C_si. If there is more than one crowd with same number of common users, select the older crowd.
        Create a new blue node and add a directed edge from the parent crowd in K_{s-1} to C_si.
    end if
end for
for Every crowd C_{s-1j} ∈ K_{s-1} that does not have a child node do
    Change the color of C_{s-1j} to red from blue.
end for
```

4. EXPERIMENTS AND RESULTS

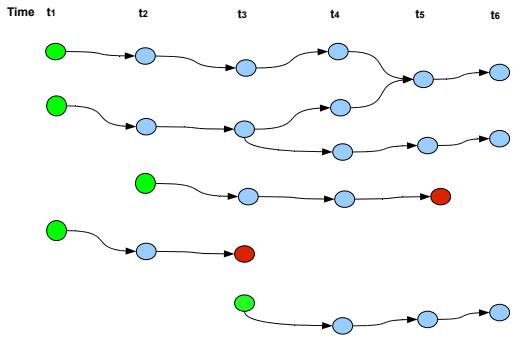
In this section we present the results of four sets of experiments: (i) we first explore the impact of locality-based crowd discovery compared to the static graph clustering approach without the locality optimizations; (ii) then we investigate the impact of the tunable edge decay parameter; (iii) we then examine the features of the discovered crowds, including size and lifespan; and (iv) we illustrate some crowd-based trends that differ from trends aggregated from individual users.

4.1 Twitter Dataset

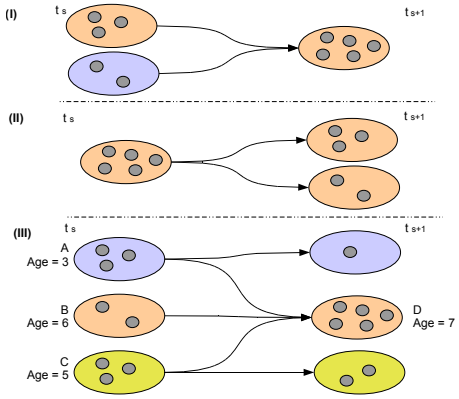
To study crowd detection in a real-world setting, we focus on the Twitter micro-blogging service. Through a mix of crawling and API calls to the Twitter service, we collected a sample of tweets from October 1^{st} to December 31^{st}, 2008, accounting for 2208 hours (see Table 1 for details). The dataset includes over 710,000 users and over 61.3 million status updates ("tweets") of 140 characters or less. Users can annotate their tweets via the inclusion of hashtags (e.g., "#redsox") to indicate a particular topic. Similarly, users can include *@mentions* of the form @$\langle$*username*$\rangle$ within a tweet to reference another user. While these *@mentions* can serve many purposes, the most popular use is as a simple messaging framework, so that a message posted by user u_1 including @$\langle u_2 \rangle$ is considered a message from u_1 to u_2.

Property	Total	Per hour avg.
Users	711,612	18,713
Total tweets	61,314,203	27,769
Messages (@$< u >$)	20,394,030	9,236
User pairs	3,756,619	9,310

Table 1: Twitter dataset properties.



(a) Crowd tracking graph. The green nodes represent start of a crowd and the red node shows the dispersing of the crowd.



(b) Examples of crowd modification events. Arrows indicate crowds in t_s that contribute to crowds in t_{s+1}. Color of the crowd in t_{s+1} indicates its parent in t_s.

Figure 3: Crowd tracking.

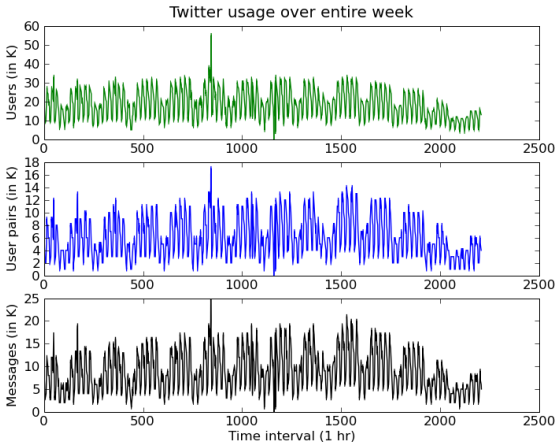


Figure 4: Users, user pairs and messages in Oct-Dec, 2008.

Of the, 61.3 million tweets in the dataset, 20.4 million contain the @⟨*username*⟩ syntax and are considered messages from one user to another. 3.7 million pairs of users are connected by these messages. The hourly distribution of tweeting users, user pairs, and messages sent is shown in Figure 4. All are strongly correlated, following a clear daily and weekly patterns.

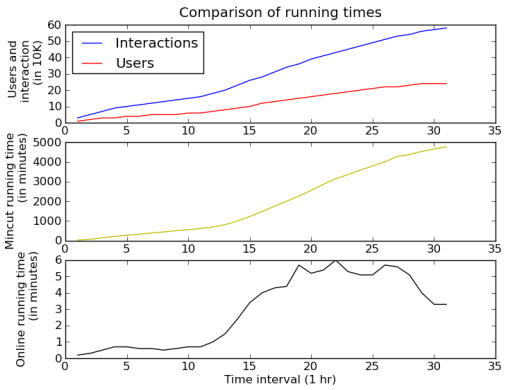
4.2 Performance of Crowd Discovery Algorithm

In the first set of experiments, we investigate the efficiency and quality of the proposed locality-based clustering approach for crowd discovery. Since social messaging systems are large with a high rate of new messages, it is important for crowd discovery to be efficient; but efficiency must be balanced with the quality of the discovered crowds. As a baseline for comparison, we considered the min-cut clustering algorithm [5] without the locality-based optimizations. Since min-cut clustering is designed for static graphs, we took snapshots of the time-evolving communication network every hour and then ran min-cut clustering over each of these hourly snapshots, resulting in 2208 total crowd sets.

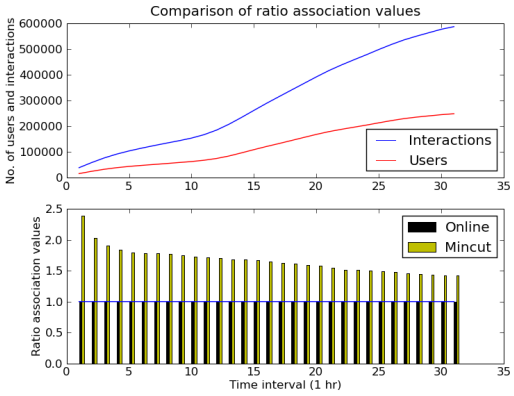
Running time: In Figure 5(a), we show the running time comparisons between min-cut clustering and the locality-based crowd discovery approach (note that we focus on the first 30 hours for presentational detail; the general trends hold across the duration). The top plot in Figure 5(a) shows the growth in users and messages; the middle plot shows the running time of min-cut clustering; the bottom plot shows the running time of online clustering algorithm. The first observation is that the proposed approach is at least 100 times faster than non-locality optimized approach in all cases, and upwards of 1,000 times faster in some cases. Next, we observe the impact the growing number of users and interactions has on the running time of these algorithms. We see that the running time of the min-cut algorithm is proportional to the increase in users and interactions, while our algorithm, because of its locality optimizations, has almost a constant running time. Spatial locality allows our algorithm to cluster a relatively small part of the graph and temporal locality reduces the number of edges by removing old edges.

Crowd quality: Although the proposed locality-based approach results in a much faster crowd discovery, there may be a cost in terms of crowd quality. To gauge this cost, we measure the quality of the discovered crowds using the ratio-association value [4], which seeks to maximize the weight of edges within a cluster: maximize $\sum_{i=1}^{k} \frac{c(C_i,C_i)}{|C_i|}$. Using this objective, we measure the ratio-association values for both min-cut clustering and the proposed approach. In Figure 5(b), we show the *ratio* of ratio-association values for both algorithms versus the proposed approach; the ratio-association value for local-clustering (online) is indicated using black bars of height 1. We see that during the initial intervals, the ratio-association of the min-cut algorithm is more than that for the locality-based approach, but the ratio continues to decrease with time. We see significant improvements by the time we reach the 30^{th} interval. This shows that as the size of the graph grows the quality of clusters generated by the locality-based approach increases.

Empirically, we find that the locality-based approach supports efficient crowd discovery while maintaining crowds of



(a) Running time comparison.



(b) Quality comparison using ratio association.

Figure 5: Performance of crowd discovery algorithm.

relatively high quality (within 50% of the ideal case using static graph clustering).

4.3 Varying the Edge-weight Decay Coefficient

In the second set of experiments, we analyze the performance of the algorithm as the decay coefficient is modified, from 0.5 to 1.0 to 1.5. The decay coefficient is an important tunable parameter that determines the rate at which crowds disperse. We first show the impact varying this parameter has on the number of crowds discovered and the size of these crowds. We then investigate the impact of this parameter on the speed of crowd discovery and the quality of the crowds discovered.

Impact on number of crowds discovered and crowd sizes: The effect of varying decay co-efficient on crowd size and count is shown in Figure 6(a). We find that the number of crowds discovered for coefficient of 0.5 is more than the ones discovered for 1.0 and 1.5. In the case of larger coefficient values the crowds disperse quickly and hence we find fewer crowds. Coefficients 1.0 and 1.5 discover almost the same number of crowds. This might be because the crowds that are discovered at 1.0 stay together even at 1.5. It is possible that they disperse at higher co-efficients. We also observe larger crowd sizes at lower coefficient values as the crowds disperse slowly.

Impact on ratio association values: The effect on quality of crowds discovered at different decay coefficient values is shown in Figure 6(b). To observe the quality of crowds discovered we use ratio association, as defined before. We observe that the best crowds are obtained when the decaying coefficient is 1.0. Hence, for the rest of the experiments we set the coefficient to 1.0.

Impact on crowd discovery time: We observe that the running time of the algorithm is not dependent on the coefficient (see Figure 6(c)). This is an important result because we can now use our algorithm to observe crowds at degrees of granularity by changing the coefficient without affecting the running time performance of the algorithm.

4.4 Transient Crowd Analysis

In the third set of experiments, we explore the characteristics of the discovered crowds using the proposed crowd discovery and tracking approach. We identify topics for a particular crowd (akin to the "Crowd Analysis" column in the example in Figure 1) using a simple approach in which we characterize the topic of a crowd by extracting the nouns from the messages (tweets) exchanged by a crowd.

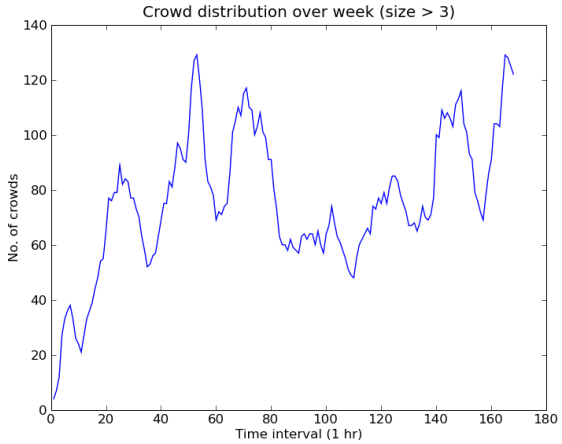
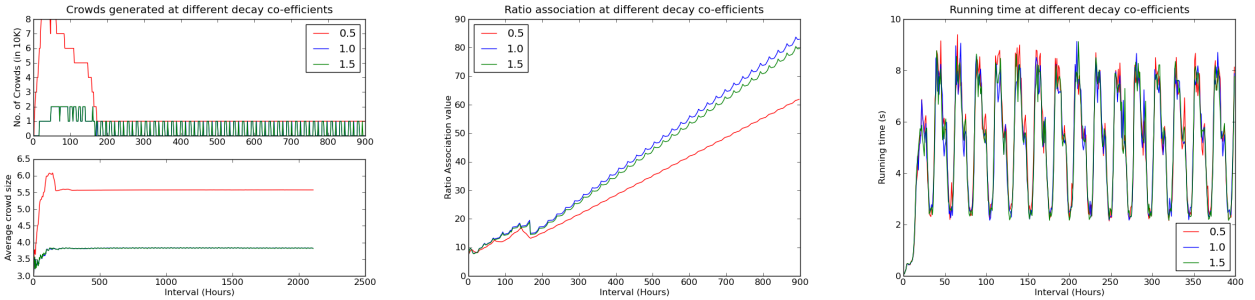


Figure 7: Crowds at each time interval.

Time-dependent crowding patterns: We first consider the number of crowds discovered in each time interval. This knowledge can yield insights into crowding patterns in social networks. Figure 7 shows the distribution of crowds during a particular week. Like the user and message frequency in Figure 4, we observe crowds following a daily pattern. But unlike the previous case, where we saw high and uniform usage throughout afternoon and evening, we observe the largest number of crowds forming in the evening. We are interested to explore this tension between crowding behavior and overall Twitter usage in our continuing work.

Crowd lifespan: Next, we consider the lifespan of crowds. The lifespan for a crowd can be obtained from the crowd tracking graph discussed in Section 3.4. The length for which a crowd lasts is an indicator of its activeness. For example, a crowd that is constantly communicating lives for a longer time than an inactive crowd which disperses. We il-



(a) Crowd count and size.

(b) Quality of clustering.

(c) Running time of the clustering algorithm.

Figure 6: Effect of varying edge-weight decay co-efficient

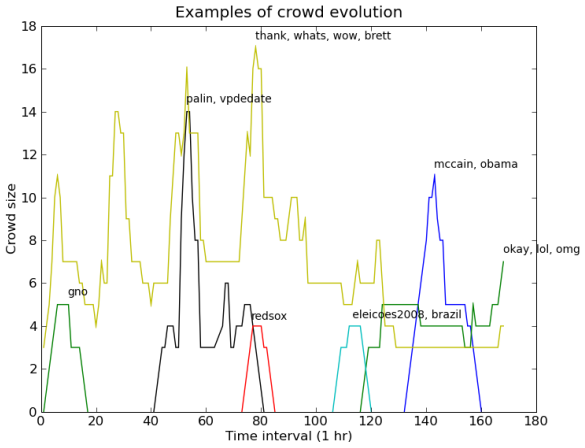


Figure 8: Examples of the crowds discovered in the dataset.

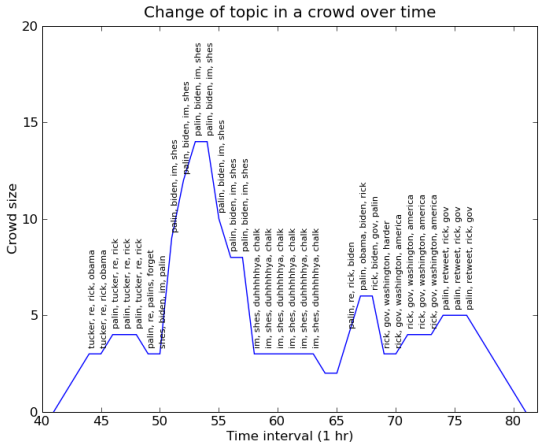


Figure 9: Topic evolution in a crowd over time.

lustrate some of the discovered crowds and their lifespans in Figure 8, with an annotation next to the crowd peak showing the topic of discussion. We see a crowd (shown in black) discussing Sarah Palin and the Vice-Presidential debate from the 40^{th} hour to 80^{th} hour that peaks around the time of the actual debate. We observe that crowds that talk about general everyday things have a greater lifespan than crowds discussing specific events. For example in Figure 8, a crowd (annotated with *thank, whats, wow*) discussing everyday things lives through the entire week, while, during the same period we observe several event-specific crowds, like crowds discussing the Red Sox, Sarah Palin, and Girl's Night Out (gno) forming and dispersing. These event-specific crowds start forming just before the event and die a few intervals after the completion of that event. This distinction between the crowds discovered clearly indicates two types of Twitter usage: first, it is used as a platform to discuss and debate specific events, and second, it as also used a means of everyday communication.

4.5 Crowd-based vs. User-based Topics

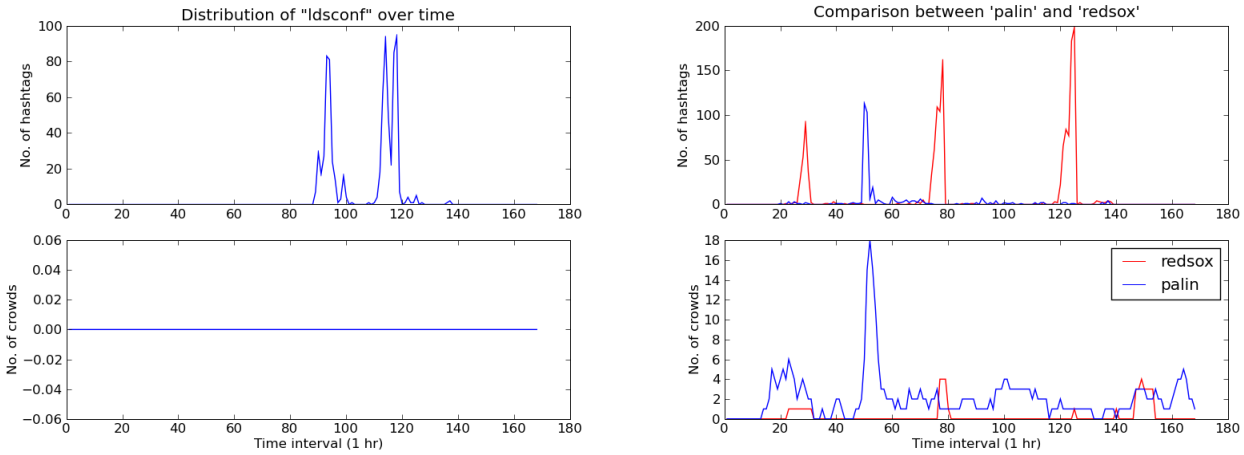
In the final set of experiments, we compare the topics that interest crowds versus topics that are discovered through the (non-crowd) aggregation of tweets from individual users.

Hashtags vs. Crowd topics: Twitter supports the inclusion of meta-data in tweets through the use of hashtags (e.g., "#redsox"). We first aggregated all of the hashtags in our dataset to see what topics were of most interest. These top hashtags are shown in Table 2. Most of the topics determined using hashtags are related to specific events, like debate-related hashtags, conference-related hashtags (*wct08, ldsconf, wjs08*) etc. This individual-based aggregation is similar to how Twitter's trending topics works (see http://search.twitter.com/).

In Table 2, we also show the topics discovered from our simple noun-based crowd analysis. We see that the crowd-

Rank	**Hashtags**	**Crowd topics**
1	vpdebate	twitter
2	current	debate
3	redsox	palin
4	vmb	money
5	ldsconf	video
6	debate08	kids
7	palin	obama
8	wcto08	school
9	wjs08	mccain
10	eleicos	office

Table 2: Top hashtags and topics observed for the week.



(a) Comparison for *ldsconf*

(b) Comparison between *redsox* and *palin*

Figure 10: Comparison between hashtags and crowd topics.

based topics are more varied and less event-specific, like *money*, *kids*, and *school*. Some topics like *ldsconf* (corresponding to the LDS Semi-annual General Conference) are hashtagged often but are part of no crowds (See Figure 10(a)). Similar results hold for the conference tags *wcto08, wjs08*, indicating lots of individual activity via tweeting about the conference, but little cohesive communication among members of a community. Another example of the difference between hashtags and topics discussed is shown in Figure 10(b). We see the distribution of the topics *palin* and *redsox*, where the number of hashtags for *redsox* is significantly more than the hashtags for *palin*, but we see that more crowds discuss *palin* than *redsox*.

Topic evolution: Finally, we track the evolution of topics within a crowd as users join and leave over time. Observing the changing topics in a crowd can give us a better understanding about the interests of a crowd and hence help us model the crowd better. An example of such a topic evolution, in a crowd discussing vice-presidential debate, is shown in Figure 9. The crowd at the beginning discusses something generic and then starts discussing the Vice-Presidential debate as it occurs (intervals 50-54). The crowd has maximum users during the actual debate and begins to lose users on completion of the debate. As we move away from the debate we see the crowd discussing other topics before dispersing.

5. CONCLUSION

In this paper, we have studied the problem of automatically discovering and tracking transient crowds in highly-dynamic social messaging systems like Twitter. We presented a locality-based clustering algorithm for a time-evolving communication network that uses two characteristics of transient crowds – temporal and spatial locality – to support efficient crowd detection. We showed how crowds at different granularity can be discovered by changing edge decay coefficient. We then analyzed these crowds to discover crowd-based topics of discussion, which are different from those identified using hashtags. Finally, with an example we showed how we can track topic evolution in a crowd. As part of future work, we plan to investigate hybrid graph clustering approaches that build on the communication-based approach presented here – e.g., by considering content-based and geographic-based similarity across users.

6. ACKNOWLEDGMENTS

This work was supported in part by a DARPA Young Faculty Award and by a Google Research Award. Any opinions, findings and conclusions or recommendations expressed in this material are the author(s) and do not necessarily reflect those of the sponsors.

7. REFERENCES

[1] A website that maintains statistical information about tweets. *http://popacular.com/gigatweet/*.

[2] S. Asur, S. Parthasarathy, and D. Ucar. An event-based framework for characterizing the evolutionary behavior of interaction graphs. In *KDD '07*, pages 913–921, New York, NY, USA, 2007. ACM.

[3] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan. Group formation in large social networks: membership, growth, and evolution. In *KDD '06*, pages 44–54, New York, NY, USA, 2006. ACM.

[4] I. Dhillon, Y. Guan, and B. Kulis. A fast kernel-based multilevel algorithm for graph clustering. In *KDD '05*, pages 629–634, New York, NY, USA, 2005. ACM.

[5] G. W. Flake, R. E. Tarjan, and K. Tsioutsiouliklis. Graph clustering and minimum cut trees. *Internet Mathematics*, 1(4):385–408, 2004.

[6] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum flow problem. In *STOC '86*, pages 136–146, New York, NY, USA, 1986. ACM.

[7] R. E. Gomory and T. C. Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961.

[8] K. Y. Kamath and J. Caverlee. Identifying hotspots on the real-time web. In *CIKM '10*, New York, NY, USA, 2010. ACM.

[9] M. E. J. Newman. Fast algorithm for detecting community structure in networks, September 2003.

[10] B. Saha and P. Mitra. Dynamic algorithm for graph clustering using minimum cut tree. In *ICDMW '06*, pages 667–671, Washington, DC, USA, 2006. IEEE Computer Society.

[11] J. Sun, C. Faloutsos, S. Papadimitriou, and P. S. Yu. Graphscope: parameter-free mining of large time-evolving graphs. In *13th ACM SIGKDD' 07*, pages 687–696, New York, NY, USA, 2007. ACM.

[12] S. Van Dongen. Graph clustering via a discrete uncoupling process. *SIAM J. Matrix Anal. Appl.*, 30(1):121–141, 2008.