Identifying Hotspots on the Real-Time 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

We study the problem of automatically identifying "hotspots" on the real-time web. Concretely, we propose to identify highly-dynamic ad-hoc collections of users – what we refer to as crowds – in massive social messaging systems like Twitter and Facebook. The proposed approach relies on a message-based communication clustering approach over time-evolving graphs that captures the natural conversational nature of social messaging systems. One of the salient features of the proposed approach is an efficient locality-based clustering approach for identifying crowds of users in near real-time compared to more heavyweight static clustering algorithms. 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 efficiently and effectively identify Twitter-based crowds relative to static graph clustering techniques at a fraction of the computational cost.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Clustering; H.3.4 [**Systems and Software**]: Current awareness systems

General Terms

Algorithms, Performance, Experimentation

Keywords

Real-time, Social, Graph, Clustering

1. INTRODUCTION

This paper focuses on identifying emerging "hotspots" on the real-time 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. 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, content personalization, social information discovery, 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*.

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?

2. CROWD IDENTIFICATION

We propose to model crowd formation through a message-based communication clustering approach over time-evolving graphs. The efficient locality-based clustering approach for identifying crowds of users is developed on the notion that changes in a small region of a graph should not affect the entire graph (reflecting locality in the messaging system).

2.1 Preliminaries

Historically, direct communication between people has been mostly unobservable or unavailable for large-scale web min-

CIKM'10, October 26–30, 2010, Toronto, Ontario, Canada.

ing. 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]. Based on these observable communication patterns, we study how to efficiently discover 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. For concreteness, 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.

Our goal then is to discover the set of transient crowds K_t that exist in the communication network $G_t(V,\ E)$ at time t. For practical crowd discovery in a large time-evolving communication network, however, we face two 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 [3]). 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.

2.2 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. 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 3, 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.

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 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 [6], multilevel graph clustering [2], etc. For concreteness in this paper, we consider min-cut clustering [3, 4], a popular graph clustering algorithm that has shown good success across real-world datasets like web pages, citation networks, etc. 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 [4]. 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 [3] 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:

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

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).

2.3 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 [5]. 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.

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. 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.

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 [5], 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.

3. EXPERIMENTS AND RESULTS

In this section we explore the impact of locality-based crowd discovery compared to the static graph clustering approach without the locality optimizations.

3.1 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.

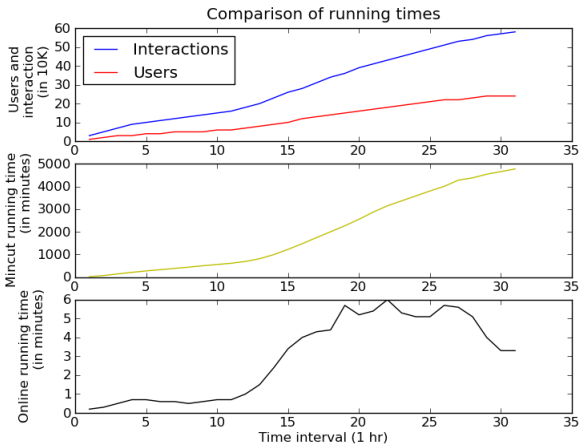
Property	Total	Per hour avg.
Users	711,612	18,713
Total tweets	61,314,203	27,769
Messages (@$< u >$)	20,394,030	9,236
User pairs	3,756,619	9,310

Table 1: Twitter dataset properties.

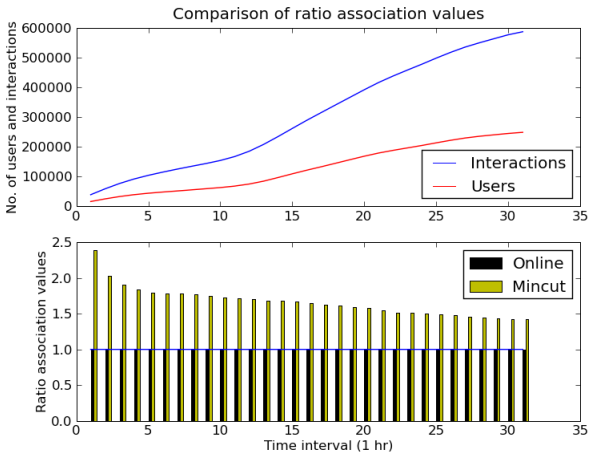
3.2 Performance of Crowd Discovery Algorithm

Of the, 61.3 million tweets in the dataset, 20.4 million contain the @$\langle username \rangle$ syntax and are considered messages from one user to another. 3.7 million pairs of users are connected by these messages.

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



(a) Running time comparison.



(b) Quality comparison using ratio association.

Figure 1: Performance of crowd discovery algorithm.

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 [3] 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 1(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 1(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.

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 [2], 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 1(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.

4. CONCLUSION

Toward the goal of detecting hotspots on the real-time web, we have studied the problem of automatically discovering transient crowds in highly-dynamic social messaging systems like Twitter. We presented a locality-based clustering algorithm for a time-evolving communication network that relies on the inherent spatial locality of transient crowds to support efficient crowd detection. As part of future work, we plan to investigate hybrid approaches that build on the communication-based approach presented here – e.g., by considering content-based and geographic-based similarity across users to serve as the basis of crowd formation.

5. REFERENCES

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

[2] 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.

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

[4] 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.

[5] 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.

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