Summarization via Pattern Utility and Ranking: A Novel Framework for Social Media Data Analytics

Xintian Yang
Google Inc.
xyang@google.com

Yiye Ruan, Srinivasan Parthasarathy
Dept. of Computer Science and Engineering
The Ohio State University
{ruan,srini}@cse.ohio-state.edu

Amol Ghoting
IBM T. J. Watson
Research Center
aghoting@us.ibm.com

Abstract

The firehose of data generated by users on social networking and microblogging sites such as Facebook and Twitter is enormous. The data can be classified into two categories: the textual content written by the users and the topological structure of the connections among users. Real-time analytics on such data is challenging with most current efforts largely focusing on the efficient querying and retrieval of data produced recently. In this article, we present a dynamic pattern driven approach to summarize social network content and topology. The resulting family of algorithms relies on the common principles of summarization via pattern utilities and ranking (SPUR). SPUR and its dynamic variant (D-SPUR) relies on an in-memory summary while retaining sufficient information to facilitate a range of user-specific and topic-specific temporal analytics. We then follow up by describing variants that take the implicit graph of connections into account to realize the Graph-based SPUR variant (G-SPUR). Finally we describe scalable algorithms for implementing these ideas on a commercial GPU-based systems. We examine the effectiveness of the summarization approaches along the axes of storage cost, query accuracy, and efficiency using real data from Twitter.

1 Introduction

Social networking and microblogging sites are ubiquitous nowadays, and an increasing number of organizations and agencies are turning to extract and analyze useful nuggets of information from such services to aid in various functions. However, a fundamental challenge for effectively analyzing social network data is its sheer scale. Twitter, for instance, has over 200 million users and several hundred million tweets per day. Supporting interactive querying and analytics requires novel approaches for summarizing and storing such data.

The data generated by social networking services can be classified into two categories: the textual content written by the users (e.g. tweets in Twitter) and the link structure of user connections (e.g. follower – followee relationship in Twitter). The textual content carries the information that people want to share with their friends. It is large-scale and streaming in nature. The link structure captures how the textual content will spread through the social network of users. While user connections are relatively stable compared with the high speed content stream, it is also large-scale because of the enormous size of user base.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

We recently proposed the SPUR (Summarization via Pattern Utility and Ranking) family of algorithms to build a queryable summary of social network content stream [10]. Here we briefly overview some of the main features of SPUR and its variant Dynamic-SPUR (D-SPUR) and show how the summarization created by these algorithms can fit in a limited memory budget, and can help answer complex queries. In this work we describe extensions to SPUR wherein effective compression and efficient querying on the link structure of social network can also be supported called Graph-based SPUR (or G-SPUR). Furthermore, with the advent of general-purpose computing using Graphical Processing Units (GPUs) , we discuss strategies for leveraging such technology in the context of the SPUR family of algorithms.

We begin by briefly describing SPUR and D-SPUR as preliminaries in Section 2. Then we present G-SPUR (including GPU speed-up) in Section 3. In Section 4, we discuss how D-SPUR is adapted to work with G-SPUR for supporting content and time aware network queries Two particular query tasks, PageRank and clustering, will be described. Finally, we present experiment results in Section 5 and conclude.

2 SPUR: Summarization via Pattern Utility and Ranking

In our previous work [10], we described a method of summarizing the user generated content from a social network. The network content is in the form of a high speed message stream fluxing into the data processing system. We propose a novel stream processing framework to summarize the input stream with efficient, incremental summary construction and budgeted memory footprint. Given the input message stream with proper word stemming and stop-word removal performed, we divide it into approximately equal-sized batches, e.g. one hour per batch (the first arrow in Figure 1).

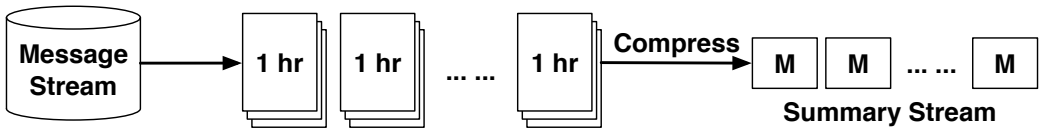
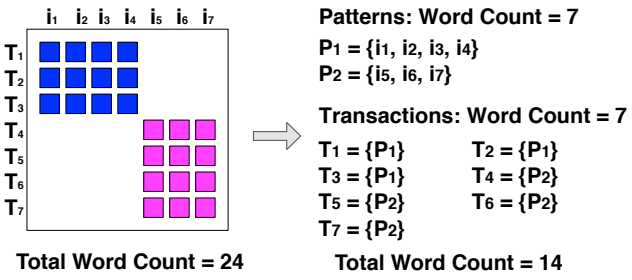


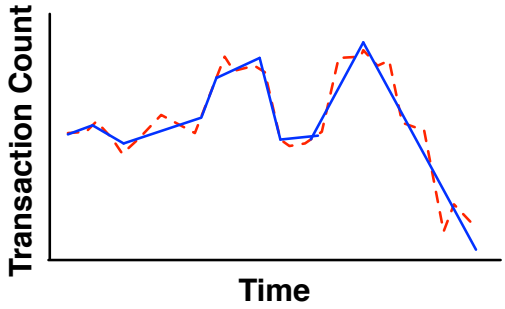
Figure 1: Division and compression of message stream

To compress each batch of messages into a summary object which can fit in a constant memory budget M (the second arrow in Figure 1), we replace individual words with frequently used phrases that can cover the original content of incoming messages (Figure 2(a)). Our approach represents each message as a transaction of words and a batch as a set of transactions. Therefore the challenge of finding frequent phrases becomes a frequent itemset mining problem. The utility of each pattern is represented by the reduction of storage cost if it would have been used. We also take into account the impact of using one pattern on other patterns, and perform dynamic ranking adjustment to reflect such changes. Patterns are selected in a greedy fashion, based on their utility values, until the size after compression satisfies the memory budget.

To guarantee the summary size grows logarithmically with time instead of linearly, we enhance and modify the pyramidal time window suggested by Aggarwal *et al.* [1] for clustering in data streams. The key operations of the system, D(ynamic)-SPUR, are merging two time windows, and managing time information. When merging time windows, patterns from both time windows are ranked together by their utility values, and ones with lower values are dropped until the merged summary can fit in the memory budget. The purpose of maintaining time information for the transactions in a summary is to be able to effectively answer a query about an arbitrary time interval. To this end, we store distinct transactions in the summary and associate a count with each transaction to indicate how many times a transaction appeared in a batch. When merging two time windows, if two transactions contain the same set of patterns, they must be from two different batches, because within a batch, we only keep distinct transactions. Instead of summing the count of these two transactions, we could concatenate their counts in time order and form a time series with two points. As D-SPUR combines more summaries, we concatenate more points to each transaction. A time series that spans batches (i.e. the red dashed line in Figure 2(b)) is therefore formed for each transaction, enabling



(a) A batch of messages compressed to a summary



(b) An example of the time series of a transaction

Figure 2: Illustration of message compression and time information management

reconstruction of the exact count in any time interval.

3 Compression of Network Topology with G-SPUR

While SPUR and D-SPUR algorithms are designed to compress social content streams, we are also interested in compressing the link structure of social networks themselves. We assume that the network topology is relative stable, and can therefore take snapshots of it at different times. The complete topological information of a social network snapshot can be modeled as a directed graph $G = (V, E)$, where each node $v \in V$ represents a user and a directed edge $(v, u) \in E$ indicates user u is a follower of v in the social network (i.e. the direction reflects the information flow). The storage space of such a graph is proportional to the number of edges, and can easily become overwhelming for large networks. To serve real-time queries related to user link structures, it is desirable to have an in-memory compact summary of them.

Our solution to this problem is to represent the adjacency list of a user as a transaction of items, where each item is a follower of this user. Then the entire social graph G can be seen as a batch of user-follower transactions. We would like to apply our SPUR algorithm to a batch of such transactions and compress its storage space. However, the SPUR algorithm is not directly applicable to the graph summarization problem for two reasons. First, SPUR produces a lossy compression by dropping infrequent items (that are, edges in the graph data), which can possibly disconnect a graph and introduce errors to various mining algorithms. Second, the frequent itemset mining stage of SPUR would not be scalable to graphs with hundreds of millions of nodes and billions of edges, if the whole graph would to be processed at once.

To address those issues, we propose the G-SPUR algorithm with two modifications of SPUR to enable lossless and fast summarization of large-scale graph data. A graph G is represented by a database of transactions, where each transaction is the adjacency list of a vertex and items in the transaction are the neighbors connected to the vertex. G-SPUR drop nodes whose numbers of adjacent edges are below a support threshold σ, and use a separate graph G_{infreq} to preserve those infrequent edges. The frequent edges will be stored in another graph G_{freq} for further processing. If G, G_{infreq} and G_{freq} are represented as adjacency matrices, we can see the above process decomposes G as the sum of G_{infreq} and G_{freq} (Equation 2). Because

G_{infreq} only contains the edges connected to vertex with in-degree below the support threshold, we directly store it as a sparse matrix.

$$G = G_{infreq} + G_{freq} \quad (2)$$
$$= G_{infreq} + T \times P \quad (3)$$

After the above separation of infrequent and frequent edges, we run SPUR on G_{freq} without loss of information because all edges in G_{freq} are frequent. The SPUR algorithm will compress G_{freq} to a pattern set P and a transaction set T. Each pattern $p \in P$ contains a set of vertices from G_{freq}. Each transaction $t \in T$ contains a set of patterns from P, corresponding to the compressed representation for the original adjacency list. To reconstruct the adjacency list of a vertex from G_{freq}, we can take the union of patterns in a transaction. In a binary sparse matrices representation of the pattern set P and transaction set T, the SPUR algorithm essentially decomposes the frequent graph G_{freq} to the product of transaction set T and pattern set P (shown in Equation 3). By using G_{infreq}, T and P, we are able to store the original graph G with smaller storage size and reconstruct G without information loss.

Note that G_{freq} contains most of the edges in the original graph G. Therefore, the input to the SPUR algorithm will be as large as hundreds millions of transactions and billions of items. To maintain a scalable solution, we use minwise independent hashing [3] to partition the data into small samples. The SPUR algorithm will operate on each partition independently and produce a summary for each partition. We can generate the final solution by merging the pattern sets and transaction sets from the partitions. This method has been used by Buehrer *et al.* [4] to improve the scalability of large-scale web graph compression problems.

3.1 Speeding up PageRank with G-SPUR

The PageRank algorithm models the link structure of web graphs by the random walk behavior of a *random surfer* [2,6]. The web graph can be represented by a directed graph $G = (V, E)$, and its adjacency matrix A is defined as $A(u, v) = 1$ if edge $(u, v) \in E$; otherwise, $A(u, v) = 0$. Matrix W denotes the row normalized matrix of A. The PageRank vector p is computed iteratively using the following equation until convergence:

$$p^{(k+1)} = cW^T p^{(k)} + (1 - c)p^{(0)} \quad (4)$$

where c is a damping factor (set to 0.85 in our experiment), and $p^{(0)}$ is initialized as a n by 1 vector with all elements set to $1/n$. The major computational cost of Equation 4 is to compute the product of sparse matrix W^T and vector $p^{(k)}$. Previous work [5] shows that graph mining algorithms such as PageRank and SALSA can be directly computed from compressed graphs and the performance can be improved because the total number of computations can be reduced due to compression. From Equation 4, we can see that the PageRank algorithm can be implemented by iteratively calling the sparse matrix and vector multiplication (SPMV) kernel on a graph G. Since G-SPUR decomposes a graph G into G_{infreq}, T, and P, all of which can be stored as sparse matrices, we can directly implement the PageRank algorithm as iterative SPMV on $G_{infreq} + T \times P$.

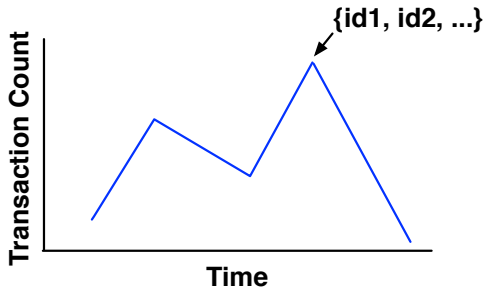
4 Content and Time Aware Network Topology Queries

In the previous sections, we introduce methods to create summarization of social network content stream and link structure. Besides those topics, another important analytical task is to investigate the network topology of the users who have written or read messages about a topic. Given a topic or keyword, example queries can be as simple as finding users who wrote or read this topic. More insights into the network topology can be obtained if we can find the social connections among these users. Furthermore, with these user connections, we can find which users are influential writers about this topic, whether there is any community structures among the users.

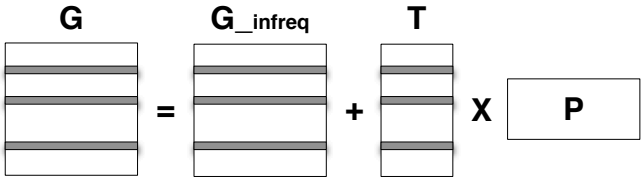
We can use the compact storage of the network content (SPUR, D-SPUR) and topology (G-SPUR) to answer the above queries in two major steps. In the first step, given a query keyword and time interval, we extract a subgraph of the entire network topology which contains all the users who either wrote or read a message about this keyword during the query time interval. In the second step, we run various graph mining algorithms on the extracted subgraphs to find influential users, and community structures in the network topology.

4.1 Content and Time Aware Subgraph Construction

First, we present our method of constructing a user subgraph given a query keyword and time interval. We achieve this by querying on the summaries built by the D-SPUR and G-SPUR algorithms.



(a) Modification to the D-SPUR algorithm



(b) Extract subgraph with G-SPUR

Figure 3: Construction of Content and Time Aware Subgraph

4.1.1 Incorporate Author Information into D-SPUR

Given a content keyword, we first find the list of users who have written messages about this keyword during the query time interval. We can slightly modify our D-SPUR algorithm to fulfill this query requirement as follow. First, we query the summary objects within the query time interval from the pyramidal time window; Second, in each summary object, we retrieve the patterns that contain the query keyword, and the transactions that include these patterns. These transactions represent all the messages containing the query keyword in the query time interval. Third, we need to find the writers of these messages. These users are who have written messages about the query keyword during the query time interval. Here we need to slightly change the D-SPUR algorithm to retrieve these users. In the original D-SPUR algorithm, IDs of users in each transaction is dropped. To preserve user information, we modify it by adding the list of user IDs at each point in the time series. Figure 3(a) illustrates this modification. Therefore, at a given time and a specific transaction, we can know which users wrote the transaction. With the above steps, we can extract the complete list of the writers of messages with a query keyword during a query time interval.

4.1.2 Extract Subgraph from Compressed Social Network

With the list of users U who have written messages about a query keyword during the query time interval, we then find their social connections by extracting a subgraph of them and their followers from the network topology. We will use the compressed network topology built by G-SPUR to find these social connections.

Suppose a graph G represents the original network topology which contains the social connections among all users. Equations 2 and 3 in Section 3 show that our G-SPUR algorithm decomposes the adjacency matrix of G into the adjacency matrix of a graph G_{infreq} and the product of a *transaction-pattern* matrix T and a *pattern-item* matrix P. In the original graph G, this subgraph of users U corresponds to a subset of the rows $G(U,:)$ in the adjacency matrix of G, where each row represents the followers of a user in U. For example, the three highlighted rows in the matrix G in Figure 3(b) contains the social connections of three users and their followers. Because of the G-SPUR decomposition $G = G_{infreq} + T \times P$, we can extract a subset of the rows $G(U,:)$ as $G(U,:) = G_{infreq}(U,:) + T(U,:) \times P$. This means we can get the rows of users in U from matrices G_{infreq} and T first, then use the rows from T to multiply with the pattern matrix P and then add to the rows from G_{infreq}. For instance, to get the three highlighted rows from G in Figure 3(b), we first get three rows from G_{infreq} and T respectively, then multiply the three rows from T with P, and then add the result to the three rows from G_{infreq}, the final results will be equivalent to the three highlighted rows in the original G. Therefore, we can efficiently extract a subgraph of the network topology from the D-SPUR and G-SPUR summaries by querying a content keyword and a time interval.

4.2 Mining Algorithms on Subgraphs of Network Topology

With a subgraph of a network topology conditioned on a query keyword in a query time interval, we can perform static analysis such as finding relevant users or communities of users on a topic during the query time interval. Here, we introduce two example mining queries to perform these tasks under our proposed framework.

4.2.1 Content and Time Aware PageRank

To rank users' importance regarding a topic keyword w during a time interval t, we can extract the subgraph $G(w,t)$ from G. This directed subgraph captures the social connections among all users who wrote messages about w during t. We then iteratively run PageRank (Equation 4) on the adjacency matrix of $G(w,t)$. The computation can be accelerated by the GPU based high performance computing platform introduced in Section 3.1.

4.2.2 Clustering

Given a subgraph $G(w,t)$ of the entire network topology G, we would like to find communities in such subgraphs to capture the topological relationships among the users who write or read the content keyword w in time interval t. This is meaningful because users have different interests in different topics and form different community structures. In this section, we introduce a new type of complex query to find such communities. Given a content keyword w and a time interval t, the query will return a clustering result C of the *active* users who write or read messages about the keyword w during time interval t in the social network. The general idea of answering such query is to first extract the subgraph $G(w,t)$, then apply graph clustering algorithms on the subgraph to find user communities. However, there are several challenges to execute the above process:

- **Scalability**: The scale of the subgraph $G(w,t)$ varies, and can become very large if a popular keyword, a long query time interval or a high-degree user is involved. A scalable clustering algorithm is thus needed to answer large number of queries efficiently.
- **Noise**: Previous work [9] has shown that there are two types of users in modern social network such as Twitter: a small fraction of influential users (e.g. celebrities, organizations), and a large number of auxiliary users (mostly followers of the influential ones). While community kernels [9] formed by influential users are more related to the network content, auxiliary users also form clusters and create noise in finding community kernels. The presence of auxiliary users also slow down the graph clustering algorithms.

- **Connectivity**: The influential users usually have a lot of followers but they rarely follow back. The connections among the influential users are weak as they rarely follow back, and it becomes hard to find dense communities if not including the auxiliary users. The auxiliary users who follow different influential users are essential to improve the connectivity of our subgraph $G(w,t)$.
- **Directionality**: The follower relationships among users are directed, so is the subgraph $G(w,t)$. Satuluri *et al* [8] show that it is non-trivial to cluster directed graphs by using graph clustering algorithms designed for undirected graphs.

To overcome the above challenges, we use a simple but effective preprocessing approach to symmetrize the directed graph $G(w,t)$ and adjust the edge weight to improve connectivity among influential users and reduce the noise from auxiliary users.

1. Symmetrization: We use the bibliometric symmetrization [8] method to transform our asymmetric subgraph $G(w,t)$ to a symmetric graph $SG(w,t)$. Given the adjacency matrix $A(w,t)$ of $G(w,t)$, bibliometric symmetrization essentially calculates the adjacency matrix of $SG(w,t)$ as $A(w,t)A(w,t)^T + A(w,t)^T A(w,t)$. This transformation not only removes the directionality of edges in $G(w,t)$ but also adds edges to vertices sharing similar set of in- or out- links. In $SG(w,t)$, edges can exist between two influential users who are not directly connected, but share common followers. This improves the connectivity among influential users.

2. Edge Re-weight: Edge weights in $SG(w,t)$ are only based on the topological information. We would like to incorporate the network content information to down-weigh the connections among auxiliary users who rarely contribute content to the network, and to reduce the noise. Given the keyword w and a node i, we calculate the weight $W(i)$ for node i by counting the number of times user i mentioned the keyword w in the time interval t. Suppose we have an edge from node i to node j, and the edge weight in the symmetrized graph $SG(w,t)$ is $SG_{w,t}[i][j]$. We adjust the weight of edge $< i,j >$ by multiplying $SG_{w,t}[i][j]$ with the sum of $W(i)$ and $W(j)$. In this way, we can construct a new weighted symmetric graph $WSG(w,t)$ with edge weight $WSG_{w,t}[i][j] = SG_{w,t}[i][j] \times (W(i) + W(j))$. The intuition of this edge re-weight method is that the node weight $W(i)$ captures how much interest user i has on the keyword w. Therefore, we want to boost up the weight of edges connected to nodes with strong interest on the query keyword w, especially for the edges that both the follower and the followee express strong interests.

After the above preprocessing on our subgraph $G(w,t)$, we have a weighted undirected graph $WSG(w,t)$ where the nodes of similar influential users are connected together and the links to noisy auxiliary nodes are down-weighted. We then run scalable graph clustering algorithms such as MLR-MCL [7] to efficiently find dense communities of influential users.

5 Experimental Results

In this section, we present results for an extensive set of experiments we conducted to evaluate the G-SPUR algorithm. We discuss the compression performance of several large web graphs and the follower-followee graph of twitter. To show the benefits of graph compression in speeding up graph mining kernels, we also implement the PageRank algorithm using the summarized graph with CPUs and GPUs.

We gathered 2100 hours of Twitter message streams from June to September in 2010 [1], and crawled the follower lists of all the users in the above message stream [2]. We construct the follower-followee graph of Twitter from this dataset. There are about 131 million vertices and 3.8 billion directed edges. We also use four other web graphs, shown in Table 9. All algorithms were implemented in C++.

[1] As provided by Twitter, it is a 15% random sample of all messages.

[2] Some of the users' follower information is not available because of their privacy settings

Graph	Nodes	Edges	Edges/Node	Density	Power-law?
it-2004	41,291,594	1,150,725,436	27.9	6.75×10^{-7}	Yes
sk-2005	50,636,154	1,949,412,601	38.5	7.60×10^{-7}	Yes
uk-union	133,633,040	5,507,679,822	41.2	3.08×10^{-7}	Yes
web-2001	118,142,155	1,019,903,190	8.6	7.31×10^{-8}	Yes

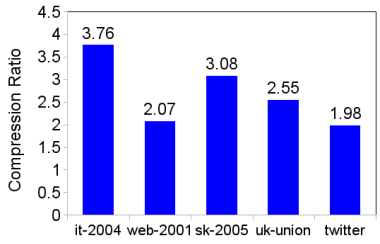
Table 9: Web Graph Datasets

5.1 Graph Compression with G-SPUR

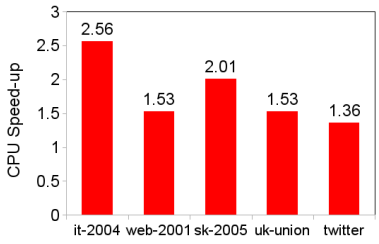
We use min-wise hashing to cluster the graph into partitions with partition size less than 1000 and we ran G-SPUR algorithm on each partition with absolute support value at 5. Figure 4(a) shows the compression ratio of the G-SPUR algorithm on these graphs. We can see that the G-SPUR algorithm can compress the storage size of large-scale web graphs to as low as 4 times smaller than the original graphs. In the Twitter follower-followee graph, our G-SPUR algorithm can still reduce the storage size by half.

5.2 Graph Mining Speed-up on CPU and GPU

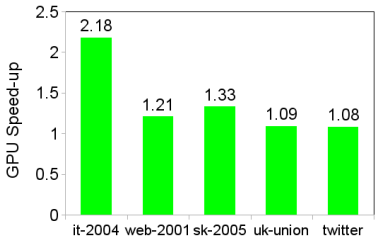
Next, we present some experimental results of speeding up the PageRank algorithm with G-SPUR. On CPU, we store the three sparse matrices G_{infreq}, T and P in CSR format and perform matrix and vector multiplication. Figure 4(b) plots the speed-up numbers of compressed graph over uncompressed on the four large web graphs and the twitter social graph. We can achieve from 1.4x to 2.6x speed-ups on these dataset. On GPU, we store the three sparse matrices in our optimized composite storage format. To compare with the performance of our multi-GPU SPMV kernel, we distribute matrices G_{infreq} and T to multiple GPUs, and each GPU will keep a copy of P because it is needed by all nodes of the GPU cluster. Figure 4(c) plots the speed-up numbers of the five datasets on GPUs. The PageRank algorithm can achieve from 1.1x to 2.2x speed-ups on the compressed graph over the original graphs. The major computational cost of the PageRank algorithm is the iterative SPMV kernel whose running time is proportional to the storage size of the graph. Our G-SPUR algorithm can effectively compress the size of the graph by decomposing it into three smaller matrices. Therefore, we can conclude that the speed-ups of the PageRank algorithm come from the compression of the graphs by the G-SPUR algorithm.



(a) Compression ratio of graphs



(b) CPU speed-up of PageRank



(c) GPU speed-up of PageRank

Figure 4: Compression ratio and speed-up of G-SPUR

Rank	Screen Name	Truncated Account Profile Description
1	Lonely Planet	Tweeting (& retweeting) the best in travel
2	GWPStudio	Photography, Socialmedia & sharing ... Love to travel & connect with people
3	American Airlines	The official channel of American Airlines
4	Sean Gardner	Digital Media Consultant
5	Gary Arndt	Traveler, blogger and photographer. A one man National Geographic. Been to over 100 countries ...
6	Tavelzoo	Travelzoo deal experts research and evaluate thousands of deals each week, selecting only the best ...
7	SmarterTravel	SmarterTravel.com is the largest online travel resource for unbiased travel news, deals, and timely ...
8	WhereIveBeen	Travel industry's leading social networking travel platform
9	TravelDeals	Use Twitter to save on travel in popular locations. Get a customized feed of travel deals near you
10	USATodayTravel	USA TODAY Travel offers the latest travel news, deals and consumer features about flights, hotels, ...
11	Andreas Susana	A guy from Austria, who writes about his trips and his website concerning books, castles, ...
12	Melvin	Love to travel, to discover the world, to travel free & untroubled & still be informed like an insider! ...
13	JD Andrews	World Traveler, Dad, 3xEmmy winner, Video, Adventure, Photographer, love dogs, Sharing & Caring ...
14	British Airways	Official British Airways global account
15	Get a Travel Deal	I find the best travel deals so you don't have to. Life's Short Travel Often!
16	Eagles Nest Wine	San Diego's Medal winning-ist Boutique Winery! Share an Authentic Wine Lifestyle with us! ...
17	Chicago Events	Real-time local buzz for live music, parties, shows and more local events happening right now in Chicago!
18	travelblggr	TV Host. Writer. Videographer. Travelista.
19	TravelGreen	Tips for sustainable travel and green living. Exploring the world, trying new foods & being green.
20	Tourism Malaysia	The official Tourism Malaysia Twitter account.

Table 10: Top 20 ranked users about the keyword "travel"

5.3 PageRank on Subgraph

Next, we show experimental results of content aware PageRank queries on the Twitter social network data we crawled from June 2010 to September 2010. We extracted the subgraph of users who mentioned the Twitter hashtag "#travel". We run the PageRank algorithm on this subgraph and rank accounts by their PageRank value from high to low. Table 10 lists the top 20 accounts, which can be classified into the following categories:

- Free information sources that people follow to find and share travel information, such as #1 Lonely Planet, #8 WhereIveBeen, #10 USATodayTravel and #19 TravelGreen.
- Travel deal websites, including #6 Travelzoo, #7 SmarterTravel, #9 TravelDeals and #15 Get a Travel Deal. These results are from a subgraph queried during the time of summer 2010. We know that people often have their vacation trips in summer and they want to reduce their travel expenses. Therefore, it is expected that those travel deal websites are active and popular on Twitter during the summer.
- Airline companies such as #3 American Airlines and #14 British Airways, because information of air transportation is a huge factor for travelers to plan their itinerary.
- Interesting travel destinations including #16 Eagles Nest Winery, #17 Chicago Events and #20 Tourism Malaysia to promote their travel packages. Since our dataset is only a random sample from all tweets, we did not find any tweets written by accounts representing famous places of interest in our dataset.
- Famous individual bloggers to share their experiences. This category includes #2 GWPStudio, #4 Sean Gardner, #5 Gary Arndt, #11 Andreas Susana, #12 Melvin and #13 JD Andrews. Some commonalities among these accounts are that they have large number of followers, they almost always follow back to their followers, they also write a lot of tweets and post photos to share their own traveling experiences.

From the above example, we can see that by running the PageRank algorithm on the subgraphs of a social network, we can find popular and influential account representing organizations, companies or individuals related to a content keyword in a time interval.

Cluster 1		Cluster 2			
PrizeDrawsUK	PiggyCodeUK	coupons2grab	CouponsInfo	SavvyPCDeals	internetshopper
Deals4UK	CodesUK	slickwallet	CouponNet	Spaffin_ebay	CouponCodeFeed
TopUKDeals		CouponSpy	Deals_Vista	DirectCoupons	redtagdeals

Table 11: Clustering results for the keyword "coupons"

5.4 Clustering on Subgraph

Here, we present experimental results on graph clustering queries. Table 11 shows the influential users from two clusters we obtained when querying the subgraph for the keyword "coupons" in the time interval from July 1st, 2010 to July 31st 2010. We can see from the screen names of these users that they are all related to coupons and deals in online shopping. Furthermore, we can see that the user names listed in cluster 1 are the Twitter accounts for online shopping websites in UK whereas the user names in cluster 2 are mostly in US with some global online shopping websites. Both cluster 1 and cluster 2 can be considered as community kernels because the user accounts have a lot of followers who are their customers. Also this cluster arrangement is reasonable because the accounts in different clusters have different follower populations. The followers of cluster 1 are mostly customers from UK whereas the followers of cluster 2 are mostly in US. Such clustering analysis is useful for online marketing with a targeted customer population.

6 Conclusions

We proposed G-SPUR, a novel algorithm to compress social network topology with low compression ratio, high quality and fast running time. The compressed link structures require less storage space, and can be directly used to speed up a series of graph mining kernels. It can also be used together with compressed social content stream to answer content and time aware network queries.

References

[1] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for clustering evolving data streams. *VLDB 2003*.

[2] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Comput. Netw. ISDN Syst.*, 30(1-7):107–117, 1998.

[3] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. *Journal of Computer and System Sciences*, 60(3):630–659, 2000.

[4] G. Buehrer and K. Chellapilla. A scalable pattern mining approach to web graph compression with communities. *WSDM 2008*.

[5] C. Karande, K. Chellapilla, and R. Andersen. Speeding up algorithms on compressed web graphs. *WSDM 2009*.

[6] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, 1999.

[7] V. Satuluri and S. Parthasarathy. Scalable graph clustering using stochastic flows: applications to community discovery. *KDD 2009*.

[8] V. Satuluri and S. Parthasarathy. Symmetrizations for clustering directed graphs. *EDBT 2011*.

[9] L. Wang, T. Lou, J. Tang, and J. E. Hopcroft. Detecting community kernels in large social networks. *ICDM 2011*.

[10] X. Yang, A. Ghoting, Y. Ruan, and S. Parthasarathy. A framework for summarizing and analyzing twitter feeds. In *SIGKDD 2012*. ACM.