This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2016.2542804, IEEE Transactions on Knowledge and Data Engineering

Personalized Influential Topic Search via Social Network Summarization

Jianxin Li, Chengfei Liu *Member, IEEE*, Jeffrey Xu Yu *Member, IEEE*
Yi Chen *Member, IEEE*, Timos Sellis *Fellow, IEEE*, and J. Shane Culpepper

Abstract—Social networks are a vital mechanism to disseminate information to friends and colleagues. In this work, we investigate an important problem - the *personalized influential topic search*, or PIT-Search in a social network: Given a keyword query q issued by a user u in a social network, a PIT-Search is to find the top-k q-related topics that are most influential for the query user u. The influence of a topic to a query user depends on the social connection between the query user and the social users containing the topic in the social network. To measure the topics' influence at the similar granularity scale, we need to extract the social summarization of the social network regarding topics. To make effective topic-aware social summarization, we propose two random-walk based approaches: random clustering and an L-length random walk. Based on the proposed approaches, we can find a small set of representative users with assigned influential scores to simulate the influence of the large number of topic users in the social network with regards to the topic. The selected representative users are denoted as the social summarization of topic-aware influence spread over the social network. And then, we verify the usefulness of the social summarization by applying it to the problem of personalized influential topic search. Finally, we evaluate the performance of our algorithms using real-world datasets, and show the approach is efficient and effective in practice.

1 Introduction

The importance of social networks such as Twitter, Facebook and WeChat in providing a convenient platform for users to share information continues to grow. The dynamic nature of information and user connectivity within these networks have presented many interesting open research problems in recent years, such as the influence maximization problem [5, 8, 9, 11], and the topic detection problem [24, 28]. However, a user issuing a keyword query can easily be overwhelmed by the number of query-related topics. This is because a large social network may contain millions of social users sharing comments on various events or topics, and these comments lead to new topics. In this context, identifying a small set of topics that are relevant to the query is a challenging problem. The most widely-accepted method is to select the relevant topics based on the term relevance between topics and the query in a manner similar to a typical keyword search [26, 27]. An alternative approach is to recommend the latest query-related topics for a query [14], but this may not be suitable in practical applications since it does not consider older topics or discussions.

- *Jianxin Li is with the School of Computer Science and Information Technology, RMIT, Australia. jianxin.li@rmit.edu.au*
- *Chengfei Liu is with the Faculty of Science, Engineering and Technology, Swinburne University of Technology, Australia. cliu@swin.edu.au*
- *Jeffrey Xu Yu is with the Department of Systems Engineering & Engineering Management, The Chinese University of Hong Kong, China. yu@se.cuhk.edu.hk*
- *Yi Chen is with College of Computing Sciences, New Jersey Institute of Technology, USA. yi.chen@njit.edu*
- *Timos Sellis is with the Faculty of Science, Engineering and Technology, Swinburne University of Technology, Australia. tsellis@swin.edu.au*
- *Shane Culpepper is with the School of Computer Science and Information Technology, RMIT, Australia. shane.culpepper@rmit.edu.au*

To address this problem, we would like to select a set of representative users that adequately summarize a given topic, where the representative users are evaluated based on their closeness to the users (topic users) whose posted messages contain the topic terms. In this case, if the query user is likely to be reached or influenced by the selected representative users, then the query user has a high probability of following or trusting the topics recommended by the selected representative users. In this paper, we refer to this problem as *Personalized Influential Topic Search*, or more succinctly PIT-Search. Our goal is not to find how a topic influences a group of users, rather it is to find how important topics and influential users might be better leveraged to meet a specific user's information need. In other words, if users are in similar social contexts in the social network, they would see the same results when they issue the same keyword query. PIT-Search can be directly applied in many personalized services in social networks, such as personalized recommendation and search, target advertising, or personal product promotion.

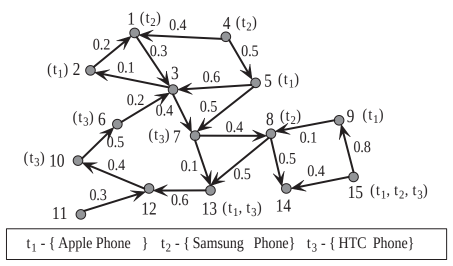


Fig. 1. An Example of Social Network Graph

Example 1 (PIT-Search). Consider the small social network depicted in Figure 1, where a node represents a user, and the edge

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2016.2542804, IEEE Transactions on Knowledge and Data Engineering

weights represent the *influence* users have over neighbors. The direction of an edge shows the influential relationship between two nodes. Here, assume users have expressed positive opinions about topics being discussed related to mobile phones, such as *Apple Phone* for t_1, *Samsung Phone* for t_2 and *HTC Phone* for t_3. A user may mention several different phones like User 13. Now suppose that User 3 wants to know which phone is the best choice for her by checking her social network via a query $q = \{\text{Phone}\}$. In this case, all three phones t_1, t_2 and t_3 are q-related topics, and there are three neighborhood users (User 1 - t_2, 5 - t_1, 6 - t_3) who can influence User 3, making the choice a difficult one.

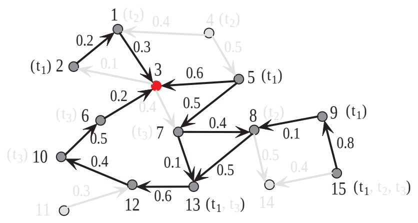


Fig. 2. Demonstration of Computing Influence of t_1 for User 3

We address thi challenge as follows. Given topic t_1 in Figure 2, the influence transition paths and corresponding transition probabilities are shown below. Since there are five users containing t_1, each node is assumed to have an equal local weight – $\frac{1}{5}$. The final influence score for a topic to a specific node can be calculated by multiplying the local weight and the transition probabilities.

Path	Transition Probability
$2 \rightarrow 1 \rightarrow 3$	0.060
$5 \rightarrow 3$	0.600
$5 \rightarrow 7 \rightarrow 13 \rightarrow 12 \rightarrow 10 \rightarrow 6 \rightarrow 3$	0.000
$13 \rightarrow 12 \rightarrow 10 \rightarrow 6 \rightarrow 3$	0.024
$9 \rightarrow 8 \rightarrow 13 \rightarrow 12 \rightarrow 10 \rightarrow 6 \rightarrow 3$	0.001
$15 \rightarrow 9 \ldots 6 \rightarrow 3$	0.001
Aggregation $\times \frac{1}{5}$	Final Score $= 0.137$

Similarly, the influence of topics t_2 and t_3 can be computed for User 3 – 0.188 and 0.065. Using the intermediate results, a PIT-Search would return t_2 (Samsung Phone) as the top-1 result for User 3 because t_2 is the most influential topic in the social context of User 3. If User 7 issues the same query q, then t_3 (HTC Phone) will be the top-1 result. Similarly, t_2 (Samsung Phone) is returned if q is issued by User 14.

As shown in Example 1, PIT-Search can return different topic results for different users even if they are issuing the same query. There are three issues to be addressed when performing an efficient and effective PIT-Search. First, a social network may have a large number of q-related topics. Secondly, for each q-related topic, a PIT-Search must compute the influence from a larger number of topic-related users for the given user and query. Thirdly, any given search can be mistakenly dominated by a few influental neighbors. To reduce this bias, we need to derive topic driven social summarizations for a social network by selecting representative users.

To address the above challenging issues, we investigate the problem of PIT-Search by developing the following three techniques: topic-sensitive representative node selection, a topic-to-representative user index, and a personalized propagation index. In order to perform topic-sensitive representative node selection, and build topic-to-representative user index, we develop two novel random-walk based approaches that can represent all topic nodes using only a subset of topic-sensitive representative nodes based on the influence propagation of a topic in the social network. Once we identify the relations among topic nodes and representative nodes, the corresponding topic-to-representative user index can be then built to capture the topic-sensitive local influence of small communities in social networks. We then construct a personalized propagation index that maintains a subset of nearby nodes and the associated propagation probabilities for every node in the graph. The personalized propagation index is independent of both topics and queries. With the support of both indexes, we can efficiently assess the influence for any q-related topic on-the-fly. To efficiently find the top-k most influential q-topics for a user, we also present a dynamic top-k PIT-Search algorithm that identifies the quality q-related topics in the top-k result set based on intermediate results. Low-quality topics are pruned from the result set by probing as few nodes as possible. Using the three techniques, we can greatly accelerate the performance of PIT-Search in large social networks.

In summary, the contributions of this paper are as follows.

- We develop two novel random-walk based approaches to select topic-sensitive representative nodes in a social network. This captures the local influence for a topic in different parts of the social network.
- We design a top-k PIT-Search algorithm to efficiently compute the k most influential q-related topics for a user based on our proposed pre-computed index.
- We evaluate the efficiency and effectiveness of our proposed techniques using a large Twitter dataset, and show the benefits of our methods by comparing with three baselines adapted from previous work on social networks.

The remainder of this paper is organized as follows. Section 2 presents the formalization of the PIT-Search problem. In Section 3 and Section 4, we present two novel approaches for choosing topic-sensitive representative nodes from social networks. We present the procedures for building a personalized propagation index and describe an efficient top-k PIT-Search algorithm in Section 5. The experimental results are presented and discussed in Section 6. Finally, we review related work in Section 7 and conclude this paper in Section 8.

2 Problem Definition

Consider a social network graph $G = (V, E, T, \Lambda)$ where V is the node set representing the social users in the social network, and E is the edge set depicting links between users. T is a topic space where each social user may have a set of topics extracted from the user's posted messages, e.g., $T(v) = \{t_1, ..., \}$ for a node v. Λ maintains the transition probability of edges in E.

Definition 1. [Social Summarization] Consider a social network $G = (V, E, T, \Lambda)$. Given a topic $t \in T$, a t-aware social summarization of G is the selection of a specified number of nodes V^* from V that represent the influence of the topic nodes V_t which satisfy the following condition:

$$\arg \min_{V* \subseteq V, |V*| \leq specified} \sum_{v \in V} |\mathcal{I}(t, v) - \mathcal{I}^*(t, v)| \quad (1)$$

where $\mathcal{I}(t, v) = \frac{1}{|V_t|} \sum_{u \in V_t} \sum_{p^i \in P_u^v} Pr(p^i)$ and $\mathcal{I}^*(t, v) = \sum_{u \in V^*} weight(u, t) \sum_{p^i \in P_u^v} Pr(p^i)$. P_u^v includes all possible paths from u to v. p^i represents the i-th path, and $Pr(p^i)$ is the propagation probability of path p^i by multiplying its edges' transition probabilities maintained in Λ. For each selected representative node u and a topic t, $weight(u,t)$ aggregates the influence of a set of topic nodes clustered by the central node u. The weight can be then taken as the initial propagation power of the representative node when evaluating the influence to the query user for the topic.

Definition 2. [**PIT-Search Problem**] Given a keyword query q issued by a user v, a social network G, and the number k of items to return, a PIT-search returns the k most influential q-related topics to v by satisfying:

$$\arg\max_{T^k \subseteq T} \{\sum \mathcal{I}^*(t,v) | t \in T^k\} \tag{2}$$

where $\mathcal{I}^*$ is the aggregate influence of topics in T^k to v, T^k is a k-size of topic subset selecting q-related topics from T.

The problem of computing social summarization can be reduced to the optimal subset selection with constraints, which is an NP-complete problem [1]. Therefore, computing social summarization is also an NP-complete problem. In this paper, we first address the challenging problem of selecting representative nodes V^* in Section 3 and Section 4 in two different ways. Then, we develop an online PIT-Search algorithm by using these pre-selected representative nodes in Section 5.

3 Approximate Random Clustering (RCL-A)

In order to select a set of representative nodes for the topic nodes, one approximate solution is to cluster the topic nodes, and then identify a central node for each group. Lastly the local influence for all unselected topic nodes is migrated to the central node. The central nodes are then taken as the representative node set. The aggregated local influence score for each representative node can be used to evaluate the influence for topics to each social user.

3.1 Clustering Topic Nodes

The main idea of clustering topic nodes is to first measure the common reachability of topics nodes for a given sample node set, and then group the topic nodes into different clusters based on the common reachability. Here, the reachability of a node is simply the number of sample nodes that can be reached by the node. Given two nodes, the common reachability is evaluated as the number of sample nodes reached by the two nodes together. The fundamental assumption is that if two nodes can be grouped into one cluster, then there is a high likelihood that these two nodes can reach a certain number of common neighbors. In other words, the more common neighbors nodes can reach, the higher the probability of grouping the two nodes together.

To decide if two nodes can be grouped together or not, the **RCL-A** solution is based on the following notion. First, select a sampling node set V' from V. Three variants are used to measure the grouping relation of the two nodes u and v:

- $GP^+(u,v) = \frac{|\{x|x \rightarrow^L u \text{ and } x \rightarrow^L v, x \in V'\}|}{|V'|}$ where $x \rightarrow^L u$ and $x \rightarrow^L v$ mean node x can reach nodes u and v in L hops. $GP^+(u,v)$ represents the probability of grouping u and v with regard to the sampling node set V'.
- $GP^-(u,v) = \frac{|\{x|x \rightarrow^{>L} u \text{ if } x \rightarrow^L v \text{ or } x \rightarrow^{>L} v \text{ if } x \rightarrow^L u, x \in V'\}|}{|V'|}$ where $x \rightarrow^{>L} u$ means x cannot reach u within L hops. $GP^-(u,v)$ represents the probability of splitting u and v into different groups with regard to the sampling node set V'.
- $GP^*(u,v) = \frac{|\{x|x \rightarrow^{>L} u \text{ and } x \rightarrow^{>L} v, x \in V'\}|}{|V'|}$ represents the probability of not knowing if nodes u and v can be grouped. This is equivalent to $1 - GP^+(u,v) - GP^-(u,v)$.

To group topic nodes, the following clustering rules are applied:

- Rule 1: Two nodes u and v are grouped if $GP^+(u,v) \geq GP^-(u,v)$ and $GP^+(u,v) \geq GP^*(u,v)$ hold.
- Rule 2: Two nodes u and v are not grouped if $GP^-(u,v) \geq GP^+(u,v)$ and $GP^-(u,v) \geq GP^*(u,v)$ hold.
- Rule 3: Two nodes u and v are grouped based on a probability if $GP^*(u,v) > GP^+(u,v) \geq GP^-(u,v)$ hold. The probability is calculated as $\frac{GP^+(u,v)}{GP^+(u,v)+GP^*(u,v)}$ that is also equal to $\frac{GP^+(u,v)}{1-GP^-(u,v)}$.
- Rule 4: A node can only appear in one group (clustering is *hard*).

Rules 1, 2 and 4 are clear. For Rule 3, there is an implicit property.

Property 1. Grouping Probability: Given two nodes u and v, if $GP^+(u,v) \geq GP^-(u,v)$, then $\frac{GP^+(u,v)}{GP^+(u,v)+GP^*(u,v)}$ must be larger than or equal to $\frac{GP^-(u,v)}{GP^-(u,v)+GP^*(u,v)}$.

Property 1 guarantees that the nodes can be approximately grouped when they are in neither a "clearly in" state, nor a "clearly out" state.

Algorithm 1 RANDOM_CLUSTER(G,t,L)

input: A graph $G = (V,E)$, a topic t, a max hop length L, and the number C_Size of clusters
output: A set of topic node groups S

```
 1: Get topic node set $V_t$ for $t$ from inverted node index
 2: Random node selection-based sampling subset $V' \subseteq V$
 3: Initialize an array $GPLabel[|V_t|][|V_t|] \leftarrow \varnothing$
 4: Initialize a temporary array $Temp\_V_t \leftarrow V_t$
 5: for each $u \in V_t$ do
 6:     $Temp\_V_t \leftarrow \{Temp\_V_t - u\}$
 7:     for each $v \in Temp\_V_t$ do
 8:         $V_{u,L} \leftarrow$ All nodes that can reach $u$ in $L$ hops //Use indexed $I_L[u]$ in Algorithm 6
 9:         $V_{v,L} \leftarrow$ All node that can reach $v$ in $L$ hops
10:         $G^+ \leftarrow \frac{|V_{u,L} \cap V_{v,L} \cap V'|}{|V'|}$
11:         $G^- \leftarrow \frac{|V_{u,L} \cap V' - V_{v,L}|}{|V'|} + \frac{|V_{v,L} \cap V' - V_{u,L}|}{|V'|}$
12:         if $G^+ \geq G^-$ and $G^+ \geq \frac{1-G^-}{2}$ then
13:             $GPLabel[u][v] \leftarrow 1$
14:         if $G^- \geq G^+$ and $G^- \geq \frac{1-G^+}{2}$ then
15:             $GPLabel[u][v] \leftarrow 0$
16:         if $G^+ \geq G^-$ and $G^+ < 1 - G^+ - G^-$ then
17:             $Pr \leftarrow \frac{G^+}{1-G^-}$
18:             if $rand() \leq Pr$ then
19:                 $GPLabel[u][v] \leftarrow 1$
20:             else
21:                 $GPLabel[u][v] \leftarrow 0$
22: $SETree \leftarrow$ SET_ENUMERATION_TREE$(GPLabel[u][v])$
23: $S \leftarrow$ NO_OVERLAP_GROUPING$(SETree, C\_Size)$
24: return $S$
```

Algorithm 1 shows the procedure of grouping topic nodes based on a random sampling node set. First, the topic nodes V_t are retrieved from an inverted node index, and then a set of nodes V' are sampled from the original node set V. In this work, each node is sampled with a probability proportional to the degree of the node, which will be further discussed in Section 6. Next, two arrays, *GPLabel*$[|V_t|][|V_t|]$ for labeling if two nodes grouped together, and *Temp_*V_t for maintaining the list of unprocessed nodes, are initialized. In order to efficiently compute the grouping probability, the node set for which every node can reach an entry node in L hops is pre-computed and indexed. The procedure of index construction is shown in Algorithm 6. Line 16-Line 21 handle node pairs that are not clearly in/out of a group. Here, the node pairs are labeled by comparing the calculated grouping probability Pr with a random generated value between 0 and 1. As $Pr \to 1$, the likelihood of a random grouping increases. Lastly, the topic node groups are generated by calling SET_ENUMERATION_TREE, and finalized by calling NO_OVERLAP_GROUPING.

Algorithm 2 SET_ENUMERATION_TREE()

```
 1: Create a tree with root $r \leftarrow \varnothing$
 2: for each $u$ in $V_t$ do
 3:     $u \leftarrow$ false
 4:     $r$.addChild($u$)
 5: for each $v_x$ in the tree $r$ do
 6:     if $v_x \neq$ true then
 7:         $Left_{sibling} \leftarrow v_x$.getSiblings()
 8:         for each sibling $v_y \in Left_{sibling}$ do
 9:             if CHECK_GROUPING($v_x$, $v_y$) using the records in GPLabel then
10:                 new $\leftarrow v_x \cup v_y$
11:                 $v_x$.addChild(new)
12:         $v_x \leftarrow$ true
13: return Set enumeration tree $r$
```

In the Function SET_ENUMERATION_TREE shown in Algorithm 2, a set enumeration tree data structure is used to maintain the possible groupings. The tree is commonly used to represent and/or enumerate sets in a best-first fashion [23]. The SEtree can be used as an index to efficiently identify the grouping relations of nodes. The root node of the tree is initially an empty set. Each topic node is set as a child node of the root node. Then, a check is made to see if a tree node (denoted as current node set) and one of the right-side sibling nodes (denoted as probed node set) can be merged. This operation minimizes repetition in node sets. If the two node sets can be merged, then the merged node set is inserted as a child node of the current node set in the tree. At most one node can be different between the current node set and the probed node set as there is a sibling relationship. Therefore, merging two node sets is equal to adding one additional node into the current node set. Before the node can be inserted, the function CHECK_GROUPING checks if the node can be grouped with one of the existing node sets already in the tree index.

In the function NO_OVERLAP_GROUPING shown in Algorithm 3, the leaf nodes of the SETree are sequentially accessed to generate the no-overlapping groups. If a node is in an existing group, then the node will be removed from all other groups. By repeating this process, all no-overlapping topic node groups are generated.

3.2 Selecting Centroid Nodes

Given the social network graph $G = (V, E)$, and a topic node group $V_g \subseteq V$, a node $v_c \in V$ is selected as the "central" node of the group based on the closeness centrality metric where v_c may or may not be a topic node in V_g. A central node for a group is

Algorithm 3 NO_OVERLAP_GROUPING()

```
 1: Compute the approximate group size as $|V_t|/C\_Size$
 2: repeat
 3:     Set Found $\leftarrow$ false
 4:     while not Found do
 5:         $s \leftarrow r$.leftMostChild()
 6:         if $|s| > \lceil |V_t|/C\_Size \rceil$ then
 7:             $r$.removeNode($s$)
 8:         else
 9:             Set Found $\leftarrow$ true
10:     Add $s$ as the first group in $S$
11:     for each $v \in$ group $s$ do
12:         $r$.removeNode($\{v\}$)
13: until $r$.hasChildren()==false
14: return The set of groups $S$
```

important as it can reach the nodes in V_g more quickly than other nodes in V. Therefore, the importance of a central node candidate can be measured by how close it is to all the topic nodes in V_g.

Given a node $v_i \in V$ and a topic node group V_g, the average distance of v_i to V_g is

$$D_{avg}(v_i) = \frac{1}{|V_g|} \sum_{v_j \in V_g} distance(v_i, v_j) \tag{3}$$

where $distance(v_i, v_j)$ is the minimal hops walking from v_i to v_j on the directed graph G.

Definition 3. **Closeness Centrality of Node:** The closeness centrality of $v_i \in V$ for a topic node group V_g is defined:

$$\begin{aligned} C_C(v_i) &= [\frac{\sum_{v_j \in V_g} distance(v_i, v_j)}{|V_g|}]^{-1} \\ &= \frac{|V_g|}{\sum_{v_j \in V_g} distance(v_i, v_j)} \end{aligned} \tag{4}$$

Although the closeness centrality of a node is equivalent to the all pair shortest path problem, it is not practical to process large graphs as an exact solution for the problem costs $\Theta(|V|^3)$. Different from the work in [18], our goal is to select a node close to a given group of topic nodes and the maximal distance of any two nodes in the group is limited to $2L$. Therefore, many nodes in V can be pruned from the calculation. To efficiently find the central node for a group of topic nodes, a heuristic approach is proposed that first identifies a few central candidates without using Equation 4, and then find the best one by computing the centrality of the subset of candidates.

The brief procedure of central node selection is presented in Algorithm 4. From Line 2-Line 5, the common nodes reached by the topic nodes in V_g within L hops are computed, and the number of votes (using *VoteCount*) of topic capable of reaching a common node is stored. In other words, $VoteCount$ represents the hitting frequency of a common node. The nodes with the most votes in *VoteCount* become the candidate set. For each candidate, we compute the closeness centrality using Equation 4. Function CENTRALITY_COMPUTATION is used to compute the centrality of a node for a topic node group in G. In Line 10-Line 14, the best central node is determined by identifying each possible node in the selected candidate set, and stored as *best*, The best central node may change through the following optimizations: (1) The candidate set is further reduced by repeating Line 2-Line 5 with smaller L; (2) The identified central node from the candidate set can be further adjusted by probing the nearest neighbor nodes

Algorithm 4 SELECT_CENTRAL(G, V_g)

```
1: Initialize an array, denoted as VoteCount
2: for each topic node $v_i$ in $V_g$ do
3:     Get the node set $V_{v_i,L}$ reaching $v_i$ in $L$ hops // Use indexed $I_L[v_i]$ in Algorithm 6
4:     for for each node $v_x \in V_{v_i,L}$ do
5:         VoteCount$[v_x]$ += 1
6: SORT(VoteCount)
7: CandidateSet ← get the top nodes from VoteCount where the value is equal to MAX(VoteCount)
8: Set best ← ∅
9: Set $c \leftarrow 0$
10: for each candidate $v_c \in$ CandidateSet do
11:     $C_{v_c} \leftarrow$ CENTRALITY_COMPUTATION($v_c$, $V_g$, $G$)
12:     if $C_{v_c} > c$ then
13:         best ← $v_c$
14:         $c \leftarrow C_{v_c}$
15: return best
```

until the new centroid cannot be increased. Further enhancements to efficiently select the best centroid are beyond the scope of this paper.

3.3 RCL-A Algorithm and Limitations of RCL-A

The **RCL-A** algorithm is presented in Algorithm 5, which includes an offline stage to pre-compute representative nodes for topics, and the online PIT-Search based on the pre-selected representative nodes and query-related topics. This offline process is independent of online queries issued by users. The space cost is dominated by $O(argmax\{|V_t|^2\})$ where t is any topic in the maintained topic space, and $|V_t|$ is the number of nodes containing the topic t in the social network G.

Algorithm 5 RCL-A

```
1: // Offline pre-processing
2: Generate topic node groups $S$ using RANDOM_CLUSTER() in Algorithm 1
3: for each group $g \in S$ do
4:     Generate the central node as a representative node using SELECT_CENTRAL() in Algorithm 4
5:     Weight the central node based on the number of topic nodes in the group $g$
6: // Online PIT-Search
7: Generate Top-$k$ PIT List using PERSONALIZED_SEARCH() in Algorithm 10
8: return Top-$k$ PIT List
```

While **RCL-A** is a useful heuristic to solve the problem, there are a few limitations to the approach.

- Limiting one central node produced from one group may increase the influence skew between a central node in a large group and a central node in a small group. This could contribute to central nodes over-estimating or under-estimating the local influence in the social network.
- Limiting each group to one topic node may not be precise when allocating the influence to the representative nodes. This is because a topic may be have different representatives with different probabilities.
- **RCL-A** is based on a time-homogeneous approach to clustering. It may not be sufficient to only consider the "important" paths when determining the centroids for the given topic nodes. This is because some paths to be visited frequently will play important roles when evaluating the centrality of nodes.
- Computing a central node with **RCL-A** is a costly computation, and the number of generated groups may be very large.

4 APPROXIMATE L-LENGTH RANDOM WALK (LRW-A)

To address the limitations of **RCL-A** in Section 3, we now present a novel approach to select representative nodes for topics based on the technique of an $L-$length random walk [13]. First, we take R samples of L-length random walks for each node. The pre-computed values can then be used to reduce the cost of generating representative node sets for any topic in the social network. Finally, the local influence of topic nodes can be migrated to the representative nodes using an absorbing random walks.

4.1 LRW-A Index

To index the random walk samples, R inverted lists are constructed, each of which contain n sublists. The sample size R can be bounded by applying the Hoedding inequality [10], which balances the tradeoff between the sample size and the accuracy of estimation using sampled data. For each node u, a sublist contains the L-length random walks originating from node u. In addition, a time-variant visiting frequency index $H[L][n]$ is maintained to track the maximum frequency required to reinforce the PageRank ranking score in Section 4.2. The visiting frequency of a node with regards to parameter L measures the frequency of the node to be visited in the L iterations. Here, we simulate the time-variant states using a number of iterations. The resulting visiting frequencies are used to compute representative nodes when performing a vertex-reinforced random walk [20].

Algorithm 6 depicts how to construct the index when performing a sample-based random walk over G. The R inverted lists, denoted by $I[R][n]$, are organized as a two dimensional array where $I[i][w]$ indexes the i-th L-length random walk starting at node w. Here $H[L][n]$ is used to maintain the time-variant visiting frequency of each node in the time period $[1, L]$, i.e., Iteration-1 to Iteration-L. First, the algorithm initializes the arrays $I[R][n]$ and $H[L][n]$. Then, for each node w in V, the algorithm performs R L-length random walks starting with node w. For each L-length random walk, the path is chosen by randomly selecting a neighbor of the node u (u=w in the first iteration), and iteratively replacing the node u with a selected neighbor v. To avoid repeating nodes in the selected path, a *visited*$[v]$ array is used to track the status of a node v. At the same time, the maximal visiting frequency for each node is selected for a given iteration, and maintained in H. In addition, $I_L[n]$ is used to index all the nodes that can reach to node n within L hops.

4.2 Effective Representative Node Selection

Given a set V_t of nodes related to topic t in G, the task in this subsection is to select a set of nodes that will become the representative nodes for the node set V. The selected set of nodes should be a near-optimal node set, and be as close to V_t as possible. To do so, a novel personalization model similar to PageRank is developed to rank nodes based on centrality (prestige) and diversity.

Algorithm 6 INVERTTVHIT_INDEX(G, L, R)

```
input: A graph $G = (V, E)$, two parameters $L$ and $R$
output: An inverted index $I[R][n]$, a time-variant visiting frequency index $H[L][n]$, and reachable index $I_L[n]$
 1: Initialize $I[R][n] \leftarrow \varnothing$
 2: Initialize $H[L][n] \leftarrow \varnothing$
 3: Initialize $I_L[n] \leftarrow \varnothing$
 4: for each node $w \in V$ do
 5:     for $i \leftarrow 1$ to $R$ do
 6:         Initialize $visited[n] \leftarrow 0$
 7:         $u \leftarrow w$
 8:         $visited[u] \leftarrow 1/R$
 9:         for $j \leftarrow 1$ to $L$ do
10:             $v \leftarrow$ Randomly selected neighbor of $u$
11:             if $visited[v] = 0$ then
12:                 $visited[v] \leftarrow 1/R$
13:                 $I[i][w] \leftarrow v$
14:                 $I_L[v] \leftarrow w$ // Index $I_L[v]$ for Algorithms 1 and Algorithm 4
15:             else
16:                 $visited[v] \leftarrow visited[v] + 1/R$
17:             if $H[j][v] < visited[v]$ then
18:                 $H[j][v] \leftarrow visited[v]$
19:             $u \leftarrow v$
20: return $I[R][n]$, $H[L][n]$, and $I_L[n]$
```

Building on the PageRank strategy [19] and the idea of a vertex-reinforced random walk [16, 20], a diversified PageRank ranking function can be formalized as follows:

$$P_{T+1}(v) = (1-\lambda)P^*(v) + \lambda \sum_{(u,v)\in E} \frac{P_0(u,v) \times N_T(v)}{D_T(u)} P_T(u) \tag{5}$$

where

- $P^*(v)$ is a random jump probability representing the prior preference of visiting a topic node v given topic t. Assume there are m nodes relating to t. In this work, $P^*(v)$ is set to $1/m$ if v is a topic node related to t, otherwise, $P^*(v) = 0$. For different topics, $P^*(v)$ can vary.
- $P_0(u, v)$ is the "organic" transition probability prior to any reinforcement. In this work, $P_0(u, v)$ derived from the topic-related transition probability of an edge. Note that $P_0(u, v)$ is only sensitive to the overall topics of social users, as discussed in Section 2.
- $N_T(v)$ is the time-variant visiting frequency of node v in the random walks at Iteration (or time) T.
- $D_T(u) = \sum_{(u,v_i)\in E} P_0(u, v_i) \times N_T(v_i)$ is used to normalize the reinforced PageRank score in Equation 5.

The key idea of selecting representative nodes is to rank the nodes based on time-variant hitting times and closeness to the topic nodes V_t. Since the influence of a node is limited by distance, i.e., L-hops, the PageRank-style algorithm need only be ran using L-iterations. By doing so, each node weight is based on the nodes within an L-length radius.

The detailed procedure of selecting the representative node set is shown in Algorithm 7. First, all the nodes for topic t are retrieved,denoted as V_t. Two arrays, *PR*$[n]$ for tracking the updated PageRank score of each node, and *PStar*$[n]$ for maintaining the personalized preference of visiting a node given topic t, are initialized. Line 4-Line 10 set the initial values of *PStar*$[n]$ and *PR*$[n]$. Each element in *PR*$[n]$ is an object consisting of three values: (1) the node id v; (2) The PageRank score v.*previous*

Algorithm 7 REPNODES$(G, t, \lambda, \mu, H[L][n])$

```
input: A graph $G = (V, E)$, a topic $t$, a parameter $\lambda$, a percentage value $\mu \in (0, 1)$, and a pre-computed time-variant visiting frequency index $H[L][n]$
output: The selected representative nodes $V_{r,t}$
 1: Topic node set $V_t$ from $V$ for $t$ retrieved from an inverted node index
 2: Initialize $PR[n] \leftarrow \varnothing$
 3: Initialize $PStar[n] \leftarrow \varnothing$
 4: for each node $v \in V$ do
 5:     if $v \in V_t$ then
 6:         $PStar(v) \leftarrow \frac{1}{|V_t|}$
 7:     else
 8:         $PStar(v) \leftarrow 0$
 9:     $PR[v].previous \leftarrow 1$
10:     $PR[v].current \leftarrow \varnothing$
11: for $i \leftarrow 1$ to $L$ do
12:     for each node $v \in V$ do
13:         Get inlink neighbor node set $V_{in}$ of $v$ in $G$
14:         for each inlink node $u \in V_{in}$ do
15:             Get outlink neighbor node set $V_{out}$ of $u$ in $G$
16:             for each outlink node $w \in V_{out}$ do
17:                 $DTValue$ += $P_0(u, w) \times H[i][w]$
18:             $PNTValue$ += $\frac{P_0(u,v) \times H[i][v]}{DTValue} \times PR[v].previous$
19:         $PR[v].current \leftarrow (1 - \lambda) \times PStar(v) + \lambda \times PNTValue$
20:     for each node $v \in PR$ do
21:         $PR[v].previous \leftarrow PR[v].current$
22:         $PR[v].current \leftarrow \varnothing$
23: Copy $v \rightarrow v.previous$ pairs in $PR$ to a new array $tempPR$
24: SORT($tempPR$)
25: $cutPosition \leftarrow \mu \times |V_t|$
26: $V_{r,t} \leftarrow$ GETSUBARRAY($tempPR, cutPosition$)
27: return $V_{r,t}$
```

for the previous moment; and (3) The PageRank score v.*current* for the current moment. The remainder of the loop calculates the diversified PageRank score for each node in L-length iterations. In Line 14-Line 18, the normalized, reinforced PageRank score is calculated for the node v by accessing every inlink neighbor node, and the indexed time-variant hitting time $H[i][v]$ for node v at the moment i. Here, *PR*$[v]$.*previous* is the time-variant PageRank score of the node v at the moment $i - 1$. Line 19 is used to compute the adapted PageRank score of the node v at moment i using Equation 5. After all the nodes are processed, the previous PageRank score of a node is replaced by the current score as shown in Line 20-Line 22. After L iterations, the nodes are highly scored if they have large hitting times, and are close to the topic nodes in V_t. Finally, the nodes are sorted based on the diversified PageRank score, and the top-ranked nodes are returned as the representative nodes for t as shown in Line 23-Line 27.

4.3 Local Influence Migration of Topic Nodes

Now we propose a novel method to effectively migrate the local influence of topic nodes to the corresponding topic-related representative nodes without running the actual clustering algorithm shown in Section 3. The key idea of influence migration is as follows: Given a topic node, a number of random walk paths are generated starting from the topic node. For each random path, a check is made to see if it contains any representative nodes. If so, the first representative node encountered is treated as an *absorbing* state for an Absorbing Markov Chain [7]. Once an absorbing state is entered, it cannot be left. After all of the random paths for a topic node are traversed, a set of representative nodes that are locally close to the topic node based on all of randomly selected paths is determined. The procedure is then repeated for each representative node, and all topic nodes encountered first and absorbed are found. Here, a topic node in an absorbed state represents means it will be absorbed by the starting node for the path. By repeatedly determining forward and backward absorption, the likelihood of discovering the closest representative nodes for each topic node is maximized. Then, the local influence weight for each topic node is migrated to the local representative nodes based on a probability distribution of closeness between each topic - representative node pair.

To compute the closeness probability distribution for topic - representative node pairs, an association matrix $M(i, j)$ is used. Each entry represents the distance between the current topic node i and the current representative node j. The simplest distance function being $[\mathcal{D}(i, j)+1]^{-1}$. The shorter the distance, the higher the closeness between the topic node and the representative node. As such, the representative j will absorb more local influence from the topic i. After all of the random paths for topic nodes and representative nodes have been processed, the matrix M is normalized as M' where $M'(i, j) = \frac{M(i,j)}{\sum_{j=1}^{\#Rep} M(i,j)}$. Assume there are $\#Rep$ representative nodes and m topic nodes for t. Based on the normalized matrix, the aggregated influence of each representative j is calculated as $\frac{1}{m}\sum_{i=1}^{m} M'(i, j)$ where each topic node is assumed to have a uniform local influence weight $\frac{1}{m}$. It is easy to see that a topic node can be represented by different representative nodes with different probabilities.

Algorithm 8 INFLUENCEMIGRATION$(t, I, V_{r,t})$

```
input: An inverted index $I[R][n]$, a representative node set $V_{r,t}$ for a topic $t$
output: A weighted representative node set $V_{r,t}$ : node $\rightarrow$ weight
1: Topic node set $V_t$ from $V$ for $t$ retrieved from an inverted node index
2: Initialize $|V_t| \times |V_{r,t}|$ adjacency matrix $M \leftarrow \varnothing$
3: for each topic node $v_i \in V_t$ do
4:     Get $R$ random paths from $I[R][v_i]$
5:     for each path $p$ in $I[R][v_i]$ do
6:         if $p$ contains a representative node $v_r \in V_{r,t}$ and $M(v_i, v_r) < \frac{1}{\mathcal{D}(v_i,v_{r,p})+1}$ then
7:             $M(v_i, v_r) \leftarrow \frac{1}{\mathcal{D}(v_i,v_{r,p})+1}$;
8: for each representative node $v_r \in V_{r,t}$ do
9:     Get $R$ random paths from $I[R][v_r]$
10:    for each path $p$ in $I[R][v_r]$ do
11:        if $p$ contains a topic node $v_t \in V_t$ and $M(v_t, v_r) < \frac{1}{\mathcal{D}(v_t,v_{r,p})+1}$ then
12:            $M(v_t, v_r) \leftarrow \frac{1}{\mathcal{D}(v_t,v_{r,p})+1}$
13: Initialize $|V_t| \times |V_{r,t}|$ adjacency matrix $M' \leftarrow \varnothing$
14: for $i \leftarrow 1$ to $|V_t|$ do
15:     for $j \leftarrow 1$ to $|V_{r,t}|$ do
16:         row_weight += $M[i][j]$
17:     for $j \leftarrow 1$ to $|V_{r,t}|$ do
18:         $M'[i][j] \leftarrow M[i][j]/$row_weight
19: for $j \leftarrow 1$ to $|V_{r,t}|$ do
20:     for $i \leftarrow 1$ to $|V_t|$ do
21:         column_weight += $M'[i][j]$
22:     Update $V_{r,t}$ by $v_j \rightarrow$ column_weight $* \frac{1}{|V_t|}$
23: return Updated $V_{r,t}$
```

The procedure for migrating local influence of topic nodes to representative nodes is detailed in Algorithm 8. First, the topic node set V_t is retrieved from the topic-based inverted node index. The nodes in V_t are the same as originally generated in Algorithm 7, and can be reused in an implementation. Then, an adjacency matrix M is used to track absorbing relationships between representative nodes and topic nodes. For sparse graphs, an adjacency list could also be used here. In Line 3-Line 7, the absorbing representative nodes are identified for each topic node based on the R random walks. In Line 8-Line 12, the topic nodes are identified that can be absorbed by a representative node based on the randomly selected R paths of the given representative node. Line 13-Line 18 are normalization of the closeness probability distribution for each topic node relative to its representative nodes. After all the sampled paths are processed, the aggregated score of each representative node is calculated as the overall influence migrated from the locally related topic nodes as shown in Line 19-Line 22. Lastly, the updated and weighted representative node set $V_{r,t}$ is returned as the sampling node set for evaluating the influence of topic t for each user.

4.4 LRW-A Algorithm

Algorithm 9 LRW-A

```
1: // Offline pre-processing
2: Generate topic-based representative nodes $V_{r,t}$ using REPNODES() in Algorithm 7
3: Weight the selected representative nodes using INFLUENCEMIGRATION() in Algorithm 8
4: // Online PIT-Search
5: Generate Top-$k$ PIT List using PERSONALIZED_SEARCH() in Algorithm 10
6: return Top-$k$ PIT List
```

The **LRW-A** algorithm is presented in Algorithm 9, and consists of two stages: The first is an offline stage to pre-compute representative nodes for topics, and assign the weight to each representative node using the absorbing strategy; The second is the online PIT-Search based on the pre-selected representative nodes and query-related topics. Note that the offline pre-processing is updated after a period of time when the social network and topics have changed. It is independent of online queries issued by users. The space cost is dominated by $O(|R|n)$ where $|R|$ is the number of inverted lists, and n is the total number of nodes in the social network G.

5 PERSONALIZED INFLUENCE OF TOPICS

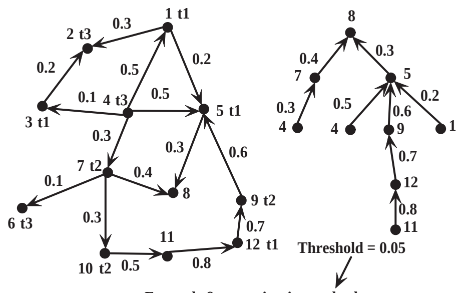
In this section, a materialization algorithm is presented that efficiently computes the personalized top-k topics for each user by evaluating the influence of the representative nodes. To do this, two significant problems must be overcome: (1) Every node must be materialized; and (2) An efficient top-k algorithm that is topic-specific is needed. These two problems are addressed in Subsection 5.1 and Subsection 5.2, respectively.

5.1 Personalized Influence Propagation Index

Different from the general problem of finding the shortest paths in a graph, personalized search of topics only requires nodes “nearby” a given social user node in the social graph. Here, “nearby” means a node that can reach a given social user with a transition probability above a fixed threshold. Since the transition probability of the propagation path decreases as the path distance increases, only the nearby nodes are selected in personalized search. Therefore, shortest paths for all node pairs in the graph are not necessary, and only a small subset of nodes for each starting position must be materialized.

The key idea of the materialization is as follows. For each node v in G, v is initially selected as the root of a backward-based tree, and a reverse breadth-first-search (BFS) is performed. For each node u linked to v, u is inserted as a child of v if the transition probability of the current path (branch) $u \hookrightarrow v$ is above θ. Then, the nodes u' linking to the nodes u are probed. If u' does not appear in the path $u \hookrightarrow v$, and the transition probability of $u' \hookrightarrow u \hookrightarrow v$ is greater than θ, then u' is inserted as a child of u in the tree. If the transition probability of $u' \hookrightarrow u \hookrightarrow v$ is less than θ, then the expansion of the branch is terminated. Note that a node is allowed to appear on different paths (branches) in the tree. Until all possible nodes are fully probed, the tree maintains all of the nodes where each node should have at least one propagation path reaching v, and the propagation probability is greater than θ. For any node in the tree, if it has additional incoming nodes but these incoming nodes can not be included due to a low transition probability, then the node will be marked as a potential node may be expanded in an online search.

Finally, a hash map is created for the node v. Because different occurrences of a node u in the tree represent u can influence v through different paths, the transition probabilities of these paths are aggregated as the propagation value of u to v. We insert u and the propagation value into a lookup table indexed by v. All nodes in the tree are processed in a similar manner. The hash map is the materialized index for servicing personalized search for a starting node v. The set of nodes in the hash map is denoted as $\Gamma(v)$.



For node 8, we maintain a hashmap :

nodes	transition probability
1	0.06
4	0.27
5	0.30
7	0.40
9	0.18
11*	0.10
12	0.13

Fig. 3. Personalized Influence Propagation Index Construction

Consider the example in Figure 3 with 12 nodes and a transition probability on each directed edge. For each graph node, the nearby nodes able to reach a given graph node for a single path traversal when the transition probability is above θ is recorded. Assume that $\theta = 0.05$, and the starting node is Node 8. Based on a reverse breadth-first-search from Node 8, the tree in the second figure in Figure 3 would be produced. By aggregating the influence of different paths for each node, the lookup table in the table in Figure 3 can be generated. Note that the transition probability of each node does not consider the local weight. To compute the final influence for a given node for a topic, the transition probability and the local weight are multiplied. Recall that the local weight can be computed using the methods in Section 3 and Section 4. In this example, only Node 11 will be marked as a potential node to be expanded when refining the top-k personalized influential topics online. This is because Node 4 has no incoming nodes. Although Nodes 1, 5, 7, 9, and 12 have incoming nodes, the nodes are already included in the index. By doing this, unnecessary expansion is avoided, and the overall computation is more efficient.

5.2 Computing Top-k Personalized Influential Topics

Given a set of topics T related to a query q issued by v, the key idea of a top-k PIT-Search algorithm is to first select the top-k topic candidates by probing the materialized node index of v, and the representative node sets of T. After all of the representative nodes appearing near v have been processed completely, some topics can be pruned from T if the topic cannot be in the top-k topic candidates based on the upper bound of the aggregate influence score. If there are still possible topic candidates that may make it into the top-k topic list, then additional neighbor nodes of v must be probed until the top-k topic candidates can no longer be affected.

Algorithm 10 PERSONALIZED_SEARCH(q, v)

```
input: A keyword query $q$ issued by $v$, Index $\Gamma$, Topic Space $T$, and
Topic-aware representative node sets $S$
output: Top-$k$ PIT List $T^k$
 1: Get query-related topics $T_q$ from topic space $T$
 2: Get representative node sets $S_q \leftarrow \{S_1, ..., S_{|T_q|}\}$
 3: $T' \leftarrow T_q$
 4: for each topic $t_i \in T_q$ do
 5:     vInner $\leftarrow S_i \cap \Gamma(v)$;
 6:     // $S_i$ is the representative node set of topic $t_i$ and $\Gamma(v)$ is the
        indexed nearby nodes of $v$, obtained from $v$.hashmap
 7:     for each node $u \in$ vInner do
 8:         influence += $v$.hashmap$(u) \times S_i[u]$
 9:         // $S_i[u]$ is the local weight of $u$ representing the local topic
            nodes to be calculated by methods in Section 3 and Section 4
10:     $W_r[t_i] \leftarrow 1 - S_i[u]$
11:     heap$[t_i] \leftarrow$ influence
12:     UPDATE$(T^k$, heap$)$
13:     $S_i \leftarrow S_i \setminus$ vInner
14: $\Gamma^*(v) \leftarrow \{u^* \in \Gamma(v)\}$
15: // $\Gamma^*(v) \subseteq \Gamma(v)$ is the subset of marked nodes with potential
    capacity to be expanded
16: maxEP $\leftarrow \max\{v.$hashmap$(u^*) | u^* \in \Gamma^*(v)\}$
17: for each topic $t_i \in T_q$ do
18:     if $S_i = \varnothing \vee \min(T^k) \geq W_r[t_i] \times$ maxEP + heap$[t_i]$ then
19:         $T' \leftarrow T' \setminus t_i$
20:         Remove $S_{t'}$ from $S_q$
21: if $T' \setminus T^k \neq \varnothing$ then
22:     EXPAND$(\Gamma^*(v), T', S_q, T^k, W_r$, heap, maxEP$)$
23: return Top-$k$ PIT List $T^k$
```

The procedure of finding the top-k topics is presented in Algorithm 10. At the beginning, the q-related topics T are retrieved and the materialized representative node sets $S = \{S_1, ..., S_{|T|}\}$. Before the topics are processed, a copy topic set T' is created to track the remaining unprocessed topics. In Line 4-Line 13, for each topic $t_i \in T$, the influence of t_i to v is computed if there are representative nodes of t_i occurring nearby (stored in $\Gamma(v)$), and the value is stored as *vInner* $\leftarrow S_i \cap \Gamma(v)$ in Line 5. For each representative node u appearing in $\Gamma(v)$, the influence to v is calculated by multiplying the local weight $S_i[u]$ for t_i and the transition probability propagation to v. A heap maintains the current influence of topics on v.

After each topic t_i has been processed, the visited nodes (*vInner*) are removed from the representative node set S_i. This is because all influence starting from $u \in$ *vInner* to v is already counted during the construction of node influence propagation in Subsection 5.1. The remaining local weight of each topic is recorded in Line 10.

Line 14-Line16 are used to find the marked nodes that may be expanded and the upper bound of the transition probability of the expanded nodes. Then, the topics that cannot be in the top-k are pruned from T' based on the intermediate results as shown in Line 4-Line 13. A topic t_i can definitely be pruned under two conditions: (1) No remaining representative nodes are in S_i; (2) The minimal value $\min(T^k)$ is larger than or equal to the influence upper bound of t_i, where the upper bound value of t_i is the aggregate of the currently computed influence in $heap[t_i]$, and the maximum possible influence of the remaining representative nodes. This can be estimated by $W_r[t_i] \times$ the maximum transition probability propagation of the marked nodes with remaining expansion capacity in the index of v.

Finally, the algorithm terminates when $T' = T^k$. Otherwise, function EXPAND is called to explore nodes further away from v in order to make sure that any remaining topics are considered. The function EXPAND is shown in Algorithm 11, and shares several key variables with Algorithm 10. The main difference is that the termination of the algorithm must be checked when each topic in Line 2-Line 14 is processed. The function UPDATE is easy to implement, and keeps the current top-k topic candidates and influence scores in T^k according to the current heap status.

Algorithm 11 EXPAND($\Gamma^*(v), T', S_q, T^k, W_r,$ *heap*, *maxEP*)

```
1:  for each node $u \in \Gamma^*(v)$ do
2:      for each topic $t' \in T'$ do
3:          $uInner \leftarrow S_{t'} \cap \Gamma(u)$
4:          for each node $x \in uInner$ do
5:              $influence$ += $u.hashmap(x) \times S_{t'}[x]$
6:              $W_r[t'] \leftarrow 1 - S_{t'}[x]$
7:          $heap[t']$ += $influence$
8:          $S_{t'} \leftarrow S_{t'} \setminus uInner$
9:          UPDATE($T^k, heap$)
10:         if $S_{t'} = \varnothing \vee \min(T^k) \geq W_r[t'] \times maxEP + heap[t']$ then
11:             $T' \leftarrow T' \setminus t_i$
12:             Remove $S_{t'}$ from $S_q$;
13:             if $T' \setminus T^k = \varnothing$ then
14:                 break
15:     $\Gamma^*_{new} \leftarrow \Gamma^*_{new} \cup \Gamma^*(u)$;
16:     Record $maxEP$ in $\Gamma^*_{new}$
17: if $T' \setminus T^k \neq \varnothing$ then
18:     EXPAND($\Gamma^*_{new}, T', S_q, T^k, W_r, heap, maxEP$)
```

Using an example social network and index in Figure 3, the procedure of finding personalized influential topics is traced. Assume that Node 8 issues a query related to the three topics t_1, t_2, and t_3, and k is set as 1. The sets $T_q = \{t_1, t_2, t_3\}$ and $S_q = \{S_1 = \{1, 3, 5, 12\}, S_2 = \{7, 9, 10\}, S_3 = \{2, 4, 6\}\}$ are instantiated. Here, $\Gamma(8) = \{1,\ 4,\ 5,\ 7,\ 9,\ 11^*,\ 12\}$. For t_1, *vInner* $= S_1 \cap \Gamma(8) = \{1, 5, 12\}$. For each node in the node set *vInner*, the influence to Node 8 is calculated. If each node in S_1 is assumed to have 0.25 as the local weight, then the aggregated influence of $\{1, 5, 12\}$ relative to Node 8 is $8.hashmap(1) \times 0.25 + 8.hashmap(5) \times 0.25 + 8.hashmap(12) \times 0.25 = 0.055$. The remaining local weight of t_1 is 0.25. Similarly, the influence of t_2 relative to Node 8 is computed as $8.hashmap(7) \times 0.33 + 8.hashmap(9) \times 0.33 = 0.19$. The remaining local weight of t_2 is 0.33. For t_3, the influence to Node 8 is $8.hashmap(4) \times 0.33 = 0.09$. The remaining local weight of t_3 is 0.67. At this stage, the intermediate results are *heap* $= \{t_1 = 0.055; t_2 = 0.19; t_3 = 0.09;\}$, $T^{k=1} = \{t_2\}$, $S_1 = \{3\}$, $S_2 = \{10\}$, $S_3 = \{2, 6\}$, $\Gamma^* = \{11\}$, and *maxEP* $= 0.10$. Based on the intermediate results, the topics that are not able to be the top-1 topic are pruned. Since S_1 and S_3 are not empty, their influence upper bounds are calculated as $W_r[t_i] \times maxEP + heap[t_i]$. For t_1, the upper bound is $W_r[t_1] \times maxEP + heap[t_1] = 0.25 \times 0.1 + 0.055 = 0.08$. Because the upper bound (0.08) of t_1 is less than the current influence value of t_2, t_1 can be pruned from further computations. Similarly, t_3 can be pruned safely since the upper bound is 0.157. Therefore, t_2 is returned as the top-1 personalized influential topic for node 8.

6 EXPERIMENTS

6.1 Experimental Setup

All algorithms are implemented in Java and ran on a 3.0 GHz Intel Pentium 4 machine with 3GB RAM running Windows 7.

Twitter Dataset: We use the large dataset (data$_{3m}$ including 3 million social users) to evaluate the performance of PIT-search, and show the average running time and space cost of building indexes. In this work, we use the dataset originally described by [12] which contains 284 million "following" relationships, 3 million user profiles, and 50 million tweets. The dataset was collected in May 2011 by the FORWARD research group at University of Illinois at Urbana-Champaign.

In addition, we generate synthetic datasets from the larger dataset. Using a similar node degree distribution, three synthetic datasets are produced from the nodes with degree range 51-100, 101-500, and 500-1000, respectively. The three datasets are denoted as data$_{350k}$ for 350k nodes, data$_{1.2m}$ for 1.2m nodes, and data$_{3m}$ including 3m nodes. We also generate a small dataset (data$_{2k}$) with 2000 social users selected from the original dataset using random selection. This is primarily used to compare against the ground-truth method. To ensure each generated dataset is a connected graph, a few synthetic edges among the close nodes across disconnected components are added. The summary of these datasets is shown in Figure 4.

Dateset	Size	Node Degree	Type
data$_{3m}$	3 million	0-695,509	Real
data$_{1.2m}$	1.2 million	101-500	Synthetic
data$_{350k}$	350,000	51-100	Synthetic
data$_{2k}$	2,000	1-500	Synthetic

Fig. 4. Summary of Datasets Used

Topic Generation: In this work, we use a collaborative method to generate a set of topics for each Twitter user. Given a Twitter user, we first treat the posted messages as a document, and apply a simple LDA (Latent Dirichlet Allocation) topic model to the document to generate a bag of terms (normally 16 terms) to be topic seeds of this user. Then, we refine the topic seeds for each user using 53,388 tags in the benchmark dataset released at the 2nd International Workshop on Information Heterogeneity and Fusion in Recommender Systems (HetRec 2011). Each tag was frequently bookmarked by 1,867 web search users. By doing this, we have a reasonable set of topic seeds for each Twitter user. By repeating the above process on each Twitter user, we can produce a large number of meaningful topics. For example, the total size of the generated topic space is 4.8 million where each user has about 200 topics extracted from 450 tweets in the large Twitter data.

Baselines: In order to evaluate the effectiveness and efficiency of the new PIT-search methods (**RCL-A** and **LRW-A**), we use three different baselines.

- **BaseMatrix** is used to produce ground-truth results over a small Twitter dataset. The idea is similar to [15]. For each q-related topic, the influence is propagated to the social users through a number of matrix multiplication iterations (set to 6 in this work). As such, the influence of the topic on the query user can be aggregated at the end of the matrix multiplication. The propagation process is repeated for each q-related topic. After that, the top-k q-related topics can be determined by comparing the aggregated values of different topics on the query user.
- **BaseDijkstra** first computes the shortest path from each topic node to the query user using Dijkstra's algorithm [4], and then replaces a sub-path in the shortest path with an alternative path that can connect the two end points of the sub-path. By repeating the replacement operation, we can generate a number of distinct paths from the topic node to the query user node.
- **BasePropagation** is a heuristic method, which is used to process the large Twitter dataset because **BaseMatrix** is too space inefficient on the larger dataset, requiring 120GB of RAM, and **BaseDijkstra** is too computationally expensive. The basic idea of **BasePropagation** is to calculate the propagation influence of each topic node for a given user using only the personalized influence propagation index described in Section 5.1.

6.2 PIT-Search Efficiency

To test PIT-Search performance, we select 100 tags to represent a user's keyword queries. Each tag would produce 500+ topics for the Twitter dataset in the topic generation process. Then, we randomly select an additional 49 users, but keep the 100 sampled keyword queries unchanged. The average over all of the runs are used to fairly assess the PIT-Search performance.

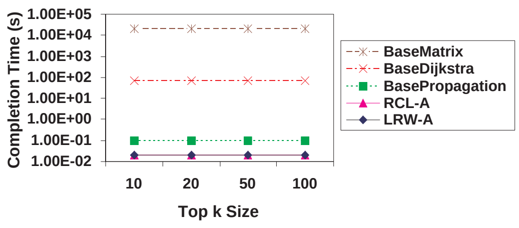


Fig. 5. Time Cost of PIT-Search using Data$_{2k}$

Figure 5 shows the time cost for the methods **BaseMatrix**, **BaseDijkstra**, **BasePropagation**, **RCL-A**, and **LRW-A** over the small Twitter dataset when the $k = 10$, 20, 50, and 100. The performance gap between each approach is quite evident. For instance, **BaseMatrix** and **BaseDijkstra** take around 5 hours and 1 minute, respectively. This is because **BaseMatrix** performs many matrix multiplications in a sparse array when computing the influence for all the possible paths from topic nodes to the user node. **BaseDijkstra** spends the majority of the processing time computing the shortest paths. Here, the length of iterations is set as 6 in **BaseMatrix**. Obviously, the high-latency is not acceptable in an online search. However, **BasePropagation** only requires 100 ms to return the top-k personalized influential topics. This is because **BasePropagation** uses a materialized index to obtain the influence from a topic node to a query user with no further on-the-fly path computations. **RCL-A** and **LRW-A** are the fastest approaches, requiring only 20 ms to determine the top-k. In addition, our experiments also show that all these methods consume reasonable memory space when processing the small dataset, i.e., **BaseMatrix** required up to 129MB, and the other methods used up to 93MB. In further timing experiments, **BaseMatrix** is omitted since the performance is so poor. In this small dataset, varying k does not change the time cost for any of the methods. This is because **BasePropagation**, **RCL-A** and **LRW-A** have to access most of the nodes to identify the top-k results. **BaseMatrix** must perform matrix multiplications in all L iterations. The time cost of **BaseDijkstra** is dominated by the cost of computing the shortest paths. Therefore, the running time is insensitive to k in the small dataset.

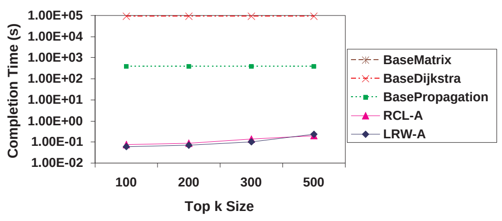


Fig. 6. Time Cost of PIT-Search using Data$_{3m}$

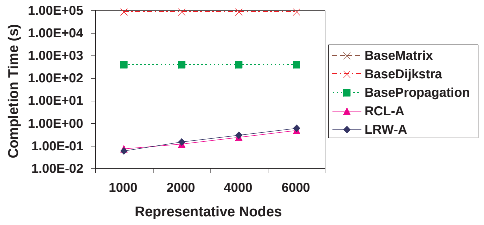


Fig. 7. Time Cost of PIT-Search for the Top-100 Topics using a different number of Representative Nodes and the Data$_{3m}$ collection.

For a selected query, the large Twitter dataset has around 3000 q-related topics on average, and each q-related topic has $20{,}000$ topic nodes. So, we maintain 1000 representative nodes for each topic in the materialized index. Figure 6 shows the average processing time of the proposed methods when $k = 100$, 200, 300, and 500. **BaseDijkstra** requires around 25 hours to complete the task, **BasePropagation** needs 6.6 minutes, and **RLC-A** / **LRW-A** use only 230 ms. From Figure 6, we observe that the average query time of **RLC-A** and **LRW-A** grow very slowly as k increases. The low-latency is still promising for online PIT-search.

From the experimental result, the algorithms are insensitive to k because personalized search can prune the low quality topics and filter the expanded nodes at the same time based on the intermediate results.

Figure 7 compares the completion time when we vary the materialized sizes of representative nodes for each topic. Since **BaseDijkstra** and **BasePropagation** have to evaluate the influence of each topic node relative to the query user, they are not affected by the change of representative node set size. They have the same time cost as that in Figure 6. However, the performance of both **RCL-A** and **LRW-A** improve when the number of representative nodes varies. For instance, a PIT-search can be stopped in 70 ms if we materialize 1000 representative nodes for each topic. But if we maintain 6000 representative nodes for each topic, then both **RCL-A** and **LRW-A** methods require 600 ms before the PIT-search terminates.

6.3 Scalability of PIT-Search

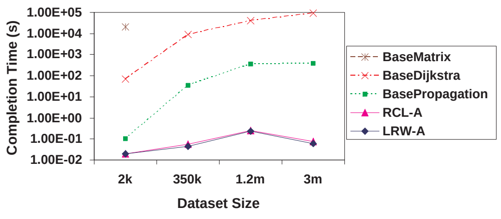


Fig. 8. Scalability of PIT-Search for the Top-100 Topics over all datasets using 1000 sampled representative nodes for each user.

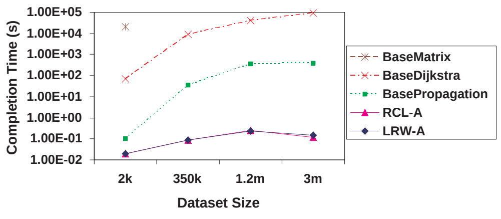


Fig. 9. Scalability of PIT-Search for the Top-100 Topics over all datasets using 2000 sampled representative nodes for each user.

In order to evaluate the scalability of PIT-Search, we assessed the average time cost for each method over all 4 datasets. Figure 8 shows the results when $k = 100$ and the number of selected representative nodes is 1000. Figure 9 shows what happens when we change the number of representative nodes to 2000. From the two figures, we can see that **RCL-A** and **LRW-A** are not sensitive to the change of dataset size, while the other approaches become even more inefficient as the collection size grows. From the comparison of this two figures, we can observe that change from 1000 representative nodes to 2000 representative nodes does not incur a noticeable performance decrease. The comparison also shows that the efficiency of PIT-Search over data$_{3m}$ is better than that of data$_{1.2m}$. This is because the average node degree of data$_{1.2m}$ is much larger than in data$_{3m}$. So, the expansion operations incurred by higher edge degrees in the graph can result in a measurable performance degradation.

6.4 PIT-Search Effectiveness

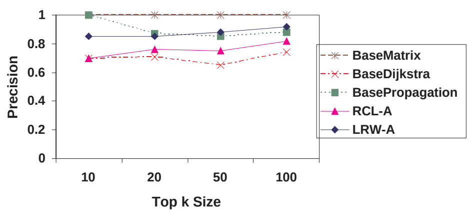


Fig. 10. Effectiveness of PIT-Search on Data$_{2k}$

To measure the effectiveness of the PIT-search approximation methods, we treat the generated results from **BaseMatrix** as the ground-truth for small Twitter dataset since it exhaustively processes all paths. **BaseMatrix** selects the personalized influential topics based on the exact influence computation for topics. For simplicity, we consider only Precision in Figure 10. By comparing the topics returned with **BaseMatrix**, we can see that **BaseDijkstra** has the lowest precision, followed by **RCL-A**. The precision is around 0.7. From Figure 10, we can see that **BasePropagation** and **LRW-A** have similar precision – around 0.85 when the k is between 20–100. **BasePropagation** can achieve nearly same result set as **BaseMatrix** when the $k = 10$. This is mainly because **BasePropagation** is also an exact-computation method using a personalized influence index to improve efficiency. In contrast to **BaseMatrix**, **BasePropagation** may mis-appropriate topic node influence when probing expanded topic nodes.

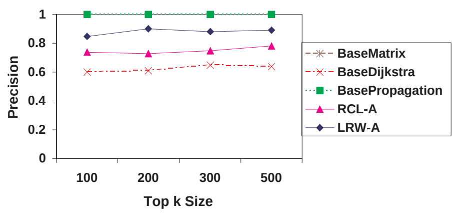


Fig. 11. Effectiveness of PIT-Search on Data$_{3m}$

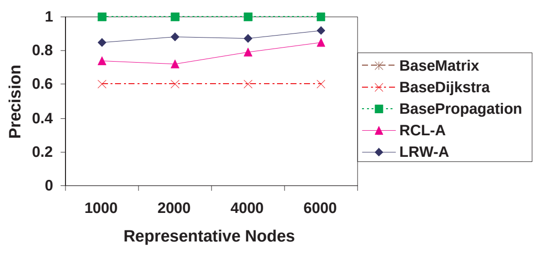


Fig. 12. Effectiveness of varying the number of Representative Nodes on Data$_{3m}$ and $k = 100$

Since it is not feasible to use **BaseMatrix** on the larger dataset, we compare the approximation algorithms to **BasePropagation** in Figure 11 instead. Figure 12 shows the impact of varying the number of representative nodes. **BaseDijkstra** has the lowest precision, followed by **RCL-A** as seen previously. **LRW-A** can achieve the best precision – above 0.8. The precision of **RCL-A** can be further improved by increasing the size of the materialized representative node set. Increasing to 0.82 for 6000 representative nodes in Figure 12. However, **LRW-A** increased precision comes at a performance cost as discussed previously. This observation corroborates our claim that the effectiveness of **LRW-A** at selecting representative nodes is better than **RCL-A**. If we select enough quality representative nodes for a topic with **LRW-A**, then simply increasing the number of representative nodes cannot bring much more benefit to **LRW-A** when assessing a topic's influence.

6.5 PIT-Search Space Cost

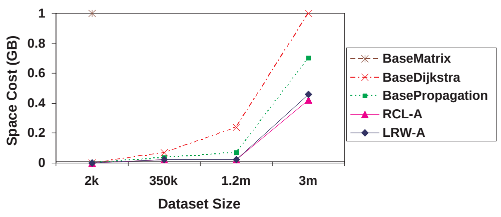


Fig. 13. Space cost when searching with $k = 100$ on all datasets with 1000 sampled representative nodes.

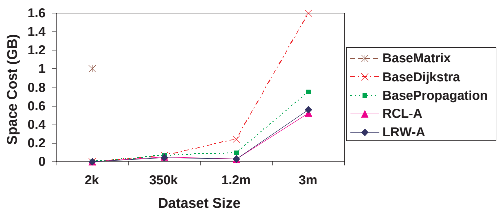


Fig. 14. Space cost when searching with $k = 100$ over all datasets with 2000 sampled representative nodes.

In Figure 13 and Figure 14, we demonstrate the space cost when searching for the top 100 influential topics over all datasets. Here, each topic's influence for a query user is calculated using 1000 or 2000 pre-computed representative nodes. From the experimental results, we can see that **BaseMatrix** consumes significant space. All other methods have reasonable space usage during a PIT-Search. In addition, we can find that the space cost of **RCL-A** and **LRW-A** processing queries over the data$_{3m}$ collection increases quickly because the dataset contains a large number of query-related topics. Since there are many possible matches, a large amount of space is consumed when loading the representative nodes at the beginning of Algorithm 10. But the space cost is still less than **BaseDijkstra** and **BasePropagation** because **BaseDijkstra** has to maintain many intermediate results when computing the shortest paths, and **BasePropagation** needs to retrieve all topic nodes into the memory at the beginning of each query evaluation.

6.6 Index Construction

To demonstrate that the materializing cost for the topic-to-representative node index is reasonable, we report the time and space cost when we vary the sample rate of **RCL-A**, the size R of L-length random walks in **LRW-A**, and L for both.

$\frac{\lvert V'\rvert}{\lvert V\rvert}$ in **RCL-A**	1%	5%	10%
Time Cost (s)	450	540	560
Space Cost (GB)	2	2	2

R in **LRW-A**	100	200	300
Time Cost (s)	14	14	14
Space Cost (GB)	2	3	4

Fig. 15. Effect of Sample Rate on PIT-Search approximation algorithms

Given a topic, Figure 15 provides the average time and space cost as sample rate varies. For instance, when we sample different numbers of nodes when grouping the topic nodes, the time of selecting central nodes in **RCL-A** does not exhibit any noticeable change. The main reason is that **RCL-A** spends most of the time on the central node computation, and not on the node pair grouping determination. Therefore, increasing the sample ratio does not affect the pre-computation time. In all three cases, the space cost jumps up to 2GB, which is mainly dominated by the space required to load nodes and the L-length node set. When compared with **RCL-A**, **LRW-A** needs more space as R increases. However, R is often a reasonably small value (200) in practice. This is because the average degree is 76 in the large Twitter data graph. The advantage of **LRW-A** is that it only takes 14 seconds to identify the representative node set for a topic, and the effect of R to the running time is negligible. This is because most of the processing time of **LRW-A** is spent on the process of computing ranking score for every node in the graph in order to select representative nodes for a topic. Once the representative nodes are selected, the absorbing procedure can be quickly finished. Here, we do not include the time cost of constructing the L-length random walk index (Algorithm 6 discussed in Section 4.1) since the sampling index only needs to run one time for a dataset for both algorithms. In our experiments, building the L-length random walk index required around seven hours and 3GB of space when $R = 200$. Since it is only ran once, this cost is amortized.

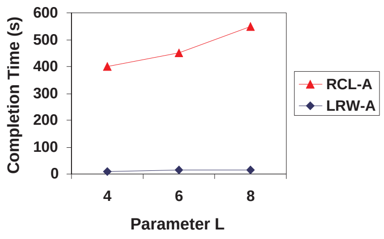


Fig. 16. Index construction time for Data$_{3m}$

Figure 16 shows the time required to compute the central nodes or representative nodes as L varies for **RCL-A** and **LRW-A**. As L increases, **RCL-A** requires more time to compute the central nodes. The main reason is that large L values may lead to large group sizes. Computing the central node of a large group is much more expensive than for many small groups because the large group may bring in many non-topic nodes into the selection phase. Different from **RCL-A**, the processing time of **LRW-A** changes much less than in **RCL-A**. Therefore, **LRW-A** is the preferred approach for materializing the topic-to-representative node index.

7 Related Work

A significant amount of prior work exists in the study of influence diffusion in social networks. For example, Richardson and Domingos [22] and Kempe et al. [8] formally defined the problem of influence maximization as finding a small subset of nodes in a social network that can maximize the spread of influence based on the independent cascade (IC) propagation model. The problem is further studied by Chen et al. [3] who consider the degree discount during the seed selection process. Goyal et al. [5] extended the influence maximization problem by deriving influence propagation with time decay using action-based traces. Zhuang et al. [29] addressed the problem of maximizing influence diffusion when the social networks are updated frequently. Guo et al. [6] proposed and studied the problem of personalized influence maximization. The problem is defined as follows: Given a target user, find a small subset of nodes which can maximize the influence spread to the given target user in a social network. However, all the above work can neither be applied to select representative users from a social network with regards to a topic, nor to PIT-search problem.

Several approaches to personalization in social networks exist. A re-ranking method was presented by Noll and Meinel [17] based on user tag profiles which are derived from a user's *del.icio.us* bookmarks. The tags of each search result on the site are matched against a user profile. The authors go on to investigate how accurate user profiles can be generated from *del.icio.us* data. Similarly, Carmel et al. [2], Vosecky et al. [25], and Qian et al. [21] developed personalized social search methods based on user profiles and topics of interest. By comparing the search results against the user profiles and topic of interest, the most relevant results can be retrieved. The limitation of these approaches is that personalization prefers results matching with a user profile or query log. However, users are often reluctant to provide private information as it can be used expose and harvest personal details. Li et al. [14] is the most recent work to study the problem of personalized search over social networks. A search returns a user's posts, and not matched topics as in this work. Li et al. used the shortest distance to simulate the social relevance between users rather than influence between users as described in this work. PIT-search focuses on topic-based inter-influence among social users over social networks. In other words, if a search topic is hotly discussed by influential connections, then this search topic will be highly recommended to the user even if it does not match the user's profile, query logs, or shortest distance.

8 Conclusions

Personalized influential topic search in social networks is an increasingly important problem. In this paper, we proposed the problem of personalized influential topic search (PIT-Search) in social networks. To make the PIT-Search more effective and efficient, we then developed two approximation approaches capable of selecting representative users as a social summarization for a given topic. We have also designed and presented a personalized influence propagation index and a top-k PIT-Search algorithm. Our experimental evaluation has verified the effectiveness of the approximate approaches and the efficiency of the top-k index.

9 Acknowledgments

This work was supported by the ARC Discovery Projects under Grant No. DP120102627 and DP140101587, and partially supported by DP160102114. Shane Culpepper is the recipient of an Australian Research Council DECRA Research Fellowship (DE140100275).

References

[1] P. K. Atrey, M. S. Kankanhalli, and B. J. Oommen. Goal-oriented optimal subset selection of correlated multimedia streams. *TOMCCAP*, 3(1), 2007.

[2] D. Carmel, N. Zwerdling, I. Guy, S. Ofek-Koifman, N. Har'El, I. Ronen, E. Uziel, S. Yogev, and S. Chernov. Personalized social search based on the user's social network. In *CIKM*, pages 1227–1236, 2009.

[3] W. Chen, Y. Wang, and S. Yang. Efficient influence maximization in social networks. In *KDD*, pages 199–208, 2009.

[4] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[5] A. Goyal, F. Bonchi, and L. V. S. Lakshmanan. A data-based approach to social influence maximization. *PVLDB*, 5(1): 73–84, 2011.

[6] J. Guo, P. Zhang, C. Zhou, Y. Cao, and L. Guo. Personalized influence maximization on social networks. In *CIKM*, pages 199–208, 2013.

[7] H. G. Kemeny and J. L. Snell. Ch. 3: Absorbing markov chains. In *Finite Markov Chains (Second ed.)*, page 224. New York Berline Heidelberg Tokyo:Springer-Verlag, 1976.

[8] D. Kempe, J. M. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *KDD*, pages 137–146, 2003.

[9] J. Kim, S. Kim, and H. Yu. Scalable and parallelizable processing of influence maximization for large-scale social networks? In *ICDE*, pages 266–277, 2013.

[10] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. M. VanBriesen, and N. S. Glance. Cost-effective outbreak detection in networks. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Jose, California, USA, August 12-15, 2007*, pages 420–429, 2007.

[11] H. Li, S. S. Bhowmick, A. Sun, and J. Cui. Conformity-aware influence maximization in online social networks. *VLDB Journal*, 24(1):117–141, 2015.

[12] R. Li, S. Wang, H. Deng, R. Wang, and K. C.-C. Chang. Towards social user profiling: unified and discriminative influence model for inferring home locations. In *KDD*, pages 1023–1031, 2012.

[13] R. Li, J. X. Yu, X. Huang, and H. Cheng. Random-walk domination in large graphs. In *ICDE*, pages 736–747, 2014.

[14] Y. Li, Z. Bao, G. Li, and K. Tan. Real time personalized search on social networks. In *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015*, pages 639–650, 2015.

[15] L. Liu, J. Tang, J. Han, M. Jiang, and S. Yang. Mining topic-level influence in heterogeneous networks. In *CIKM*, pages 199–208, 2010.

[16] Q. Mei, J. Guo, and D. R. Radev. Divrank: the interplay of prestige and diversity in information networks. In *KDD*, pages 1009–1018, 2010.

[17] M. G. Noll and C. Meinel. Web search personalization via social bookmarking and tagging. In *ISWC*, pages 367–380, 2007.

[18] P. W. Olsen, A. G. Labouseur, and J. Hwang. Efficient top-k closeness centrality search. In *ICDE*, pages 196–207, 2014.

[19] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999.

[20] R. Pemantle. Vertex-reinforced random walk. *Probability Theory and Related Fields*, 92(1), 1992.

[21] X. Qian, H. Feng, G. Zhao, and T. Mei. Personalized recommendation combining user interest and social circle. *IEEE Trans. Knowl. Data Eng.*, 26(7):1763–1777, 2014.

[22] M. Richardson and P. Domingos. Mining knowledge-sharing sites for viral marketing. In *KDD*, pages 61–70, 2002.

[23] R. Rymon. Search through systematic set enumeration. In *KR*, pages 539–550, 1992.

[24] T. Takahashi, R. Tomioka, and K. Yamanishi. Discovering emerging topics in social streams via link-anomaly detection. *IEEE Trans. Knowl. Data Eng.*, 26(1):120–130, 2014.

[25] J. Vosecky, K. W. Leung, and W. Ng. Collaborative personalized twitter search with topic-language models. In *SIGIR*, pages 53–62, 2014.

[26] W. Webber. Evaluating the effectiveness of keyword search. *IEEE Data Eng. Bull.*, 33(1):54–59, 2010. URL http://sites.computer.org/debull/A10mar/webber-paper.pdf.

[27] H. C. Wu, R. W. P. Luk, K. Wong, and K. Kwok. Interpreting TF-IDF term weights as making relevance decisions. *ACM Trans. Inf. Syst.*, 26(3), 2008. doi: 10.1145/1361684.1361686. URL http://doi.acm.org/10.1145/1361684.1361686.

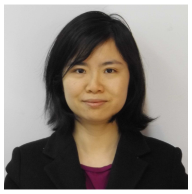
[28] H. Yin, B. Cui, H. Lu, Y. Huang, and J. Yao. A unified model for stable and temporal topic detection from social media data. In *ICDE*, pages 661–672, 2013.

[29] H. Zhuang, Y. Sun, J. Tang, J. Zhang, and X. Sun. Influence maximization in dynamic social networks. In *ICDM*, pages 1313–1318, 2013.

Jianxin Li received his BE and ME degrees in computer science, from the Northeastern University, China, in 2002 and 2005, respectively. He received his PhD degree in computer science, from the Swinburne University of Technology, Australia, in 2009. He is currently a lecturer in the School of Computer Science and Information Technology, RMIT. His research interests include database query processing & optimization, and social network analytics.

Chengfei Liu received the BS, MS and PhD degrees in Computer Science from Nanjing University, China in 1983, 1985 and 1988, respectively. Currently he is a Professor in the Faculty of Science, Engineering and Technology, Swinburne University of Technology. His current research interests include keywords search on structured data, query processing and refinement for advanced database applications, query processing on uncertain data and big data, and data-centric workflows. He is a member of IEEE, and a member of ACM.

Jeffrey Xu Yu received the BE, ME, and PhD degrees in computer science, from the University of Tsukuba, Japan, in 1985, 1987, and 1990, respectively. Currently he is a Professor in the Department of Systems Engineering and Engineering Management, The Chinese University of Hong Kong. His major research interests include graph mining, graph database, social networks, keyword search, and query processing and optimization. He is a senior member of IEEE, a member of the IEEE Computer Society, and a member of ACM.



Yi Chen is an Associate Professor and Henry J. Leir Chair in the School of Management, with a joint appointment with the College of Computing Sciences, New Jersey Institute of Technology. Her research interests focus on data management technologies and their applications in healthcare, business and Web, such as information search, recommendation systems, social computing, and workflow management. She was a general chair of SIGMOD'2012, and is serving an associate editor for PVLDB, DAPD, and ECRA. Yi Chen is a recipient of Peter Chen Big Data Young Researcher Award, a Google Research Award, IBM Faculty Award and an NSF CAREER Award. She received her Ph.D. from the University of Pennsylvania and her B.S. from Central South University in 2005 and 1999, respectively.

Timos Sellis received the PhD degree in computer science from the University of California, Berkeley, in 1986. Till the end of 2012 he was the Director of the Institute for the Management of Information Systems (IMIS) and a Professor at the Nat. Techn. Univ. of Athens, Greece. He is currently a Professor in the Faculty of Science, Engineering and Technology, Swinburne University of Technology. His research interests include big data, data streams, personalization, data integration, and spatio- temporal database systems. He is an IEEE Fellow and an ACM Fellow.

J. Shane Culpepper completed a PhD at The University of Melbourne in 2008. Since then he has been a faculty member at RMIT University, with research interests in designing efficient algorithms and data structures for a wide variety of information storage and retrieval problems.