Real Time Personalized Search on Social Networks

Yuchen Li #, Zhifeng Bao *, Guoliang Li +, Kian-Lee Tan #
#*National University of Singapore* **University of Tasmania* +*Tsinghua University*
{liyuchen, tankl}@comp.nus.edu.sg baoz@utas.edu.au liguoliang@tsinghua.edu.cn

***Abstract*—Internet users are shifting from searching on traditional media to social network platforms (SNPs) to retrieve up-to-date and valuable information. SNPs have two unique characteristics: frequent content update and small world phenomenon. However, existing works are not able to support these two features simultaneously. To address this problem, we develop a general framework to enable real time personalized top-k query. Our framework is based on a general ranking function that incorporates time freshness, social relevance and textual similarity. To ensure efficient update and query processing, there are two key challenges. The first is to design an index structure that is update-friendly while supporting instant query processing. The second is to efficiently compute the social relevance in a complex graph. To address these challenges, we first design a novel 3D cube inverted index to support efficient pruning on the three dimensions simultaneously. Then we devise a cube based threshold algorithm to retrieve the top-k results, and propose several pruning techniques to optimize the social distance computation, whose cost dominates the query processing. Furthermore, we optimize the 3D index via a hierarchical partition method to enhance our pruning on the social dimension. Extensive experimental results on two real world large datasets demonstrate the efficiency and the robustness of our proposed solution.**

I. INTRODUCTION

With the rise of online social networks, Internet users are shifting from searching on traditional media to social network platforms (SNPs) to retrieve up-to-date and valuable information [1]. For example, one may search on Twitter to see the latest news and comments on Malaysia Airlines flight MH17. However, searching SNP data is rather challenging due to its unique properties: not only containing user-generated contents from time to time, but also having a complex graph structure. In particular, there are two distinguishing characteristics:

- **High Update Rate**: Twitter has over 288M active users with 400 million posts per day in 2013[1].
- **Small World Phenomenon**: People in the social network are not very far away from each other. For Facebook, the average degrees of separation between users are 4.74[2]. Moreover, unlike traditional road networks where each vertex has a small degree, the vertex degree of a social network follows the power law distribution [2].

Top-k keyword search is an important tool for users to consume Web data. However, existing works on keyword search over social networks [3], [4], [5], [6], [7] usually ignore one or many of the above characteristics of SNPs and lead to several drawbacks, e.g. returning user with meaningless or outdated search results, low query performance, providing global major trends but not personalized search results which may easily result in biased views [8]. Thus, it calls for a new and general framework to provide real time personalized search over the social network data that leverages all unique characteristics of SNPs.

As a preliminary effort to support the above two characteristics, we focus on three most important dimensions on SNPs: *Time Freshness*, *Social Relevance* and *Text Similarity*. *Time Freshness* is essential as outdated information means nothing to user in a highly dynamic SNP. *Social Relevance* must be adopted in the search framework as well. Leveraging the personal network to rank results will greatly improve the search experience as people tend to trust those who are "closer" and will also enable more user-interactions. E.g., a basketball fan is more likely to respond to a post on "NBA Finals" posted by a friend than unfamiliar strangers. Lastly, *Text Similarity* is the fundamental dimension where keywords are used to distinguish the results from available records.

There are many challenges to support the search over these three dimensions. First, it is challenging to design an index structure that is update-friendly while supporting powerful pruning for instant query processing. Specifically, when a new record is posted, it must be made available immediately in the search index rather than being periodically loaded. The second challenge lies in the query evaluation based on the index. In particular, how to enable an efficient computation along the social dimension, whose performance dominates its counterparts on the other two dimensions. The social distance is usually modeled as the shortest distance on the social graph [9], [10], [6], [5], [7]. These solutions either compute the distance on-the-fly [7] or pre-compute all-pairs distances [5]. The first is extremely inefficient for large networks, which renders it unacceptable for real time response, while the latter requires prohibitively large storage. A natural way is to develop query processing techniques based on index with reasonable size. However, existing distance indices are unable to handle massive social network because they neglect at least one of the unique characteristics of SNPs.

To address these challenges, we present a novel solution to support real time personalized top-k query on SNPs. The contributions of this paper are summarized as follows:

- We present a 3D cube inverted index to support efficient pruning on the three dimensions (time, social, textual) simultaneously. Such index is update-efficient, while flexible in size w.r.t. different system preferences.
- We design a general ranking function that caters to various user preferences on the three dimensions, and

[1]http://www.statisticbrain.com/twitter-statistics/
[2]http://www.facebook.com/DegreesApp/

devise a cube threshold algorithm (CubeTA) to retrieve the top-k results in the 3D index by this ranking function.
- We propose several pruning techniques to accelerate the social distance computation, and a single distance query can be answered with an average of less than $1\mu s$ for a graph with $10M$ vertices and over $230M$ edges.
- We optimize the 3D index via a hierarchical partition method to enhance the pruning on the social dimension. A deeper partition tree will lead to better query performance and the flexibility of the index allows the space occupied to be the same as the basic 3D index.
- We conduct extensive experimental studies on two real-world large datasets: Twitter and Memetracker. Our proposed solution outperforms the two baselines with average 4-8x speedups for most of the experiments.

Our framework has a general design of ranking function, index and search algorithm. It enables opportunities to support personalized-only (regardless results' freshness) and real-time-only (regardless of social locality) search. It is also a good complement to the global search provided by existing SNPs. Our vision is: search results are not the end of story, instead they will be fed into data analytic modules, e.g. sentiment analysis, to support real-time social network analysis ultimately.

We present related work in Sec. II and problem definition in Sec. III. We propose the 3D inverted index in Sec. IV and the CubeTA in Sec. V. In Sec. VI we present several pruning methods to speed up the distance query evaluation. We propose a hierarchical partition scheme for our 3D inverted index to optimize CubeTA in Sec. VII, and report experiment results in Sec. VIII. Finally we conclude in Sec. IX.

II. Related Work

Search on Social Media Platform. Facebook developed Unicorn [11] to handle large-scale query processing. While it supports socially related search, the feature is only available for predefined entities rather than for arbitrary documents. Moreover, Unicorn is built on list operators like *LISTAND* and *LISTOR* to merge search results. This can be very costly especially for real time search because the search space is huge. Twitter's real time query engine, Earlybird [3], has also been reported to offer high throughput query evaluation for fast rate of incoming tweets. Unfortunately, it fails to consider social relationship. Therefore, our proposed method can complement existing engines by efficiently handling real time search with social relevance.

Search on Social Network. Several research works have been proposed for real time search indices over SNPs. Chen et al. introduced a partial index named TI to enable instant keyword search for Twitter [4]. Yao et al. devised an index to search for the microblog messages which are ranked by their provenance in the network [12]. However, none of them offers customized search for the query user. Although indices on social tagging network offer the social relevance feature [5], [6], [7], they rely on static inverted lists that sort documents by keyword frequencies to perform efficient pruning. These indices cannot handle high rate of documents ingestion because maintaining sorted lists w.r.t keyword frequencies requires huge number of random I/Os. Besides, [6] only considers the documents posted from a user's direct friends while we consider all documents in SNPs. Thus, there is a need to design novel indices that are update-efficient and support efficient search with social relevance feature.

Distance Indices on Social Networks. Road network distance query has been well studied in [13], [14]. However, they cannot work on large social networks because the vertex degree in road networks is generally constant but dynamic in social networks due to the **power law** property. Existing distance indices for social networks [15], [16], [17], [18] cannot be applied to our scenario for several reasons. First, the schemes in [15], [16], [17] assume un-weighted graphs and are not able to handle weighted graphs. Second, the mechanisms in [15], [16] are only efficient for social graphs with low tree-width property. Unfortunately, as reported in [19], the low tree width property does not hold in real world social graphs. We also made this observation in our real datasets. Lastly, personalized top-k query requires one-to-many distance query whereas the methods in [17], [16] are applicable only for one-to-one query and the solution in [18] only supports one-to-all query. It is hard to extend these schemes to fit our case. It therefore motivates us to design pruning methods to overcome the distance query problem on large social networks.

III. Problem Definition

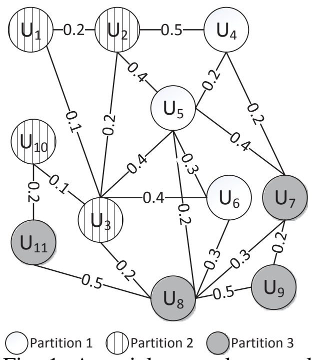
Data Model. Let $G = (V, E)$ be an undirected social graph, where V is the set of vertices representing users in the network and E is the set of edges in G denoting social links. For each vertex v, R_v is a set of records attached to v (e.g., microblogs published by the user). A record $r \in R_v$ is formulated as $r = \langle r.v, r.W, r.t\rangle$ where:
- $r.v$ shows which vertex this record belongs to.
- $r.W$ is a set of keywords contained in r.
- $r.t$ denotes the time when the record is published by $r.v$.

Each edge is associated with a weight, which quantifies the social distance between the two vertices. In this paper, we adopt the *Jaccard Distance* which has been widely used [20], [21], [2]; however, our method can be easily extended to other functions. The weight between user v and its neighbor v', denoted as $D(v, v')$, is the *Jaccard Distance* between their neighbor sets $N(v)$ and $N(v')$ where $N(x) = \{n|(x, n) \in E\}$.

$$D(v, v') = 1 - \frac{|N(v) \cap N(v')|}{|N(v) \cup N(v')|} \; s.t.(v, v') \in E \qquad (1)$$

Query Model. A top-k query q on the social graph G is represented as a vector $q = \langle q.v, q.W, q.t, q.k\rangle$ where:
- $q.v$ is the query user.
- $q.W$ is the set of query keywords.
- $q.t$ is the time when the query is submitted.
- $q.k$ is the number of desired output records.

Ranking Model. Given a query q, our objective is to find a set of $q.k$ records with the highest relevance. To quantify the relevance between each record and the query, we should consider the following aspects.



rid:user	**TS**	Keywords
$r0$:u_{10}	0.1	(“icde”,0.8),(“nus”,0.1)
$r1$:u_1	0.1	(“icde”,0.9)
$r2$:u_7	0.1	(“icde”,0.1),(“nus”,0.5)
$r3$:u_2	0.2	(“icde”,0.6),(“nus”,0.2)
$r4$:u_3	0.3	(“icde”,0.7),(“nus”,0.2)
$r5$:u_5	0.4	(“icde”,0.4)
$r6$:u_4	0.6	(“icde”,0.8)
$r7$:u_7	0.7	(“icde”,0.8),(“nus”,0.1)
$r8$:u_2	0.7	(“icde”,0.7)
$r9$:u_5	0.8	(“icde”,0.2),(“nus”,0.2)
$r10$:u_9	0.9	(“icde”,0.1),(“nus”,0.4)
$r11$:u_{11}	1.0	(“icde”,0.7)

Fig. 1: A social network example including records posted. Records are ordered by time from old to recent (used throughout this paper).

(1) *Social Relevance*: The social distance for two vertices $v \leftrightarrow v'$ is adopted as the shortest distance [9], [10], [6], [5].

$$\mathbf{SD}(v,v') = \min_{\text{path} v=v_0 \ldots v_k=v'} \sum_{i=0..k-1} D(v_i, v_{i+1})/maxDist \quad (2)$$

The social relevance is computed as $\mathbf{SR}=\max(0, 1-\mathbf{SD})$ and $maxDist$ is the user-tolerable maximal distance.

(2) *Textual Relevance*: We adopt the well-known tf-idf based approach [22]. Let $tf_{w,r}$ denote the frequency of keyword w in r whereas idf_w is the inverse frequency of w in the entire document collection. We represent textual relevance as a cosine similarity between $q.W$ and $r.W$:

$$\mathbf{TS}(q.W, r.W) = \sum_{w \in q.W} tf_{w,r} \cdot idf_w \quad (3)$$

Specifically, $tf_{w,r} = z_{w,r}/(\sum_{w \in r.W} z_{w,r}^2)^{\frac{1}{2}}$ where $z_{w,r}$ is the number of occurrences of w in $r.W$; and $idf_w = z_w/(\sum_{w \in q.W} z_w^2)^{\frac{1}{2}}$, $z_w = ln(1 + |R|/df_w)$ where $|R|$ is the total number of records posted and df_w gives the number of records that contain w.

(3) *Time Relevance*: The time freshness score **TF** is the normalized time difference between $q.t$ and $r.t$. In particular, let t_{min} be the pre-defined oldest system time, then

$$\mathbf{TF}(q.t, r.t) = \frac{r.t - t_{min}}{q.t - t_{min}} \quad (4)$$

Overall Ranking Function. Now, with the social, textual and time relevances normalized to [0,1], our overall ranking function is a linear combination of these three components.

$$\Re(q,r) = \alpha \mathbf{TS}(q.W, r.W) + \beta \mathbf{SR}(q.v, r.v) + \gamma \mathbf{TF}(q.t, r.t) \quad (5)$$

where α, β, γ are user preference parameters for general weighting functions, and $\alpha, \beta, \gamma \in [0, 1]$. Tuning the parameters is a well-studied problem in information retrieval and a widely-adopted solution is by user click-throughs [23]. In this paper we focus on devising efficient algorithms for query processing and leave the parameter tuning as a future work.

Example 1. *Fig. 1 is an example social network where all records posted are listed. For each record* r, *its user id (UID), time score (**TS**), keywords and their frequencies are included. Suppose* α=β=γ=1 *and* u_1 *expects to get the top-1 record that contains "icde". By Equation 5,* $r11$ *is the desired result as* $\Re(q_{u_1}, r11) = 1.0 + (1.0 - 0.4) + 0.7 = 2.3$ *has the maximum value among all records.*

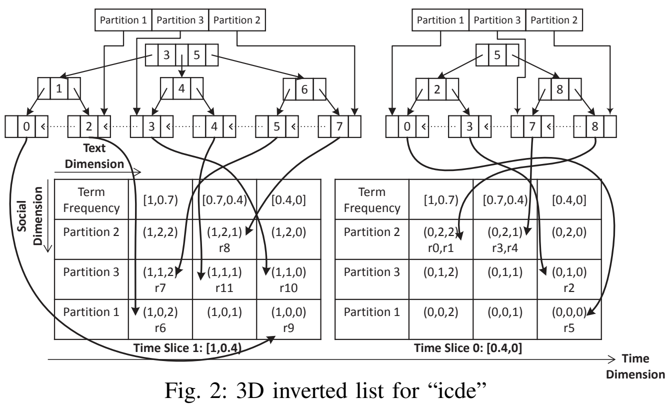


Fig. 2: 3D inverted list for “icde”

IV. 3D Index

To support the **high update rate** for SNPs, we propose a memory based 3D inverted list. Time is the primary dimension. Textual and social dimensions are also included to support efficient pruning. Before introducing the 3D list design, we would like to present how to handle the social dimension first as it is not straightforward to order the records according to the social relevance without knowing who the query user is.

Graph Partition. Given a social graph G like Fig. 1, we first divide the vertices into c partitions using k-way partition [24] with minimum cut utility. For each P_i, a pivot vertex pn_i is selected among the vertices in P_i and the shortest distance from pn_i to each vertex in G is pre-computed. In addition, for any two partitions P_i, P_j, we pre-compute $\mathbf{SD}(P_i, P_j) = \min_{x \in P_i, y \in P_j} \mathbf{SD}(x, y)$. This index is used for estimating the social distance between the query user $q.v$ and a partition. As the partitions along the social dimension do not have an ordering which depends on the query, we rank P_i by $\mathbf{SD}(P_{q.v}, P_i)$ in query time where $P_{q.v}$ contains $q.v$. The space complexity of the index for social distance is $O(|V| + c^2)$.

3D Inverted List. Each keyword w is associated with a 3D list of records which contain w. From Example 1, Fig. 2 shows the 3D list for keyword “icde”. The primary axis of the 3D list is time, from latest to oldest, which is divided into slices. Each time slice contains a 2D grid consisting of the social and textual dimensions. The social dimension corresponds to the c partitions constructed on G. The textual dimension is discretized to m intervals w.r.t. the keyword frequencies tf by using equi-depth partition on historical data [25]. This results in a cube structure of the 3D list where each cube may contain 0 or many records. Note that in Fig. 2, the time and textual dimensions are sorted offline whereas the social partitions will only be sorted against user u_1 when u_1 issues a query at runtime in Example 1.

To support efficient retrieval of records from the 3D list (as discussed later in Sec. V), we define the neighbours of a cube:

Definition 1. *We assign each cube with a triplet:* (x, y, z), *where x,y,z refer to time, social and textual dimensions. The three neighbours of cube* $cb(x, y, z)$ *are* $cb(x-1, y, z)$, $cb(x, y-1, z)$, $cb(x, y, z-1)$ *respectively.*

For example in Fig. 2, there are 18 cubes and the dimension indices (x, y, z) are listed in the first row of a cube. The cube $cb(1,1,1)$ which contains $r11$ has neighbours: $cb(0,1,1)$, $cb(1,0,1)$, $cb(1,1,0)$ along the time, social and textual dimensions respectively. To avoid storing empty cubes in the 3D list, we deploy a B$^+$-tree for each time slice as shown in Fig. 2. In each B$^+$-tree, the value of a leaf node represents a cube's linearized dimension index in the 2D grid. Suppose a cube has a grid index entry (y, z) where y and z represent the entries of the social and textual dimensions respectively, then a leaf node with value $ym + z$ is added to the B$^+$-tree. To facilitate random access, each B$^+$-tree keeps a reference to the leaf node with index $(y, \max(z))$ for each partition y.

Index Update. When a new record r arrives, it is inserted into w's 3D list $\forall w \in r.W$. r is always allocated to the latest time slice and mapped to the corresponding cube. When the number of records in the latest time slice exceeds a system determined threshold, a new time slice will be created. Let us take $r6$ for an example and assume the time slice threshold is 6. When $r6$ arrives, there are already 6 records in time slice 0 (see Fig. 2), so slice 1 is created, which $r6$ is inserted into. In social dimension, since $r6$ belongs to user u_4 (see Fig. 1), $r6$ should go to partition 1. In textual dimension, "icde" has a keyword frequency of 0.8 in $r6$, which falls in the frequency interval $[1, 0.7)$. Finally, $r6$ should be inserted into the cube $cb(1,0,2)$ that maps to leaf node 2 after linearization. The B$^+$-tree is updated accordingly.

Complexity. Since the record is inserted into B+tree and each B+tree has at most $c{\cdot}m$ leaf nodes, the complexity of inserting a new record is just $O(|r.W| \cdot log(c \cdot m))$.

Forward Index. To enable efficient random access, a forward list is built for each record r. Each list stores the keywords in r and their keyword frequencies. The user who posts it and the time posted are stored as well. The table in Fig. 1 is an example of the forward lists for the social network.

V. CubeTA algorithm

Given the 3D list design, we are now ready to present our basic query evaluation scheme, CubeTA (Algorithm 1). CubeTA extends the famous TA algorithm [26] by introducing a two-level pruning upon the 3D list to further speed up the query processing, i.e. at record level and at cube-level. We maintain two data structures in CubeTA: (1) The cube queue CQ which ranks the cubes by their estimated relevance scores (computed by **EstimateBestScore** function in line 15); (2) the min heap H which maintains the current top-k candidates. The main workflow is as follows: In each iteration, we first access the 3D list for each keyword and get the cube cb with the best estimated scores among all unseen cubes in CQ (line 4). Next we evaluate all the records stored in cb (lines 8-12), then we keep expanding the search to the three neighbors of cb (lines 13-16), until the current top-k records are more relevant than the next best unseen cube in CQ. Following Equation 5 in computing the score of a record, Equation 6 illustrates how **EstimateBestScore** estimates the score of a cube cb:

$$\Re(q, cb) = |q.W|(\alpha \mathbf{TS}_{cb} + \frac{\beta \mathbf{SR}_{cb} + \gamma \mathbf{TF}_{cb}}{|q.W|}) \quad (6)$$

Algorithm 1: CubeTA Algorithm

```
Input: Query $q = \langle q.v, q.W, q.t, q.k \rangle$, CubeQueue $CQ$
       which is initialized by inserting the first cube for
       each of the $q.W$'s inverted list.
Output: Top $q.k$ records that match the query $q$.
1  MinHeap $H \leftarrow \phi$          /* $q.k$ best records */
2  $\varepsilon \leftarrow 0$              /* $q.k^{th}$ record's score */
3  while $!CQ.empty()$ do
4      Cube $cb = CQ.pop()$
5      if EstimateBestScore$(q, cb) < \varepsilon$ then
6          return $H$
7      else
8          foreach record $r$ in $cb$ do
9              if $r$ has not been seen before then
10                 if GetActualScore$(q, r, \varepsilon) > \varepsilon$ then
11                     $H.pop()$ and Push $r$ to $H$
12                     $\varepsilon \leftarrow H.top()$'s score w.r.t $q$
13         foreach of the three neighbour cubes $nc$ of $cb$ do
14             if $nc$ has not been seen before then
15                 if EstimateBestScore$(q, cb) > \varepsilon$ then
16                     Push $nc$ to CQ
```

The social score of cb is $\mathbf{SR}_{cb} = 1 - \mathbf{SD}(P_{q.v}, P_{cb})$, where $P_{q.v}$ is the partition containing the query user and P_{cb} is the partition containing the cube cb. The time freshness $\mathbf{TF}_{cb}$ and text similarity $\mathbf{TS}_{cb}$ are the maximum values of cb's time interval and frequency interval. It is easy to see that the total estimated score $\mathbf{SR}_{cb}$ is actually an upper bound of all the unseen records in the cube, so if it is still smaller than the current k^{th} best record's score ε, we can simply terminate the search and conclude the top-k results are found (lines 5-6). This stopping condition is presented in Theorem 1.

Theorem 1. *Let* **cb** *be the next cube popped from CQ. The score estimated by Equation 6 is the upper bound of any unseen record in the 3D lists of all query keywords q.W.*

Proof: Let r be any record that exists in any of the 3D lists and whose score has not been computed. Given a query q, let $\Delta = \beta\mathbf{SR}(q.v, r.v) + \gamma\mathbf{TF}(q.t, r.t)$ and $\delta_w = \alpha \cdot tf_{w,r} \cdot idf_w$ where $w \in q.W$. The overall score $\Re(q, r_x)$ of r w.r.t. q is:

$$\begin{aligned}\Re(q, r_x) &= \Delta + \sum_{w \in q.W} \delta_w = \sum_{w \in q.W} (\delta_w + \frac{\Delta}{|q.W|}) \\ &= \sum_{w \in q.W \cap r.W} (\delta_w + \frac{\Delta}{|q.W|}) + \frac{\Delta |q.W \setminus r_x.W|}{|q.W|} \end{aligned} \quad (7)$$

But note that r_x must exist in one of the 3D lists, say w^*. Then it follows Equation 7:

$$\begin{aligned}\Re(q, r_x) &\leq \sum_{w \in q.W \cap r.W} (\delta_w + \frac{\Delta}{|q.W|}) + |q.W \setminus r_x.W|(\delta_{w^*} + \frac{\Delta}{|q.W|}) \\ &\leq |q.W| \cdot \max_{cb \in q.W} (\alpha \mathbf{TS}_{cb} + \frac{\beta \mathbf{SR}_{cb} + \gamma \mathbf{TF}_{cb}}{|q.W|}) \quad \blacksquare \end{aligned}$$

GetActualScore (Algorithm 2) computes the exact relevance of a certain record. With the forward list mentioned

```
Algorithm 2: GetActualScore Algorithm
    Input: Query $q$, record $r$ and threshold $\varepsilon$
    Output: $\Re(q,r) > \varepsilon\ ?\ \Re(q,r) : -1$
1   Compute TF and TS using the forward list of $r$
2   Compute the Social Relevance Lower Bound
    $\min \mathbf{SR}_r = (\varepsilon - \alpha\mathbf{TS} - \gamma\mathbf{TF})/\beta$
3   $\mathbf{SR}_r = 1 - \mathbf{ShortestDistance}(q.v, r.v, \min \mathbf{SR}_r)$
4   return $\alpha\mathbf{TF} + \beta\mathbf{TS} + \gamma\mathbf{SR}$
```

in Sec. IV, we can compute the exact text similarity and time freshness. Since we have the k^{th} best score ε among the evaluated records, a lower bound for social relevance (i.e. the distance upper bound) can be computed for the current record r before evaluating the distance query (in line 3). This bound enables efficient pruning which we will later discuss on how to compute the exact social relevance score in Section VI. Example 2 shows a running example of how CubeTA works.

Iteration	Processing Cube	Candidates	min**SR**	max**SD**	Best Cube:Score	Top 1
1	(1,2,2)				(1,1,2): 2.8	
2	(1,1,2)	***r7:0.7+(1-0.6)+0.8=1.9***	***0***	***1***	(1,0,2): 2.6	r7
3	(1,0,2)	r7:0.7+(1-0.6)+0.8=1.9 ***r6:0.6+(1-0.7)+0.8=1.7***	***0.5***	***0.5***	(1,2,1): 2.6	r7
4	(1,2,1)	***r8:0.7+(1-0.2)+0.7=2.2*** r7:0.7+(1-0.6)+0.8=1.9 r6:0.6+(1-0.7)+0.8=1.7	***0.4***	***0.6***	(1,1,1): 2.5	r8
5	(1,1,1)	***r11:1+(1-0.4)+0.7=2.3*** r8:0.7+(1-0.2)+0.7=2.2 r7:0.7+(1-0.6)+0.8=1.9 r6:0.6+(1-0.7)+0.8=1.7	***0.2***	***0.8***	(0,2,2): 2.3	r11

Fig. 3: Example of CubeTA (The highlighted text indicates that the records are being evaluated in their current iteration.)

Example 2. *By Example 1, the social partitions are sorted by their distances to u_1 when u_1 issues the query. Fig. 2 shows the reordered 3D list of keyword "icde". Detailed steps for CubeTA are illustrated in Fig. 3. In iteration 1, cube $cb(1,2,2)$ is first evaluated. Since no record is in $cb(1,2,2)$, three neighbors of $cb(1,2,2)$ are inserted into the cube queue. In iteration 2, $cb(1,1,2)$ is popped from the cube queue and before expanding its neighbor cubes into the queue, we evaluate $r7$ and insert it into the candidate max heap. This procedure terminates at iteration 5 because $r11$ is found to have an equal score to the best cube score in the cube queue.*

Efficient Cube Traversal

As discussed in Sec. IV, traversing the 3D list may encounter many empty cubes. To speed up the traversal in CubeTA (line 13), we define the boosting neighbors for a cube $cb(x,y,z)$:

Definition 2. *Suppose there are c social partitions and m frequency intervals, the boosting neighbors of $cb(x,y,z)$ are:*

- $cb(x-1,y,z)$ *if* $y = c \wedge z = m$.
- $cb(x,y-1,z)$ *if* $z = m$.
- $cb(x,y,max\{z' | z' < z \wedge cb(x,y,z')$ *is non-empty*$\})$

Boosting neighbors essentially can identify all the non-empty cubes, as illustrated below.

Theorem 2. *All cubes are covered by traversing via boosting neighbors only.*

Proof: Let $l = max(x)$. As the traversal always starts from $cb(l,c,m)$, we need to prove that any $cb(x,y,z)$ is reachable from $cb(l,c,m)$. Along the textual dimension, $cb(x,y,z)$ is reachable from $cb(x,y,m)$. $cb(x,y,m)$ is reachable from $cb(x,c,m)$ along the social dimension. Lastly $cb(x,c,m)$ is reachable by $cb(l,c,m)$ along the time dimension. ■

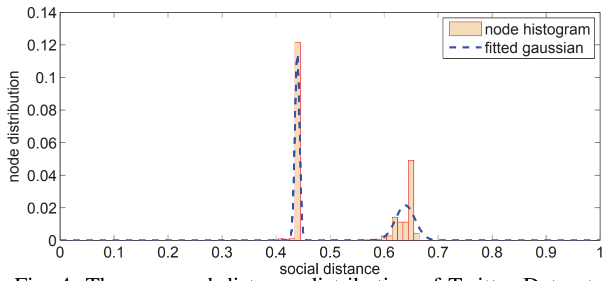


Fig. 4: The averaged distance distribution of Twitter Dataset.

VI. Social-Aware Pruning

In CubeTA, to process a query issued by a user v, a large number of distance queries need to be evaluated between v and the users who post records containing the query keywords in **GetActualScore**. In order to accelerate such one-to-many distance query evaluation, we develop a series of optimizations that operate in three categories: (1) Quick computation of the social relevance scores for records that cannot be pruned; (2) Immediate pruning for records that are not among the final top-k records; (3) Fast construction of the pruning bound at the initial rounds of the graph traversal to facilitate pruning as early as possible.

A. Observation

First, we highlight an observation on the real SNP datasets that motivates our pruning idea. Fig. 4 is the distance distribution which shows the percentage of vertices that have the corresponding social distances to a randomly sampled vertex in the social network. The distribution is constructed by uniformly sampling 1000 vertices from the Twitter dataset which is extensively studied in our experiments. We observe that there exist layers of vertices with different distances from a random vertex. Due to the **small world** property, the number of layers is usually small. This finding has also been verified in the news dataset that we have studied. Moreover, this is consistent in real life context where close friends are rare and the rest just scatter around, which means our defined social distance measure may enable more pruning power on real world SNPs. If we avoid computing the distances between the source vertex and those vertices that are positioned at these layers, the performance will be greatly improved.

B. Baseline Solution: Direct Pruning

For a given source vertex v, Dijkstra Algorithm (DA) [27] is the most widely-adopted approach to find the distance between v and every other vertex in the graph. The key idea of DA is to maintain a set S and a priority queue PQ. S contains all the vertices whose distances to v have been determined while PQ orders the remaining vertices by their estimated distances to v, i.e. $\mathbf{SD}^*(v,u), u \in PQ$. In each iteration, the

TABLE I: Notations used across Sec. VI

Notation	Meaning
v, u	v is the query user, u is any other user in the social network
r_u	the record posted by user u
r^*	the kth best record among evaluated records
S	the set contains users with determined social distances to v
PQ	the priority queue contains users with undetermined social distances to v
$\mathbf{SD}(v, u)$	the actual distance between v and u
$\mathbf{SD}^*(v, u)$	the estimated $\mathbf{SD}(v, u)$ in PQ
$\min \mathbf{SR}_{r_u}$	the minimum $\mathbf{SR}(v, u)$ allowed for r_u s.t. $\Re(v, r_u) \geq \Re(v, r^*)$
$\max \mathbf{SD}_{r_u}$	$1 - \min \mathbf{SR}_{r_u}$
n	$argmin_{x \in PQ} \mathbf{SD}(v, x)$
$mdist$	**SD**(v,n)
DA	Dijkstra's Algorithm

vertex n with the smallest $\mathbf{SD}^*(v, n)$ is popped from PQ. At this moment, we are sure that $\mathbf{SD}(v, n) = \mathbf{SD}^*(v, n)$ and we denote $\mathbf{SD}(v, n)$ by $mdist$. Then n is inserted into S and for each $n' \in PQ$ where $(n, n') \in E$, $\mathbf{SD}^*(v, n')$ is updated to $\min(\mathbf{SD}^*(v, n'), mdist + \mathbf{SD}(n, n'))$. Therefore, in order to answer such one-to-many distance query, we choose to use DA as the backbone for our traversal while presenting a series of pruning methods to access as few nodes as possible.

In particular, whenever a record r_u posted by u contains any query keyword(s), we may need to evaluate $\mathbf{SD}(v, u)$. If $u \in S$, no further computation is needed because $\mathbf{SD}(v, u)$ has already been determined. Otherwise, by Algorithm 2 (line 2), we can get a lower bound for u's social relevance: $\min \mathbf{SR}_{r_u}$, i.e. the minimum possible $\mathbf{SR}(v, u)$ for r_u to be no less relevant than the current k^{th} best record r^*. Equivalently, $\max \mathbf{SD}_{r_u}$ denotes the maximum allowed $\mathbf{SD}(v, u)$ in order for r_u to be a potential top-k candidate. Therefore, once we find $mdist \geq \max \mathbf{SD}_{r_u}$, it means r_u is definitely not a top-k candidate and we can simply terminate our search and return. This forms our baseline method called *Direct Pruning*.

Example 3. *In Example 2, CubeTA needs to evaluate $r7$, $r6$, $r8$, $r11$ before confirming the top-1 result. For each record evaluated, its $\min \mathbf{SR}$ and $\max \mathbf{SD}$ are listed in Fig. 3, and the user who posted it is in Fig. 1. When direct pruning is enabled, the search proceeds as below. We first evaluate $r7$ posted by u_7. Since no record has been computed, $\min \mathbf{SR}_{r7} = 1$ and original DA visits the vertices in the order of u_1, u_3, u_2, u_{10}, u_8, u_{11}, u_5, u_6 and lastly reach u_7 to get $\mathbf{SR}(u_1, u_7) = 1 - \mathbf{SD}(u_1, u_7) = 0.4$. When we proceed to Iteration 3 to evaluate $r6$ posted by u_4, the current top-1 result is r_7, so we have $mdist = \mathbf{SD}(u_1, u_7) = 0.6$ and $\max \mathbf{SD}_{r6} = 0.5$. We find $mdist > \max \mathbf{SD}_{r6}$, which means $r6$ needs to be pruned from the top-1 candidate.*

C. In-circle Pruning

Since DA traverses the graph in a closest-vertex-first fashion, we must avoid traversing to any vertex that has a large social distance from the query user v so that the layers in Fig. 4 are never reached. Otherwise evaluating just one distance query may require traversing the entire social graph and the performance can be as bad as original DA, i.e. $O(|E|+|V|log|V|)$. We will discuss three issues, namely **Far True Candidates**, **Far False Candidates** and **Cold Start**, where direct pruning fails to avoid the distance computation from v to far vertices.

1) Far True Candidates: These refer to records that are socially far from the query user v but cannot be pruned at the current stage of query processing. Thereby we have to compute the exact social distances for these far candidates. Since direct pruning only prunes the candidates rather than computing the exact social distance, i.e. $\mathbf{SD}(v, u)$, we propose the first pruning method called *early-determination* to quickly compute the social distance as outlined in category (1)'s optimization. According to DA, the estimated distance $\mathbf{SD}^*(v, u)$ is only updated to its actual distance $\mathbf{SD}(v, u)$ when there exists a u' that is popped from PQ in previous iteration(s) and is also a neighbor of u. Therefore, given the vertex $n = argmin_{x \in PQ} \mathbf{SD}(v, x)$ that is being popped from PQ and the $mdist = \mathbf{SD}(v, n)$, we can get a lower bound of $\mathbf{SD}(v, x)+\mathbf{SD}(x, u), \forall x \in PQ$. If this bound is no smaller than the current $\mathbf{SD}^*(v, u)$, then we can simply stop and determine that $\mathbf{SD}(v, u) = \mathbf{SD}^*(v, u)$.

Theorem 3. *Let $\mathrm{SD}^*(v, u)$ be the estimated $\mathbf{SD}(v, u)$ for a vertex u popped from PQ. If $mdist + \min_{u'} \mathrm{SD}(u, u') \geq \mathrm{SD}^*(v, u)$ where $(u, u') \in E$, then $\mathrm{SD}(v, u) = \mathrm{SD}^*(v, u)$.*

Proof: If $mdist + \min_{u'} \mathrm{SD}(u, u') \geq \mathrm{SD}^*(v, u)$, then $\forall x \in PQ$, $\mathrm{SD}(v, x) + \mathrm{SD}(x, u) \geq mdist + \min_{u'} \mathrm{SD}(u, u') \geq \mathrm{SD}^*(v, u)$. It means that the estimated distance cannot be updated to a smaller value via any path that contains vertices in PQ, so the social distance between v and u has been determined, i.e. $\mathrm{SD}(v, u) = \mathrm{SD}^*(v, u)$. ■

Example 4. *Recall Example 3, the direct pruning needs to traverse 9 vertices to get the social score for $r6$. With early-determination applied, the traversal still starts from u_1, and when popping u_8 from PQ, we update $\mathbf{SD}^*(u_1, u_7) = 0.6$ as u_7 is a neighbour of u_8. We can stop the traversal at u_{11} and determine $\mathbf{SD}(u_1, u_7) = 0.6$ because $\mathbf{SD}(u_1, u_{11}) + \min_{u'} \mathbf{SD}(u_7, u') = 0.6 \geq \mathbf{SD}^*(u_1, u_7) = 0.6$. As a result, early-determination terminates by traversing 6 vertices only.*

To make the pruning more effective, it is critical to obtain a precise estimation of $\mathbf{SD}(v, u)$ before traversing. Thus we use $\mathbf{SD}^*(v, u) = \min\{\mathbf{SD}(v, pn_u)+\mathbf{SD}(pn_u, u), \mathbf{SD}(u, pn_v)+\mathbf{SD}(pn_v, v)\}$ as an upper bound of $\mathbf{SD}(v, u)$ and pn_u, pn_v are pivot vertices from partitions P_v, P_u ($v \in P_v$, $u \in P_u$).

2) Far False Candidates: These refer to records that are socially far away from v and should be pruned from the current top-k result set. For a record r_u, early-determination only computes $\mathbf{SD}(v, u)$ without exploiting $\max \mathbf{SD}_{r_u}$ for pruning r_u. In Fig. 4, suppose we want to evaluate r_u where $\mathbf{SD}(v, u) = 0.8$ and the nearest neighbor of u has a distance of 0.1 from u, then the processing by early-determination is slow because $\mathbf{SD}(v, u)$ cannot be determined until we pop a vertex from PQ that has a distance of 0.7 from v. This motivates us to propose the second pruning method, namely *early-pruning* as outlined in category (2), to complement the early-determination method. Like early-determination that estimates a lower bound that $\mathbf{SD}^*(v, u)$ could be updated to by vertices in PQ, early-pruning essentially prunes the record r_u by exploiting $\max \mathbf{SD}_{r_u}$, as described in Theorem 4.

Theorem 4. *If $mdist + \min_{u'} \mathbf{SD}(u, u') \geq \max \boldsymbol{SD}_{r_u}$ and $\mathbf{SD}^*(v, u) \geq \max \boldsymbol{SD}_{r_u}$, then $\mathbf{SD}(v, u) \geq \max \boldsymbol{SD}_{r_u}$.*

Proof: If $mdist + \min_{u'} \mathbf{SD}(u, u') \geq \max \mathbf{SD}_{r_u}$, similar to Theorem 3, $\mathbf{SD}^*(v, u)$ will not be updated to a distance smaller than $\max \mathbf{SD}_{r_u}$ by any remaining vertices in PQ. Thus it is possible for $\mathbf{SD}(v, u) < \max \mathbf{SD}_{r_u}$ if $\mathbf{SD}^*(v, u) < \max \mathbf{SD}_{r_u}$. So by ensuring $\mathbf{SD}^*(v, u) \geq \max \mathbf{SD}_{r_u}$, we can determine $\mathbf{SD}(v, u) \geq \max \mathbf{SD}_{r_u}$. ■

Example 5. *From Example 4, the traversal stops at u_{11} after evaluating $r7$. The next record to compute is $r6$ posted by u_4. $\boldsymbol{SD}^*(u_1, u_4) = 0.7$ as $u_2 \in S$ and u_2 is a neighbor of u_4. We also know $\max \boldsymbol{SD}_{r6} = 0.5$ in Fig. 3. However, we cannot early determine $\boldsymbol{SD}(u_1, u_4)$ at u_{11} because $\boldsymbol{SD}(u_1, u_{11}) + \min_{u'} \boldsymbol{SD}(u_4, u') = 0.6 < \boldsymbol{SD}^*(u_1, u_4) = 0.7$. Instead we can use early-pruning to eliminate $r6$ because $\boldsymbol{SD}(u_1, u_{11}) + \min_{u'} \boldsymbol{SD}(u_4, u') > \max \boldsymbol{SD}_{r6}$ and $\boldsymbol{SD}^*(u_1, u_4) > \max \boldsymbol{SD}_{r6}$.*

3) Cold Start: This refers to scenarios where $\max \mathbf{SD}$ is not small enough for efficient pruning at the early stage of query processing. Suppose the first record to evaluate is posted by the furthest vertex in the social graph w.r.t v, early-pruning cannot help as the pruning bound is trivial. Although early-determination can reduce the query time to some degree, it is still highly possible to visit almost every vertex.

Thus, we propose the third pruning method called *warm-up queue*, which aligns with the optimization of category (3). The warm-up queue WQ is meant to evaluate the records that are nearer to v first and to obtain a decent bound for further pruning. WQ is constructed as follows: we push a number of records to WQ before computing any social distance. WQ ranks the records with an estimated distance, which is computed by exploiting the pivot vertices as the transit vertices between v and the authors of the records. When the size of WQ exceeds δ, all records in WQ will be popped out and their exact scores are computed followed by the original CubeTA.

A key problem is to determine δ. We wish that, among δ records in WQ, there are at least $q.k$ records whose social distances to v are smaller than the left most layer in the vertex-distance distribution. Based on the observation from Fig. 4, we model the vertex-distance distribution as a mixture of gaussians, which is a weighted sum of M normal distributions: $p(x) = \sum_{i=1}^{M} w_i g(x|\mu_i, \sigma_i)$ where $g(x|\mu_i, \sigma_i)$ is the probability density of a normal distribution with mean μ_i and variance σ_i^2. In the context of social network, the number of layers in the vertex-distance distribution is small due to the **small world** property and it makes the training complexity low. The model is established by the standard expectation maximization method. Given the mixture model and a random record r_u, the probability p_{opt} that $\mathbf{SD}(q, u) < \mu_{min}$ where μ_{min} is the mean of the left most normal component which means $\mu_{min} \leq \mu_i, \forall i = 1..M$.

$$p_{opt} = \int_{-\infty}^{\mu_{min}} \sum_{i=1}^{M} w_i g(x|\mu_i, \sigma_i) dx \tag{8}$$

Assuming the record authors are independent and identically distributed random variables w.r.t their distance to v in the social graph, the probability of having at least $q.k$ records whose social distances to v are smaller than μ_{min} follows the binomial distribution:

$$p(\delta) = 1 - \sum_{i=0}^{q.k} \binom{\delta}{i} (1 - p_{opt})^{\delta - i} {p_{opt}}^{i} \tag{9}$$

In this work we aim to ensure $p(\delta) > 99.9\%$ so that, in most cases, the first $q.k$ records in WQ have social distances that are less than μ_{min}.

Algorithm 3: Optimized Distance Query Computation

```
Input: Query user $v$, record $r_u$, set $S$, priority queue
       $PQ$, $\max \mathbf{SD}_{r_u}$ and $mdist$
Output: $\mathbf{SD}(v,u) < \max \mathbf{SD}_{r_u}$ ? $\mathbf{SD}(v,u)_{r_u}$ : $-1$
1  if $u \in S$ then
2      return $\mathbf{SD}(v,u) < \max \mathbf{SD}_{r_u}$ ? $\mathbf{SD}(v,u)_{r_u}$ : $-1$
3  $u'' \leftarrow$ nearest 2-hop neighbour of $u$.
4  for $(u,u') \in E$ do
5      if $\boldsymbol{SD}^*(v,u') + \boldsymbol{SD}(u',u) < \boldsymbol{SD}^*(v,u)$ then
6          $\mathbf{SD}^*(v,u) \leftarrow \mathbf{SD}^*(v,u') + \mathbf{SD}(u',u)$
7          Update $\mathbf{SD}^*(v,u)$ of $u$ in $PQ$
8  while $PQ$ is not empty do
9      if $mdist + \boldsymbol{SD}(u,u'') \geq \boldsymbol{SD}^*(v,u)$ then
10         return $\mathbf{SD}^*(v,u) < \max \mathbf{SD}_{r_u}$ ? $\mathbf{SD}(v,u)$ : $-1$
11     if $mdist + \boldsymbol{SD}(u,u'') \geq \max \boldsymbol{SD}_{r_u} \wedge \boldsymbol{SD}^*(v,u) \geq$
       $\max \boldsymbol{SD}_{r_u}$ then
12         return $-1$
13     Vertex $n \leftarrow PQ.pop()$
14     $mdist \leftarrow \mathbf{SD}^*(v,n)$
15     $S \leftarrow S \cup n$
16     for $(n',n) \in E$ and $n' \notin S$ do
17         if $\boldsymbol{SD}^*(v,n) + \boldsymbol{SD}(n,n') < \boldsymbol{SD}^*(v,n')$ then
18             $\mathbf{SD}^*(v,n') \leftarrow \mathbf{SD}^*(v,n) + \mathbf{SD}(n,n')$
19             Update $\mathbf{SD}^*(v,n')$ of $n'$ in $PQ$
20         if $(n',u) \in E$ then
21             if $\boldsymbol{SD}^*(v,n') + \boldsymbol{SD}(n',u) < \boldsymbol{SD}^*(v,u)$ then
22                 $\mathbf{SD}^*(v,u) \leftarrow \mathbf{SD}^*(v,n') + \mathbf{SD}(n',u)$
23                 Update $\mathbf{SD}^*(v,u)$ of $u$ in $PQ$
```

D. Out-of-circle Pruning

In Theorems 3 and 4, the nearest neighbor u' of the target vertex u and its distance to u play a critical role to quickly determine $\mathbf{SD}(v, u)$, However, when u' has a very short distance to u, none of the aforementioned pruning techniques is effective. So by exploiting the 2-hop nearest neighbour of u in the above pruning methods, in particular *early-determination* and *early-pruning*, the pruning power could be enhanced as compared to its in-circle counterparts. This is because the 2-hop nearest neighbour has a longer distance from u, which results in an earlier determination of $\mathbf{SD}(u, v)$. We refer to this approach as *out-of-circle pruning*, and will demonstrate its merit over the in-circle pruning in Example 6.

Example 6. *By Examples 4 & 5, in-circle early-determination needs to traverse 6 vertices to evaluate* $\textbf{SD}(u_1, u_7)$*, while with the out-of-circle early-determination we only need to traverse 3 nodes. The nearest 2-hop neighbour of* u_7 *is* u_3 *and* $\textbf{SD}(u_7, u_3) = 0.5$*. Before any traversal, we use* u_8*, which we assume it to be the pivot vertex of partition 3, to estimate* $\textbf{SD}^*(u_1, u_7) = \textbf{SD}(u_1, u_8) + \textbf{SD}(u_8, u_7) = 0.6$*. Then the out-of-circle early-determination takes effect when we reach* u_2*: now* $\textbf{SD}(u_1, u_2) + \textbf{SD}(u_7, u_3) = 0.7 > \textbf{SD}^*(u_1, u_7) = 0.6$*; so by Theorem 3 we guarantee* $\textbf{SD}(u_1, u_7) = 0.6$*. Furthermore, we can use out-of-circle early-pruning to eliminate* $r6$*. Since* u_4 *posted* $r6$*, we first identify the 2-hop nearest distance of* u_4 *to be* $\textbf{SD}(u_4, u_9) = 0.4$*. In addition we know that* $\textbf{SD}^*(u_1, u_4) = 0.7$ *since* u_2 *was reached when evaluating* $r7$*. Then by out-of-circle early-pruning we are sure* $r6$ *is not the top-1 candidate at* u_2 *because* $\textbf{SD}(u_1, u_2) + \textbf{SD}(u_4, u_9) > \max \textbf{SD}_{r6}$ *and* $\textbf{SD}^*(u_1, u_4) > \max \textbf{SD}_{r6}$*.*

As a result, by enabling more powerful out-of-circle pruning upon the DA traversal, we present a complete solution of the social distance computation in Algorithm 3. In particular, lines 9-12 extend the idea of Theorems 3 and 4 by replacing the nearest neighbour distance by the nearest 2-hop distance; lines 4-7 and 20-23 are meant to guarantee the correctness of pruning: if n' is a neighbor of u, $\textbf{SD}^*(v, u)$ gets updated via a path that contains n'. The rest part is in line with the original DA: specifically, Lines 1-2 return the social distance if $\textbf{SD}(v, u)$ has been determined in S; Lines 13-19 follow the graph traversal and distance estimate in DA.

Example 7. *In Example 6, we visit* u_1*,* u_3*,* u_2 *in order after evaluating* $r7$*,* $r6$ *while* $r8$ *and* $r11$ *remain to be evaluated. The total score of* $r8$ *is 2.2 and we continue to evaluate* $r11$ *posted by* u_{11}*. By adopting the same assumption from Example 6, we use* u_8 *to be the pivot vertex of partition 3. Then we obtain* $\textbf{SD}^*(u_1, u_{11}) = \textbf{SD}(u_1, u_8) + \textbf{SD}(u_8, u_{11}) = 0.8$*. According to lines 4-7, we need to update* $\textbf{SD}^*(u_1, u_{11}) = \textbf{SD}^*(u_1, u_{10}) + \textbf{SD}(u_{10}, u_{11}) = 0.4$ *when we traverse to* u_3 *as* u_{10} *is a neighbour of both* u_{11} *and* u_3*.*

If we do not perform lines 4-7, this means $\textbf{SD}^*(u_1, u_{11}) = 0.8$ *instead of* 0.4*. Since* $r8$ *is the current top-1 candidate,* $\max \textbf{SD}_{r11} = 0.5$*. As the traversal stops at* u_2 *and nearest 2-hop neighbour of* u_{11} *is* u_3*, the out-of-circle early-pruning will eliminate* $r11$ *because* $\textbf{SD}(u_1, u_2) + \textbf{SD}(u_3, u_{11}) = 0.5 \geq \max \textbf{SD}_{r11}$ *and* $\textbf{SD}^*(u_1, u_{11}) > \max \textbf{SD}_{r11}$*. But* $r11$ *is the ultimate top-1 result which should not be pruned. lines 20-23 in Algorithm 3 have a similar reason to guarantee the correctness of the out-of-circle pruning.*

Out-of-circle pruning requires the pre-computation of the nearest 2-hop distance from each vertex which is retrieved by using DA. The worst time complexity is $O(|V| + |E|)$ and the process can easily be parallelized. The space complexity is $O(|V|)$ which brings almost no overhead for using out-of-circle pruning compared to in-circle pruning. One may wonder whether the pruning can be further extended by using 3-hop nearest distance and beyond. The answer is that it brings more

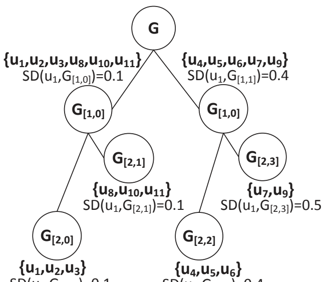


Fig. 5: Tree partition of the social graph in Fig. 1.

Insert **r6-r10**		[1,0.7)	[0.7,0.4)	[0.4,0]
	$G_{<2,0>}$		r8	
	$G_{<2,2>}$	r6		r9
	$G_{<2,3>}$	r7		r10
Insert **r11**		[1,0.7)	[0.7,0.4)	[0.4,0]
	$\boldsymbol{G_{<1,0>}}$		r8,r11	
	$G_{<2,2>}$	r6		r9
	$G_{<2,3>}$	r7		r10

Fig. 6: Example of building time slice 1 of the 3D list for "icde" on the partition tree in Fig. 5.

complexity in both time and space. If we use 3-hop nearest distance, one has to ensure $\textbf{SD}^*(v, u)$ is updated via a path that contains n' if n' is 2-hop away from u. However, to check if n' is a 2-hop neighbour of u, we must either store all 2-hop neighbours for each vertex or validate 2-hop relationship on the fly. Storing all 2-hop neighbours are not realistic for large graphs whereas computing on the fly will trivially slow down the query processing. Therefore, it can be concluded that out-of-circle pruning achieves the maximum pruning along the social dimension.

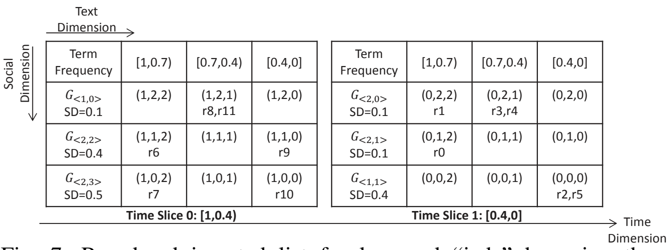
VII. 3D INDEX OPTIMIZATION

In the 3D index, the social dimension is statically partitioned. Intuitively, the larger the number of partitions is, the more accurate the estimate of the social score will be; and this will translate to more effective pruning. However, a large number of partitions on the social dimension will severely tax the system resources and this is not scalable for large social networks. Moreover the static partition strategy does not capture the nature of online social activities. Given a period of time, people who are socially related are likely to publish similar information on the online network. A fine-grained yet time-aware social partition is required for more efficient pruning. Therefore we develop a dynamic division strategy on social dimension using the hierarchical graph partition.

A. Hierarchical Graph Partition Index

We improve static social division scheme by a hierarchical graph partition using a binary tree denoted as $pTree$. Fig. 5 serves an example of a partition tree of the social graph mentioned in Fig. 1. The root of $pTree$ contains all vertices in G. Each child node in the tree represents a sub partition of G. A sub partition is represented as $G_{[h,idx]}$ where h denotes the level in $pTree$ and idx is the position of $G_{[h,idx]}$ at level h. For the 3D list, we still keep c partitions as described in Sec. IV. The improvement is that the c partitions are formed by nodes in $pTree$ instead of uniform graph partitions.

The 3D list dynamically updates the social dimension when new records are inserted. The update procedure maintains c partitions within a time slice. When u_r posted a new record r, within each time slice where r is inserted into, we try to find the sub partition $G_{[h,idx]}$ to store r. Traversing from the root of $pTree$, if $G_{[h',idx']}$ has already contained some records or $G_{[h',idx']}$ is a leaf node of $pTree$, we insert r into $G_{[h',idx']}$. Otherwise we traverse to the child partition


Text Dimension
Social Dimension
Time Dimension

Term Frequency	[1,0.7)	[0.7,0.4)	[0.4,0]
$G_{<1,0>}$ SD=0.1	(1,2,2)	(1,2,1) r8,r11	(1,2,0)
$G_{<2,2>}$ SD=0.4	(1,1,2) r6	(1,1,1)	(1,1,0) r9
$G_{<2,3>}$ SD=0.5	(1,0,2) r7	(1,0,1)	(1,0,0) r10

Time Slice 0: [1,0.4)

Term Frequency	[1,0.7)	[0.7,0.4)	[0.4,0]
$G_{<2,0>}$ SD=0.1	(0,2,2) r1	(0,2,1) r3,r4	(0,2,0)
$G_{<2,1>}$ SD=0.1	(0,1,2) r0	(0,1,1)	(0,1,0)
$G_{<1,1>}$ SD=0.4	(0,0,2)	(0,0,1)	(0,0,0) r2,r5

Time Slice 1: [0.4,0]

Fig. 7: Reordered inverted list for keyword "icde" by using the hierarchical partition w.r.t. user u_1.

of $G_{[h',idx']}$ that contains u_r. For any two nodes $G_{[x,idx_x]}$ and $G_{[y,idx_y]}$ on $pTree$, we denote their Lowest Common Ancestor by $LCA(G_{[x,idx_x]}, G_{[y,idx_y]})$. After insertion, if we find $c+1$ non-empty sub partitions, we merge two sub partitions G_{left} and G_{right} to form $G_{[h^*,idx^*]} = LCA(G_{left}, G_{right})$, and $G_{[h^*,idx^*]}$ has the lowest height h^* among all possible merges. If there is a tie in h^*, the merge that involves the least number of records will be executed.

Recall Example 2, Fig. 6 demonstrates how to divide the social partition dynamically using a hierarchical partition for the 3D list of the keyword "icde" in Fig. 2. Since we are still building three partitions within a time slice, $r6 - r10$ are inserted into the leaves of the partition tree when they arrive chronologically. When $r11$ arrives, we first identify that it should be inserted into $G_{[2,1]}$. However we can only keep three partitions, so $G_{[2,0]}$ and $G_{[2,1]}$ are merged to form $G_{[1,0]}$ shown in Fig. 6 when inserting $r11$. Note that in a time slice we only store sub partitions that contain at least one record, and the tree structure is just a virtual view.

The advantage of our hierarchical partition is that, we have a fine-grained social partitions which improve the estimation of the cube's relevance score. This enables more opportunities for powerful pruning along the social dimension. At the same time, we still have c partitions and the resource requirement does not increase. Again in Fig. 5, the score under each tree node represents the social distance estimated from u_1. The static partition scheme estimates the distance from u_1 to u_7 as 0.2 where the hierarchical partition scheme gives 0.5 which is closer to the true value (0.6).

B. CubeTA on Hierarchical Graph Partition Index

CubeTA has to be extended to incorporate the hierarchical partition to improve the pruning efficiency. Since the partition scheme is improved from a single layer to multi layers, we need to change the method to estimate social distances. In the pre-processing step, all leaf nodes of partition tree $pTree$ are computed for distance to each other. When a user u submits a query, we first identify the leaf node G_u that this user matches. Then the distance $\mathbf{SD}(u, G_{[h,idx]})$ from u to any partitions $G_{[h,idx]}$ is estimated using $\min \mathbf{SD}(u, G_{[h',idx']})$ and $G_{[h,idx]}$ is an ancestor of $G_{[h',idx']}$ in $pTree$. Suppose user u_1 submits a query, then the social distances from u_1 to other sub partitions are estimated in Fig. 5. The distances from G_{u_1} to all leaf nodes are first retrieved from the pre-computed index and the value is shown below the nodes. Then distance from u_1 to $G_{[1,0]}$ and $G_{[1,1]}$ are estimated as 0.1 and 0.4 respectively by taking the minimum value of their leaf nodes.

TABLE II: All parameter settings used in the experiments.

Parameters	Pool of Values					
Datasets	Twitter			News		
# of users	**<u>1M</u>**	5M	10M	**<u>0.1M</u>**	0.5M	1M
max vertex degree	**<u>0.7M</u>**	0.7M	0.7M	**<u>16K</u>**	29K	30K
average vertex degree	**<u>81.6</u>**	82.5	46.1	**<u>9.2</u>**	6.8	7.0
# of records	**<u>10M</u>**,15M,20M			**<u>0.5M</u>**,1M,5M		
keyword frequency	low,medium,**<u>high</u>**					
degree of query user	low,medium,high					
top-k	1,**<u>5</u>**,10,15,. . .,50					
dimension weight (α, β, γ)	**<u>(0.1,0.1,0.1)</u>**(0.1,0.3,0.5)(0.1,0.5,0.3)(0.3,0.1,0.5) (0.3,0.5,0.1)(0.5,0.1,0.3)(0.5,0.3,0.1)					

After reordering the social dimension w.r.t. user u_1, we can visualize the 3D list of "icde" as Fig. 7. For each social partition, the estimated social distances are listed in the cell. The partitions may vary across different time slice but the number of partitions remains the same. CubeTA can be applied directly to the hierarchical index.

We also see the feasibility of the 3D list and CubeTA which two of them easily incorporate the hierarchical index into efficient query processing.

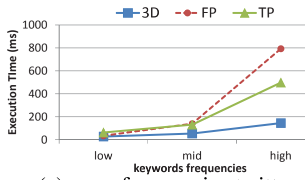
VIII. Experiment Results

We implemented the proposed solution on a CentOS server (Intel i7-3820 3.6GHz CPU with 60GB RAM) and compared with the baseline solutions on two large yet representative real world datasets: Twitter and Memetracker from SNAP[3]. The original Twitter dataset contains 17M users, 476M tweets. Memetracker is an online news dataset which contains 9M media and 96M records. Twitter encapsulates a large underlying social graph but short text information (average number of distinct non-stopped keywords in a tweet is 7); Memetracker has a smaller social graph but rich in text information (average number of distinct non-stopped keywords in a news is 30). The datasets with different features are used to test our proposed solution. Since both raw social graphs have a lot of isolated components, we sampled the users that formed a connected component to demonstrate the effectiveness of our solution. Accordingly we filtered the documents/tweets based on the sampled users, resulting in the datasets used in our experiments. Table II contains all the parameters used in our experiments, and those highlighted in bold denote the default settings unless specified otherwise. For scalability tests, we make three samples of the social graph and three samples of the text records for each dataset. For query keywords, we randomly sampled keywords with length 1, 2, 3. We are aware of four factors that may have impact on the overall performance: keyword frequency, query user, top-k and weights for different dimensions for the ranking function (Equation 5).

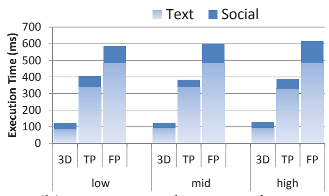
As no existing work supports searching along all three dimensions, we would like to compare the proposed solution with several baseline methods to demonstrate the effectiveness.

- **Time Pruning (TP)**: The state of the art on real time social keyword search [3], [12], [4] sorts the inverted list by reverse chronological order, in order to return the latest

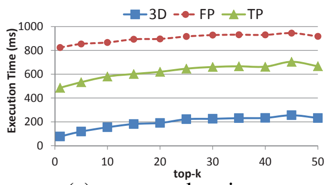
[3]http://snap.stanford.edu/



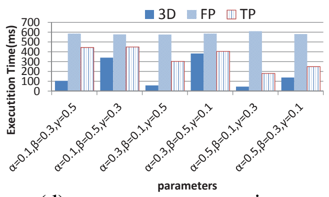
(a) vary frequencies:twitter



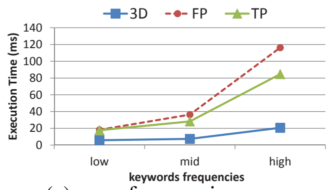
(b) vary user degree:twitter



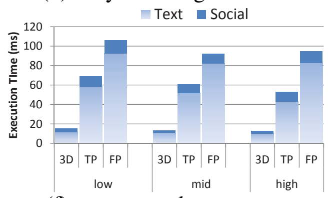
(c) vary top-k:twitter



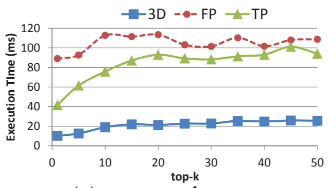
(d) vary parameters:twitter



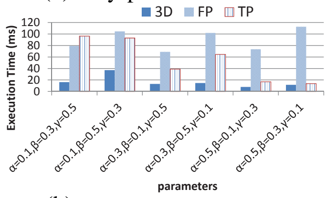
(e) vary frequencies:news



(f) vary user degree:news



(g) vary top-k:news



(h) vary parameters:news

Fig. 8: Performance for various settings

result. Thereby, we implement a TA approach that can take advantage of such order to retrieve the top-k results.

- **Frequency Pruning (FP)**: Without considering efficient updates in the real time social network, traditional approach for personalized keyword search sorts the inverted list by keyword frequencies [5], [6]. **FP** is the implementation of TA on this type of inverted list.
- **3D Index (3D)**: It is the complete solution that we proposed in this paper, i.e. CubeTA (Algorithm 1) with our optimized distance query computation (Algorithm 3).

All methods consist of two parts of computation for retrieving the top-k results. We refer the time to evaluate all candidates' social relevance scores as *social* and the rest is denoted as *text*. Two notions are adopted throughout all figures in this section.

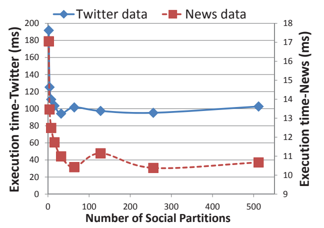


Fig. 9: Effect of increasing social partitions for both datasets

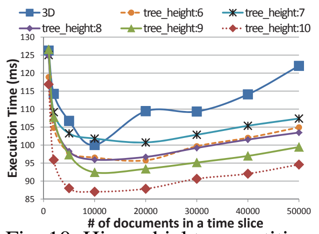


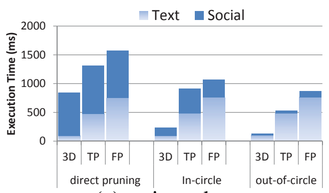
Fig. 10: Hierarchial tree partition vs. static partition

A. Discussion on Social Partitions

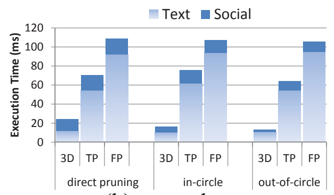
Before presenting the results to compare all the methods, we investigate the query performance with different social partitions in the 3D list. Although more social partitions in the 3D list bring better accuracy in distance estimates, such improved accuracy has slowed down due to the small world phenomenon and the high clustering property of the social network. Moreover, the query processor needs to visit more cubes due to the additional social partitions. Thus, as shown in Fig. 9 there is an optimal setup for the number of social partitions for both datasets. For twitter dataset, the optimal number of social partitions is 32 while that for news dataset is 64. Even though twitter dataset has a larger social graph than news dataset, twitter has a higher average degree resulting in a higher degree of clustering. This brings more difficulties in distinguishing the vertices in terms of their social distances.

We also study the impact of hierarchical partition (proposed in Sec. VII) on query processing. Three factors will impact on the performance: (1) Number of documents within a time slice, (2) Height of the partition tree $pTree$, (3) Number of partitions to keep in the 3D list. Since the memory is usually constrained by a maximum limit, we cannot keep too many partitions. Therefore, we fix the number of partitions as the optimal setting just mentioned. Fig. 10 shows the performance results on twitter dataset when we vary the number of documents within a time slice and also the height of the partition tree. We find: (1) more fine-grained social partition leads to better query performance but it will slow down the index update as allocating records to the 3D list requires tree traversal; (2) an optimal setup exists for number of documents to be allocated in a time slice (10000 in this case).

The hierarchical partition achieves better performance than the static partition, but it will involve many discussions on parameter settings. Due to the space constraint, the static partition is used in **3D** to compare with the two baselines: **TP** and **FP**. We have seen that the time slice size and the number of social partitions have corresponding optimal setups and we will adopt this setting throughout the experiment. Besides, we use 10 intervals for the text dimension.



(a) twitter data

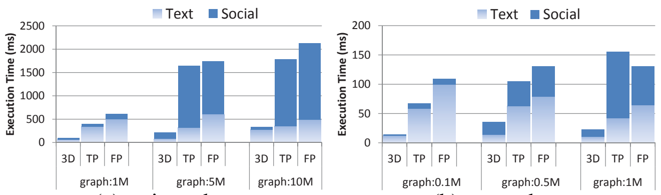


(b) news data

Fig. 11: Effect of distance pruning

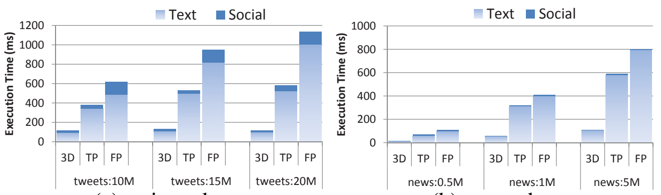
B. Efficiency Study

1) Evaluating Our Pruning Techniques: Fig. 11 shows the experimental results for both datasets with default settings. In general, **3D** outperforms the other by a wide margin. In addition, prunings based on the time dimension (**TP**, **3D**) have better performance than the pruning based on the textual dimension (**FP**) because searching along the text dimension is yet another multi-dimensional search for multiple keywords, and the well-known curse of dimensionality problem reduces



(a) twitter data (b) news data

Fig. 12: Scalability: Effect of increasing social graph size



(a) twitter data (b) news data

Fig. 13: Scalability: Effect of increasing text information

the pruning effect along the text dimension. In contrast, the time dimension is just a scalar so it is more efficient to prune.

To see the effect of the distance optimizations proposed: direct, in-circle and out-of-circle pruning, we apply them to **TP**, **FP** and **3D**. We find that, with better distance pruning methods, the time for distance queries is greatly reduced for all methods. Moreover, we have confirmed that **3D** works better with the optimized distance query computation and enables more efficient pruning compared to the other two. When all methods employ the optimized distance query computation (equipped with all pruning techniques in Sec. VI), **3D** achieves 4x speedups against **TP** and 8x speedups against **FP**.

In the rest of this section, we investigate the query processing time (of CubeTA + complete optimized distance computation) by varying the keywords frequencies, the query user, the choice of top-k and the dimension weights respectively.

2) Varying Keyword Frequency: Fig. 8a and 8e show the query processing time over different ranges of keyword frequency. Keywords with high and medium frequencies are the top 100 and top 1000 popular keywords respectively, whereas the low frequency keywords are the rest which appear at least 1000 times. In both datasets, we have the following observations: (1) Among all approaches, **3D** dominates the rest for all frequency ranges. (2) As query keywords become more popular (i.e. with higher frequencies), the performance speedup by **3D** against other methods becomes larger. Intuitively, there are more candidate documents for more popular query keywords. **3D** effectively trims down candidates and retrieves the results as early as possible.

3) Varying Query User: We further study the performance w.r.t. users with degrees from high to low. The high, medium and low degree users denote the upper, mid and lower third in the social graph respectively. We randomly sample users from each category to form queries. The results are reported in Fig. 8b and 8f. We find: (1) **3D** achieves a constant speedup compared to the rest of the methods regardless of the query user's degree. (2) The social relevance computation for **TP** and **FP** takes longer than **3D**, even though **TP** and **FP** have the same distance pruning technique as **3D**. This is because **3D** prunes more aggressively on social dimension using time and text dimension whereas the other two methods only have one dimension to prune. As illustrated later in Sec. VIII-C, such advantage is magnified when the graph goes larger.

4) Varying Top-k: We vary the top-k value from 1 to 50. As shown in Fig. 8c and 8g, the performance slowly drops with more required results. Since we are doing high dimensional search, result candidate scores tend to be close to each other. Therefore it is more meaningful to identify results quickly for smaller top-k values, which also explains why we set the default top-k as 5. **3D** retrieves the results extremely fast compared to other methods and scales against even larger top-k value. For **TP**, the performance is very poor for news dataset against larger top-k values as news dataset contains more text information, so time pruning becomes less effective. Lastly for **FP**, it almost exhausts all possible documents in the inverted lists to get the top-k results which is almost a linear scan.

5) Varying Dimension Weights: Lastly, for each dimension we consider, i.e. time, social and text, we assign different weights to them to test the flexibility of our scheme. As shown in Fig. 8d and 8h, **3D** remains superior over the other methods. Even when the weight of time dimension is small, i.e. 0.1, **3D** is still better than **TP** where both methods use the time dimension as the primary axis. **3D** does not perform well when the weight of the social dimension is the largest among all dimension weights because the **small world** property makes it hard to differentiate users effectively in term of social distance.

C. Scalability

In order to test the scalability of our proposed solution, we decide to scale along the social graph size and the number of user-posted documents respectively.

First, we test the scalability when the social graph grows while the number of records remains the same. It naturally follows the empirical scenario that users issue real time queries for the most recent posts in a very big social network. In Fig. 12, we find both **TP** and **FP** spend much longer time in the social distance computation. In contrast, **3D** limits the time for distance query to minimal due to the simultaneous 3-dimension pruning upon the 3D list. Therefore, **3D** is capable of handling increased volume of the social graph. We also tested what happens if one of the three pruning techniques, i.e. early-determination, early-pruning and warm-up queue, is missing. The results are shown in Table III where we observe that early-determination is the most powerful pruning. Nevertheless, all prunings must work together to ensure an efficient distance computation with the increasing graph size.

Second, we test each approach w.r.t. varying number of records posted while fixing the social graph sizes to the default values. As we can see from Fig. 13, **3D** remains to be the best against the rest and is able to maintain the performance to near constant. Therefore, it verifies that our proposed method is also scalable against high text volume.

D. Handling Fast Update

Lastly we study how our framework deals with fast and continuously coming new data. The experiment is conducted in this way: we measure the query time, online index update

TABLE III: Performance for evaluating social distances with different pruning disabled. The units are in milliseconds.

Twitter Graph	1M	5M	10M
No Warm-up Queue	266.1	257.4	1987
No Early-Determination	1341	2377	6018
No Early-Pruning	271.9	531.3	2407
all	31.75	124.8	69.34
News Graph	0.1M	0.5M	1M
No Warm-up Queue	2.56	38.3	38.7
No Early-Determination	17.5	67.6	129
No Early-Pruning	1.34	55.9	52.0
all	0.80	11.2	12.8

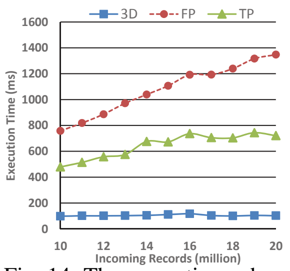


Fig. 14: The query time when records are being ingested.

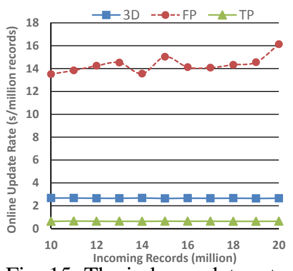


Fig. 15: The index update rate with incoming records.

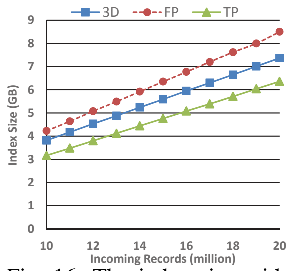


Fig. 16: The index size with incoming records.

rate and index size while new tweets are being ingested continuously from 10M to 20M tweets, and the results are presented in Fig. 14, 15, 16 respectively. Result for the news dataset is similar to Twitter, while not presented due to space limit. We find: (1) For query time, time based pruning methods achieve fairly stable performance while text pruning is getting worse when tweets are being ingested; **3D** outperforms **TP** by 5-7x speedup and **3D** is even more stable against index update. (2) For online index update time, **TP** is the most efficient method due to its simple index stricture. Nevertheless, **3D** also demonstrates its fast-update feature as it clearly outperforms **FP** and the margin against **TP** is quite small. (3) For index size, the space occupied by **3D** is close to **TP** as we stored the 3D list as time slices of trees (Sec. IV) to avoid empty cubes. For **FP**, more space is required as a large tree structure is used to maintain a sorted list w.r.t keyword frequencies.

In summary, equipped with its fast update and efficient space management features, **3D** has the advantage in handling real time personalized query against the other methods.

IX. Conclusion

In this work, we presented a general framework to support real time personalized keyword search on social networks by leveraging the unique characteristics of the SNPs. We first proposed a general ranking function that consists of the three most important dimensions (time,social,textual) to cater to various user preferences. Then, an update-efficient 3D cube index is designed, upon which we devised an efficient Threshold Algorithm called CubeTA. We further proposed several pruning methods in social distance query computation. Extensive experiments on real world social network data have verified the efficiency and scalability of our framework. In future, we would like to study how to design an effective ranking function to provide high-quality personalized results.

Acknowledgement. This work is funded by the NExT Search Centre (grant R-252-300-001-490), supported by the Singapore National Research Foundation under its International Research Centre @ Singapore Funding Initiative and administered by the IDM Programme Office. Guoliang Li is partly supported by the 973 Program of China (2015CB358700) and NSF of China (61373024 and 61422205).

References

[1] X. Hu and H. Liu, "Text analytics in social media," *Mining Text Data*, 2012.
[2] C. C. Aggarwal, Ed., *Social Network Data Analytics*. Springer, 2011.
[3] M. Busch, K. Gade, B. Larson, P. Lok, S. Luckenbill, and J. Lin, "Earlybird: Real-time search at twitter," in *ICDE*, 2012.
[4] C. Chen, F. Li, B. C. Ooi, and S. Wu, "Ti: An efficient indexing mechanism for real-time search on tweets," in *SIGMOD*, 2011.
[5] R. Schenkel, T. Crecelius, M. Kacimi, S. Michel, T. Neumann, J. X. Parreira, and G. Weikum, "Efficient top-k querying over social-tagging networks," in *SIGIR*, 2008.
[6] S. Amer-Yahia, M. Benedikt, L. V. S. Lakshmanan, and J. Stoyanovich, "Efficient network aware search in collaborative tagging sites," *PVLDB*, 2008.
[7] S. Maniu and B. Cautis, "Network-aware search in social tagging applications: instance optimality versus efficiency," in *CIKM*, 2013.
[8] M. Mathioudakis and N. Koudas, "Twittermonitor: trend detection over the twitter stream," in *SIGMOD*, 2010.
[9] M. Qiao, L. Qin, H. Cheng, J. X. Yu, and W. Tian, "Top-k nearest keyword search on large graphs," in *PVLDB*, 2013.
[10] B. Bahmani and A. Goel, "Partitioned multi-indexing: Bringing order to social search," in *WWW*, 2012.
[11] M. Curtiss, I. Becker, T. Bosman, S. Doroshenko, L. Grijincu, T. Jackson, S. Kunnatur, S. Lassen, P. Pronin, S. Sankar, G. Shen, G. Woss, C. Yang, and N. Zhang, "Unicorn: A system for searching the social graph," *PVLDB*, 2013.
[12] J. Yao, B. Cui, Z. Xue, and Q. Liu, "Provenance-based indexing support in micro-blog platforms," in *ICDE*, 2012.
[13] R. Zhong, G. Li, K.-L. Tan, and L. Zhou, "G-tree: An efficient index for knn search on road networks," in *CIKM*, 2013.
[14] K. C. K. Lee, W.-C. Lee, B. Zheng, and Y. Tian, "Road: A new spatial object search framework for road networks," *TKDE*, vol. 24, no. 3, 2012.
[15] F. Wei, "Tedi: Efficient shortest path query answering on graphs," in *Graph Data Management*, 2011.
[16] T. Akiba, Y. Iwata, and Y. Yoshida, "Fast exact shortest-path distance queries on large networks by pruned landmark labeling," in *SIGMOD*, 2013.
[17] R. Agarwal, M. Caesar, P. B. Godfrey, and B. Y. Zhao, "Shortest paths in less than a millisecond," in *WOSN*, 2012.
[18] J. Cheng, Y. Ke, S. Chu, and C. Cheng, "Efficient processing of distance queries in large graphs: A vertex cover approach," in *SIGMOD*, 2012.
[19] A. B. Adcock, B. D. Sullivan, and M. W. Mahoney, "Tree-like structure in large social and information networks," in *ICDM*, 2013.
[20] R. Li, S. Bao, Y. Yu, B. Fei, and Z. Su, "Towards effective browsing of large scale social annotations," in *WWW*, 2007.
[21] X. Li, L. Guo, and Y. E. Zhao, "Tag-based social interest discovery," in *WWW*, 2008.
[22] J. Zobel and A. Moffat, "Inverted files for text search engines," *ACM Comput. Surv.*, vol. 38, no. 2, 2006.
[23] J. Gao, W. Yuan, X. Li, K. Deng, and J.-Y. Nie, "Smoothing clickthrough data for web search ranking," in *SIGIR*, 2009.
[24] G. Karypis and V. Kumar, "Parallel multilevel k-way partitioning scheme for irregular graphs," in *Supercomputing*, 1996.
[25] V. Poosala, P. J. Haas, Y. E. Ioannidis, and E. J. Shekita, "Improved histograms for selectivity estimation of range predicates," in *SIGMOD*, 1996.
[26] R. Fagin, A. Lotem, and M. Naor, "Optimal aggregation algorithms for middleware," in *PODS*, 2001.
[27] E. W. Dijkstra, "A note on two problems in connexion with graphs," *NUMERISCHE MATHEMATIK*, 1959.