Supplementary Document to Towards Longitudinal Analytics on Social Media Data

This document supplements the ICDE submission "Towards Longitudinal Analytics on Social Media Data."In particular, this document presents detail description of the query algorithm and experimental results for the WOR and OR semantics.

1 Query Processing Algorithm on TPII

We propose an algorithm that employs the proposed *TPII* to efficiently answer *TkTK*. The algorithm sequentially scans the tuples in posting lists, which avoids random accesses and thus ensures efficiency even when the *TPII* is stored in-disk. In Algorithm 1, we present the algorithm for processing the queries based on the WOR semantic, which is the most complicated semantic among the three semantics, i.e., AND, OR and WOR. Only minor changes are needed to support the other two semantics.

The algorithm relies on the following variables and data structures. For each candidate social item, we maintain a score lower bound $scoreLB$ and a score upper bound $scoreUB$, representing the lowest and highest possible scores of the social item w.r.t. a given query Q. Both are initialized as 0. In addition, each candidate social item is associated with an interval array $reIntv[]$ that consists of $|Q.K|$ intervals, which records the remaining unvisited intervals for each keyword. All intervals in $reIntv[]$ are initialized to the query interval $Q.I$ (lines 1–2).

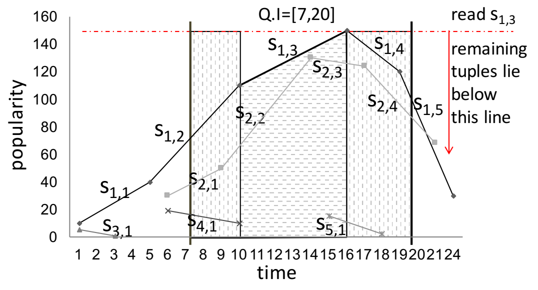
For each keyword $kw_i \in K$, we maintain a popularity upper bound $popUB$ that represents the highest popularity of the items that contains keyword kw_i. It is initialized to the largest popularity of the tuples in kw_i's posting list $PL(kw_i)$, i.e., the maximum popularity of the topmost tuple in $PL(kw_i)$ (lines 3–4).

We use a priority queue $topk$ to maintain the current top-k social items, where the priority is based on a social item's score lower bound $scoreLB$. We randomly choose k social items to initialize $topk$. In addition, we use two sets $cand$ and $pruned$ to record the social items that may become top-k social items and that cannot become top-k social items, respectively. Both sets are initialized to empty sets (line 5).

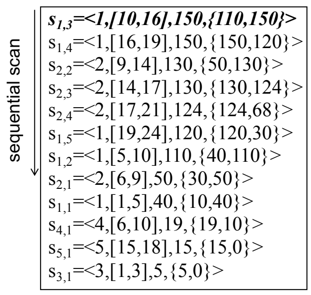
In the following, we consider a simple query $Q = \langle k, \{kw_1, kw_2\}, [7, 20], W\rangle$ on the five social items shown in Figure 1(a) to explain the query processing algorithm.

The algorithm sequentially checks the keywords' posting lists in a round robin fashion (lines 7–8). For each keyword $kw_i \in K$, we sequentially scan the tuples that are maintained in its posting list $PL(kw_i)$ and identify the next tuple tp whose interval $tp.I$ intersects $Q.I$.

When checking keyword kw_1's posting list, the first tuple $s_{1,3}$ is returned as its interval $[10, 16]$ intersects query interval $Q.I = [7, 20]$. Recall that each tuple corre-



(a) Popularity Time Series


sequential scan

$s_{1,3}$=<1,[10,16],150,{110,150}>
$s_{1,4}$=<1,[16,19],150,{150,120}>
$s_{2,2}$=<2,[9,14],130,{50,130}>
$s_{2,3}$=<2,[14,17],130,{130,124}>
$s_{2,4}$=<2,[17,21],124,{124,68}>
$s_{1,5}$=<1,[19,24],120,{120,30}>
$s_{1,2}$=<1,[5,10],110,{40,110}>
$s_{2,1}$=<2,[6,9],50,{30,50}>
$s_{1,1}$=<1,[1,5],40,{10,40}>
$s_{4,1}$=<4,[6,10],19,{19,10}>
$s_{5,1}$=<5,[15,18],15,{15,0}>
$s_{3,1}$=<3,[1,3],5,{5,0}>

(b) Posting List of kw_1

Figure 1:

sponds to a social item's curve. Then, we let p and s denote the social item and the curve that tuple tp corresponds to (lines 9–10). Next, we check whether social item p is in set $pruned$—if so, it means that p cannot be a top-k social item and thus we do not need to further consider p (line 11–12).

Based on tuple tp, we update the popularity upper bound of keyword kw_i to $w_i \cdot tp.mp$ (line 13). Following the running example, $kw_1.popUB$ is set to $w_i \cdot 150$ because $s_{1,3}.mp = 150$.

Then, we update the score lower and upper bounds of social item p (lines 14-21). In particular, we add the current score lower bound $scoreLB$ with the score that tp contributes, denoted as $area(s,\ rt_s,\ rt_e)$ that computes the area of the region that is under segment s and between rt_s and rt_e. Following the running example shown in Figure 1(a), the area under segment $s_{1,3}$ from 10 to 16 on the x-axis is computed as the score that tp contributes, which is (110+150)·(16-10)/2=780.

The score upper bound $scoreUB$ should be at least the same as the lower bound $scoreLB$ (line 16). Then, for the currently considered keyword kw_i, we update its remaining intervals that are left from tp. For the remaining intervals, we compute the largest possible score that a social item can get. For example, given the query interval $Q.I = [7, 20]$, after considering tp with interval $[10, 16]$, the remaining intervals are $[7, 10]$ and $[16, 20]$. Since the tuples in a posting list are ordered according to the descending order of the largest popularity, this means the remaining tuples can at most have the popularity that the current tuple tp has, which is recorded in $kw_i.popUB$. In other words, we consider the areas of the two regions shaded by vertical dotted lines in Figure 1(a). Specifically, based on keyword kw_1, p has a score upper bound 780+(10-7)·150+(20-16)·150=1830.

We need to compute the score upper bounds for all keywords in K and the sum of them is used as the score upper bound of p (lines 19-21). For example, since no tuples have been visited yet in the other query keyword kw_2's posting list, kw_2 is still with the whole query interval $Q.I$. Thus, it contributes $|Q.I| \cdot kw_2.popUB$ to p's score upper bound. Meanwhile, we use another variable $scoreUBunvisited$ to maintain the score upper bound for unvisited items.

Next, we check if p can be a top-k social item. If social item p's score lower bound is greater than the smallest score lower bound of the items in $topk$, this means p may become a top-k social item. Thus, we first remove social item p' from $topk$, where p' is the social item that has the smallest score lower bound and then insert p into $topk$ and insert p' into $cand$.

Algorithm 1: TemporalThresholdAlgorithm

```
Input: TemporalKeywordQuery $Q(k, K, I, W)$
Output: SocialItemSet: $result$
1  foreach social item $p_i$ do
2      $p_i.scoreLB$=0; $p_i.scoreUB = 0$; $p_i.reIntv[\,] = \{Q.I\}$;
3  foreach keyword $kw_i \in K$ do
4      tuple $tp = PL(kw_i)$.firstTuple(); $kw_i.popUB = w_i \cdot tp.mp$;
5  Initialize priority queue $topk$; $pruned = \emptyset$; $cand = \emptyset$;
6  Boolean $terminate$ =false;
7  while not terminate do
8      foreach keyword $kw_i \in K$ do
9          tuple $tp = PL(kw_i)$.next($Q.I$);
10         Identify social item $p$ and curve $s$ of tuple $tp$;
11         if $p$ is in $pruned$ then
12             Continue;
13         $kw_i.popUB = w_i \cdot tp.mp$;
14         $rt_s = \max(Q.I.t_s, tp.tic.t_L)$; $rt_e = \min(Q.I.t_e, tp.tic.t_R)$;
15         $p.scoreLB += w_i * area(s, rt_s, rt_e)$;
16         $p.scoreUB = p.scoreLB$;
17         $p.reIntv[kw_i] = p.reIntv[kw_i] \setminus [rt_s, rt_e]$;
18         $scoreUBunvisited = 0$;
19         for each keyword $kw_j \in K$ do
20             $p.scoreUB$ += $|p.reIntv[kw_j]| \cdot kw_j.popUB$;
21             $scoreUBunvisited$ += $|Q.I| \cdot kw_j.popUB$;
22         if $p.scoreLB >$ minScoreLB(topk) then
23             $p' = topk$.ExtractMin(); topk.insert($p$); $cand$.insert($p'$);
               $cand$.remove($p$);
24         else if $p.scoreUB \leqslant$ minScoreLB(topk) then
25             $pruned$.insert($p$);
26             $cand$.remove($p$);
27         else
28             $cand$.insert($p$);
29         if minScoreLB(topk) $\geqslant$ maxScoreUB(cand) $\wedge$ minScoreLB(topk) $\geqslant$
           scoreUBunvisited then
30             $terminate$ =true;
31             break;
```

If social item p's score upper bound is not greater than the smallest score lower bound of the items in $topk$, this means that p cannot become a top-k social item because there are at least k other social items whose scores should be no smaller than p's score. Thus, we do not need to further consider p by including p into set $pruned$. We also remove p from $cand$ if p used to be in $cand$. Otherwise, p may also become a top-k social item and we insert p into $cand$.

Finally, we can safely terminate the searching process if the following two conditions are met. (1) The smallest score lower bound of the items in $topk$ is greater than or equal to the largest score upper bound of the items in $cand$, meaning that the items in $cand$ can at most have a score that is no greater than the scores of items in $topk$. (2) The smallest score lower bound of the items in $topk$ is greater than or equal to $scoreUBunvisited$, meaning that the unvisited item can at most have a score that is no greater than the scores of items in $topk$.

2 Preprocessing of Dataset

Two real world social media data sets, *Weibo* and *Twitter*, are used. The *Weibo* data set is an event repository, meaning that the tweets, i.e., micro-blogs, are all related to some events. It consists of 13 million tweets which are generated by 1.7 million users between September 2009 and December 2012. The details of the crawling process are included elsewhere [1]. The *Twitter* data set is publicly available[1], which consists of 467 million tweets that are published by 20 million users from June 2009 to December 2009.

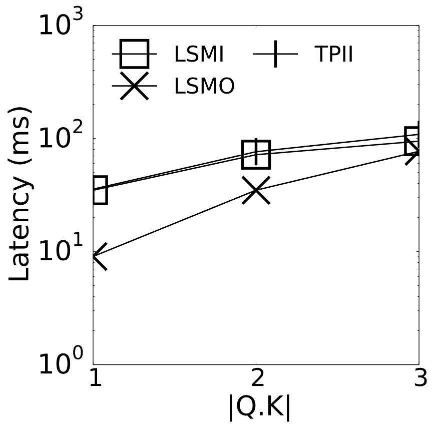
Both data sets do not directly contain the origin relationships among different tweets. We rely on retweet syntax, e.g.,“Rt @user: content” used by Twitter, “//@user: content” used by Weibo, in the textual content to recover their origin relationship to build linkage trees. For example, “a//@user1:b” published by *user2* means that *user2* replies *user1*’s tweet “b” with comment “a”. The regular expressions that is used for detecting such retweet relationships for both data sets is as following:

$$(rt|RT|Rt|Retweet|retweet|//)[\wedge @] * @([\wedge :]+)[:]$$

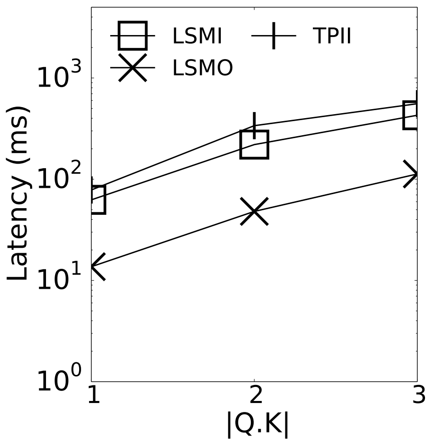
After reconstructing the linkage trees, we build a popularity time series of each tweet. Then, each popularity time series is further represented by a piecewise linear function using a well known method [2].

[1]`http://snap.stanford.edu/data/twitter7.html`

3 Experiments for WOR semantics

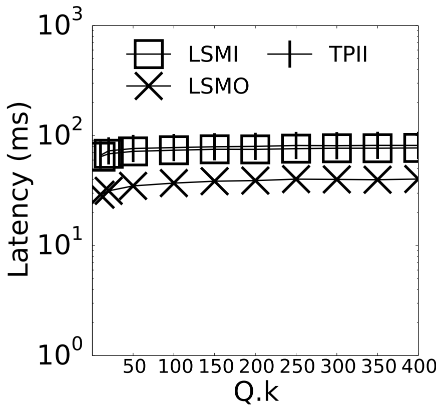


(a) Twitter

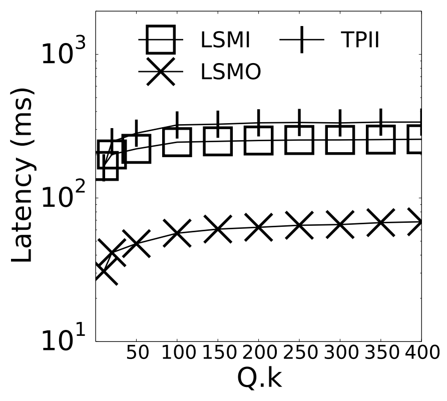


(b) Weibo

Figure 2: Impact of $|Q.K|$

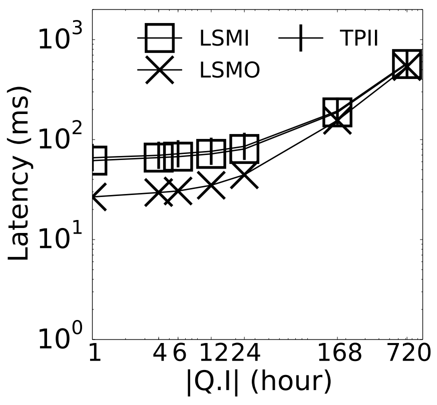


(a) Twitter

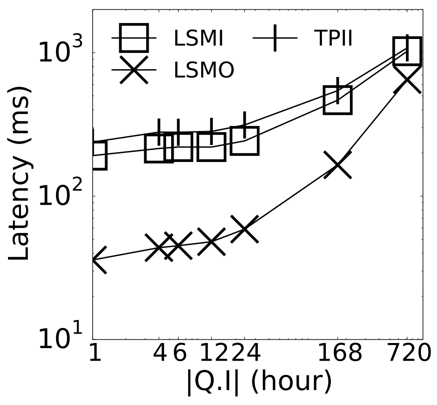


(b) Weibo

Figure 3: Impact of Q.k

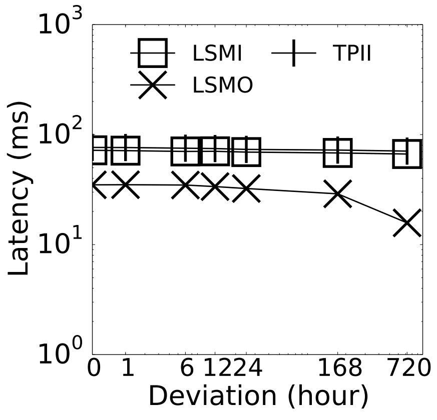


(a) Twitter

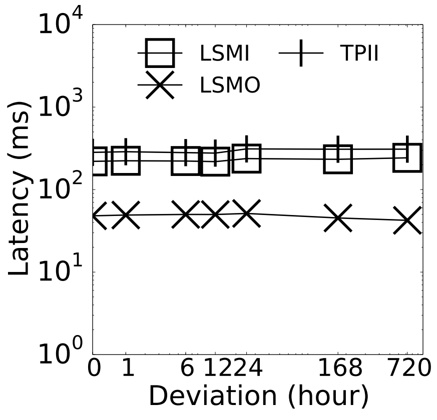


(b) Weibo

Figure 4: Impact of $|Q.I$

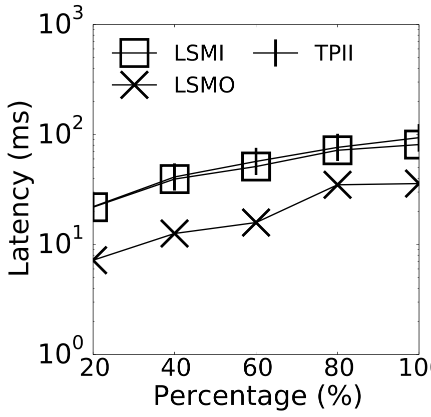


(a) Twitter

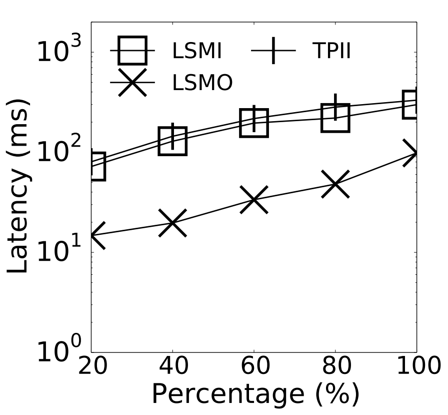


(b) Weibo

Figure 5: Impact of deviation



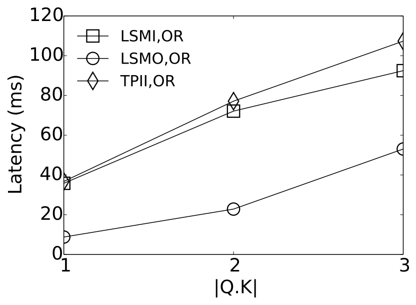
(a) Twitter



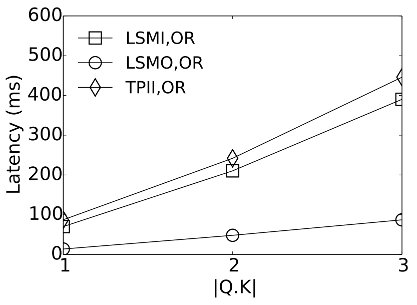
(b) Weibo

Figure 6: Query Performance of scalability experiments

4 Experiments for OR semantics

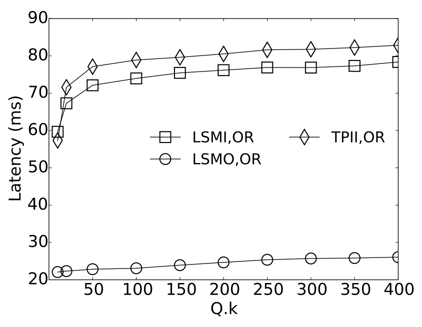


(a) Twitter

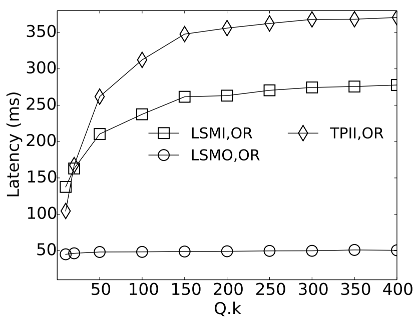


(b) Weibo

Figure 7: Impact of $|Q.K|$

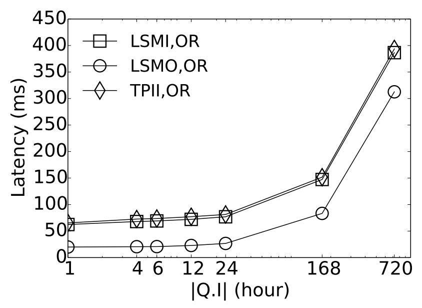


(a) Twitter

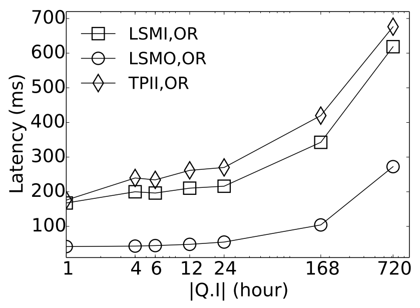


(b) Weibo

Figure 8: Impact of Q.k

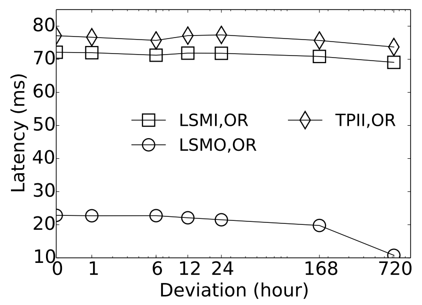


(a) Twitter

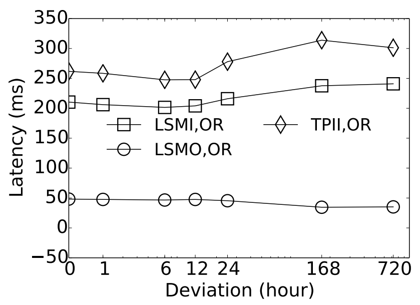


(b) Weibo

Figure 9: Impact of $|Q.I$

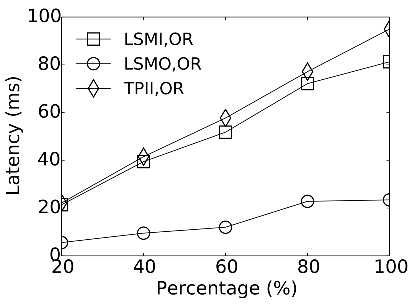


(a) Twitter

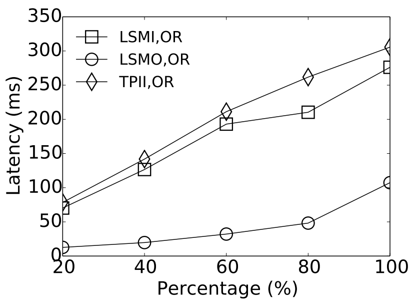


(b) Weibo

Figure 10: Impact of deviation



(a) Twitter



(b) Weibo

Figure 11: Query Performance of scalability experiments

5 Experiments for Effectiveness

We study the effectiveness of TkTK, more specifically, the effectiveness of the popularity definition. To this end, we consider two other definitions of popularity as outlined in our paper. First, the popularity of a social item is defined as the size of the whole subtree rooted at the social item, denoted as WS. Second, the popularity of a social item is defined as the generation time of the social item, denoted as GT. This means that more recent items have larger popularity.

We use Jaccard similarity to measure the overlaps between the results returned by TkTK and the results returned by WS or GT. We run 50 queries using the default parameters except $|Q.I|$ on the Weibo data set. For $|Q.I|$, we consider values from 1 to 720 as stated in the paper. The average Jaccard similarity between TkTK and WS, and that between TkTK and GT are reported in Figure 12.

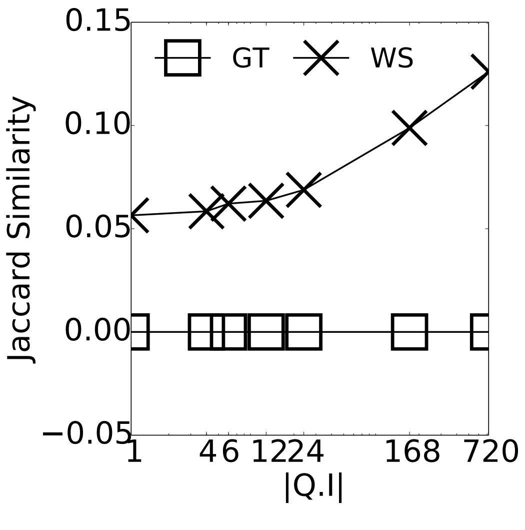


Figure 12: Results of Effectiveness Experiments

References

[1] Haixin Ma, Weining Qian, Fan Xia, Xiaofeng He, Jun Xu and Aoying Zhou. Towards modeling popularity of microblogs. FCS, vol. 7, no. 2, pp. 171-184, 2013.

[2] Eamonn J. Keogh, Selina Chu, David M. Hart and Michael J. Pazzani. An Online Algorithm for Segmenting Time Series. in ICDM, 2001, pp. 289-296.