GeoRank: An Efficient Location-Aware News Feed Ranking System*

Jie Bao[1] Mohamed F. Mokbel[2]
[1] [2]*Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455, USA*
[2] *KACST GIS Technology Innovation Center, Umm Al-Qura University, Makkah, Saudi Arabia*
{baojie, mokbel}@cs.umn.edu

ABSTRACT

News feed function becomes very popular in many social networking services and news aggregators, as it delivers the messages from users' subscribed sources. More recently, location has been introduced to the news feed function, which returns the news items relevant to the user's location. However, with the large number of the news items generated by the sources, existing news feed systems opt to return the top-k most recent ones, which completely overlooks the messages' spatial relevance and may end up in missing more geographically close ones. In this paper, we present GeoRank, an efficient location-aware news feed ranking system that provides top-k new feeds based on (a) spatial proximity, (b) temporal proximity, and (c) user preferences. GeoRank encapsulates spatio-temporal pruning techniques to improve its response time and efficiency. GeoRank is composed of two main modules, namely, *query processor* and *message updater*. The *query processor* module is triggered by the user, upon logging on to the system, to provide the top-k ranked location-based news feeds. The *message updater* module is a process running in the background, which keeps maintaining statistics used by the *query processor* module. Extensive experimental results, based on real and synthetic data sets, show the scalability and efficiency of GeoRank.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Spatial databases and GIS

General Terms

Algorithms

Keywords

location-based social networks, location-based services

1. INTRODUCTION

Social networking services, e.g., Twitter and Facebook, and news aggregators, like iGoogle and MyYahoo!, have become one of the most popular web services. One of their main functionalities is the news feeds [1], where users can receive news items posted from their subscribed sources, e.g., friends or news sources of interest. With the widely usage of location information, many news items are now associated with geographical information, e.g., geo-tagged photos or check-ins. As a consequence, the news feed also becomes location-aware [2, 3], where the news items in the news feed are not only from the subscribed sources, but also relevant to the user's location. However, due to (a) the large number of the relevant news items generated by the users' sources/friends and (b) the limited screen viewing capability, a user may not be interested in all the relevant messages. For example, a desktop user may be interested in the top-50 news items, while a mobile user may be only interested in the top-10 news items. As a result, the existing news feed systems [1, 2, 3] opt to return only the top-k most recent ones. However, with such strict temporal order, a user may end up missing very relevant news items that are very close to the user location, yet, they are not very recent. Moreover, different users may have different tastes for the relevant news items: traveling users may interested in the nearby messages, while stationary users may interested in the recent messages. Thus, there is a strong motivation calling for a new way to produce more "personalized" location-aware news feeds for the user, which include the top-k most relevant news items considering a user's preference.

In this paper, we present GeoRank, an efficient location-aware news feed ranking system, which produces the news feed based on a ranking function considering: (a) the temporal proximity, where more recent news items are favored, (b) the spatial proximity between the location of the news items and the user, where closer news items are favored, and (c) a user defined preference parameter $0 \leq u.\omega \leq 1$, which reflects the relative importance of the temporal and spatial proximity. For example, a traveling user may set a smaller $u.\omega$ to get more geographically close news items, while a stationary user may set a higher $u.\omega$ to include more recent ones.

The goal of GeoRank can be abstracted as an aggregated top-k query from multiple input lists (i.e., the news sources) based on a ranking function. One way to realize the GeoRank is to retrieve the top-k most relevant news items from each of the subscribed news sources and perform a global top-k selection afterwards. However, this approach is extremely inefficient and may introduce significant response delay, because: (1) retrieving the top-k most relevant news items from each news source can be costly, especially when it possesses a large number of news items; and (2) a user may have subscribed to many sources (e.g., a typical user in Facebook follows 150 friends [4]), which results in evaluating a large number of top-k most relevant news items queries. As users may issue the news feed request repeatedly over the time, the user response time and the system efficiency are the main concerns. To this end, Geo-

*This work is supported in part by the National Science Foundation, USA, under Grants IIS-0952977 and IIS-1218168 and by KACST GIS Technology Innovation Center at Umm Al-Qura University, Makkah, Saudi Arabia.

Rank avoids such costly approach by injecting the ranking function deep inside the news feed system. GeoRank employs spatial and temporal pruning techniques that not only avoid retrieving the messages that will not make it to the top-k news feed result, but also avoid evaluating the top-k queries to the news sources who will not contribute any messages to the news feed. In that way, GeoRank significantly reduces the user response time and system overhead.

Although injecting a top-k ranking function inside of query operations, especially those operations that read data from different lists, is a well studied problem (e.g., see [5] for a survey), unfortunately, existing techniques are not applicable to our problem due to the following four main reasons: (1) Most existing top-k techniques require that input lists are sorted on the attribute that contributes to the ranking function. Such requirement is not applicable to our case, as a news source will be subscribed by many users who want to get the news feed at different location. It is not possible to provide an universal ranked spatial order in advance. (2) Existing techniques focus on addressing one top-k query at a time. GeoRank is different, where each user in the social network poses a unique location-aware news feed query to multiple news sources upon logging on to GeoRank. So, different users may retrieve news items for the same input lists. Thus, GeoRank has the opportunity to exploit optimization and indexing techniques that will be shared by all users. (3) Existing techniques have the implicit assumption that number of items in the result (i.e., k) is significantly higher than the number of input lists, which could be only two or three lists. The environment, where GeoRank is applied, exhibits a completely opposite behavior, where a typical value of k could be 30 or 50, while the number of input lists has an average value of 150, e.g., the number of friends for an average Facebook user. Such property gives the room for different spatial and temporal pruning techniques. And (4) Most of the existing top-k techniques overlook the updates in the input list, where the input lists are static. However, in the GeoRannk, the news sources may update new messages continuously, where an efficient updating algorithm is also needed.

GeoRank consists of two main modules, namely, the *GeoRank query processor* and the *GeoRank message updater*. *GeoRank query processor* is mainly responsible for producing the top-k ranked location-aware news items. *GeoRank query processor* employs a two-stage pruning technique. On the first stage, the *query processor* module prunes those news sources (or friends) that will never contribute any news item to the news feed. On the second stage, the *query processor* module exploits spatial and temporal pruning techniques to avoid evaluating all the news items from each candidate news source, i.e., the sources that are not pruned from the first stage. The key for effective pruning techniques in the *query processor* module is the availability of pre-computed statistics that are well maintained for each user, while the user is offline.

On the other side, *GeoRank message updater* module does not really contribute to the answer of any top-k news feed requests. Instead, it is a background process with the sole responsibility of maintaining the set of pre-computed statistics used later by *query processor* module in all its pruning techniques. In a nutshell, GeoRank encounters a system overhead through its *message updater* module in continuously maintaining a set of statistics for each user. However, this offline system overhead is amortized by the savings in online query response time for each user's news feed request. *GeoRank message updater* module is mostly triggered by each submitted new message to check if it will affect any of the existing pre-computed statistics. The efficiency of the *message updater* module stems out from its ability to smartly point out those statistics that will be affected by every new message submitted to the system. Another trigger for the *GeoRank message updater* module is that the user follows a new source, where the *message updater* module will initialize the statistics for the relationship.

Extensive experimental results based on real data sets crawled from Twitter show the efficiency and scalability of GeoRank, along with the effectiveness of its pruning and optimization techniques used in both its *query processor* or *message updater* modules.

The rest of the paper is organized as follows: Section 2 highlights the related work. Section 3 presents preliminaries in GeoRank. Section 4 gives an overview of the system. The details of GeoRank major modules, *query processor* and *message updater* are presented in Sections 5 and 6, respectively. Experimental results are given in Seciton7. Finally, Section 8 concludes the paper.

2. RELATED WORK

This section highlights the related work to GeoRank in three main areas: (1) News ranking/news feed systems, (2) Efficient evaluation of top-k queries, and (3) Answer quality in top-k queries.

News ranking/news feed systems. Most of the existing news ranking systems, e.g., [6, 7], ranks the news by matching the content with the user's profile. Other systems, e.g., [8, 9], keep tracking of the users' clicking behaviors. Most recently, location information has drawn a significant attention in generating more relevant news, e.g., [10, 11], or enabling the location tagging, e.g., [12, 13]. Moreover, MobiFeed [14] provides the relevant news items based a user's predicated traveling path. However, these technique cannot be adapted in GeoRank directly, because all the above techniques assume the users are interested in the news items from all the sources. GeoRank, on the other hand, allows the users to get their news feed from the subscribed news sources.

To incorporate with the social awareness, news feed systems [1] have been proposed, whereas location-aware news feed systems [2, 3] further introduce the spatial relevance. However, these existing news feed systems limit their result as the most recent ones. At the same time, because of its important, several commercial products have also appeared to provide news feed services. However, none of them provides the similar service as GeoRank. For example, Facebook Places consider the location as just an additional tag, where their users get the same news feed regardless of user locations. FourSquare can only provide a view for the geo-tagged messages issued in a nearby venue, yet nothing about ranking the results based on a spatio-temporal function. GeoRank, on the other side, provides the users with a personalized top-k most relevant messages, considering both spatial and temporal proximity.

Efficient evaluation of top-k queries. The top-k query evaluation has been well studied in the literature, e.g., see [5, 15, 16, 17, 18]. All algorithms present extensions or variations from the famous TA algorithm that retrieves objects from a set of input lists, each ordered based on one attribute contributing to the overall ranking function [15, 16]. More recent algorithms have focused on supporting top-k queries on streaming environment and continuous queries, e.g., [19]. As mentioned in the previous section, existing algorithms are not applicable to GeoRank: (1) input lists have to be ordered on a contributing attribute to the ranking function. It is not a valid assumption in GeoRank, whereas the users issue news feed requests from different locations; (2) a top-k query is evaluated in an ad-hoc basis, which is not the case in GeoRank. Although the users issue news feed request for a unique set of sources, there are many cases they may share a part of sources [20]. Thus, shared execution techniques and index structures are needed to further optimize the system performance; (3) most of the existing techniques consider k is significantly higher than the number of input lists, which is completely the opposite in GeoRank, where k is significantly lower than the number of input lists. And (4) most of the existing top-k techniques overlook the updates in the input list. However, in GeoRank, new messages come continuously, where efficient updating method is also essential.

Answer quality in top-k queries. Several methods have been proposed to provide better quality of top-k queries that match the users' different requirements. Examples of such methods include but not limited to the skylines [21], hybrid multi-objective methods [22], and top-k dominance [23]. With the popularity of location-based services, spatial information has been introduced in the top-k rankings, e.g., k nearest neighbor queries [24], where only the distance proximity is considered. Then, spatial skylines [25] and spatial preference queries [26] incorporate other factors in the ranking function. Unfortunately, none of these techniques is directly applicable to GeoRank, as all of them assume that their input is stored in one table or index structure. This is not the case in GeoRank, as the top-k answer may be retrieved from different sources. In addition, GeoRank takes the social aspect, where the top-k messages have to come from the subscribed sources.

Most recently, recommendation techniques have been introduced [27, 28] in top-k rankings to utilize the user behaviors. The location recommendation is either based on mining the user's trajectories [28] or adjusting traditional collaborative filtering techniques to the spatial environment [27]. The latter relies on finding similar users, who do not have to be friends nor have to even know each other. This is different form GeoRank, where users only see the news from the sources of interest.

3. PRELIMINARIES

3.1 Messages, Users, and News Feed

Location-based Messages. Similar to the existing social networking services, each message M from a user u has a timestamp $M.t$ indicating its issuing time. In addition, each message has an explicit geographical coordinate $M.loc$, indicating the location, which can be either the user's current location extracted from a GPS equipped device or the most related location inferred from the contents.

System Users. A user u in GeoRank maintains four main attributes: (1) the user location $u.loc$, which can either be the user current location or a fixed location set in the user profile. In GeoRank, a user's location is static, as in the most cases, users get the news feed frequently from certain locations, e.g., office or home. If a user has more than one location for getting the news feed, we consider her as two different users in the system; (2) a set of followers $u.\mathcal{F}$ that represents the set of other GeoRank users who have indicated their interests in any location-based messages posted by u; (3) a set of sources $u.\mathcal{S}$, that represents the set of other GeoRank users, where u has indicated his/her interest in any location-based messages posted by any of them; and (4) a preference parameter $0 \leq u.\omega \leq 1$, that weights the user preference towards spatial and temporal proximity, described in Section 3.2.

Rank Aware Location-based News Feed. Once the user u logs on to GeoRank, u receives news feed messages that are: (a) issued by one of the users in u's source list, $u.\mathcal{S}$, and (b) among the top-k highest scores, according to the message ranking function, described in Section 3.2, that considers user location $u.loc$, message location $M.loc$, message time $M.t$, user log on time $u.t$ and user preference parameter $u.\omega$. k is a system wide parameter indicating the total number of messages that will be delivered. For example, a typical value of k is 50 for desktop users and 30 for a mobile devices. During a logging session, a user u may post location-based messages that may be seen later by her follower in $u.\mathcal{F}$.

3.2 Message Ranking Function

GeoRank employs a ranking function $Ranking(u, M)$ that gives a relevant score of a message M to a user u. The higher the score the more relevance is the message. The ranking function encapsulates both the temporal and spatial proximity of M to u. Since spatial and temporal proximity have two different domains, i.e., distance in miles and time in hours, the ranking function normalizes the spatial and temporal proximity to have a normalized domain score from 0 to a maximum score *Max*. The ranking function also employs a personalized preference parameter $0 \leq u.\omega \leq 1$ that weights the relative importance to a user. Formally, the message ranking function can be represented as:

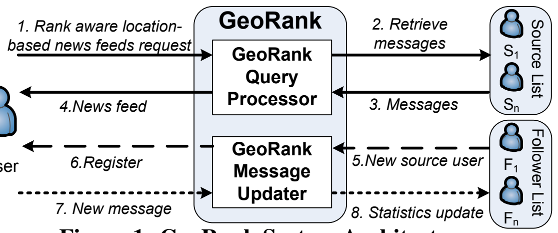


Figure 1: GeoRank System Architecture.

$$\begin{aligned} Ranking(u, M) = {} & u.\omega \times \mathit{NormTemporal}(u.t, M.t) \\ & +(1 - u.\omega) \times \mathit{NormSpatial}(u.loc, M.loc) \quad (1) \end{aligned}$$

A higher value for the user preference parameter $u.\omega$ indicates a higher weight to the temporal proximity than the spatial proximity. At one extreme, setting $u.\omega$ to 0 indicates that the user only cares about the closest k messages, among the ones issued by $u_j \in u.\mathcal{S}$, which is similar to the results of k nearest neighbors (*KNN* queries) [24]. On the other extreme, setting $u.\omega$ to 1 indicates that the user only cares about the most recent messages, which is equivalent to the traditional news feed function [1].

In the mean time, $\mathit{NormTemporal}(u.t, M.t)$ is a normalization function that normalizes the time differences between the user logging on time $u.t$, which is NOW, and the message issue time $M.t$ to scale from 0 to the maximum scoring value *Max*. The temporal normalization function decreases linearly within a range of a predetermined temporal boundary T, where the score of *Max* is given when there is no time difference between the log on time and the message issuing time, while a score of 0 is given to those messages that are older by T time units or more from the log-on time. Similarly, $\mathit{NormSpatial}(u.loc, M.loc)$ is a normalization function that normalizes the Euclidean space difference between the user location $u.loc$ and the message location $M.loc$ to scale from 0 to a maximum value *Max*. The spatial normalization function also decreases linearly, up to a predetermined spatial boundary S.

3.3 Problem Definition

Our problem in GeoRank can be formulated as: "*Given a set of users $\mathcal{U}$ in the system, where each user u has a list of followers $u.\mathcal{F}$, a list of sources $u.\mathcal{S}$, and a preference parameter $u.\omega$. Once a user u logs on the system, GeoRank finds the top-k most relevant news items, based on her current location $u.loc$, log on time $u.t$, and the preference parameter $u.\omega$ from her sources*".

The main contribution of GeoRank is not to create an alternative ranking method for news feed, but to improve its efficiency. In that way, users can enjoy news feed with short response times, while the system is not overwhelmed.

4. GEORANK SYSTEM OVERVIEW

4.1 System Architecture

Figure 1 depicts the system architecture of GeoRank, which is composed of two main modules, namely, the *GeoRank query processor* and the *GeoRank message updater*, briefly described below:

GeoRank query processor. The *query processor* module is automatically triggered for a user u, once u logs on to GeoRank system. The *query processor* module first consults the source list $u.\mathcal{S}$ to find and rank the relevant messages, using the user preference parameter $u.\omega$ and the ranking function $Ranking(u, M)$ (Arrows 1 and 2 in Figure 1). The output of the *query processor* module is a set of k messages as the top-k highest ranked messages among all the messages posted from the user's subscribed sources $u.\mathcal{S}$ (Arrows 3 and 4 in Figure 1). The *query processor* module employs a set of smart pruning techniques to avoid scanning all possible sources and messages. In addition, it relies on pre-computed statistics that significantly enhance the query performance. Details of the *query processor* module will be described later in Section 5.

GeoRank message updater. The main purpose of the *message updater* module is to track the set of statistics used later by the *query processor* module for improving the efficiency. The statistics include the most relevant (or highest ranked) messages for u from each user in its sources $u.\mathcal{S}$. The *message updater* module is triggered by: (1) Adding a new source user, where user u_f follows the message updates from user u (Arrow 5 in Figure 1). In this case, we initialize the most relevant message, among the ones posted from u, for u_f, along with with few other statistics; and (2) A new message M posted by user u (Arrow 7 in Figure 1). In this case, we update the statistics for the user's followers in $u.\mathcal{F}$, with internally used statistics. It is important to note that the *message updater* module does not directly contribute to the answer. Instead, it is a process running in the background to facilitate the mission of the *query processor* module, whenever called. Details of the *message updater* module will be described in Section 6.

4.2 Data Structure

In addition to the typical user data for each user u that includes the user id $u.id$ and location $u.loc$, GeoRank maintains the following data structure for each user u:

A preference parameter ($u.\omega$), which gives the user preference towards the spatial or temporal domains, as described in the ranking function in Section 3.2.

The source list ($u.\mathcal{S}$), as the set of other users that u is interested to receive messages from.

The follower list ($u.\mathcal{F}$), as the set of other users who are interested in receiving messages from u.

List of posted location-based messages ($u.\mathcal{M}$), as the list of prior posted messages from user u. Each message $M \in u.\mathcal{M}$ has a location $M.loc$ and timestamp $M.t$.

Spatial grid index ($u.\mathcal{G}$)**,** as the underlying spatial index structure, which consists of $n \times n$ equal area grid cells. A grid cell $C \in u.\mathcal{G}$ includes all the user followers $u.\mathcal{F}$, sources $u.\mathcal{S}$, and posted massages $u.\mathcal{M}$, whose locations fall within the cell area boundary.

5. GEORANK QUERY PROCESSOR

The query processor is responsible for providing the top-k relevant news feed for user u. A straightforward way to support this functionality is to decompose a user's news feed request into $|u.\mathcal{S}|$ top-k queries, as one query for each source user $u_s \in u.\mathcal{S}$. Then, the final result is assembled by aggregating the overall top-k ranked messages among the ones retrieved from all the source users. The *query processor* module in GeoRank avoids such naive (and prohibitively expensive) solution through a two-step approach that is based on two main concepts: (a) It is not necessary to check all the source users $u_s \in u.\mathcal{S}$ for the relevant messages, as it is most likely that $k < |u.\mathcal{S}|$. For example, in Facebook, an average user has 150 friends (i.e., sources), yet k may be set to 50. This means that we can get all the top-50 ranked messages from at most 50 friends, and (b) It is not necessary to return k messages from each checked source user $u_s \in u.\mathcal{S}$, as most of such returned messages will not qualify for the final top-k answers.

In order to exploit these two concepts, GeoRank query processor follows a two-step approach. The first step, termed *candidate sources selection* (Section 5.2), exploits the first concept to early prune a large number of source users, who will never contribute to any of the top-k news items. Non-pruned sources are considered as candidate sources and checked further in the second step. The second step, termed *news feed aggregation* (Section 5.3), exploits the second concept by using spatial and temporal pruning techniques, along with an early termination condition, to minimize the number of retrieved messages to produce the top-k ranked news feed.

Algorithm 1 gives the pseudo code for *GeoRank query processor*, with its two main steps. The input to the algorithm is the user location $u.loc$, preference parameter $u.\omega$, source list $u.\mathcal{S}$, list of the most relevant messages $u.\mathcal{R}$, grid index structure $u.\mathcal{G}$, and log on time $u.t$, which is set as NOW. Notice that we do not need anything related to the set of user's followers $u.\mathcal{F}$ within the *query processing* module, as none of them will contribute to the messages that the user u will receive in her news feed. The output of the algorithm is the set of highest ranked k location-based messages based on their spatial and temporal relevance to the user u according to u's preference parameter $u.\omega$ and the message ranking function.

5.1 Additional Data Structure

In addition to the data structure described in Section 4.2, the *query processing* module maintains the following data structure:

List of most relevant messages from the sources ($u.\mathcal{R}$), as one message from each source user $u_s \in u.\mathcal{S}$. A most relevant message R_s, posted by a source user u_s, is pre-computed and continuously maintained in GeoRank. R_s represents the highest ranked message posted from the source user u_s with respect to user u according to the user's message ranking function and preference parameter $u.\omega$. This is a key structure in the GeoRank query processor as it plays a major role in all the pruning techniques. It is important to note that the *query processor* module in GeoRank just uses this key data structure to produce the results. However, the computation and the continuous maintenance of this data structure is all done by the *GeoRank message updater* module, described in Section 6.

5.2 STEP 1: Candidate Sources Selection

The *candidate sources selection* step aims to exploit the fact that the number of messages k is highly likely to be less than the number of source users, i.e., $k < |u.\mathcal{S}|$. This means that at least $|u.\mathcal{S}| - k$ sources will never contribute to the top-k news feed. The objective of this step is to find out these $|u.\mathcal{S}| - k$ sources and exclude them, along with their messages, from any further consideration.

Main idea. The main idea of this step relies on the list of the most relevant messages $u.\mathcal{R}$ to prune $|u.\mathcal{S}| - k$ sources. Since this list includes exactly one message from each source user, then we can just take the highest k ranked messages, and only consider these sources users. It means that any source user that does not contribute to the highest k messages in $u.\mathcal{R}$ will never contribute any message to the news feed. The reason is that we can easily create a news feed with k messages (aggregating the top-k ranked messages from the list of the most relevant messages $u.\mathcal{R}$), where all the messages there have higher ranking scores than the highest ranked ones from the excluded source users. As a result, if we can get these top-k ranked source users based on their most relevant message scores, we can easily avoid a significant amount of computations in the news feed processing. However, producing such top-k list based on the most relevant messages from the source users is not trivial, as the only thing that we can pre-compute is the source's most relevant message itself. The reason we cannot store the message score is that the ranking score is mainly depend on the user's log-on time (i.e., $u.t$), which is needed in the temporal normalization, i.e.,

Algorithm 1 GeoRank Query Processing

Input: user location $u.loc$, preference parameter $u.\omega$, source list $u.\mathcal{S}$, most relevant messages $u.\mathcal{R}$, grid index $u.\mathcal{G}$, and log on time $u.t$.
Output: Top-k highest ranked messages.

```
1: //Step 1. Candidate source selection
2: minscore ← 0; count ← 0; $CandList$ ← $\phi$
3: for each message $M_i \in u.\mathcal{R}$ do
4:     score ← $u.\omega \times$ NormTemporal$(u.t, M_i.t)$ + (1-$u.\omega$) $\times$ NormSpatial$(u.loc, M_i.loc)$
5:     if score > minscore OR count < $k$ then
6:         $CandList \leftarrow CandList \cup (u_i$, score$)$
7:         if count > $k$ then
8:             Remove ($u_{min}$, minscore)
9:         end if
10:        minscore ← minimum score in $CandList$
11:        count ← count + 1
12:    end if
13: end for
14: //Step 2. GeoRank news feed aggregation
15: $N \leftarrow k$; Result ← All messages in $CandList$
16: for each source $u_i \in CandList$, ordered by ranking score do
17:     $\mathcal{D} \leftarrow$ NormSpatial$^{-1}(\frac{minscore - u.\omega \times Max}{1 - u.\omega})$
18:     $\mathcal{T} \leftarrow$ NormTemporal$^{-1}(\frac{minscore - (1-u.\omega) \times Max}{u.\omega})$
19:     Result ← Result $\cup$ top-$N$ messages from $u_i$ within $\mathcal{D}$ & $\mathcal{T}$ (retrieved from $u.\mathcal{G}$, ranked by score)
20:     $N \leftarrow k-$ number of items in Result with a score higher than $u_{i+1}.score$
21:     if $N \leq 0$ then
22:         Return top-$k$ messages in Result as the final result
23:     end if
24:     minscore ← The score of the $k$th item in Result
25: end for
26: Return top-$k$ messages in Result as the final answer
```

NormTemporal$(u.t, M.t)$, as a part of the message ranking function (described in Section 3), and not known as a priori. This calls for a online computing for the *current score* of each message in $u.\mathcal{R}$, once u is logged on, as only then, we know about $u.t$ and the score for temporal normalization. The correctness of this approach comes from the fact that even without knowing the score of each message $R_s \in u.\mathcal{R}$, we are still confident that this R_s has the highest ranking score among all messages produced by the source user u_s. We will elaborate more on it, when discussing the *message updater* module (Section 6), which ensures the sanity of $u.\mathcal{R}$.

Algorithm. Lines 2 to 13 in Algorithm 1 gives the pseudo code of the *candidate sources selection* step. The algorithm iterates over all the messages in $u.\mathcal{R}$, while calculating the score of each message based on the user's location, log-on time, and the underlying message ranking function. The source users that may contribute to the top-k highest ranked messages are stored, along with their ranking scores, in the list $CandList$, for the further processing. Meanwhile, *minscore* is set as the minimum score we have in $CandList$.

5.3 STEP 2: News Feed Aggregation

Given a set of *candidate sources* (i.e., $CandList$), produced from Step 1, a naive way to produce the top-k news feed for a user is to just get the local top-k messages from each source $u_s \in CandList$, and then proceed to find the global top-k messages across all the candidate sources. The *news feed aggregation* step aims to avoid such naive way by: (a) minimizing the number of retrieved items from each source $u_s \in CandList$ using spatial and temporal pruning techniques, and (b) avoiding checking all the sources $u_s \in CandList$ through an early termination condition. The output of this step is the requested top-k messages for user u.

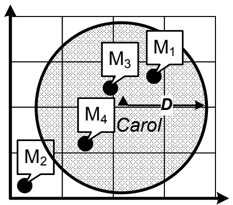
Main idea. The main idea of the *news feed aggregation* step is to use the current minimum score (*minscore*) of all available message ranking scores in $CandList$, to compute both spatial and temporal boundaries that limit the number of messages retrieved from each candidate source. In addition, we incrementally maintain a set of valid candidate messages along with their *minscore*, which is used to update the spatio-temporal boundaries and limit the number of further processed messages from each candidate source user. This can be summarized in the following three ideas:

- *Spatial boundary*. Given that the kth highest ranked message we have so far for u has the score *minscore*, then for a message M to make it among the top-k messages for u, M has to have a higher score than *minscore*, i.e., per Equation 1, $u.\omega \times$ *NormTemporal*$(u.t,\ M.t)$ + (1- $u.\omega$) $\times$ *NormSpatial*$(u.loc,\ M.loc) >$ *minscore*. In order to get a spatial boundary of where the message location $M.loc$ should be, we assume that M has the highest possible temporal score *Max*. In this case, in order for M to make it to the highest top-k messages, M has to be located inside the spatial area $\mathcal{D}$, as follows:
$$\mathcal{D} = NormSpatial^{-1}(\frac{minscore - u.\omega \times Max}{1 - u.\omega}), \quad (2)$$
where *NormSpatial*$^{-1}()$ is the inverse function of the spatial normalization in the user's message ranking function. This means that if a message M is located outside of the area $\mathcal{D}$, M will have no chance in having a higher score than *minscore*, hence will not make it to the top-k items.
- *Temporal boundary*. Similar to the case of determining a spatial boundary, in order to get a temporal boundary of when the message was posted, we assume that M has the highest possible spatial score *Max*. In this case, in order for M to make it to the highest top-k messages, M has to be posted in the last $\mathcal{T}$ time units, computed as follows:
$$\mathcal{T} = NormTemporal^{-1}(\frac{minscore - (1 - u.\omega) \times Max}{u.\omega}), \quad (3)$$
where *NormTemporal*$^{-1}()$ is the inverse function of the temporal normalization in the user's message ranking function. This means that if M was posted older than $\mathcal{T}$ time units, M will have no chance in making it to the top-k items.
- *Number boundary*. We start by the objective of getting the top-k messages. Then, as we visit each source user in the candidate sources, $CandList$, we start to confirm that a certain number of message, x, will definitely be among the top-k ones. In this case, for the next source user to visit, we only look for retrieving at most $|k - x|$ messages. As we keep lowering our number boundary, we early terminate our search when the number boundary reaches 0.

Algorithm. Lines 15 to 26 in Algorithm 1 gives the pseudo code of the *news feed aggregation* step. We initially set our number boundary N as k, and the output result set as the list of messages in $CandList$. Then, we iterate over each source $u_i \in CandList$, in descending order of their most relevant message scores, as the source user with higher score has a higher chance to contribute messages to the news feed result. For each source, we calculate both the spatial and temporal boundaries $\mathcal{D}$ and $\mathcal{T}$, per Equations 2 and 3, respectively. Then, we exploit the spatial grid index structure $u.\mathcal{G}$ to retrieve the highest N ranked messages that lie within our spatial and temporal boundaries from source u_i. The retrieved messages, which could be at most N messages, are inserted to our current result set. Our number boundary N is set by subtracting k with the number of messages in the current result set that has a higher score than the score of the next source user. This is mainly as these messages are guaranteed to be in the top-k result. If the number boundary becomes lower than or equal to zero, we just terminate the algorithm and return the top-k messages in the current result set as the news feed. Otherwise, we set the value of the

MSG	Time	Score
Alice.M$_4$	19:30	8.5
Alice.M$_3$	15:00	6.9
Alice.M$_2$	9:00	-
Alice.M$_1$	8:00	-

Alice's Messages



Alice's Spatial Index

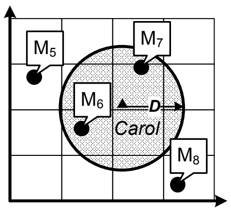
MSG	Score
Alice.M$_4$	8.5
David.M$_7$	7.0
Alice.M$_3$	6.9
minscore	6.9

Carol's Result Set

(a) GeoRank News Feed Aggregation in source user *Alice*

MSG	Time	Score
David.M$_8$	19:30	-
David.M$_7$	17:00	7.0
David.M$_6$	16:40	6.2
David.M$_5$	15:00	-

David's Messages



David's Spatial Index

MSG	Score
Alice.M$_4$	8.5
David.M$_7$	7.0
Alice.M$_3$	6.9
minscore	6.9

Carol's Result Set

(b) GeoRank News Feed Aggregation in source user *David*

Figure 2: Example of GeoRank News Feed Aggregation Step.

new minimum score (*minscore*) as the score of the kth item in the current result set, and proceed to check on the next source u_{i+1}. The algorithm continues till we either have our number boundary $N \leq 0$, or go through all the sources in the $CandList$. The latter case corresponds to the unlikely case, where each source $u_i \in CandList$ contributes exactly one message to the user's news feed.

Example. *Carol* requests her top-3 location-based news feed at time 20:00. Then, assuming that the *candidate sources selection* step returns three candidate sources, i.e., *Alice*, *David*, and *Bob*, with the their most relevant message scores of 8.5, 7.0, and 6.7, respectively. Thus, the minimum score (*minscore*) is set to 6.7. Figure 2 illustrates the *news feed aggregation* step that will be executed for *Carol*, where we go through the three candidate sources based on their scoring order, i.e., we first start with user *Alice*. Figure 2a depicts *Alice*'s four messages, with their issuing times and locations, marked as black dots on her grid spatial index. Assuming that we have calculated the temporal boundary $\mathcal{T}$ using Equation 3 to be 15:00 and the spatial boundary $\mathcal{D}$ using Equation 2 to be the circle around *Carol*'s location in *Alice* spatial index. Since *Alice* is the first user to check for, our number boundary N is initialized by k=3. Based on the temporal, spatial, and number boundaries, we only need to retrieve two messages from *Alice*, namely, M_4 and M_3, with scores 8.5 and 6.9, respectively. Among M_4 and M_3, we know for sure that M_4 will make it to the final answer as it scores higher than the most relevant message score of the next source user, i.e., *David*. However, we are not yet sure about the fate of M_3. Since M_3 has a lower score than the highest scored message from *David*, then there is a probability that *David* may have two messages higher than M_3. With this, we update our number boundary to be $N = 2$, indicating that we are still looking for two more messages from *David*. Also, the minimum score is updated to 6.9 as the kth ranked message we have so far, which is M_3.

Figure 2b depicts *David*'s messages. With the updated minimum score, the temporal boundary $\mathcal{T}$ becomes tighter as 16:00 while the spatial boundary $\mathcal{D}$ is depicted by a smaller circle. Only two messages satisfy the new spatial, temporal, and number boundaries, namely, M_7 and M_6 with scores 7.0 and 6.2, respectively. With this, we know that for sure M_7 and M_3 will be in the final result, as both of them score higher than 6.7, which is the highest message score from *Bob*. So, we just update our number boundary to be 0. As this is our stopping criteria, we terminate the algorithm without visiting *Bob*, where news feeds include M_4, M_7, and M_3, in order.

6. GEORANK MESSAGE UPDATER

As discussed in the previous section, the *query processor* module mainly relies on the list of most relevant messages, $u.\mathcal{R}$, in all its pruning techniques. However, the *query processor* module has dealt with this list as a given input, and has nothing to do with computing and maintaining it. In this section, we discuss *GeoRank message updater* module, where its main purpose is to ensure the sanity and accuracy of the list $u.\mathcal{R}$. For a user u, computing the most relevant message R_s from a source $u_s \in u.\mathcal{S}$ includes two steps, *initialization* and *maintenance*. The *initialization* step takes place when a new source user u_s is followed by the user u. Then, u will need to compute an initial value of R_s, based on the messages from the source user u_s, i.e., $u_s.\mathcal{M}$. The *maintenance* step is triggered with each new message posted from the source user $u_s \in u.\mathcal{S}$, where we will need to check if the new message has a higher score to u than the current most relevant message from u_s.

The challenge here comes from the fact that each new message from a source user $u_s \in u.\mathcal{S}$ is not only relevant to u, but, it is also relevant to all those users that consider u_s as one of their sources, i.e., all the users in the follower list $u_s.\mathcal{F}$. A straightforward solution can go as follows: Once a source user u_s submits a new message, we scan all the followers $u_f \in u_s.\mathcal{F}$ to check if we need to update the most relevant message with the new message for any of these users. This straightforward solution can be extremely inefficient, where users may have large number of followers and produce large number of messages, e.g., according to the Facebook statistics [4], an average user has 150 friends and creates over 90 pieces of content each month. Doing an exhaustive search of every posted message over every user's follower may be prohibitively expensive. *GeoRank message updater* module avoids such prohibitively expensive operations by employing spatial filters that limit the number of followers to check for. This is done by associating a set of $|u.\mathcal{F}|$ monitoring areas for user u, as one monitoring area per follower $u_f \in u.\mathcal{F}$. Then, whenever u posts a new message M at location $M.loc$, we do not need to check for all u's followers. Instead, we only check on those followers, whose monitoring areas overlaps with the new message location $M.loc$.

6.1 Additional Data Structures

In addition to the data structures described in Section 4.2, the *message updater* module maintains the following data structure:

List of monitoring areas for the followers ($u.\mathcal{A}$), as one monitoring area for each user's follower $u_f \in u.\mathcal{F}$. A monitoring area A_f is initialized and maintained by the *message updater* module to significantly reduce the computational costs for updating the list of most relevant messages for that follower $u_f.\mathcal{R}$. A monitoring area A_f basically says that in order for a new message M from user u to make it to the list of most relevant messages for the follower u_f, then, M has to be located inside A_f. All the follower's monitoring areas are laid out in the user's spatial grid index structure, $u.\mathcal{G}$.

6.2 Initialization: New Source Update

Whenever a user u_1 decides to follow the message updates from another user u_2, u_2 is added to the list of source users of u_1, i.e., $u_1.\mathcal{S}$ = $u_1.\mathcal{S} \cup u_2$, and, at the meanwhile, u_1 is added to the list of followers of u_2, i.e., $u_2.\mathcal{F}$ = $u_2.\mathcal{F} \cup u_1$. Then, the *message updater* module needs to take two actions: (1) initialize the most relevant message $R_2 \in u_1.\mathcal{R}$, where R_2 is the most relevant (highest ranked) one to user u_1, among all other messages posted from u_2; and (2) initialize the monitoring area $A_1 \in u_2.\mathcal{A}$ in user u_2, which says that any newly posted message from u_2 that is located outside area A_1 will never make it to the list of most relevant messages of

u_1, $u_1.\mathcal{R}$. The two initialization actions are outlined below:

Action 1: Initializing the most relevant message $R_2 \in u_1.\mathcal{R}$. To perform this action, we need to find out the most relevant message to u_1, among all the ones posted by $u2$. Since all posted messages by u_2, $u_2.\mathcal{M}$, are sorted by their issuing times, we just scan the list $u_2.\mathcal{M}$, and calculate the score of each message based on the temporal order. We first consider the latest message as the most relevant one. Assuming that the score of this message is *MaxRelScore*, then we update the most relevant message whenever we find a message with a higher score than *MaxRelScore*, and accordingly adjust the value of *MaxRelScore*. Notice that a message M_2 with a lower temporal score than message M_1 can still have a higher overall score than M_1, if M_2 has a higher spatial score than M_1. We early terminate our scanning process if the next message to visit in $u_2.\mathcal{M}$ cannot score higher than *MaxRelScore*, regardless of its proximity to the location of u_1. In this case, we know that none of the subsequent messages will score higher than *MaxRelScore*, also regardless of its proximity to u_1. To judge on this early termination procedure, we will assume that the next message to visit M_n has the maximum spatial score *Max*. Then, M_n score will be: *Ranking*(u_1,M_n) = $u_1.\omega$ × *NormTemporal(u_1,M_n)* + $(1 - u_1.\omega)$ × *Max*. This means that we can early terminate, if this score is less than *MaxRelScore*, i.e., the temporal score of the next message M_n is less than $\frac{MaxRelScore-(1-u_1.\omega)\times Max}{u_1.\omega}$. Finally, it is important to note that we only store the most relevant message R_2 in $u_1.\mathcal{R}$ without its score *MaxRelScore*. This is mainly because this score will be irrelevant when time advances, as its value is based on the temporal score, which can only be computed with the current time.

Action 2: Initializing the monitoring range area $A_1 \in u_2.\mathcal{A}$. To perform this action, we will need to find an area A_1 such that if a new incoming message M from u_2 is located outside A_1, then we can safely conclude that M will never make it to the list of most relevant messages for user u_1, $u_1.\mathcal{R}$. We will use the most relevant message R_2, computed in Action 1, as an estimation of the initial value of A_1. We can safely say that if a new incoming message M has less score than that of R_2, then, M will have no chance in replacing R_2. Although we know the current exact score of R_2, we cannot rely on this score when comparing it with the score of the new message M, as the score of R_2 decays over time, where its issuing time becomes further. To make R_2 and M comparable, we use a very conservative estimation of the score of R_2 at the time when u_2's new message M is issued. The conservative estimation is obtained by considering that the temporal score of R_2 has decayed to its lowest possible value 0. In this case, the minimum possible score of R_2 is *MinScore* = (1-$u_1.\omega$) × *NormSpatial*($u_1.loc$,$R_2.loc$). Then, with another conservative assumption, we assume the incoming message M has the highest possible temporal score *Max*. Then, for M to have a higher score than R_2, the following criteria should hold: *Ranking*(u_1,M) > *MinScore*, i.e., $u_1.\omega$ × *Max* + $(1-u_1.\omega)$ × NormSpatial(u_1,M) > *MinScore*. This means that A_1 is a circular centered at $u_1.loc$ with a radius $A_1.r$, computed per the following equation:

$$A_1.r = \mathit{NormSpatial}^{-1}(\frac{\mathit{MinScore} - u_1.\omega \times \mathit{Max}}{1 - u_1.\omega}), \qquad (4)$$

Finally, the computed monitoring area A_1 is laid out on the source user's grid spatial index $u_2.\mathcal{G}$.

6.3 Maintenance: New Message Update

Whenever a user u posts a new message M, u needs to check if M will affect any of the most relevant messages R_u for the followers, $u.\mathcal{F}$. If so, we update the most relevant messages accordingly, along with their corresponding monitoring areas stored at u.

Main idea. The main idea of the maintenance step is to exploit the grid spatial index structure maintained at u, $u.\mathcal{G}$, to early prune followers that will not be affected by the new message M. Such early pruning avoids an exhaustive scan over all followers of u. For those followers who are not pruned, we still do an extra check to see if M actually affects their most relevant message from u, as we used the conservative way to calculate the monitoring areas. If this is the case, we update the most relevant message for the follower with the new message. Finally, we use the updated most relevant message to calculate a new monitoring area for such followers, in a similar way to the initialization module discussed in section 6.2.

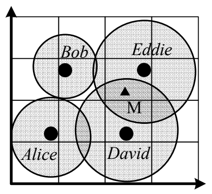
Algorithm 2 GeoRank Message Updating

```
Input: A message $M$ posted by user $u$
 1: $AffectedList \leftarrow \phi$
 2: $C \leftarrow$ The grid cell in $u.\mathcal{G}$ that includes $M.loc$
 3: for each monitoring area $A_f \in u.\mathcal{A}$ located in cell $C$ do
 4:     if $M.loc$ is inside $A_f$ then
 5:         $AffectedList \leftarrow AffectedList \cup u_f$
 6:     end if
 7: end for
 8: for each follower $u_f \in AffectedList$ do
 9:     $R_u \leftarrow$ Retrieve $u$'s most relevant message to $u_f$ from $u_f.\mathcal{R}$
10:     $MostRelScore \leftarrow u_f.\omega \times NormTemporal(\text{NOW}, R_u.time) + (1-u_f.\omega) \times NormSpatial(u_f.loc, R_u.loc)$
11:     $NewMsgScore \leftarrow u_f.\omega \times Max + (1-u_f.\omega) \times NormSpatial(u_f.loc, M.loc)$
12:     if $NewMsgScore > MostRelScore$ then
13:         $R_u \leftarrow M$ in $u_f.\mathcal{R}$
14:         $MinScore \leftarrow (1-u_f.\omega) \times NormSpatial(u_f.loc, M.loc)$
15:         $A_f.r \leftarrow NormSpatial^{-1}(\frac{MinScore - u_f.\omega \times (Max)}{1-u_f.\omega})$
16:     end if
17: end for
```

Algorithm. Algorithm 2 gives the pseudo code for the *message updater* module upon receiving a new message M from user u. The pseudo code has two main steps:

(1) Finding out the list of followers that may be affected by the new message M (Lines 1 to 7 in Algorithm 2). We do so by first locating the cell C in $u.\mathcal{G}$ that includes the message location $M.loc$. Then, for any follower u_f whose monitoring area A_f is registered in cell C, we do an extra check to see if $M.loc$ is located inside A_f. If this is the case, u_f will be added in the list of affected followers.

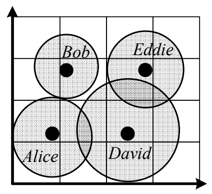
(2) For each affected follower u_f, we update its most relevant message R_u with the new message M posted from u, if needed. In case that the message update takes place, we also update the monitoring area A_f at user u (Lines 8 to 17 in Algorithm 2). We do so by doing the following for each user u_f in the list of affected followers: (a) We retrieve R_u from the list $u_f.\mathcal{R}$ as the current most relevant message from user u to follower u_f, (b) As we do not have the score of R_u, we will need to calculate R_u score (i.e., *MostRelScore*) based on the current time and the follower location. It is important to note that we could not store this score with R_u as it decays over time, and has to be recomputed with every time instance. Thus, we opt to compute it only when needed, (c) We compute the score of the new message M (i.e., *NewMsgScore*) considering that the temporal score is of a maximum value as M is just posted now, while the spatial score is computed based on the proximity of message location to the follower's location $u_f.loc$, and (d) We compare the score of the most relevant message (*MostRelScore*) against that of the new message M (*NewMsgScore*). If the new message has a lower score, we just do nothing, as the new message will not affect anything in our maintained data structure. On the other hand, if the new message has a higher score than that of the currently most relevant message, we first replace the currently most relevant message R_u by the new message M. Then, in a similar way to what we have done in section 6.2, we



(a) *Affect Follower Selection*

Follower	R_u	Most Rel	New Msg
Alice	M_3	-	-
Bob	M_4	-	-
David	M_2	7.5	7.3
Eddie	M	7.1	7.9

(b) *Most Relevant Message Update*



(c) *Monitoring Range Update*

Figure 3: Example of Message Update in GeoRank.

calculate the minimum possible score of M using only its spatial score, and use this value to update the radius of the monitoring area A_f for that follower in the user's spatial index $u.\mathcal{G}$.

Example. Figure 3 gives an example of the maintenance algorithm in *GeoRank message updater*. Figure 3a depicts the spatial grid index at user *Carol*, where she maintains four circular monitoring areas that correspond to her four followers *Alice*, *Bob*, *David*, and *Eddie*, along with the message M posted from *Carol* (depicted by a small triangle). As the location of message M is located outside the monitoring areas of *Alice* and *Bob*, we just early prune these two followers, as we know for sure that their most relevant message will not be affected by M. Figure 3b depicts the further actions taken on the remaining users, *David* and *Eddie*, where we calculate their scores of the new message M based on their preference parameters and locations, which, ended up to be 7.3 and 7.9, respectively. Assume that the prior most relevant scores at *David* and *Eddie* are 7.5 and 7.1, respectively. By comparing the new computed scores for M by the old most relevant scores, we find that M has a lower score than what *David* already has, so, we just exclude *David* from any further considerations. In the mean time, we find that M actually gives a higher score that what *Eddie* already has. In this case, we do two actions: (1) Update the most relevant message R_{carol} at *Eddie* to be M, and (2) Update the monitoring area of *Eddie* to be tighter based on the new message M (Figure 3c).

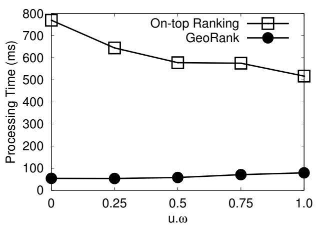
7. EXPERIMENTAL EVALUATION

Experimental evaluations of GeoRank are based on an actual system implementation in PostgreSQL database management system [29]. Experiments are based on a set of 10 Million geo-tagged Twitter messages (i.e., tweets) issued within the state of Minnesota, US, generated as follows: First, we crawled the twitter message data via Twitter Search API[1] for one week. Then, we got $\approx$ 650K distinct geo-tagged tweets, where the geographical information is represented as either a semantic location, e.g., a city name or latitude/longitude coordinates. In the former case, we use Google GeoCoding API[2] to convert into latitude/longitude coordinates. We make use of these real 650K tweets to get to know the real spatial and temporal distributions the tweets. Finally, we generate synthetic 10,000 GeoRank users, where each user is associated with 1,000 geo-tagged tweets. The locations and issuing times of all the 10 Million messages mimic the spatial and temporal distributions of our crawled real tweets. Also, locations of the 10,000 users are static and set randomly based on the locations of the real tweets.

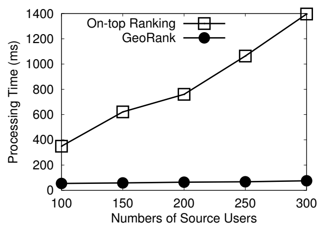
Mostly taken from Facebook statistics [4], and unless mentioned otherwise, each user is following an average of 150 users (i.e., sources) and is also followed by another 150 users (i.e., followers), randomly picked from the user set. We set the default k as 30, which means the user likes to see the top-30 messages in the news feeds. We use a simple ranking function, where *Max* is set to 10, and every one mile or one hour is equivalent to 0.1. The default user preference parameter ω is set to 0.5. Finally, a 10×10 grid structure is constructed for each user to index message locations and the monitoring areas. All experiments were evaluated on a server computer with Intel Core 2 Quad CPU 2.83GHz processor and 8 GB RAM running Ubuntu Linux 10.04.

[1]Twitter Search API: http://search.twitter.com.

[2]Google GeoCode:http://maps.googleapis.com/maps/api/geocode/



(a) Different User Preferences.



(b) Different Source Users.

Figure 4: Compare With "On-Top" Approach.

7.1 GeoRank Overall Performance

In this section, we compare the overall performance of GeoRank against the simple *on-top* approach. In the *on-top* approach, we issue one top-k query to each source user and aggregate the top-k results to produce the news feed. We still optimize in the *on-top* using the temporal order of the messages for the early termination.

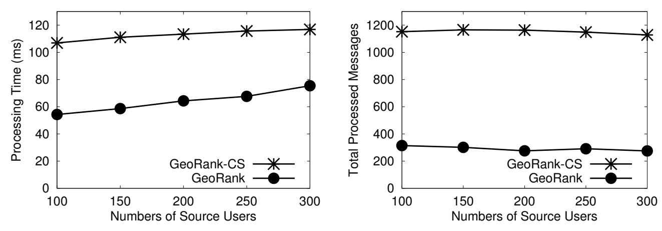
Figure 4a gives the performance of both approaches when varying the user preference parameter $u.\omega$ from 0 to 1. GeoRank consistently gives 6 to 10 times better performance than the *on-top* approach. This is mainly because GeoRank reduces the number of evaluated source users, i.e., we do not have to check on each source user, and also reduce the number of processed messages at each source. Another thing to note is that with the increasing of $u.\omega$, the *on-top* approach gets a better performance. This is because the temporal early termination technique is more effective with the more weight to the temporal domain. This is not the case in GeoRank as it gives the same good performance for all ω.

Figure 4b gives the performance, when varying the number of source users from 100 to 300 to simulate the users following different numbers of sources. In the experiment, each user follows randomly selected source users and we evaluate the average news feed processing time. As shown in the figure, GeoRank scales up well with the increase of the number of source users, while the performance of the *on-top* solution deteriorates significantly. Moreover, the performance gains by GeoRank increases significantly, when the querying user follows more source users. Especially for the case of 300 sources, GeoRank gives 28 times better performance than the *on-top* approach. The main reason is because of the pruning power at GeoRank avoiding all the disqualified source users, while *on-top* approach still needs to check on all source users. Essentially, GeoRank does not care about the total number of source users followed by the querying user, it will always process up to k source users for the news feed. As a result, we can infer that GeoRank will be even more efficient comparing with the *on-top* approach for the active users, who may follow hundreds or thousands of other users in a real social networking system.

Based on the above experiments, GeoRank outperforms the *on-top* approach significantly in all the cases. Thus, we can safely conclude that GeoRank will reach a better throughput than the *on-top* approach. Moreover, we find that the *on-top* approach is impractical due to its unacceptable performance (i.e., the response times), and hence will not consider it in the further experiments.

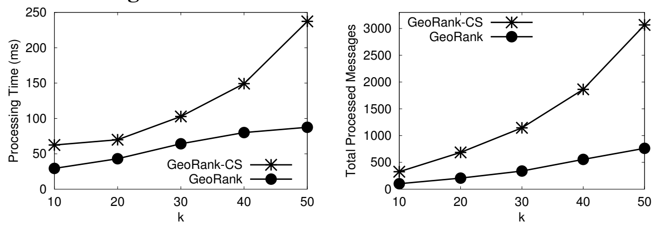
7.2 GeoRank Query Processor Performance

This section studies the performance of the internals of GeoRank *query processor*, i.e., the processing time to produce the news feed. As discussed in Section 5, GeoRank *query processor* has two main steps: *candidate sources selection* and *news feed aggregation*. To study the effect of each step separately, we compare two versions



(a) Processing Time. (b) Total Processed Messages.

Figure 5: Different Numbers of Source Users



(a) Processing Time. (b) Total Processed Messages.

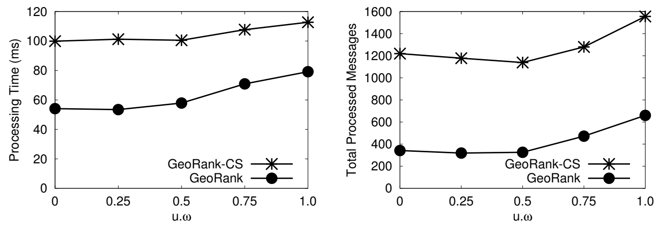
Figure 6: Different Numbers of K Values.

of GeoRank. The first version, termed *GeoRank-CS* applies only the *candidate sources selection*, while the second one is equipped with both the pruning steps. The comparison is done when varying the number of source users, k, and ω.

Number of source users. Figure 5 gives the performance of both GeoRank and GeoRank-CS when varying the number of source users from 100 to 300. The performance is measured in terms of processing time (Figure 5a) and number of processed messages (Figure 5b). The processing time for both GeoRank and GeoRank-CS increases with the number of sources, as we need more time to calculate and rank the scores of most relevant messages in the first step. The performance gap between GeoRank and GeoRank-CS shows the effect of the *news feed aggregation* step, employed by GeoRank, where it: (a) prunes more sources than GeoRank-CS with the early termination condition, and (b) employs both spatial and temporal pruning techniques to avoid retrieving k messages from each candidate source. With number of processed messages (Figure 5b), it is interesting to see a consistent behavior for both GeoRank-CS and GeoRank, as the number of processed message is almost not affected by the number of sources. This is mainly as in both algorithms, the *candidate selection step* prunes the list of sources to 30, i.e, the value of k. As a result, no matter how many sources we have, we only operate on the top-30 of them.

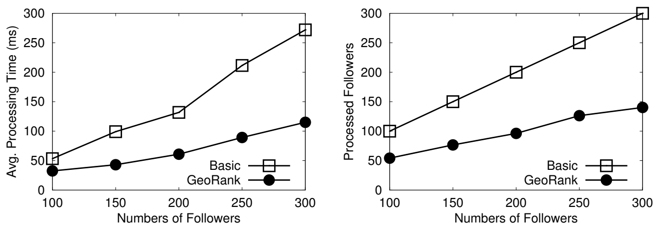
Top k values. Figure 6 gives the performance of both GeoRank and GeoRank-CS, when varying k from 10 to 50, in terms of processing time (Figure 6a) and number of processed messages (Figure 6b). With the increase of k, Both GeoRank and GeoRank-CS take more time to produce the answer and process more messages, as both of them need to check more sources and messages. However, it is interesting to see that the performance gain of GeoRank increases with k. This shows that with larger k, the *news feed aggregation* step in GeoRank becomes even more effective, because of the pruning and early termination. Also, it is obvious that the trend of processing time is similar to the trend of the number of processed messages, which gives insight that the processing overhead is mainly due to the processed messages.

User preference parameter u.ω. Figure 7 gives the performance of both GeoRank and GeoRank-CS when varying ω from 0 (spatial proximity is mostly favored) to 1 (temporal proximity is mostly favored), in terms of processing time (Figure 7a) and number of processed messages (Figure 7b). For all values of ω, there is a consistent performance gap between GeoRank and GeoRank-CS. Although ω plays a major role in determining both the spatial and



(a) Processing Time (b) Total. Processed Messages.

Figure 7: Different User Preferences.



(a) Processing Time. (b) Processed Followers.

Figure 8: Different Numbers of Followers.

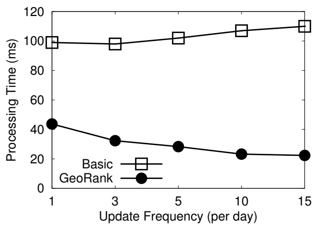
temporal boundaries in the *news feed aggregation* step, increasing or decreasing the value of ω just tightens one boundary and relaxes the other. Overall, the performance gain becomes consistent. In the mean time, for $\omega > 0.5$, we can see that the number of processed messages increases for both GeoRank and GeoRank-CS, hence, the performance degrades. This is mainly due to the distribution of messages is more clustered towards the time domain, and hence favoring the temporal domain results in processing more messages.

7.3 GeoRank Message Updater Performance

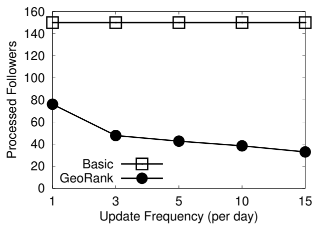
This section studies the performance of GeoRank *message updater* module, which is the overhead to maintain the statistics. As discussed in Section 6, GeoRank *message updater* employs a set of monitoring areas, indexed by a spatial grid index, to determine which followers will be affected by the new message. To evaluate the main idea of the *message updater* module, we compare the full version of GeoRank against a *basic* approach for the *message updater* module, which scans all the followers for most relevant message updates whenever a new message is posted. We discuss the results varying the number of followers, update frequencies, and ω.

Number of followers. Figure 8 gives the performance of both GeoRank and *basic* approaches when varying the number of followers $|u.\mathcal{F}|$ from 100 to 300. In terms of average processing time for each posted message (Figure 8a), both GeoRank and *basic* encounter more time with the increase of the number of followers. However, *basic* approach uses more time with more followers, as it needs to check all the followers. On the other side, GeoRank is more efficient, as it utilizes the monitoring areas and only updates a subset of followers, as we plot the number of followers (Figure 8b). For example, in the case of 300 followers, the *basic* approach checks on 300 followers while GeoRank checks on only 140 followers.

Message update frequencies. Figure 9 gives the performance of both GeoRank and *basic* approaches when varying the message update frequency from 1 to 15 messages per day. In terms of average processing time for each posted message (Figure 9a), the *basic* approach suffers from an increase in processing time with higher frequencies, where we need to check on every follower for update. In the mean time, GeoRank shows much better scalability where it actually gives better performance with higher update frequency. The main reason behind this scalability is that with higher update frequencies, GeoRank can easily update the monitoring areas for the user followers to be tighter and more accurate. Thus, higher update frequency results in a much better performance for GeoRank. In

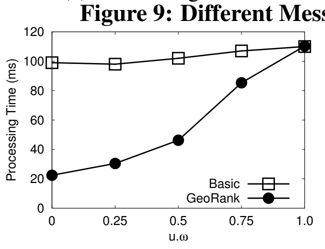


(a) Processing Time.

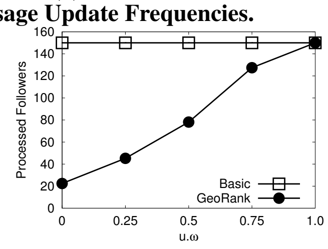


(b) Processed Followers.

Figure 9: Different Message Update Frequencies.



(a) Processing Time.



(b) Processed Followers.

Figure 10: Different User Preference Parameters.

terms of the number of followers to check on (Figure 9b), the *basic* approach gives steady state performance as it basically checks for all the 150 followers. In the mean time, GeoRank checks for lower number of followers with the increase of the update frequency. This is mainly due to the tighter and more accurately calculated monitoring areas, as previously discussed.

User preference parameter. Figure 10 gives the impact of different follower's preference parameter $u.\omega$ on the message update performance of GeoRank and *basic* approach, where we set the same preference parameter for all the user's followers varying from 0 (only cares about location) to 1 (only cares about time). Figure 10a gives the average processing time, and Figure 10b gives the average number of followers processed by each approach for one message update. We have the following observations: (1) the basic approach is almost not affected by the ω at all, as it checks the most relevant message update for all the followers, regardless of their preference parameter ω, (2) GeoRank is more scalable than the *basic* approach as it only updates a subset of the followers for each message update, and (3) GeoRank has a better performance, when the follower's preference parameter is smaller (cares more of the spatial domain). This is because the monitoring areas are calculated based on Equation 4, where with a larger preference parameter $u_f.\omega$, we will have a smaller monitoring area registered at the user's spatial index. As a result, less number of followers is selected for most relevant message updates, when a new message is posted from the user. On the other hand, when the follower's preference parameter is larger (i.e., cares more about time), the processing time of GeoRank increases, as each follower has a larger monitoring area and more followers are selected for update when a new message is posted. For example, in the extreme case $u_f.\omega = 1$ (meaning that the followers only cares the most recent message from the source users), GeoRank has the same performance as the basic approach, because, in this case, the monitoring area is set as the whole space, and all the followers need to be selected for the most relevant message update.

8. CONCLUSION

In this paper, we have presented GeoRank; an efficient location-aware news ranking system. GeoRank provides the top-k relevant news items considering: (a) the spatial proximity, (b) the time recency of the message, and (c) a preference function, that weights the relative importance of the above factors. The basic ideas in GeoRank is fairly simple but highly effective, which tries to avoid the unnecessary computations for processing the disqualified the news sources and their messages. GeoRank is composed of two main modules, namely, *query processor* and *message updater*. The *query processor* module retrieves top-k most relevant news feed. On the other side, the *message updater* module is a process running in the background to maintain the statistics. Such statistics ensure efficiency of the *query processor*, and hence GeoRank. Extensive experimental results, based on real and synthetic data sets, confirm that GeoRank significantly reduces the response time for news feed and improves the efficiency by at least 6 to 10 times.

9. REFERENCES

[1] A. Silberstein, J. Terrace, B. F. Cooper, and R. Ramakrishnan, "Feeding Frenzy: Selectively Materializing User's Event Feed," in *SIGMOD*, 2010, pp. 831–842.

[2] J. Bao, , M. F. Mokbel, and C.-Y. Chow, "GeoFeed: A Location-aware News Feed System," in *ICDE*, 2012, pp. 54–65.

[3] M. Sarwat, J. Bao, A. Eldawy, J. J. Levandoski, A. Magdy, and M. F. Mokbel, "Sindbad: A Location-based Social Networking System," in *SIGMOD*, 2012, pp. 649–652.

[4] "Facebook Statistics," http://www.facebook.com/press/info.php?statistics, 2010.

[5] I. F. Ilyas, G. Beskales, and M. A. Soliman, "A Survey of Top-k Query Processing Techniques in Relational Database Systems," *ACM Computing Surveys*, vol. 40, no. 4, p. 11, 2008.

[6] G. M. D. Corso, A. Gull, and F. Romani, "Ranking a stream of news," in *WWW*, 2005, pp. 97–106.

[7] L. Li, W. Chu, J. Langford, and R. E. Schapire, "A Contextual-Bandit Approach to Personalized News Article Recommendation," in *WWW*, 2010, pp. 661–670.

[8] A. Das, M. Datar, A. Garg, and S. Rajaram., "Google news personalization: scalable online collaborative filtering," in *WWW*, 2007, pp. 271–280.

[9] J. Liu, P. Dolan, E. Ronby, and Pedersen, "Personalized News Recommendation Based on Click," in *IUI*, 2010, pp. 31–40.

[10] B. E. Teitler, M. D. Lieberman, D. Panozzo, J. Sankaranarayanan, H. Samet, and J. Sperling, "NewsStand: a new view on news," in *SIGSPATIAL*, 2008, p. 18.

[11] G. Quercini, H. Samet, J. Sankaranarayanan, and M. D. Lieberman, "Determining the spatial reader scopes of news sources using local lexicons," in *SIGSPATIAL*, 2010, pp. 43–52.

[12] L. Aalto, N. Göthlin, J. Korhonen, and T. Ojala, "Bluetooth and WAP Push Based Location-Aware Mobile Advertising System," in *MobiSys*, 2004, pp. 49–58.

[13] Y. Cai and T. Xu, "Design, Analysis, and Implementation of A Large-scale Real-time Location-based Information Sharing System," in *MobiSys*, 2008, pp. 106–117.

[14] W. Xu, C.-Y. Chow, M. L. Yiu, Q. Li, and C. K. Poon, "MobiFeed: A Location-Aware News Feed System for Mobile Users," in *SIGSPATIAL*, 2012.

[15] U. Güntzer, W.-T. Balke, and W. Kießling, "Optimizing Multi-Feature Queries for Image Databases," in *VLDB*, 2000, pp. 419–428.

[16] S. Nepal and M. Ramakrishna, "Query Processing Issues in Image (Multimedia) Databases," in *ICDE*, 1999, pp. 22–29.

[17] J. Lu, P. Senellart, C. Lin, X. Du, S. Wang, , and X. Chen, "Optimal Top-k Generation of Attribute Combinations based on Ranked Lists," in *SIGMOD*, 2012, pp. 409–420.

[18] P. Cao and Z. Wang, "Efficient Top-k Query Calculation in Distributed Networks," in *PODC*. ACM, 2004, pp. 206–215.

[19] A. Yu, P. K. Agarwal, and J. Yang, "Processing a Large Number of Continuous Preference Top-k Queries," in *SIGMOD*, 2012, pp. 397–408.

[20] Y. Li, Z.-L. Zhang, and J. Bao, "Mutual or Unrequited Love: Identifying Stable Clusters in Social Networks with Uni- and Bi-directional Links," in *WAW*, 2012, pp. 113–125.

[21] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, "Skyline with presorting," in *ICDE*, 2003, pp. 717–719.

[22] W.-T. Balke and U. Guntzer, "Multi-objective Query Processing for Database Systems," in *VLDB*, 2004, pp. 936–947.

[23] M. L. Yiu and N. Mamoulis, "Efficient Processing of Top-k Dominating Queries on Multi-Dimensional Data," in *VLDB*, 2007, pp. 483–494.

[24] G. R. Hjaltason and H. Samet, "Distance Browsing in Spatial Databases," *ACM TODS*, vol. 24, no. 2, pp. 265–318, 1999.

[25] M. Sharifzadeh and C. Shahabi, "The Spatial Skyline Queries," in *VLDB*, 2006, pp. 751–762.

[26] M. L. Yiu, H. Lu, N. Mamoulis, and M. Vaitis, "Ranking Spatial Data by Quality Preferences," *TKDE*, no. 99, pp. 1–1, 2010.

[27] J. Bao, Y. Zheng, and M. F. Mokbel, "Location-based and preference-aware recommendation using sparse geo-social networking data," in *SIGSPATIAL*, 2012, pp. 184–195.

[28] V. Zheng, Y. Zheng, X. Xie, and Q. Yang, "Collaborative Location and Activity Recommendations with GPS History Data," in *WWW*, 2010, pp. 1029–1038.

[29] "PostgreSQL ," www.postgresql.org.