Geoinformatica (2020) 24:301–337
https://doi.org/10.1007/s10707-019-00380-z

# Local trend discovery on real-time microblogs with uncertain locations in tight memory environments

**Abdulaziz Almaslukh[1] · Amr Magdy[1] · Ahmed M. Aly[2] · Mohamed F. Mokbel[3] · Sameh Elnikety[4] · Yuxiong He[4] · Suman Nath[4] · Walid G. Aref[5]**

Received: 15 January 2019 / Revised: 4 August 2019 / Accepted: 16 August 2019 /
Published online: 10 September 2019


**Abstract**
This paper presents *GeoTrend+*; a system approach to support scalable local trend discovery on recent microblogs, e.g., tweets, comments, online reviews, and check-ins, that come in real time. *GeoTrend+* discovers top-$k$ trending keywords in arbitrary spatial regions from recent microblogs that continuously arrive with high rates and a significant portion has uncertain geolocations. *GeoTrend+* distinguishes itself from existing techniques in different aspects: (1) Discovering trends in arbitrary spatial regions, e.g., city blocks. (2) Considering both exact geolocations, e.g., accurate latitude/longitude coordinates, and uncertain geolocations, e.g., district-level or city-level, that represents a significant portion of past years microblogs. (3) Promoting recent microblogs as first-class citizens and optimizes different components to digest a continuous flow of fast data in main-memory while removing old data efficiently. (4) Providing various main-memory optimization techniques that are able to distinguish useful from useless data to effectively utilize tight memory resources while maintaining accurate query results on relatively large amounts of data. (5) Supporting various trending measures that effectively capture trending items under a variety of definitions that suit different applications. *GeoTrend+* limits its scope to real-time data that is posted during the last $T$ time units. To support its queries efficiently, *GeoTrend+* employs an in-memory spatial index that is able to efficiently digest incoming data and expire data that is beyond the last $T$ time units. The index also materializes top-$k$ keywords in different spatial regions so that incoming queries can be processed with low latency. In peak times, the main-memory optimization techniques are employed to shed less important data to sustain high query accuracy with limited memory resources. Experimental results based on real data and queries show the scalability of *GeoTrend+* to support high arrival rates and low query response time, and at least 90+% query accuracy even under limited memory resources.

**Keywords** Microblogs · Trend · Spatial · Real-time · Indexing · Query processing · Adaptive memory optimization · Uncertainty · Uncertain location

✉ Abdulaziz Almaslukh
aalma021@ucr.edu

Extended author information available on the last page of the article.

## 1 Introduction

Timely discovering and understanding localized trending events from online microblogs, e.g., tweets, comments, and check-ins, have become a reality. In fact, news agencies and people have referred to Twitter (a prime microblogging service) to get timely news about various events, e.g., Michael Jackson death [39], Boston explosions [6], tracking health issues [21], and China floods [11]. This is so popular that it outstrips TV as a news source for young people [5]. As a result, Twitter has released its own feature of localized trending hashtags [48], which shows current trending hashtags in a country or a city. Following the needs and importance of such a feature, various research efforts were dedicated to online local event discovery from microblogs [1, 8, 17, 33, 45]. Unfortunately, current efforts are tailored to finding events in pre-defined areas, where one needs to first specify the areas of interest, e.g., California, then start to detect events and news in these areas. In order to have worldwide high resolution coverage of such feature, there is a real need for a detection technique that: (a) covers arbitrary ad-hoc areas that are not pre-specified to the system, and (b) covers high resolution areas, e.g., finding events within part of the city, or events at the street level.

Up to our knowledge, there are two main attempts to support localized trend discovery with *arbitrary* spatial regions [26, 45]. However, one of these techniques ([45]) is built on two simplistic assumptions: (1) It assumes a very simplistic definition of "trending" queries as "frequent" queries, which can be computed through simple counting techniques, and (2) it assumes that the underlying system has unlimited memory. Hence, it does not account for expiring data from memory, which is crucial to ensure the accuracy of trending queries on recent data. Meanwhile, the second approach ([26]) is designed in a generic way to support trending queries for various contexts, where *location* can be considered as a context. Due to its generic nature, it has two main drawbacks: (1) It does not take advantage of the distinguishing characteristics of the spatial dimension, and (2) It is mainly designed to handle queries on arbitrarily large historical time periods, which makes it poor in handling queries on recent data in terms of both query performance and memory consumption, while recent data is the most important in discovering timely trends.

In this paper, we present *GeoTrend+*; a holistic system approach that supports *online trending* queries for *arbitrary ad-hoc* areas with *limited memory* resources. *GeoTrend+* abstracts localized trending queries to be in the form: *"Find the top-k trending keywords in the last T time units in area R"*, where $R$ is an arbitrary ad-hoc area and the *keyword* search is a proxy for trending events. For example, a tourist who visits *New York* city wishes to discover what is trending in Manhattan district instead of the whole city. She might get "Hamilton" and "Wicked" as local trending shows on Broadway. Another example is getting first-hand access to local news, so users and news agencies search for local news in Boston through social media during the Boston Marathon Explosions in 2013 [6]. In specific, people might find out more information about the incident such as the suspects and the victims as their names might be trending. A third example is localizing the tracking of health issues, that is currently adopted by the US Department of Human and Health Services [21], in different districts within the city for earlier and faster response for epidemics. To serve such diverse applications, *GeoTrend+* adopts a wide definition of *trending* keywords that goes beyond the simple counting assumption (i.e., *frequent* keywords) to consider trending as the growth in number of appearances over the query period $T$. It is likely that trending keywords are not among the frequent ones. For example, the keyword "*love*" is consistently frequent in Twitter, and it appears much more frequent than the keyword "*elections*", while the latter

is considered trending over the election week. This particular property along with the focus on supporting recent trending queries (i.e., last $T$ time units) are the main distinctions of *GeoTrend+* over its main competitors [26, 45].

*GeoTrend+* employs an in-memory partial pyramid index structure [4] that is able to digest incoming real-time microblogs with high arrival rates. The partial pyramid hierarchy divides the entire space into a set of multi-layers cells, where cells in each layer are non-overlapping. To accommodate incoming data in limited memory resources, each index cell is equipped with a novel and efficient count aggregation technique that maintains count-based measures over the last $T$ time units and expires data that is outside $T$. Injecting the concept of *expiration* in our aggregation is a key to *GeoTrend+* success, as it ensures discovering trends from only recent data and ensures continuous digestion of fresh microblogs in the limited memory. *GeoTrend+* count aggregation technique distinguishes itself from all previous sliding-window counting techniques (e.g., [3, 13, 20, 31]) by its simple and efficient structure that uses low-overhead update techniques to digest/expire microblogs with high rates; up to an order of magnitude higher than Twitter rate. In particular, it uses a constant memory per keyword regardless of the length of time span $T$. This is in contrast to existing techniques that have memory overhead proportional to $T$. This enables *GeoTrend+* to support arbitrarily large time spans with millions of keywords while using much less memory. For scalable query processing, each *GeoTrend+* index cell maintains a materialized list of top-$k$ trending keywords that appear within the cell spatial boundaries. Then, incoming queries with arbitrary spatial regions efficiently merge the materialized top-$k$ lists to come up with a final top-$k$ list.

*GeoTrend+* extends *GeoTrend* [35] to provide two new functionalities: indexing microblogs with uncertain locations and providing adaptive memory optimization that enables scalable accurate queries in tight memory environments. Uncertain locations, e.g., geotagging a microblog with a whole city instead of an exact point location, have become more popular with more restrictive privacy policies that prevent sharing exact user locations by default. In fact, the statistics on real Twitter data over the past three years show that 85% of geotagged tweets are associated with uncertain locations. All existing trend discovery techniques, including *GeoTrend* [35], approximate an uncertain location with a representative point location, e.g., city center. However, such approximation is simplifying both indexing and main-memory overheads as it avoids the expensive operations that replicate the same data record in multiple locations. For example, a single city typically spans several index cells and requires a microblog with city-level location to be represented in all these cells, which adds overhead to real-time indexing and memory usage. Our extended techniques in this paper provide indexing and memory optimization techniques that are able to support uncertain locations in real time with limited resources.

Despite the low memory overhead that is provided by *GeoTrend+* count aggregation technique, the overwhelming amount of microblogs data still encounter high memory footprint in peak times. Thus, *GeoTrend+* supports different settings; one of the most important settings is employing memory optimization techniques that exploit the nature of user-generated data to smartly select and shed less important keywords that are unlikely to contribute to any incoming query. The new techniques extend *GeoTrend* [35] memory optimizer to include parameter adaptivity that treats spatial areas at different spatial levels differently to allow maximum memory saving with almost no loss in query accuracy as verified in our experimental evaluation.

*GeoTrend+* is experimentally evaluated based on a real system prototype with a real-time feed of tweets and locations of Bing Mobile search queries. Our experiments show that *GeoTrend+* digests microblogs in high rates up to an order of magnitude higher than Twitter

rate, provides average query latency of few milli-seconds, and achieves much less memory consumption than its competitors with 90+% query accuracy.

The rest of this paper is organized as follows. Section 2 highlights related work, Section 3 introduces our trending measures, and Section 4 gives an overview of *GeoTrend+*. The *GeoTrend+* indexing, memory optimization, and query processing are discussed in Sections 5, 6, and 7, respectively. Section 8 presents the experimental evaluation and Section 9 concludes the paper.

## 2 Related work

Related work to *GeoTrend+* spans various areas, which include: trending items in data streams, spatial queries on microblogs, and spatial aggregate queries.

**Trending items in data streams** Discovering trending items in data streams [7, 10, 25, 38] is a well-studied topic. However, the main focus of existing techniques on the entire data stream, i.e., no support for old data expiration. Furthermore, there is no support for the spatial aspect of incoming data streams. This renders all techniques in this category not applicable for the problems addressed in *GeoTrend+*, which are: spatial querying with uncertain geolocations, promoting recent data that encounter a high fraction of queries, expiring old data as a necessity for digesting new microblogs, and providing efficient indexing in tight memory environments.

**Spatial queries on microblogs** Microblog locations are exploited for either *visualization*, where microblogs are plotted on the map [43, 50], *geotagging*, where geotags are extracted from the microblog contents [24, 32], *modeling*, where a model is built between users, locations, and topics [22], local topic discovery, where collections of data items that relate to a certain topic or event are discovered either in online or offline fashion [23, 28, 29, 44], or real-time query processing [2, 8, 36, 45]. The topic discovery techniques mostly focus on discovering cohesive clusters of related data items that are grouped into *topics* or *events*, and generally involve expensive computations that make them inappropriate for handling real-time streaming data continuously. The last category that addresses real-time query processing is the most related to our work. However, none of the existing techniques addresses either discovering trending items on recent microblogs or considering uncertain geolocations. In particular, Mercury [36] searches individual microblogs and does not support any aggregate query. GeoScope [8] addresses an interesting, yet orthogonal, problem of finding correlated location-topic pairs. Using GeoScope, we can support neither getting top-$k$ trending keywords as no ranking is employed nor handling arbitrary query regions as the locations are considered as a predefined discrete set, e.g., cities. Finally, AFIA [45] supports getting the top-$k$ *frequent* keywords on real-time data within arbitrary spatial regions. However, AFIA [45] techniques cannot be extended to discover trending items as they keep *only* top-$k$ frequent keywords in their index with no other information about any other keywords. kFST [2] extends AFIA to support large datasets that cannot fit in main-memory, yet, it is still limited for only top-$k$ frequent terms with no support for trending items. The literature has other related work that is orthogonal from *GeoTrend+*. Firefly [53] is an example that addresses the problem of data sparsity to identify effective keywords that are related to local news. RefTopicSketch [56] is another example that focuses on topic coherence of extracted trends. RevTopk [16] focuses on reverse frequent queries and frequency estimation of certain keywords. [49] focuses on parallelizing query processing

while consuming small memory overhead. [51, 52] identify users who have potential to enhance news detection applications. All this work either focuses on different queries or optimization goals and can be combined with *GeoTrend+* to provide finer and more efficient functionality.

**Spatial aggregate queries** There exists a lot of work in spatial aggregate queries, e.g., see [30, 34, 42, 46, 55], where the main focus is on building spatial index structures for disk-resident data. Aggregate information is precomputed and maintained for easy retrieval. Data is infrequently updated, and hence it is acceptable to use traditional spatial index structures without additional features for high arrival rates. Unfortunately, none of these works can support fast microblogs streams where high rates of digestion and expiration are cores issues to address.

## 3 Trending measures

Discovering trending items in microblogs currently depends on keyword count [8, 38, 47], within a limited time period, due to its simple computations that scales for massive numbers of microblogs. However, absolute count measure does not capture *trending* items effectively. In fact, it promotes keywords that are immortally top frequent ones, e.g., *job* and *love*, while ignoring other keywords that encounter considerable increasing count over time but they are not among the top frequent ones. For example, consider two keywords *love* and *elections*. Taking their count in hourly basis, over the last three hours, *love* has appeared 1000, 1150, and 950 times, while *elections* has appeared 200, 400, and 600 times. While *love* is the most frequent, it is clear that *elections* is a trending one. Yet, depending on absolute count does not capture this.

To overcome such limitation, trending items in the broader context of streaming data [10, 25] are detected based on changes in items behavior over time. This correctly detects rising keywords even if they are not top frequent. However, existing popular measures usually include expensive computations, e.g., Singular Value Decomposition, which is not efficient to maintain incrementally. In fact, efficient incremental computations is crucial for microblogs environments scalability, so that measures are not recomputed with new arrivals of keywords that come in fast rates. For this, *GeoTrend+* uses an efficient and effective measure that is based on the keyword *rate of count increase over time*. Count is easy to compute and maintain incrementally over time. So, measures that depend on count are suitable to scale in microblogs environments. *GeoTrend+* can adapt several trending measures as long as each of them is based on counting. Thus, *GeoTrend+* is equipped with two measures: either *rate of count increase over time*, or *weighted count over recent time period*, introduced in Sections 3.1 and 3.2, respectively.

### 3.1 Rate of increase measure

Rate of count increase over time is measured using a trend line slope that is computed based on the statistical linear regression [27]. Assume the last $T$ time units are divided into $N$ equal time intervals, trend line slope gauges the increase in keyword count in recent intervals compared to the oldest interval as follows:

$$Trend_{reg} = \frac{6\sum_{i=1}^{N-1}[i \times (c_i - c_0)]}{N(N+1)(2N+1)} \quad (1)$$

Where $N$ is the number of time intervals on which the count change is gauged over the last $T$ time units. $c_i, 0 \leq i < N$, is the count at time interval $i$, and for all $i > j$, interval $i$ is more recent than interval $j$, so that $c_0$ is the oldest counter and $c_{N-1}$ is the most recent counter. The detailed derivation of Eq. 1 based on linear regression slope is shown in Appendix.

The value of $N$ controls the accuracy of discovering trending items, as it represents the number of counts for which the regression slope is calculated. The higher the value of $N$, the more accurate the regression output. Setting $N$=$T$ gives the highest accuracy, yet it is the most expensive computationally and memory-wise. On the contrary, setting $N$=2 is the least expensive option that divides the whole $T$ time units into two intervals, yet it provides the least accuracy and might miss the actually rising keywords.

$Trend_{reg}$ measure that is presented in Eq. 1 is also efficiently maintainable in an incremental way on the arrival of new appearances of the keyword. As a new keyword appearance increase the count of just the most recent counter $c_{N-1}$, the only affected term in $Trend_{reg}$ would be $(N-1) \times (c_{N-1} - c_0)$. With increasing $c_{N-1}$ by one, this term is increased by $(N-1)$ and thus the whole $Trend_{reg}$ value is increased by $\frac{6(N-1)}{N(N+1)(2N+1)}$ (per Eq. 1). In case $N$ value is fixed through the processing of a microblog stream, which is the realistic case, the increase in $Trend_{reg}$ is a constant value that guarantees efficient incremental maintenance of $Trend_{reg}$ in real-time environments.

### 3.2 Weighted count measure

As an extensible framework for any count-based aggregate measure, *GeoTrend+* can employ *weighted count over recent time period* to detect *frequent* keywords in different spatial regions. Assume the last $T$ time units are divided into $N$ equal time intervals, keyword weighted count can be measured as follows:

$$Trend_{freq} = \sum_{i=0}^{N-1} c_i \times w^{N-1-i} \tag{2}$$

Where $0 < w \leq 1$ is a weighting parameter, and $N$ is the number of time intervals on which the count is gauged over the last $T$ time units. $c_i, 0 \leq i < N$, is the count at time interval $i$, and for all $i > j$, interval $i$ is more recent than interval $j$, so that $c_0$ is the oldest counter and $c_{N-1}$ is the most recent counter.

$Trend_{freq}$ is an exponentially weighted sum of the $N$ counters, where recent keyword counts have higher weight than older ones. The weight of counter $c_i$ is $w^i$, where $i = (N-1)$ is the most recent time period that has the highest weight $w^0 = 1$, regardless the value of $w$. Smaller $w$ gives lower weight to older counts, and setting $w$ to 1 gives equal weights to all counts and produces total count over the last $T$ time units. Similar to $Trend_{reg}$, the value of $Trend_{freq}$ is also efficiently maintainable in an incremental way where each new instance of a keyword simply adds one to both $c_{N-1}$ and $Trend_{freq}$ values.

For presentation simplicity, we assume to maintain a single trending measure $Trend_{reg}$ (Eq. 1). However, *GeoTrend+* can easily maintain more than one measure simultaneously to support queries that get either recently rising keywords or absolute frequent keywords using the same indexing data structures.

## 4 *GeoTrend+* Overview

This section gives *GeoTrend+* system architecture (Section 4.1) and query formulation (Section 4.2).

### 4.1 System architecture

Figure 1 gives the architecture of *GeoTrend+*, which consists of a *preprocessor* and three main components: an in-memory spatial index that embeds *count aggregation and expiration* module, a *memory optimizer* module, and a *query processor* module.

**Preprocessor** Each incoming microblog first goes through a preprocessor that extracts its timestamp, location, and keywords. A microblog location could be exact latitude/longitude coordinates or uncertain location represented with a minimum bounding rectangle (MBR) that determines a spatial range in which the microblogs is located, e.g., Chicago boundaries. This location could be directly associated with the incoming microblog, if available, or associated with the user profile who issued the microblog. Keywords are taken from hashtags associated with microblogs, if present, or a random word of its text.

**In-memory Index** The preprocessed microblogs are digested, with high arrival rates, in the in-memory spatial index with both exact and uncertain locations. The index divides the space into multiple levels, each level consists of a set of non-overlapping cells. Each index cell is equipped with efficient *count aggregation and expiration* module that maintains trending measures for the cell's keywords over the last $T$ time units. So, any data that is older than $T$ is expired and thrown out of memory. Details of indexing are presented in Section 5.

**Memory Optimizer** In case of scarce memory resources, the *memory optimizer* module is invoked on all index cells to shed keywords that are less likely to contribute to query answers. This saves significant memory space while keep highly accurate queries. The memory optimizer employs both fixed and adaptive parameters that provide different levels of memory savings and query accuracy based on the available resources. The adaptive parameters allow to deal differently with different spatial levels to shed the maximum amount of useless data without hurting the query accuracy. Details of memory optimizations are presented in Section 6.

**Query Processor** Users post their queries to the *query processor* module, that efficiently exploits the index materialized aggregate measures to return query answers to the users. Instead of processing excessive lists of all keywords, only $k$ local keywords in each query sub-region are exploited to aggregate the final top-$k$ list in the query region. Details of query processing are presented in Section 7.

### 4.2 Query formulation

*GeoTrend+* users can post queries in the form "*Find the most trending keywords within a spatial region R*." Internally, the system beefs up this query with three parameters: (1) $k$; the

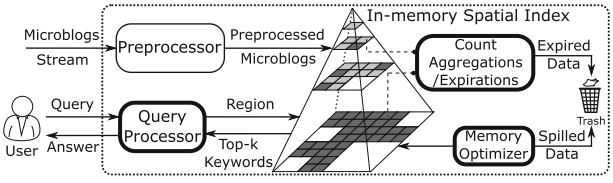



**Fig. 1** *GeoTrend+* Architecture

number of keywords to be returned, (2) a time span $T$; where the trending keywords should be posted within the last $T$ time units, and (3) a trending measure $Trend$; where the returned $k$ keywords should be highest ranked based on $Trend$. The query answer is then retrieved based on indexed locations, either precise point locations or uncertain locations, that are extracted from microblogs through a pre-processing step (as highlighted in Section 4.1). Formally, *GeoTrend+* query is defined as follows:

**Query Definition** *Given an arbitrary spatial region $\underline{R}$, an integer $\underline{k}$, a time span $\underline{T}$ time units, and a trending measure $\underline{Trend}$, GeoTrend+ finds k keywords such that: (1) The k keywords are posted within the region R. (2) The k keywords are posted within the last T time units. (3) The k keywords are the highest ranked based on Trend measure among keywords that are posted within R and T.*

Our query limits its answer size to $k$ as a natural consequence for the plethora of keywords that come with microblogs, which calls to selectively provide end users with the most relevant results (top-$k$ items) based on a certain ranking function. In fact, for the same reason, all research efforts on microblogs are limiting their answer size to $k$ [8, 9, 36, 45, 54] to be useful for end users. Furthermore, our query retrieves its answer from only *recent* keywords that are posted within the last $T$ time units. This basically promotes real-time nature of microblogs as a first-class citizen, which is a distinguishing property for nowadays microblogging services, to discover trends that are happening *now* on social media websites.

Upon initialization, a system administrator sets default values for parameters $k$, $T$, and $Trend$. Users may still change the default values of $k$ and $T$, yet a query may have less performance if the new values consider larger search space than the default values. Optimizing the index performance for a pre-set parameter values is a common design choice for major web services. For example, Twitter gives the most recent $k$ tweets to a user, where $k$=10, and so in a keyword search result. If a user would like to get more than $k$ results, an extra query response time will be paid on demand.

## 5 Real-time indexing

*GeoTrend+* employs a spatial pyramid index [4] to efficiently support queries in arbitrary spatial regions. The index divides the space into multi-layers cells of different spatial granularity, where each layer consists of a set of non-overlapping cells. For each incoming microblog in real time, *GeoTrend+* stores *only* its keywords and their aggregate information, distributed over all index cells that span the microblog location, rather than storing the microblog itself. To support fast digestion of microblogs streams and low query latency, the index is wholly resident in main-memory. However, main-memory resources are limited and cannot accommodate microblogs aggregate information for infinite time. Consequently, *GeoTrend+* limits its index content to data that arrived only in the last $T$ time units, where old data that is outside the time span $T$ is expired. The length of window $T$ depends on the available memory resources, and typical values range from several hour to few days of microblogs data. Indexing data of multiple days, that have hundreds of millions of microblogs, is feasible as the index does not store individual data records, but stores only aggregate information. Yet, such fast data rates impose scalability challenges on both index insertion and expiration.

Insertion and expiration in microblogs environments are so challenging that residing in main-memory is not enough to scale for microblogs high arrival rates. In particular, inserting keywords can be performed in a traditional way [4], where each new keyword is traversing

the pyramid structure from its root cell passing by all intermediate cells reaching leaf cells that include its location to update cells contents, either point location that is located in a single leaf cell or uncertain location that could span multiple cells. However, this is expensive given the large number of keywords that arrive every second in microblogs, where a significant portion span multiple index cells. To overcome this, *GeoTrend+* employs a bulk insertion technique that reduces the insertion cost so that it scales for digesting high arrival rates. Similarly, expiring contents from *GeoTrend+* index should be ideally performed in similar rates like insertion so that the index storage is stable in the system steady state. With large number of cells and high data rates, proactive expiration that iterates all index cells is very expensive and put significant overhead on the index performance in real time. To scale up, *GeoTrend+* employs a lazy expiration that dramatically reduces the expiration cost.

The rest of this section presents the index structure (Section 5.1), insertion (Section 5.2), and expiration (Section 5.3).

### 5.1 Index structure

*GeoTrend+* pyramid index structure is similar to a partial quad tree and consists of a single root cell that represents the entire geographic area, level 1 partitions the space into four equi-area disjoint cells, and so forth. As a partial tree structure, any index level could have both leaf and intermediate cells. Figure 2 depicts an instance of *GeoTrend+* pyramid index. The figure shows a partial pyramid that divides the space into three levels, where light gray cells indicate intermediate cells, dark cells indicate leaf cells, and white cells replace areas that are not actually maintained at that level. The pyramid shape is determined based on the spatial distribution of microblogs, through its *index shaping* process. Then, *index real-time operation* is started in which the index continuously digests incoming real-time data. Each stage is outlined below.

**Index shaping process** This is a one-time process that determines how pyramid cells are divided to cover the space at different levels of granularity. Areas with dense data distribution are divided into smaller cells at deeper levels of the pyramid. On the contrary, areas with sparse data are divided into large cells that span only a few pyramid levels. To determine the pyramid shape, we insert a sample of one day microblogs so that any cell stores maximum number of microblogs, called cell's $Capacity$. $Capacity$ has been chosen experimentally to range from 1000 to 2000 for fine granular space division. Thus, each cell maintains a data counter that accumulates the number of inserted microblogs in the cell. When the data

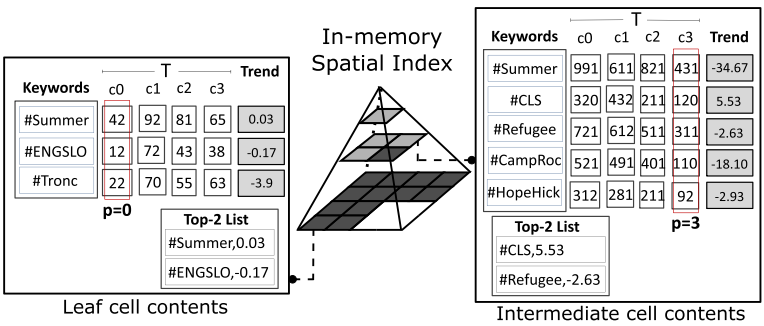



**Fig. 2** *GeoTrend+* index structure and cell contents

counter of any cell exceeds $Capacity$, the cell is divided further into four disjoint children cells. Inserted microblogs could have either exact point locations or uncertain locations represented with minimum bounding rectangles (MBR). In case of an exact point location, the microblog is inserted in a single index cell and its data counter is incremented by one. If the microblog has an uncertain location, then all cells that intersect with its MBR are considered for inserting this microblog. Although each of these cells overlaps with a fraction of the microblog uncertain location, the shaping process accommodates a conservative approach that replicates the same microblog in each cell. Therefore, the data counter in each cell is incremented by one. This conservative approach is intended not to underestimate the density of data in any index cell. In fact, it overestimates the data density so all index cells that span the city of Minneapolis, for example, replicate all microblogs that are tagged with Minneapolis although the overall Minneapolis region contains this data only once. Such overestimation leads only to dividing Minneapolis to a finer granular index cells so no cell is overloaded during the index real-time operation. The replicated microblogs are actually removed from all cells on concluding the shaping process and only aggregate information accumulates in these cells while digesting real-time data. After each insertion in a cell, the total number of microblogs is compared against the cell $Capacity$ to decide on dividing the cell. Once there are no more cell divisions, all the individual microblogs are wiped and the shaping process is concluded.

**Index real-time operation** After its shaping, the index then starts to continuously receive real-time data and stores only aggregate information about incoming keywords rather than storing individual microblogs. Keyword aggregate information are stored in both leaf and intermediate cells, so that information of the same keyword are aggregated at different levels of spatial granularity. Each index cell $C$, both leaf and non-leaf cells, stores four data structures: a hash table $H$, a sorted list $TopK$, a rotating pointer $p$, and a timestamp $t_{last}$, described below:

**Hash table *H*** Each hash entry $h \in H$ represents a single keyword arrived to cell $C$ in the last $T$ time units. With each hash entry $h$, we maintain the following:

1. A set of $N$ counters, $c_0$ to $c_{N-1}$. The $N$ counters divide the time window $T$ into a **set of equal temporal intervals**, each of $\frac{T}{N}$ time units. Each counter maintains the number of times that the hash entry $h$ has appeared in its corresponding $\frac{T}{N}$ time units. $N$ is a system parameter that trades query accuracy with computation efficiency as discussed in Section 3. A larger value of $N$ gives more accurate results, yet, it comes with processing and storage overhead in maintaining more counters. Every $\frac{T}{N}$ time units, the current counter is concluded, another current counter is created with a zero value, and the oldest counter is expired. Details of inserting and expiring data are given in Sections 5.2 and 5.3, respectively.
2. A trending value $Trend$ that is calculated based on hash entry $h$ counters $c_i$'s according to Eq. 1.

**List *TopK*** A sorted list of size $k$ that maintains the top-$k$ trending keywords in this cell ranked based on the trending value $Trend$. This is mainly to materialize the top-$k$ answer of this cell to speed up the query processing significantly.

**Rotating pointer *p*** An integer value, in the range of 0 to $(N-1)$, that points to the current (i.e., most recent) counter. Thus, the most recent counter is $c_p$ and the oldest counter is

$c_{(p-1)\%N}$. Maintaining $p$ saves huge efforts in shifting the counter values and expiring old counters every $\frac{T}{N}$ time units as discussed in Section 5.3.

**Timestamp $t_{last}$** The starting timestamp of the time interval of the last expiration of $C$ contents, where it is used to decide which counters need to be expired in the following expiration cycle.

Figure 2 shows the contents of two index cells, one intermediate cell and one leaf cell. Both cells enclose exactly the same data structures that are described earlier. The intermediate cell encounter more keyword arrivals as it lies one level higher than the leaf cell, and so it covers four times larger space area. The intermediate cell in Fig. 2 contains five hashtags, *Summer*, *CLS*, *Refugee*, *CampRoc*, and *HopeHick*, each maintains four counters, $N$=4, and $Trend$ value. It also maintains a top-2 list, *CLS* and *Refugee*, of the most trending keywords in the cell sorted based on $Trend$ value, and an integer pointer $p$=3 that indicates $c_3$ as the most recent counter in which data is digested during the latest insertion. The leaf cell in Fig. 2 contains three hashtags, *Summer*, *ENGSLO*, and *Tronc*, which also maintain four counters per hashtag and a top-2 list, *Summer* and *ENGSLO*, with similar semantic to the list in the intermediate cell. So, $N$ and $k$ values are fixed for all index cells. Yet, the leaf cell integer pointer $p$=0, which is a different value than the intermediate cell. The $p$ value is updated on data expiration, which happens in a lazy fashion piggybacked on the insertion, so different cells expire their data at different times based on insertions in the cell. The details of data expiration and updating $p$ value in each cell are discussed in Section 5.3.

## 5.2 Index insertion

To reduce the index update cost and scale for digesting high arrival rates, *GeoTrend+* spatial index employs an efficient bulk insertion technique that saves thousands of comparison operations for keyword locations with spatial cell boundaries compared to the traditional way of inserting individual data records. The bulk insertion process consists of two steps: (1) traversing pyramid index cells with batches of keywords, and (2) while traversing, keywords are inserted in their corresponding cells. Each step is described below.

**Pyramid traversal** To reduce the pyramid traversal cost, the incoming keywords are buffered for $t$ seconds before being inserted in bulk. $t$ represents a trade-off between the insertion overhead and the delay between a microblog arrival and being available to search results. Typical values of $t$ is 1-2 seconds which is an acceptable delay for real-time applications, and still sufficient to collect several thousands of keywords to insert as a batch. For example, Twitter receives ∼12,000 tweets every 2 seconds, which is a reasonable batch size that saves significant insertion cost. During the buffering, a spatial minimum bounding rectangle (MBR) is maintained around locations that are associated with the keywords, either point or uncertain locations. We then traverse the pyramid levels through comparing the MBR boundaries, instead of locations of individual microblogs, and insert keywords in the corresponding cells.

The buffered keywords are first inserted in the root cell $C$, as shown in *cell insertion* below. If $C$ is not a leaf cell, the new keywords are recursively inserted in $C$'s children cells. The new keywords are divided based on their locations into four MBRs, each MBR encloses a subset of the keywords that corresponds to one of the children cells. Then, the same *cell insertion* process is applied to each of the children cells. This leads to replicating all keywords across all index levels. Such replication significantly reduces the query latency for large query areas as it minimizes number of processed cells for large query regions.

On another hand, it increases both index insertion time and memory consumption. Our experiments study the impact of this replication on indexing overhead, query processing, and memory consumption.

Dividing buffered keywords could be based on exact point locations or uncertain MBR locations. In case of point locations, each keyword is routed to a single child cell that corresponds to its point location. In case of uncertain locations, a keyword location might correspond to one or more children cells. If the keyword location is wholly contained inside a single child cell, it is inserted in this cell similar to exact-location keywords. If the keyword location spans multiple cells, a fraction $f_{C_i}$ for each cell $C_i$ is calculated to reflect the overlap between the keyword MBR location and the cell $C_i$ boundaries. Assume an MBR $L$ that represents the keyword uncertain location, $f_{C_i} = \frac{Area(L \cap C_i)}{Area(L)}$. Thus, $f_{C_i}$ represents the ratio of intersection between $L$ and $C_i$, where $\sum_{\forall i} f_{C_i} = 1$. Then, the keyword is forwarded to each cell $C_i$, during the *cell insertion*, associated with the corresponding $f_{C_i}$ value. For example, if the keyword location $L$ spans four cells with equal intersection areas, the keyword is forwarded to each of these cells associated with a fraction of 0.25 indicating that one quarter of $L$ lies in this cell. In a general case, the keyword is associated with any values of fractions $f_{C_i}$'s so their summation remains one.

**Cell insertion** On the arrival of new keywords to any cell $C$, two steps are performed: (1) inserting the new keywords in the hashtable $C.H$, and (2) updating the list $C.TopK$ that maintains $C$'s top-$k$ keywords.

(1) *Insertion in hashtable* $C.H$. For each newly arrived keyword, if there is no corresponding hash entry in the hashtable $C.H$, it is added to $C.H$ with zero-initialized $N$ counters and $Trend$ value. Then, regardless of whether there was a prior hash entry or not, its most recent counter $c_p$ is incremented to reflect the new arrival update. The increment of $c_p$ value is dealt differently for exact and uncertain locations. For exact location, $c_p$ value is incremented by one as the keyword is wholly contained inside this cell. This leads its $Trend$ value to be incremented by $\frac{6(N-1)}{N(N+1)(2N+1)}$ (per Eq. 1). For uncertain locations, the keyword is associated with a fraction $f_C$ as described in the *pyramid traversal* phase. This fraction represents the portion of the uncertain location that overlaps with $C$. Thus, we increment the value of $c_p$ by $f_C$ instead of one. In addition, $Trend$ value is incremented by $f_C \times \frac{6(N-1)}{N(N+1)(2N+1)}$. In both cases of exact and uncertain locations, the update of $Trend$ value involve only a constant increment. This makes our trending measures very efficient to be maintained incrementally as discussed in Section 3 and suitable to scale in real-time environments such as microblogs.

The described insertion process of data with uncertain locations increases the overhead of both indexing and main-memory consumption compared with considering only exact locations as in existing techniques. The main source of overhead is replicating the same keyword in all cells that intersect with its uncertain location. This replication adds hash entries to multiple cells, at multiple spatial levels, instead of a single cell in each level, which consumes both additional memory and higher insertion time. From another hand, representing uncertain locations in all cells enhances the query accuracy as it reflects the actual location information without approximating large MBRs with a single point. Our experimental evaluation studies the effect of handling uncertain locations on indexing overhead, main-memory consumption, and query results, showing the scalability of *GeoTrend+* in terms of indexing and

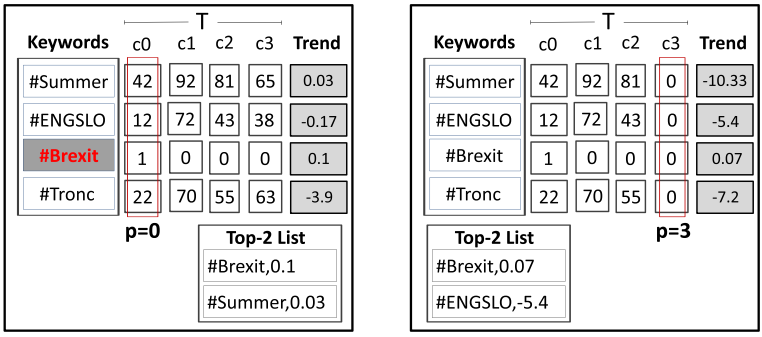
memory consumption and the effectiveness of the new query results compared to the approximate techniques with different similarity measures.

(2) *Updating list* $C.TopK$. For each new keyword inserted in $C.H$, we check its $Trend$ value to update $C.TopK$ list, if needed, so that it keeps maintaining the most trending $k$ keywords in $C$. If $C.TopK$ has less than $k$ keywords, the new keyword is inserted in $C.TopK$ directly. Once $C.TopK$ has $k$ keywords, the $Trend$ value of each new keyword is compared to $Trend_{min}$: the lowest trending value in $C.TopK$. If the new keyword's $Trend$ is larger than $Trend_{min}$, then it is inserted in $C.TopK$ replacing the keyword that corresponds to $Trend_{min}$.

*Example 1* Figure 3a shows an example for index insertion. The figure shows the content of the leaf cell shown in Fig. 2 after inserting hashtag *Brexit* with an exact location. As the hashtag is not previously present in the cell, a new entry is added to the hashtable $H$ with zero-initialized counters. Then, the most recent counter, $c_0$, is incremented and $Trend$ value is computed. As the new $Trend$ value is eligible for the top-2 list, the hashtag *Brexit* is inserted into the list.

### 5.3 Data expiration

As *GeoTrend+* index limits its contents to data of the last $T$ time units, it needs to periodically expire old data that is outside the time span $T$. Thus, every $\frac{T}{N}$ time units, *GeoTrend+* should hold on inserting new data, iterate over *all index cells*, and expire the old contents. However, this causes a significant interruption for index real-time insertion and terribly reduces its digestion rates. To prevent such interruption, *GeoTrend+* skips such an expensive expiration that expires *all cells at once* and employs a lazy expiration technique that postpones expiring any index cell contents until: (1) either an insertion occurs in this cell, or (2) a query comes to this cell and hence an expiration is necessary so as not to consider old data in the query answer. In both cases, expiration is necessary, and performed, only in a *single* cell. This minimally interrupts real-time insertion of *GeoTrend+* index as it expires *only one cell at a time*, and even consumes no index traversal cost as it piggybacks this cost on either insertion or query processing. The effect of putting this overhead on query response is minimal as expiration is performed once and it pays off for all incoming queries. However, this lazy expiration does not guarantee to expire all old contents. In fact, cells




(a) Index Insertion (b) Index Expiration

**Fig. 3** Example of *GeoTrend+* index insertion and expiration

that encounter neither insertions nor queries during the $T$ time units, e.g., low dense spatial regions like suburbs, would keep very old contents. To overcome this, *GeoTrend+* runs an additional cleaning process, every $T$ time units, that is very light and efficient, so that it does not put an overhead on the index performance. Both lazy expiration and periodic light cleaning are described below.

**Lazy expiration** The contents of a cell $C$ is expired only if it is last expired more than a complete period of $\frac{T}{N}$ time units ago. This is checked through $C.t_{last}$ timestamp, that is the starting timestamp of the period when $C.H$ is last expired. If $n_c = \left\lfloor \frac{NOW - t_{last}}{T/N} \right\rfloor \geq 1$, then the oldest $n_c$ counters need to be expired and $C.t_{last}$ is updated to be $t_{last}$=$t_{last}$+$n_c \times \frac{T}{N}$. For presentation simplicity, assumes $n_c$=1, i.e., we expire only the oldest counter. Then, the oldest counter $c_{(p-1)\%N}$ should expire for *all entries* in the hashtable $C.H$. This requires to set the value of $c_{(p-1)\%N}$ to zero, the value of pointer $p$ is decremented to be $p = (p-1)\%N$, and the aggregate $Trend$ value is recomputed. This is repeated $n_c$ times when $n_c > 1$.

Maintaining $p$ saves huge efforts in expiration. A traditional way is to shift the counter values for each hash entry. With $p$, we keep all counter values intact in their positions, and we just shift left (i.e., rotate) the value of $p$ to replace the oldest expiring counter with a new one. With this, it is always the case that counter $c_p$ represents the current $\frac{T}{N}$ time units while counter $c_{(p-1)\%N}$ represents the oldest $\frac{T}{N}$ time units within the time span $T$.

Expiring the contents of hashtable $C.H$ leads to invalidating the contents of $C.TopK$ list. Thus, $C.TopK$ is recomputed with each expiration of $C.H$ contents. However, recomputing $C.TopK$ list comes with a very little overhead on the lazy expiration process. While updating $Trend$ value of each hash entry $h$, $h$ is considered as a potential candidate for $C.TopK$. If $C.TopK$ has less than $k$ keywords, then $h$ is inserted in $C.TopK$ right away. If $C.TopK$ has $k$ keywords, then $h.Trend$ is compared to $Trend_{min}$: the lowest trending value in $C.TopK$. If $h.Trend$ is larger than $Trend_{min}$, then it is inserted in $C.TopK$ replacing the keyword that corresponds to $Trend_{min}$. This repeats for each hash entry $h$ while its counters are updated.

*Example 2* Figure 3b gives the contents of Fig. 3a cell after $c_0$'s time period expires. In this case, (a) $c_0$ is concluded, (b) the oldest counter $c_3$ would expire its old values and reset to zero for all keywords, (c) the current pointer $p$ becomes 3 as $c_3$ becomes the current active counter, and (d) $Trend$ values are recomputed based on the new counter positions, where $c_2$ is the oldest counter. Meanwhile, the top-2 list is recomputed, based on the new $Trend$ values, to include *Brexit* and *ENGSLO* keywords.

**Light cleaning** To account for sparse cells that rarely encounter insertions and queries, and hence do not encounter any lazy expiration, we run a light periodic cleaning. Every $T$ time units, a light expiration process is traversing *all* index cells. If the cell is last expired older than $T$ time units ago, then all cell contents are wiped, otherwise, nothing is done. This process intentionally overlook contents that is within the last $T$ time units but still old enough to be expired, i.e., older than $\frac{T}{N}$ time units ago. This is intended to make it very light and efficient, while this contents are left for the next lazy or periodic expiration in the cell. Although some cells would contain unneeded contents for $T$ time units, practically this does not cause much overhead as they are very sparse cells. As the light cleaning process wipe all cells contents, so no $TopK$ update is needed as $TopK$ is wiped as well.

## 6 Memory optimization

As *GeoTrend+* index is wholly resident in main-memory, it might be the case that during peak times, e.g., local events in major cities, available memory resources are limited to store the vast amount of incoming data. In that case, some applications are willing to remove a portion of memory content that minimally affects query accuracy, still sustain the index real-time performance in peak times. Thus, *GeoTrend+* employs a main-memory optimization technique, called *TrendMem*, that reduces memory footprint significantly while keep query answers highly accurate. *TrendMem* is based on a key observation that identifies a very interesting spatial property for microblogs data. Such property is used to smartly identify victim data to expel from main-memory with slight loss in query accuracy. Furthermore, *GeoTrend+* equips its index with *AdaptiveTrendMem* that extends *TrendMem* with parameter adaptivity to treat different spatial index cells differently based on their content to significantly decrease the loss in query accuracy encountered by *TrendMem* while maintaining the low memory footprint. In the rest of this section, Section 6.1 presents the key observation and key idea behind *TrendMem* and *AdaptiveTrendMem*. Then, Sections 6.2 and 6.3 present the details of realizing *TrendMem* and *AdaptiveTrendMem* inside *GeoTrend+* index, respectively.

### 6.1 Key ideas

**Key observation** Memory optimization in *GeoTrend+* takes advantage of the observation that keywords popularity in microblogs follows a Zipf distribution [12, 19, 40, 41], i.e., small percentage of keywords appear with high frequency while the majority of keywords appear very few times. Interestingly, Zipf distribution holds not only for the entire microblogs collection over the entire world, but also over those appearing in smaller spatial regions. We demonstrate such interesting property in Fig. 4. The figure shows the frequency distribution of millions of real tweets at four different levels of spatial granularity, Level 1 is the entire USA, Level 2 is the four quarters of the USA, Level 3 has sixteen tiles dividing each tile in Level 2 into four quarters, and so on. The figure shows that the majority of keywords in Twitter are infrequent across all levels of spatial granularity. Such

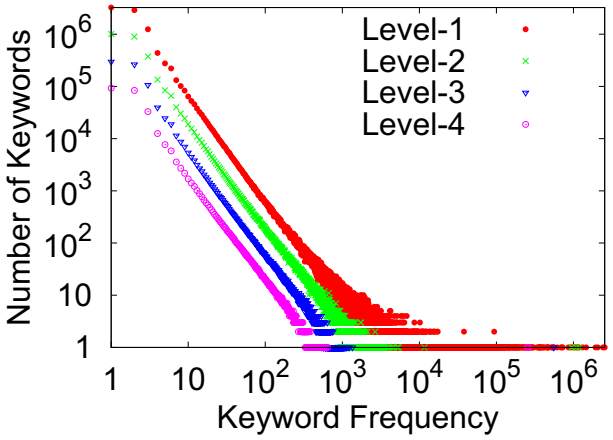



**Fig. 4** Zipf distribution of Twitter keywords at different spatial levels

majority of infrequent keywords consume large percentage of the memory for their counters. Our memory optimization techniques exploit the existence of such infrequent keywords in a smart way to identify a subset of them that are very unlikely to contribute to trending query answers. This subset is shed from main-memory without hurting the accuracy of query answers.

**Key idea** The key idea of our memory optimization techniques that some keywords with low frequency are unlikely to be trending ones. Those keywords must satisfy a crucial condition: they must encounter low frequency *in all sub-intervals of the last $T$ time units*. This condition is sufficiently working as it judges count change over time, which is the same as our trending measures (Section 3). To elaborate, if we decide on a keyword importance only through its total count during the last $T$ time units, it might be the case that a keyword encounter low total count, yet, its count is rising significantly over time. Thus, we may end up removing trending keywords from main-memory. However, if we ensure that the keyword count is low in all the sub-interval of the last $T$ time units, then it is very unlikely that *growth of count* of this keyword makes it a potential trending one. Then, it is unlikely to contribute to query results and it can be removed without affecting the query accuracy.

### 6.2 *TrendMem* technique

**Main idea** In each cell $C$ in *GeoTrend+* index, *TrendMem* periodically removes keywords that are $\epsilon$-infrequent in *all the N time intervals of the last T time units*. $\epsilon$-infrequent keyword is a keyword that has count less than $\epsilon \cdot n$, where $\epsilon$ is a small fraction, e.g., 0.001, and $n$ is the total number of keyword arrivals in cell $C$ in the corresponding time interval. For example, if $C$ received total of $n_i$ keyword arrivals during time interval $i$, $0 \leq i \leq (N-1)$, then a keyword $W$ is considered $\epsilon$-infrequent if its counter $c_i < \epsilon \cdot n_i$, for all $0 \leq i \leq (N-1)$. Removing infrequent items from a cell $C$ is invoked every $\frac{1}{\epsilon}$ insertion cycles in $C$. This ensures to limit the size of the hashtable $C.H$ to $O(\frac{1}{\epsilon}\log(\epsilon \cdot n))$ entries (inspired by the same ideas presented in LossyCounting algorithm [37]). Also, any keyword with total count $> (\epsilon \cdot n)$ at any sub-interval of $T$ is guaranteed to be maintained. In fact, checking a keyword to be infrequent in each of the $N$ sub-intervals independently ensures the consistency of the keyword infrequency along the whole time window $T$ and thus guarantees not to expel any possibly trending keywords as discussed in Section 6.1. In addition, employing a percentile threshold $\epsilon$, which means keyword importance is identified based on a percentage of frequencies of its neighbor keywords within the spatial locality. This guarantees that dense spatial areas do not affect suburb areas and leads to maintain an accurate top-$k$ keyword list in each spatial locality. This makes *TrendMem* provides highly accurate query answers.

**Impact on the index** To realize *TrendMem* inside *GeoTrend+* index, two main operations are added to the *index insertion*: (1) periodic cleaning of infrequent keywords inside each cell every $\frac{1}{\epsilon}$ insertion cycles in the cell, and (2) checking on $\epsilon$-infrequent keywords in each sub-interval to decide on removing which keywords. To this end, each index cell maintains an insertion cycles counter that is initialized to zero. The counter is incremented by one with every insertion in the cell, either the insertion is for a whole microblog with an exact point location or for a fraction of microblog with an uncertain location (as described in Section 5.2). Once it reaches $\frac{1}{\epsilon}$, the cleaning procedure is triggered and the counter is reset to zero. The cleaning procedure goes through a complete scan for all hash entries in hashtable $H$ and removes any keyword that is consistently infrequent during all the $N$ intervals. To check for the keyword infrequency in each sub-interval independently, each *cell* maintains

additional $N$ counters $n_i, 0 \leq i \leq (N-1)$, that keep the total number of keyword arrivals in each of the $N$ sub-intervals of the time window $T$. Thus, with each insertion to the cell, the counter of the current interval is incremented by the number of new keywords. Using this, the infrequency check is then performed very cheap by comparing $\epsilon \cdot c_i$ of each keyword counters to the counter $n_i$, for all $0 \leq i \leq (N-1)$. It worth noting that the new counters are maintained per a whole cell not per each individual keyword inside this cell. This means adding a negligible memory overhead for *TrendMem* compared with the significant memory saving as shown in our experimental evaluation.

The value of $\epsilon$ is fixed for all index cells. A typical value of $\epsilon$ would be around 0.001, which is considered large enough to limit the memory footprint without really affecting the accuracy of the query result. Although introducing $\epsilon$ saves significant storage, apparently, executing the periodic cleaning procedure incurs additional computational overhead during the index insertion operation. Since we adjust the triggering of our cleaning procedure to be every $\frac{1}{\epsilon}$ insertions, a lower value of $\epsilon$ implies less frequent cleaning, i.e., less insertion overhead and less storage saving, but higher query accuracy. For example, when $\epsilon$ is 0.01, we run the cleaning procedure every 100 insertions. Yet, when $\epsilon$ is 0.0001, we perform the cleaning every 10,000 insertions, which is cheaper in computation cost, achieves higher query accuracy, but consumes more memory. We study in details the effect of varying $\epsilon$ on the insertion overhead, storage saving, and query accuracy experimentally to provide a reasonable compromise for both memory consumption and insertion overhead. In addition, we develop an adaptive version of *TrendMem* (Section 6.3) that varies the value of $\epsilon$ for different cells to eliminate any unnecessary cleanup cycles and provide better selectivity for victim keywords to shed from main-memory.

Indexing uncertain locations is another factor that affects the frequency of invoking the memory optimization module. The reason is that a single microblog with uncertain location is potentially inserted in multiple cells, which increments multiple insertion counters on the contrary to an exact-location microblog that increments only one insertion counter in a single cell. This leads many insertion counters to reach the threshold $\frac{1}{\epsilon}$ much faster than the case of indexing only exact locations. Therefore, the memory optimization module is invoked more frequently, which increases the cleaning overhead during real-time insertion and gets rid of infrequent keywords faster as well.

### 6.3 *AdaptiveTrendMem* technique

**Main idea** *GeoTrend+* replicates keywords across all index levels to speed up query processing. This means that each parent cell in the index maintains a union of all keywords that are maintained in all cells in its sub-tree, which is typically a massive number up to hundreds of millions in high levels of the tree. For example, the root node maintains aggregate information for all keywords that arrive to the index, while each of its children cell maintains a fraction of these keywords (roughly a quarter of them), and each cell in each subsequent level maintains a smaller fraction and so on. So, there is a high variation in the amount of maintained information in different cells based on the level of spatial granularity. This is shown in Fig. 4 where Level 3 in the pyramid index maintains ∼5 times more keywords than Level 4, while Level 2 and Level 1 maintains higher numbers up to 10 times and 50 times number of keywords, respectively.

Despite that high variability in the amount of maintained information in different cells, what is really processed during query processing (as detailed in Section 7) is only $k$ keywords from each cell regardless the cell level. These keywords are maintained in a top-$k$ list at indexing time and represent the highest trending keywords in the cell ranked based on

$Trend$ measure. This list is a subset of highly frequent keywords and infrequent keywords do not mostly contribute to this list. In fact, the main idea of *TrendMem* that is introduced in Section 6.2 depends on removing such infrequent keywords, based on a percentile frequency threshold $\epsilon$ in each time interval, as they do not contribute to query answers. However, *TrendMem* uses the same threshold $\epsilon$ for all cells regardless their data content. Therefore, using small values of $\epsilon$ will keep many infrequent keywords in high levels cells that do not contribute to query answers and hurt the memory consumption. For example, Fig. 4 shows the root cell at Level 1 maintains 5 times infrequent keywords as much as its children cells at Level 2, and so on for subsequent levels. On another hand, using large values of $\epsilon$ could aggressively remove rising keywords from lower levels cells and hurt the query accuracy. This makes the index content is highly sensitive to $\epsilon$ value.

To overcome this problem, the *AdaptiveTrendMem* technique employs adaptive $\epsilon$ values for cells at different spatial levels of the index. This depends on the fact that the practical values of $k$ are much smaller than the number of keywords maintained in any cell. Therefore, finding the actual top-$k$ trending keywords in cells of high levels of the index uses much smaller percentage of the data compared with cells at lower levels. *AdaptiveTrendMem* then assigns large $\epsilon$ values for cells at high levels and smaller $\epsilon$ values for cells at lower levels. This allows to remove many infrequent keywords from excessive amounts of data in high levels of the index and reduce the number of cleanup cycles at lower levels where less data should be removed. This parameter adaptivity provides much better selectivity for victim keywords to spill from main-memory, which leads to almost perfect query accuracy while still ensures low memory footprints in tight memory environments, as shown in our experimental evaluation.

**Impact on the index** To realize *AdaptiveTrendMem*, *GeoTrend+* index structure maintains a different $\epsilon$ value for each index level during the index shaping process instead of using a single value for the whole index. This adds a single lookup table that maintains one $\epsilon$ value per index level and makes a minor change to the cleanup process. In particular, the cleanup process remains the same except it looks up the $\epsilon$ value to use from the new lookup table depending on the cell level. In addition, as the cleanup process is invoked every $\frac{1}{\epsilon}$ insertions in the cell, the frequency of invoking cleanup becomes different across levels and depends on its corresponding $\epsilon$ value. Thus, for high levels of the index, that have large $\epsilon$, the cell content is cleaned up more frequently to remove the plethora of infrequent keywords and improve memory consumption. On the contrary, for lower index levels, less cleanups are performed to reduce the memory optimization overhead on inserting real-time data.

A challenging problem in realizing *AdaptiveTrendMem* is selecting the correct value of $\epsilon$ for different index levels. Our realization proposes two schemes to assign these values: (1) *variable decimal digit* (VDD), and (2) *variable level number* (VLN) schemes. First, VDD scheme takes two values of $\epsilon$ from a system administrator, a minimum value and a maximum value. The maximum value is assigned to the root level. Then, each of the subsequent levels is assigned the value of the preceding level divided by ten to add a single decimal digit to the value of $\epsilon$. However, any level cannot be assigned lower than the minimum value of $\epsilon$ to limit the effect of the fast $\epsilon$ value reduction at lower levels from assigning negligible $\epsilon$ values and suppress the memory optimization effect. Second, VLN scheme takes a single value of $\epsilon$ that is assigned to the root index cell at level 1, called $\epsilon_r$. Then, any subsequent level $i$ is assigned a value $\epsilon_i$, that is $\epsilon_i = \frac{\epsilon_r}{i}$. Thus, level 2 is assigned half of $\epsilon_r$, level 3 is assigned one third of $\epsilon_r$, and so on. This scheme reduces the value degradation of $\epsilon$ at lower levels. Both schemes are evaluated in our experimental evaluation, highlighting their relative pros and cons.

## 7 Query processing

This section discusses query processing in *GeoTrend+*. As *GeoTrend+* index already materializes top-$k$ items in each spatial cell, processing top-$k$ queries is simple, efficient, and provides low response time. In fact, *GeoTrend+* query processing depends on getting top-$k$ keywords in the query region $R$ by manipulating *only* the top-$k$ lists that are maintained in the index cells that overlap with $R$. Our hypothesis is that it is highly unlikely that a keyword that did not make it to any of the top-$k$ lists in any cell would make it to the final answer. The main reason is that our trending measures are additive (per Eqs. 1 and 2), which means the trending value of a certain keyword $W$ over an arbitrary region $R$ equals the summation of $W$'s trending values in all index cells that overlap with $R$. Thus, top-$k$ items within each cell have much better chances to be the global top-$k$ items in $R$. This hypothesis is supported empirically by our experimental results, where the vast majority of queries can get the true top-$k$ trending keywords in $R$ from the ones that appear in any top-$k$ list.

*GeoTrend+* query processing is composed of two main steps. In the first step, *GeoTrend+* finds a set of pyramid index cells $S$ that cover the query spatial region $R$ in a way that minimizes the number of cells in $S$ while maximizes the coverage ratio with $R$. In the second step, it finds the top-$k$ keywords in $R$ by aggregating the values from only top-$k$ lists that are maintained in $S$ cells. Details of the two steps are described below.

Step 1 takes the query spatial region $R$ and the root cell of the spatial pyramid index as input and outputs a set of cells $S$ that completely cover $R$, such that: (a) the number of cells in $S$ is minimal, which reduces the aggregation cost in Step 2, and (b) the cells in $S$ have the highest overlap ratio with $R$, which maximizes the accuracy of the retrieved results. We define the overlap ratio between a cell $C$ and the query region $R$ as the area of the part of $C$ that is inside $R$ divided by the area of $C$, i.e., $\frac{C \cap R}{C}$. Starting at the pyramid root cell, we recursively visit the children overlapping with $R$. A cell $C$ is added to $S$ if *one* of the following two conditions is satisfied: (1) $C$ is a leaf cell, or (2) $C$ is completely inside $R$, i.e., overlap ratio of 100%. In both cases, we know that $C$ has the best covering area which is the same coverage we can get from $C$'s children. So, to minimize the number of cells in $S$, we just add $C$, and skip all its children. Otherwise, we visit children cells applying the same selection procedure.

Step 2 takes the set of cells $S$ from Step 1 as input and produces the final answer of the top-$k$ keywords that appear in $S$. In this step, we only consider keywords that have appeared in at least one top-$k$ list of all the cells in $S$. Following the spirit of Fagin's TA algorithm [15], the main idea of this step is to employ a max-heap priority queue, initiated by the top item in each list in $S$. The key of the priority queue is the trending value. Then, we keep extracting items from the queue one by one. For each extracted item $Top$, we do the following: (1) We compute the total trending value of $Top$ as the sum of its values in all cells in $S$. (2) If the total value of $Top$ is among the highest $k$ found so far, we update our final answer accordingly. (3) We replace $Top$ in the priority queue by the next item in the top-$k$ list of its cell, if any. This is repeated until either exhausting all top-$k$ lists in $S$ or the maximum possible total value for any remaining keyword is less than the $k^{th}$ entry in the current final answer. This maximum value is upper bounded by the summation of the existing keys in the max-heap. On concluding Step 2, the final top-$k$ list is returned as the query answer. Evaluating only top-$k$ lists of different cells significantly reduces the query response time as shown in our experimental evaluation.

## 8 Experimental evaluation

This section evaluates *GeoTrend+* experimentally. We compare *GeoTrend+* with AFIA [45] and GARNET [26], which are the state-of-the-art and the closest to our work in the literature. Our AFIA implementation uses two spatial grid levels of granularity of $1km \times 1km$ and $10km \times 10km$, and four levels of temporal resolution, hour, day, week, and month. GARNET is primarily proposed for queries of any generic context, where we instantiated context as *location* to use a one-level spatial grid index of resolution $10km \times 10km$ per cell. We use GARNET memory components and limit our evaluation to its in-memory performance, which is the main focus of *GeoTrend+* queries and components. With our comparison to competitor systems, we also evaluate different design choices and modules of *GeoTrend+*, including replicating keywords across index level, materializing top-$k$ list at indexing time, memory optimization techniques, and indexing uncertain locations.

The rest of this section organized as follow. Section 8.1 presents experimental setup. Section 8.2 evaluates query processing. Sections 8.3 and 8.4 evaluate both fixed parameter and adaptive memory optimization techniques. Section 8.5 evaluates indexing uncertain locations. Finally, Section 8.6 evaluates combining both adaptive memory optimization and indexing uncertain locations.

### 8.1 Experimental setup

Our experiments are based on two real *GeoTrend+* prototypes, one implemented during the course of initiating this work in Microsoft Research and the other during extending this work for uncertain locations and adaptive memory optimization, both implemented in multi-threaded servers that use latches for concurrency control. The first prototype is deployed on a server running Windows Server 2012 on Intel 2.40GHZ Core i7 CPU with 64GB RAM, while the second is deployed on a server running Ubuntu 16.04 (64 bit) on Intel Xeon(R) with CPU E5-2637 v4 (3.50 GHz) and 128GB RAM. We use 152 million geotagged tweets obtained from the Twitter archive. The tweets are used to simulate an incoming stream of microblogs with high arrival rates. Each tweet is associated with either an exact point location (latitude and longitude) or uncertain location represented with a minimum bounding rectangle (MBR). By default, exact locations and centroid points of MBRs are used to represent each tweet with a single point location. For keywords, we use hashtags (if present) or select a random word from the tweet text. For queries, we use a query log from Bing Mobile containing actual point locations (latitudes and longitudes) of user search queries on Bing. This query log is used to compose a default query load of 1000 MBR queries (centered around the point locations) with different area sizes that range from $4mi^2$ to $400Kmi^2$, containing 15% with large areas ($40Kmi^2$ to $400Kmi^2$). Unless mentioned otherwise, the default value of $k$ is 100, $N$ is 8 counters per hash entry, $T$ is 24 hours, and $\epsilon$ is 0.001.

All experimental results are collected during steady state after running *GeoTrend+* for at least $T$ time units. All measurements are done in real time, i.e., while the tweet stream is flowing. Our main performance metrics are the supported microblogs arrival rate, memory overhead, query latency, and query result accuracy. Accuracy is calculated as the percentage of entries in the received result that are included in the correct top-$k$ answer computed with infinite resources.

### 8.2 Query processing

This section evaluates *GeoTrend+* index design decisions that affects query processing. We evaluate the *GeoTrend+* pyramid index (denoted with prefix *GT*) against AFIA [45] (denoted as *AFIA*) and GARNET [26] with and without employing its $\epsilon$ memory cleaning process (denoted with prefixes *GRN-*$\epsilon$ and *GRN*, respectively). GARNET $\epsilon$-cleaning process is similar to *GeoTrend+* $\epsilon$-cleaning with a fixed $\epsilon$ value for all index cells. Section 8.2.1 evaluates the effect of replicating keywords across index levels. Section 8.2.2 evaluates the effect of maintaining top-$k$ list inside each index cell.

#### 8.2.1 Keyword replication

In this section, we evaluate the replication of keywords in all pyramid index levels. To this end, we compare the pyramid index with a partial quad-tree index [18] that has similar cell structure to the pyramid, yet, keywords are maintained only in leaf cells (denoted as *GT-QT*). The two indexing structures favor different objectives: (1) The pyramid index maintains keywords aggregates in *all* leaf and non-leaf cells, increasing both memory and insertion overhead, but its query processor accesses far fewer cells, from higher levels, to compute the final answer. (2) The quad-tree index maintains keywords aggregates *only* in leaf cells, reducing both memory and insertion overhead, but increasing the query latency as the query processor accesses many cells to compute the final answer. The experiments results show that the quad-tree would not be able to provide low query latency although it has much lower memory and insertion overhead.

Figure 5 denotes the pyramid index, with and without $\epsilon$-cleaning, as *GT* and *GT-*$\epsilon$, quad tree as *GT-QT*, and GARNET as *GRN-*$\epsilon$, excluding AFIA from query evaluation due to its different aggregate measure. Figure 5a and b show one to three orders of magnitude better query latency for *GT* and *GT-*$\epsilon$ than *GT-QT* and *GRN-*$\epsilon$ with varying answer size $k$ and query region area $R$, respectively. *GT* and *GT-*$\epsilon$ consistently outperform both *GT-QT* and *GRN-*$\epsilon$ for different $k$ values. With changing the area of spatial location under consideration, by varying the value of $R$ starting from 0.004 K $mi^2$ to 40 K $mi^2$ as shown in Fig. 5b, the improvement ratio changes: For small areas, all indexes have almost the same average query latency as the number of processed cells is similar or close. When the area increases, *GT* and *GT-*$\epsilon$ use far fewer cells than both *GT-QT* and *GRN-*$\epsilon$, as they have a chance to use larger non-leaf cells contained in $R$, and therefore they give much lower query latency. This clearly shows the superiority of *GeoTrend+* with increasing the spatial location size up to 10000 times while the query latency remains under 5 milli-seconds.

This lower query latency comes with the cost of higher insertion overhead and larger memory footprint than *GT-QT*. Figure 5c and d show that this is a favorable trade-off with affordable indexing overhead and memory footprint. For different values of $k$, *GT* and *GT-*$\epsilon$ still support up to an order of magnitude higher arrival rate than Twitter rate. Furthermore, *GT-*$\epsilon$ incurs only around three times memory overhead compared to *GT-QT*. On the contrary, *GRN-*$\epsilon$ still encounter high memory footprint due to the large number of cells in a fine-divided grid index with high resolution. This shows the effectiveness of *GeoTrend+* design decisions to provide an excellent compromise in both memory overhead and query latency.

#### 8.2.2 Materializing Top-k lists

The query answer can be computed either by using all keywords within the cell, which are expected to be huge with many keywords, or by exploiting only top-$k$ items in each

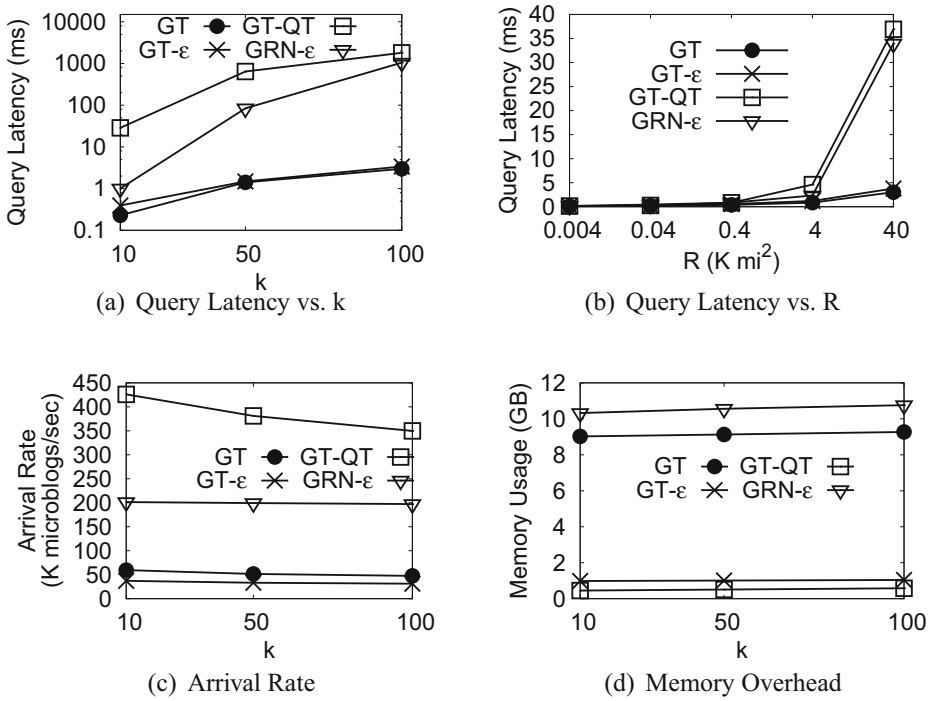



**Fig. 5** Impact of keyword replication across pyramid index levels

cell as introduced in Section 7. We show that maintaining these lists reduces query latency significantly at the cost of acceptable overhead to store and maintain the sorted lists while continuously inserting new keywords and deleting old information and acceptable reduction in query accuracy. In this section we evaluate the effect of maintaining top-$k$ lists on query latency, query accuracy, and insertion overhead, excluding memory overhead effect as the storage of top-$k$ is negligible compared to the cell all keywords storage. The experimental results show two orders of magnitude improvement in query latency with sublinear increase in insertion overhead.

Figure 6 compares *GeoTrend+* (denoted as *GT*), AFIA (denoted as *AFIA*), and GARNET (denoted as *GRN*), with and without maintaining top-$k$ lists (denoted with suffix *K* and *NK*, respectively). Note that AFIA has only top-$k$ option as this is the only maintained data structure in its index cell. It is also excluded from query measures as it supports only top-$k$ frequent queries and cannot adapt our trending measure. Figure 6a depicts the query latency of all alternatives for different $k$ values. We observe that maintaining the top-$k$ lists reduces query latency of *GeoTrend+* alternatives from 850 msec for all values of $k$ to between 1 and 3 msec, which is two orders of magnitude reduction. *GRN* query latency is consistently much higher than *GeoTrend+* for two reasons. First, the large number of cells processed from its fine-divided grid index compared to cells of high levels of *GeoTrend+* index which is much smaller in number. Such inefficient division for the space is a result for *GRN* generic framework for any context, so it is not tailored for location queries and thus cannot make maximum use of the spatial properties of the data. Second, *GRN* computes its

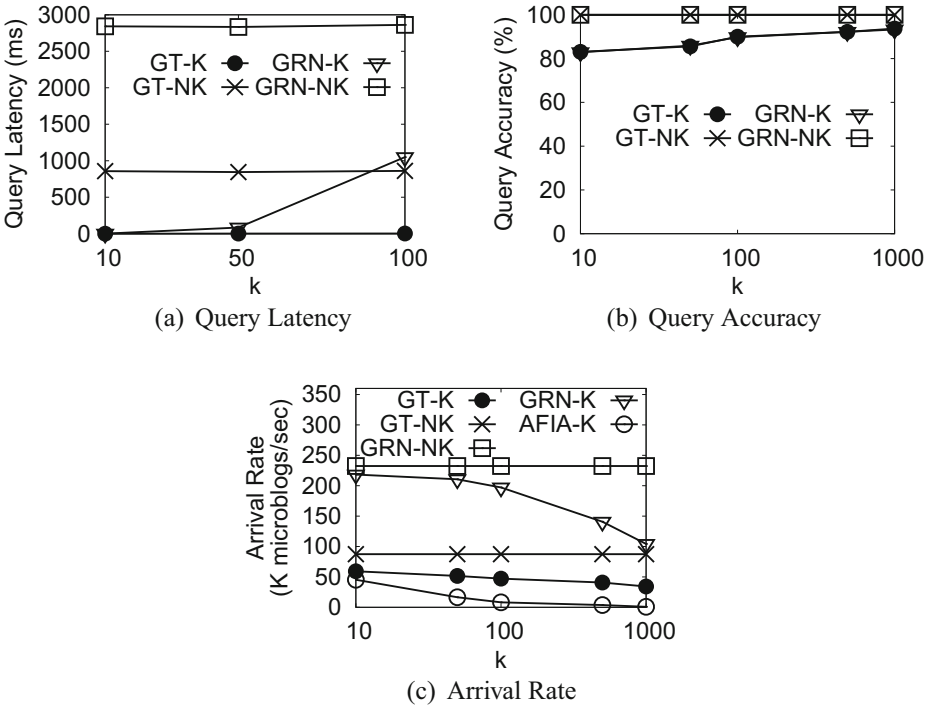



**Fig. 6** Impact of maintaining top-$k$ lists

aggregate measures from different temporal cells, as it is originally proposed and optimized for arbitrary temporal periods, which increase the aggregation time.

Figure 6b shows that for $k \geq 100$, aggregating from top-$k$ lists provides at least 90% accuracy, providing an empirical evidence for the effectiveness of using top-$k$ lists with an acceptable accuracy loss. Figure 6c show the overhead of maintaining the top-$k$ lists on index insertion. *AFIA* supports the lowest arrival rates due to its cell replication over both spatial and temporal dimensions. For *GeoTrend+* and GARNET, the significant reduction in query latency comes at the cost of 50% reduction in the supported arrival rate. For the worst case ($k$=1000) in Fig. 6c, *GeoTrend+* index supports at least 40,000 microblog/sec which is seven times the current Twitter rate.

### 8.3 Memory optimization

This section and the following section evaluate the impact of our memory optimization techniques, both *TrendMem* and *AdaptiveTrendMem*, on memory overhead, index scalability, and query accuracy. The evaluation of *TrendMem* against the competitors AFIA and GARNET uses the first system prototype while the comparing *TrendMem* with *AdaptiveTrendMem* uses the second system prototype.

Figure 7 shows the memory usage of *GeoTrend+*'s *TrendMem* against AFIA (denoted as *AFIA*) and GARNET with and without employing its $\epsilon$ memory cleaning process (denoted as *GRN-*$\epsilon$ and *GRN*, respectively). Figure 7a and b depict the memory usage for different values of $k$ and $\epsilon$, respectively. For different values of $k$ (Fig. 7a), only *AFIA* memory usage

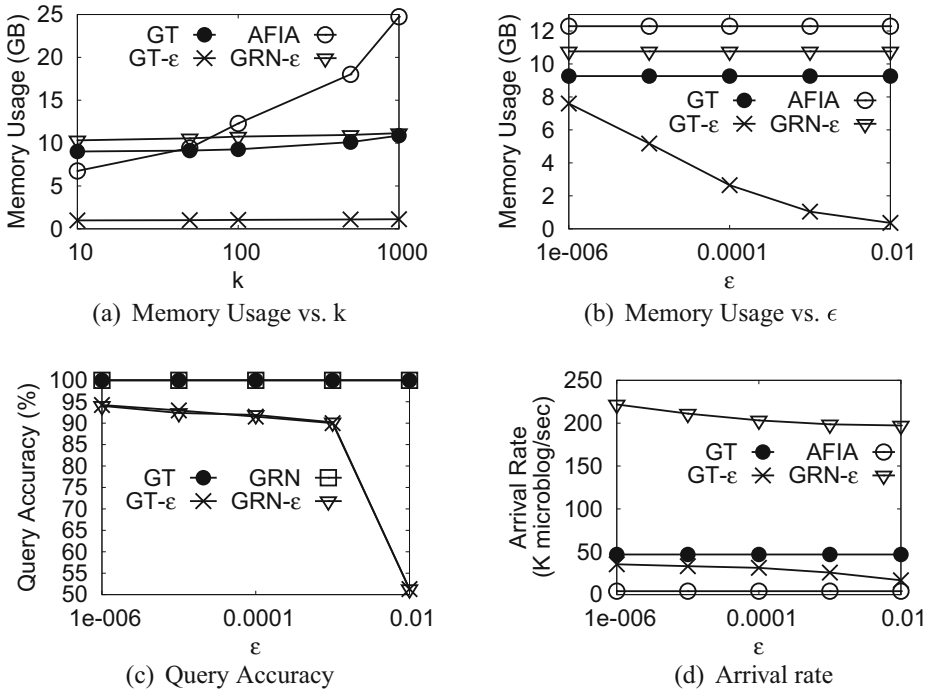



**Fig. 7** Impact of memory optimization module

increase significantly while the rest of technique encounter relatively stable memory usage. The highest *AFIA* memory usage (at $k$=1000) is around 24GB excluding programming language overhead. Such large overhead comes for two reasons. With increasing $k$, the number of items in archived dynamic summaries are increasing significantly and hence it consumes more memory. In addition, such dynamic summaries are replicated in multi-resolution over both spatial and temporal dimensions per its index structure. This even amplifies the effect of increasing $k$ and encounter high memory consumption. Both *GT* and *GRN*-$\epsilon$ encounter nearly 40% of *AFIA* memory. Yet, *GT*-$\epsilon$ can significantly improves this and consumes less than 10% of *AFIA* memory. The amount of memory saving is actually changing with different $\epsilon$ values as Fig. 7b shows. This figure shows memory usage of *GT*-$\epsilon$ is reducing dramatically with increasing $\epsilon$ as more keywords are removed from all index cells. However, *GRN*-$\epsilon$ consumes relatively high memory due to the large number of cells it maintain. Also, $\epsilon$ value does not have significant effect on its memory overhead as its spatial cell size is much smaller, then each cell receives much less keywords and so $\epsilon$ removes relatively stable amount of keywords.

The effect of reducing memory overhead is shown in Fig. 7c and d on query accuracy and supported arrival rates of incoming microblogs. *AFIA* is not included in query accuracy as it support only top-$k$ frequent queries and cannot adapt our trending measure. For different values of $\epsilon > 0.01$, query accuracy exceeds 90% for both *GT*-$\epsilon$ and *GRN*-$\epsilon$. In Fig. 7d, *GRN*-$\epsilon$ supports the highest arrival rate due to its simple index structure (one-level grid index) while *AFIA* still supports the lowest arrival rates due to its high replication overhead on both spatial and temporal dimensions as mentioned earlier. *GeoTrend+* alternatives come

in the middle of both and still can support up to 50K microblog/second which is an order of magnitude higher than current Twitter rate.

### 8.4 Adaptive memory optimization

This section evaluates the impact of *AdaptiveTrendMem* technique that changes the $\epsilon$ value over different index levels, compared to *GeoTrend+* without any memory optimization and with *TrendMem* that has a fixed $\epsilon$ value for the whole index. The evaluation includes memory overhead, index scalability, and query accuracy. *GeoTrend+* is denoted as *GT*. *TrendMem* is denoted as *GT-*$\epsilon$. *AdaptiveTrendMem* has two evaluated variations (described in Section 6.3): the *variable decimal digit* scheme (VDD) is denoted as *GT-VDD*$\epsilon$ and the *variable level number* scheme (VLN) is denoted as *GT-VLN*$\epsilon$.

Figure 8 shows the impact of the four alternatives on supported arrival rate, memory usage, and query accuracy. Figure 8a and b show that *GT-*$\epsilon$ outperforms the other three alternatives in both memory usage and supported arrival rate. This is interpreted by the small $\epsilon$ value in the deep levels of the index of *GT-VDD*$\epsilon$ and *GT-VLN*$\epsilon$, while $\epsilon$ is zero in all *GT* cells. This small value triggers less frequent periodic cleaning cycles and each cycle removes fewer keywords, which leads to slightly more content in main-memory and less efficient insertions in turn. However, the arrival rate for all alternatives, except *GT*, is increasing with increasing $\epsilon$ as the index removes more useless content more frequently. *GT-VLN*$\epsilon$ outperforms *GT-VDD*$\epsilon$ in both figures, showing the superiority of *variable level*

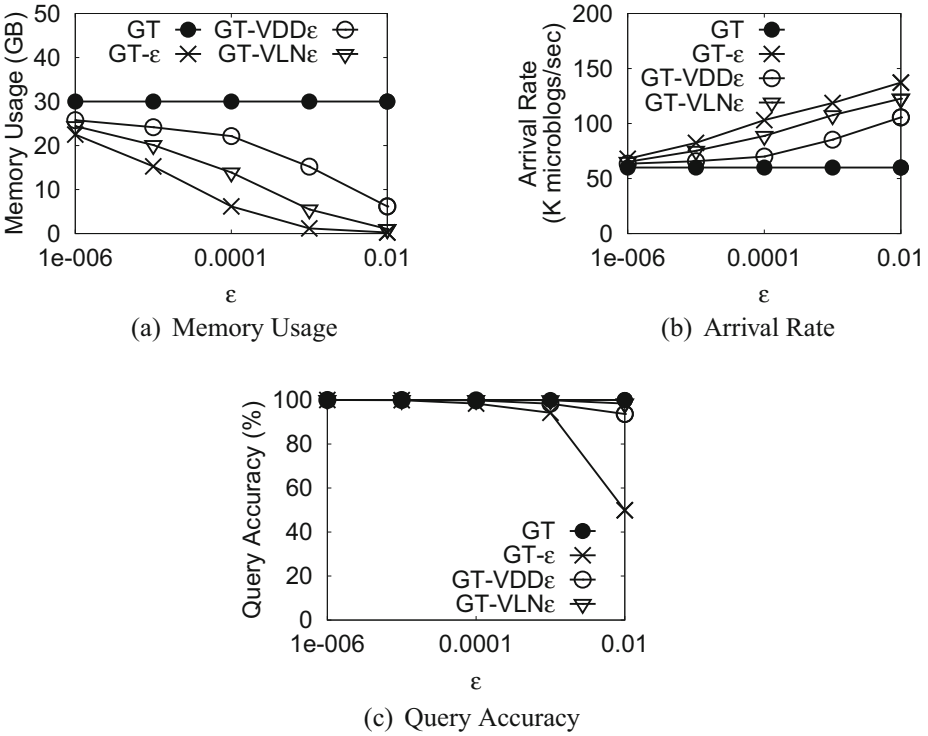


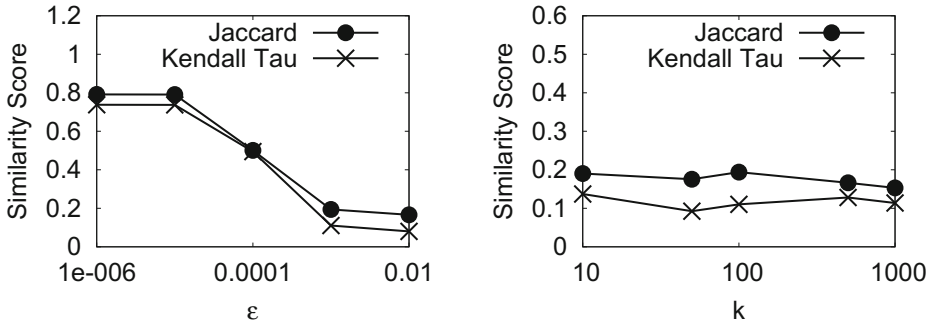

**Fig. 8** Impact of adaptive memory optimization module

*number* scheme over *variable decimal digit* scheme. The slight performance loss of adaptive optimization techniques comes with lifting query accuracy to almost 100% for all values of $\epsilon$ as shown in Fig. 8c. The figure shows robust and perfect accuracy for *GT-VLN*$\epsilon$ for all values of $\epsilon$, equivalent to *GT* that does not remove any content. This shows the ability of *GT-VLN*$\epsilon$ to spill only the useless data without spilling any data that serves incoming queries. *GT-VDD*$\epsilon$ though still performs very well with slight decrease in query accuracy at a relatively large $\epsilon = 0.01$. On the contrary, *GT-*$\epsilon$ accuracy is drastically decreased to 50% for that $\epsilon$ value. For small values of $\epsilon$, all the alternatives perform about the same. These experiments show that the introduced adaptive memory optimization techniques are able to distinguish the correct useless data to spill from main-memory, for different $\epsilon$ values, to maintain perfect query accuracy while still reducing the overall memory footprint significantly.

### 8.5 Indexing uncertain locations

This section evaluates the impact of indexing tweets associated with uncertain locations, where locations are represented as minimum bounding rectangles (MBR) that cover a spatial range, such as a district or a city, rather than an exact point location. We evaluate the impact of location uncertainty on the indexing overhead, memory usage, and query top-$k$ answer similarity compared to using exact locations. The top-$k$ answer similarity gauges the importance of considering uncertain locations. Considering uncertainty locates each tweet in as many index cells as its location spans. On the contrary, representing an MBR with the centroid point, each tweet is located only in one index cell even if it spans a wide area such as a city. We compare answers in both cases to evaluate whether considering uncertainty makes a difference or the two answers are closely similar. We use two similarity measures, Jaccard coefficient and Kendall Tau similarity [14]. Jaccard measure considers the intersection between the two top-$k$ lists regardless the rank of each item in the list. On the contrary, Kendall Tau measure considers the similarity of items' ranks in the two lists.

Figures 9 , 10 and 11 compare two versions of *GeoTrend+* memory optimized index, one that approximates all locations with an exact point (denoted as *GT-*$\epsilon$) and the other index uncertain locations as a whole area (*GT-U*$\epsilon$) as described in Section 5. Figure 9 shows the similarity between query answers that are retrieved based on both ways with different $\epsilon$ values (Fig. 9a) and $k$ values (Fig. 9b). The peak similarity between answers of the two alternatives is 80% at $\epsilon \leq 10^{-5}$, which are small values approaching zero and




(a) Answer Similarity vs. $\epsilon$ (b) Answer Similarity vs. k

**Fig. 9** Impact of indexing uncertain locations on query answers

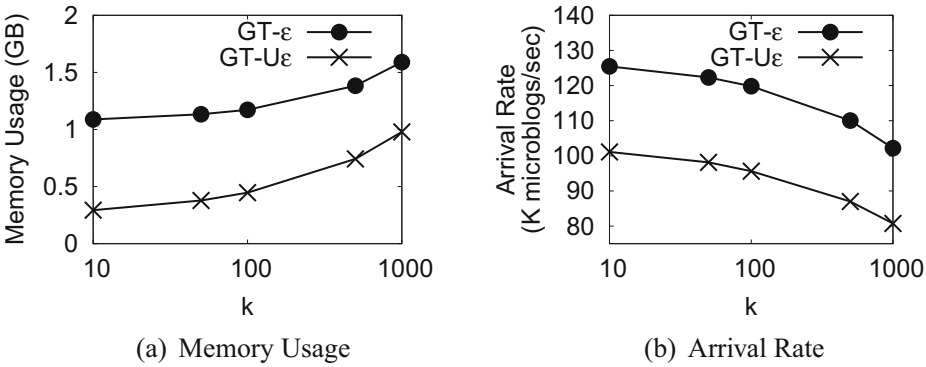



**Fig. 10** Impact of indexing uncertain locations on indexing for different $k$ values

give low memory optimization performance. This similarity is significantly decreasing with increasing $\epsilon$ to below 20%. The different values of $k$ do not highly affect this number to remain around the 20% as depicted in Fig. 9b. This means missing over 80% of the correct answer when approximating uncertain locations with a single point with low $\epsilon$ values, which might be unacceptable for several applications. These low similarity numbers motivate the importance of indexing uncertain locations to get results that reflects the actual location distributions. It is obvious that even with encountering high memory overhead of keep almost all data, at low values of $\epsilon$ in Fig. 9a, we are still missing over 20% of the correct answer when approximating uncertain locations, which is still significant. Given that our *GeoTrend+* index is encountering reasonable overhead for indexing uncertain locations, as we are going to elaborate in the rest of this section, this small additional overhead provides a great compromise for enhancing the query result.

Figure 10 shows their performance in terms of memory overhead and index arrival rate for different values of $k$. Figure 10a shows the memory consumption of the two alternatives. The memory consumption of *GT-U*$\epsilon$ is always lower than *GT-*$\epsilon$. The main reason is that insertion cycle counters are incremented in much higher frequency in case of *GT-U*$\epsilon$ because inserting a single microblog increment insertion counters in multiple cells at once. As both alternatives perform a periodic cleaning of infrequent keywords based on number

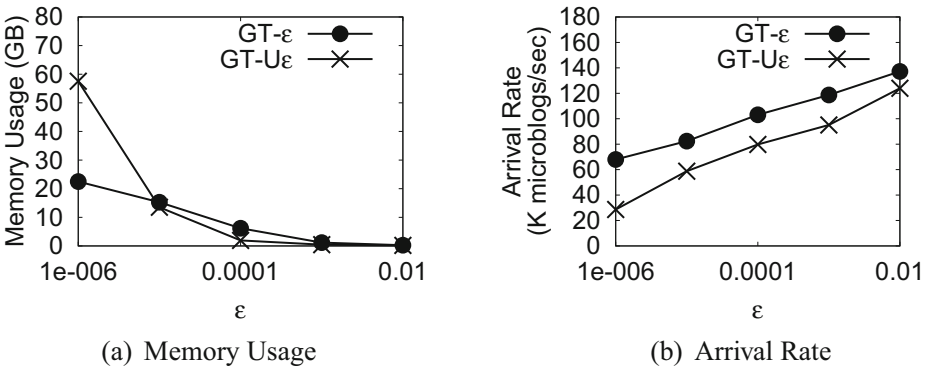
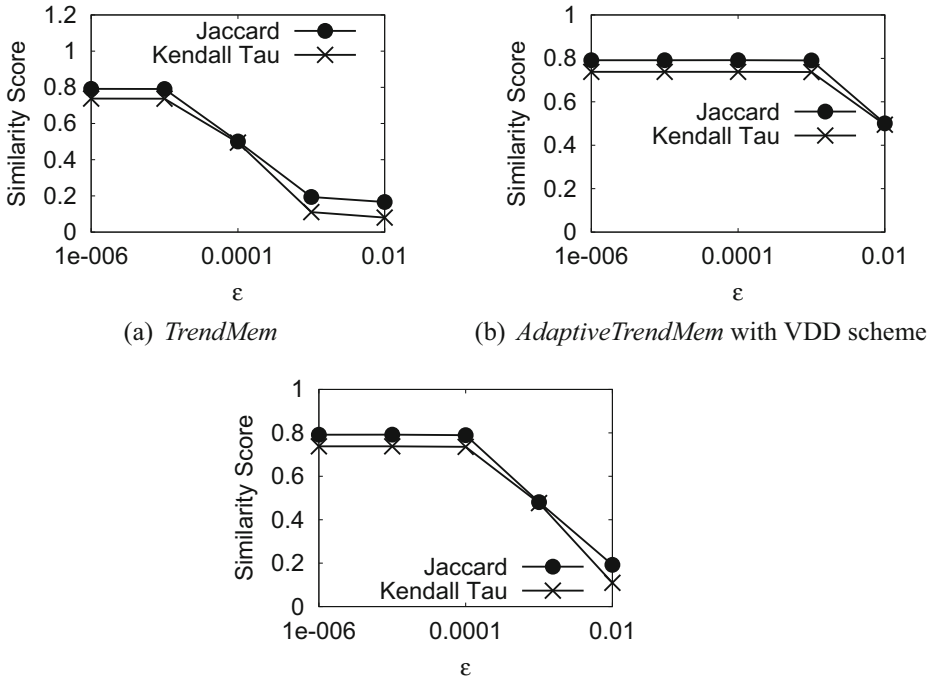



**Fig. 11** Impact of indexing uncertain locations on indexing for different $\epsilon$ values

of insertions in the cell, *GT-Uϵ* triggers this cleaning more frequently than *GT-ϵ* and wipe up more useless data from main-memory, and in turn consumes less memory overhead. Figure 10b depicts the supported arrival rates of incoming microblogs of the two alternatives for different $k$ values. *GT-ϵ* starts with 125K microblog/second (at $k$=10) and then it decreases with increasing $k$, while *GT-Uϵ* starts with 100K microblog/second (at $k$=10), and it follows the same decrease pattern as *GT-ϵ*. The reason that both alternatives support less arrival rates with increasing $k$ is the overhead of maintaining longer top-$k$ list. In addition, Fig. 10b shows that the cost of indexing uncertain location data is around 20% reduction in the supported arrival rate. The consistent lower rate supported by *GT-Uϵ* is due to the overhead of inserting the same microblog in multiple index cells instead of only one cell and invoking $\epsilon$-cleaning more frequently due to incrementing insertion counters in these multiple cells at once. However, even with this 20% reduction, we are still able to support high data rate, much higher than Twitter arrival rates. This reasonable reduction comes with advantages in query answers as discussed earlier in this section.

Figure 11 shows the effect of different $\epsilon$ values on the memory consumption and index arrival rate. Figure 11a shows the memory consumption for different $\epsilon$ values. As expected, with weak memory optimization at small values of $\epsilon$, *GT-Uϵ* encounter higher memory storage as it replicates the same keyword in multiple cells. However, slightly increasing the $\epsilon$ value reverses this situation, where *GT-Uϵ* start to invoke memory cleaning more frequently, as described earlier, and gets rid of larger amounts of useless data, and so encounter less memory overhead than *GT-ϵ*. Figure 11b shows that increasing $\epsilon$ value leads to digesting




(a) *TrendMem*

(b) *AdaptiveTrendMem* with VDD scheme

(c) *AdaptiveTrendMem* with VLN scheme

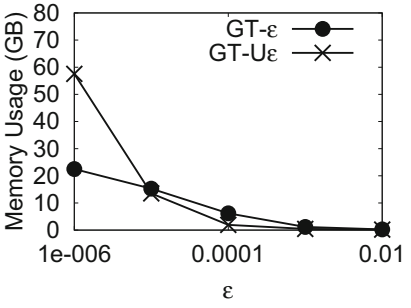
**Fig. 12** Impact of combining adaptive memory optimization and indexing uncertain locations on query answers

more data as the index becomes lighter due to triggering more frequent cleaning, and the insertion is less costly. *GT-ϵ* still outperforms *GT-Uϵ* due to the same reasons mentioned earlier. However, *GT-Uϵ* still encounter a slight decrease in supported rates and can support much higher than Twitter rates.

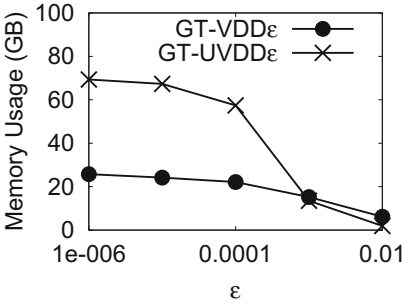
### 8.6 Combining uncertain locations with adaptive memory optimization

The previous two sections evaluate the impact of adaptive memory optimization and indexing uncertain location separately to show the pros and cons of each technique in isolation from other factors. However, *GeoTrend+* index is equipped with the two sets of techniques in the same structure. This section evaluates different versions of *GeoTrend+* index that combines both techniques revealing new insights on different performance measures.

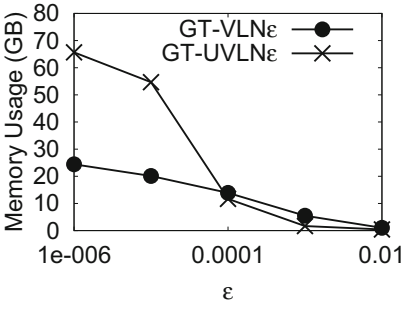
Figure 12 shows the impact of combining adaptive memory optimization and indexing uncertain locations on query answers. The similarity is measured between top-*k* answers with and without indexing uncertain locations as described in Section 8.5. Figure 12a shows the effect with *TrendMem* technique, and it is duplicated from Fig. 12a for better readability, while Fig. 12b and c show the effect of adaptive memory optimization, *AdaptiveTrendMem* technique, with the two variations: the *variable decimal digit* scheme (VDD) and the *variable level number* scheme (VLN), as described in Section 6.3. The figure shows that adaptive ϵ memory optimization enhances query answer similarity for relatively large ϵ values compared with fixed ϵ memory optimization. *AdaptiveTrendMem* with VDD (Fig. 12b)




(a) *TrendMem*




(b) *AdaptiveTrendMem* with VDD scheme

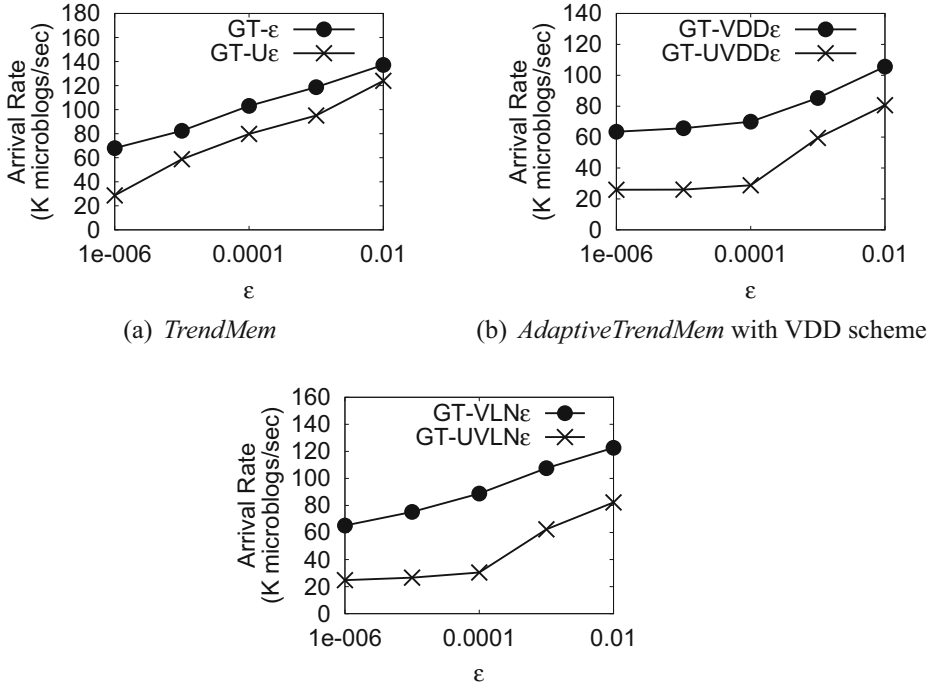



(c) *AdaptiveTrendMem* with VLN scheme

**Fig. 13** Impact of combining adaptive memory optimization and indexing uncertain locations on memory usage

is the most robust technique, followed by *AdaptiveTrendMem* with VLN (Fig. 12c), and finally *TrendMem* (Fig. 12a). Figure 12b sustains the maximum similarity, 80%, for up to $\epsilon$ =0.001, while its worst similarity at $\epsilon$ =0.01 is 40%. This is much better performance compared with other alternatives where similarity drops below 20%. This answer similarity performance is interpreted by the corresponding memory usage performance that is depicted in Fig. 13. Figure 13 denotes *GeoTrend+* with *TrendMem* as *GT-*$\epsilon$, and with *AdaptiveTrendMem* as *GT-VDD*$\epsilon$ for VDD scheme and as *GT-VLN*$\epsilon$ for VLN scheme, where all alternative has *U* to indicate uncertain locations. *AdaptiveTrendMem* with VDD (Fig. 13b) encounters higher memory footprint up to $\epsilon$ =0.001. This extra memory content sustains the answer of approximated locations similar to the answer of non-approximated locations for large variation of parameter values. On the contrary, other alternatives (in Fig. 13a and c) keep lower memory footprints by spilling content that demonstrates the effect of approximating location. In all cases, the answer similarity does not exceed 80%, which shows that approximating locations leads to missing a significant portion of the correct answer. However, varying memory content with different optimization techniques changes the robustness of this missing part for different parameter values.

Figure 14 shows the impact of combining adaptive memory optimization and indexing uncertain locations on arrival rate. *TrendMem* (Fig. 14a) shows the lowest decrease in arrival rate for indexing uncertain locations compared to *AdaptiveTrendMem* with VDD (Fig. 14b) and with VLN (Fig. 14c). The main reason is the lower values of $\epsilon$ that are employed in lower index levels in *AdaptiveTrendMem*. However, this effect is being demoted with




(a) *TrendMem*

(b) *AdaptiveTrendMem* with VDD scheme

(c) *AdaptiveTrendMem* with VLN scheme

**Fig. 14** Impact of combining adaptive memory optimization and indexing uncertain locations on arrival rate

increasing $\epsilon$ value. With the default $\epsilon$ =0.001, all techniques still digest high rates that are at least eight times Twitter rate. This shows the scalability of all *GeoTrend+* components with all different settings even with providing significant memory optimization and high query accuracy.

## 9 Conclusion

In this paper, we presented *GeoTrend+*; a scalable system that supports spatial trending queries on recent microblogs. *GeoTrend+* supports a variety of trending measures that suit different applications. It also supports queries on arbitrary spatial regions using data that has recently arrived in the last $T$ time units. For this, it employs an efficient main-memory spatial index that digests and expires data with high rates. The index is able to digest data with both exact point locations and uncertain locations. In peak times, where main-memory is overwhelmed, it employs a smart memory optimizer that sheds less important data while keeping highly accurate query answers. The memory optimizer employs both fixed and adaptive parameters to distinguish useless from useful data at different spatial levels. The experimental evaluation shows the scalability of *GeoTrend+* to digest much higher rates than prominent microblogging services, while providing average query latency of few milliseconds and sustaining high performance with limited memory. Compared to existing competitors, *GeoTrend+* has a clear advantage in both main-memory optimization and query latency in all parameter settings. These two optimization goals ensure sustainability of digesting new real-time streaming data in tight memory environments, while still serving scalable applications with low query latency. In addition, *GeoTrend+* trades-off this low resources overhead with indexing time and effective query accuracy that are still efficient and applicable in microblogs applications as they support multiple times higher rates compared to average Twitter data.

**Acknowledgments** Amr Magdy acknowledges the support of the National Science Foundation under Grants Number IIS-1849971, SES-1831615, and CNS-1837577. Mohamed Mokbel acknowledges the support of the National Science Foundation under Grants Number IIS-1525953, CNS-1512877, and IIS-1907855. Walid Aref acknowledges the support of the National Science Foundation under Grants Number III-1815796, and IIS-1910216.

## Appendix: Trend Line Slope

*GeoTrend+* uses statistical linear regression slope to measure the trendiness of a certain keyword. The following Lemma derives the equation that determines the trendiness of a keyword:

**Lemma 1** *Given a keyword consecutive frequencies vector* $f = [f_0, f_1, ..., f_N]$*, the keyword trend line can be estimated with the following formula:*

$$Trend_{reg} = \frac{\sum_{i=1}^{N}[i \times (f_i - f_0)]}{N(N+1)(2N+1)} \tag{3}$$

*Proof* The simple linear regression slope $Trend_{reg}$ of $x$ and $y$ is given with the following equation:

$$Trend_{reg} = \frac{Mean(xy)}{Mean(x^2)} \tag{4}$$

Where $Mean(x)$ is the average value of the vector and $xy$ is a vector that results from value-wise multiplication of the vectors $x$ and $y$. In *GeoTrend+*, the vector $x$ values are always constants while the vector $y$ contains the frequencies of a keyword $W$. Thus values of vector $x$ are always be $[1, 2, 3, ..., N]$ while values of vector $y$ are $[f_1, f_2, f_3, ..., f_N]$. Thus, $Mean(x^2)$ can be simplified as $\frac{(N+1)(2N+1)}{6}$. On the other hand, $Mean(xy)$ can be calculated as $\frac{\sum_{i=1}^{N} i \times f_i}{N}$. Substitutes both variables to Equation 1:

$$Trend_{reg} = \frac{\frac{\sum_{i=1}^{N} i \times f_i}{N}}{\frac{(N+1)(2N+1)}{6}} = \frac{6\sum_{i=1}^{N} i \times f_i}{N(N+1)(2N+1)} \tag{5}$$

The equation above assumes that the measurement is used from the start of the stream and each keyword $W$ starts from frequency 0. However, in *GeoTrend+*, we need to consider the start position of a keyword $W$ by using the previous frequency, namely $f_0$. Thus, the equation above can be modified to:

$$Trend_{reg} = \frac{6\sum_{i=1}^{N} [i \times (f_i - f_0)]}{N(N+1)(2N+1)} \tag{6}$$

□

## References


1. Abdelhaq H, Sengstock C, Gertz M (2013) EvenTweet: Online Localized Event Detection from Twitter. In: VLDB
2. Ahmed P, Hasan M, Kashyap A, Hristidis V, Tsotras VJ (2017) Efficient Computation of Top-k Frequent Terms over Spatio-temporal Ranges. In:s SIGMOD
3. Arasu A, Manku GS (2004) Approximate counts and quantiles over sliding windows. In: PODS
4. Aref WG, Samet H (1990) Efficient processing of window queries in the pyramid data structure. In: PODS
5. Social media 'outstrips TV' as news source for young people. http://www.bbc.com/news/uk-36528256, 2016
6. After Boston Explosions, People Rush to Twitter for Breaking News. http://www.latimes.com/business/technology/la-fi-tn-after-boston-explosions-people-rush-to-twitter-for-breaking-news-20130415,0,3729783.story, 2013
7. Budak C, Agrawal D, El Abbadi A (2011) Structural trend analysis for online social networks. PVLDB 4(10):646–656
8. Budak C, Georgiou T, Agrawal D, El Abbadi A (2014) GeoScope: Online detection of Geo-Correlated information trends in social networks. In: VLDB
9. Busch M, Gade K, Larson B, Lok P, Luckenbill S, Lin J (2012) Earlybird: real-time search at twitter In: ICDE
10. Chi Y, Tseng BL, Tatemura J (2006) Eigen-Trend: trend analysis in the blogosphere based on singular value decompositions. In: CIKM, pp 68–77
11. Weibo S China Twitter, comes to rescue amid flooding in Beijing. http://thenextweb.com/asia/2012/07/23/sina-weibo-chinas-twitter-comes-to-rescue-amid-flooding-in-beijing/, 2012
12. Cunha E, Magno G, Comarela G, Almeida V, Gonçalves MA, Benevenuto F (2011) Analyzing the dynamic evolution of hashtags on twitter: a language-based approach. In: Proceedings of the Workshop on Languages in Social Media, pp 58–65
13. Datar M, Gionis A, Indyk P, Motwani R (2002) Maintaining stream statistics over sliding windows (extended abstract). In: SODA
14. Fagin R, Kumar R, Sivakumar D (2003) Comparing Top k Lists. SIAM J Discret Math 17(1):134–160
15. Fagin R, Lotem A, Naor M (2001) Optimal aggregation algorithms for middleware. In: PODS, pp 102–113
16. Farazi S et al (2019) Top-K Spatial term queries on streaming data. In: ICDE
17. Feng W, Han J, Wang J, Aggarwal C, Huang J (2015) STREAMCUBE: Hierarchical Spatio-temporal Hashtag Clustering for Event Exploration Over the Twitter Stream. In: ICDE

18. Finkel RA, Bentley JL (1974) Quad Trees: A Data Structure for Retrieval on Composite Keys. ACTA, 4(1)
19. Gao H, Tang J, Liu H (2012) Exploring Social-Historical ties on Location-Based social networks. In: The 6th Intl AAAI Conf on Weblogs and Social Media
20. Golab L, DeHaan D, Demaine ED, López-Ortiz A, Ian Munro J (2003) Identifying frequent items in sliding windows over on-line packet streams. In: Internet Measurement Comference
21. Us department of health and human services disease tracking. https://nowtrending.hhs.gov
22. Hong L, Ahmed A, Gurumurthy S, Smola AJ, Tsioutsiouliklis K (2012) Discovering geographical topics in the twitter stream. In: WWW
23. Huang J, Peng M, Wang H, Cao J, Gao W, Zhang X (2017) A probabilistic method for emerging topic tracking in microblog stream. World Wide Web 20(2):325–350
24. Ikawa Y, Enoki M, Tatsubori M (2012) Location inference using microblog messages. In: WWW
25. Indyk P, Koudas N, Muthukrishnan S (2000) Identifying representative trends in massive time series data sets using sketches. In: VLDB, pp 363–372
26. Jonathan C, Magdy A, Mokbel M, Jonathan A (2016) GARNET A holistic system approach for trending queries in microblogs. In: ICDE
27. Kenney JF, Sydney E (1962) Keeping. Mathematics of Statistics, Part 1, chapter 15, pp 252–285. van Nostrand 3rd edn
28. Kim K-S, Kojima I, Ogawa H (2016) Discovery of local topics by using latent spatio-temporal relationships in geo-social media. Int J Geogr Inf Sci 30(9):1899–1922
29. Krumm J, Eyewitness EH (2015) Identifying local events via space-time signals in twitter feeds. In: Proceedings of the 23rd Sigspatial International Conference on Advances in Geographic Information Systems, ACM, p 20
30. Lazaridis I, Mehrotra S (2001) Progressive approximate aggregate queries with a Multi-Resolution tree structure. In: SIGMOD, pp 401–412
31. Lee L-K, Ting HF (2006) A simpler and more efficient deterministic scheme for finding frequent items over sliding windows. In: PODS
32. Li G, Jun H, Feng J (2014) Kian-lee tan effective location identification from microblogs. In: ICDE
33. Li R, Lei KH, Khadiwala R, Chen-Chuan K (2012) Chang. TEDAS: a twitter-based event detection and analysis system. In: ICDE
34. López IFV, Snodgrass RT, Moon B (2005) Spatiotemporal Aggregate Computation: A Survey. TKDE 17(2):271–286
35. Magdy A, Aly AM, Mokbel MF, Elnikety S, He Y, Nath S, Aref WG (2016) GeoTrend: Spatial Trending Queries on Real-time Microblogs. In: SIGSPATIAL
36. Magdy A, Mokbel MF, Elnikety S, Nath S, Mercury YH (2014) A memory-constrained spatio-temporal real-time search on microblogs. In: ICDE
37. Manku GS, Motwani R (2002) Approximate frequency counts over data streams. In: VLDB
38. Mathioudakis M, TwitterMonitor NK (2010) Trend detection over the twitter stream. In: SIGMOD
39. How Michael Jackson's Death Shut Down Twitter, Brought Chaos to Google, and Killed Off Jeff Goldblum. https://www.dailymail.co.uk/sciencetech/article-1195651/How-Michael-Jacksons-death-shut-Twitter-overwhelmed-Google–killed-Jeff-Goldblum.html, 2009
40. Nath S, Lin F (2013) Lenin ravindranath, and jitu padhye. Smartads: Bringing contextual ads to mobile apps. In: ACM Mobisys
41. Nguyen K, Tran DA (2011) An analysis of activities in Facebook. In: IEEE Consumer communications and networking conference (CCNC)
42. Papadias D, Kalnis P, Zhang J, Tao Y (2001) Efficient OLAP operations in spatial data warehouses. In: SSTD, pp 443–459
43. Sankaranarayanan J, Samet H, Teitler BE, Lieberman MD, TwitterStand JS (2009) News in tweets. In: GIS
44. Shin S, Choi M, Choi J, Langevin S, Bethune C, Horne P, Kronenfeld N, Kannan R, Drake B, Park H et al (2017) Stexnmf: Spatio-temporally exclusive topic discovery for anomalous event detection. In: 2017 IEEE International Conference on Data Mining (ICDM), IEEE, pp 435–444
45. Skovsgaard A, Sidlauskas D, Jensen CS (2014) Scalable top-k spatio-temporal term querying. In: ICDE, pp 148–159
46. Tao Y, Kollios G, Considine J, Li F, Papadias D (2004) Spatio-Temporal Aggregation using sketches. In: ICDE, p 214–225
47. Trends 24. http://trends24.in
48. Twitter Location Trends. https://support.twitter.com/articles/101125#Trend_Location
49. Le HV, Takasu A (2018) Parallelizing top-k frequent spatio-temporal terms computation on key-value stores. In: SIGSPATIAL

50. Weber I, Garimella VRK (2014) Visualizing user-defined, discriminative geo-temporal twitter activity. In ICWSM
51. Wei H, Sankaranarayanan J, Samet H (2017) Finding and tracking local twitter users for news detection. In: SIGSPATIAL
52. Wei H, Sankaranarayanan J, Samet H (2017) Measuring spatial influence of twitter users by interactions. In: Proceedings of the 1st ACM SIGSPATIAL Workshop on Analytics for Local Events and News
53. Wei H, Sankaranarayanan J, Samet H (2018) Enhancing local live tweet stream to detect news. In: Proceedings of the 2nd ACM SIGSPATIAL Workshop on Analytics for Local Events and News
54. Lingkun W, Lin W, Xiao X, Yabo X (2013) LSII An indexing structure for exact Real-Time search on microblogs. In: ICDE
55. Zhang Donghui, Tsotras VJ, Gunopulos D (2002) Efficient aggregation over objects with extent. In: PODS, pp 121–132
56. Zhang T, Zhou B, Huang J, Jia Y, Zhang B, Li Z (2017) A refined method for detecting interpretable and Real-Time bursty topic in microblog stream. In: WISE



**Abdulaziz Almaslukh** received the MSc degree from the Department of Computer Science, University of Southern California, Los Angeles, in 2014. He is currently pursuing a Ph.D. degree in Computer Science at the University of California, Riverside. His research interests include big data management, spatial data management, and spatio-temporal data indexing Techniques.

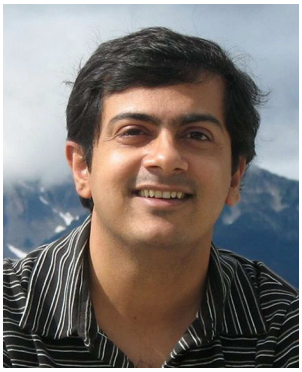
**Amr Magdy** is an Assistant Professor of Computer Science and Engineering and a co-founding faculty member of the Center for GeoSpatial Sciences at UC Riverside. His research interests include database systems, spatial data management, big data management, large-scale data analytics, indexing, and main-memory management. His research is published in prestigious research venues, including ACM SIGMOD, ACM SIGSPATIAL, IEEE ICDE, IEEE TKDE, and VLDB Journal. His research is recognized among best papers in IEEE ICDE 2014 and has been incubated by several industrial collaborators, including a patented system that is commercialized by a social media analytics company with access to all Twitter data. His research is supported with multiple NSF awards, including NSF CRII 2019 award.

**Ahmed M. Aly** obtained his PhD from Purdue University in 2015 and joined Google right afterwards. Ahmed's research interests lie in the broad area of database systems, with a focus on the problems related to query optimization as well as the management of big data. Ahmed's research has been published and demonstrated in prestigious research venues, including VLDB, IEEE ICDE, EDBT, WSDM, ACM SIGSPATIAL, and ACM TSAS.

**Mohamed F. Mokbel** received the BSc and MS degrees from Alexandria University, Egypt,and the PhD degree from Purdue University. He is a professor at the University of Minnesota. His current research interests focus on providing database and platform support for spatio-temporal data, location-based services2.0, personalization, and recommender systems. His research work has been recognized by four Best Paper Awards at IEEE MASS 2008, IEEEMDM 2009, SSTD 2011, and ACM MobiGIS Workshop 2012, and by the US National Science Foundation (NSF)CAREER award 2010. He is/was general co-chair of SSTD 2011, pro-gram co-chair of ACM SIGSPAITAL GIS 2008-2010, and MDM 2014,2011. He has served in the editorial board of the ACM Transactions on Spatial Algorithms and Systems, IEEE Data Engineering Bulletin, Distributed and Parallel Databases Journal, and Journal of Spatial Information Science. He has held various visiting positions at the Microsoft Research, Hong Kong Polytechnic University, and Umm Al-Qura University, Saudi Arabia. He was elected a chair of ACM SIGSPATIAL 2014-2017. He is a senior member of the ACM and IEEE, and a founding member of the ACM SIGSPATIAL. For more information, please visit: www.cs.umn.edu/~mokbel.

**Sameh Elnikety** received the MS degree from Rice University in Houston, Texas, and the PhD degree from the Swiss Federal Institute of Technology (EPFL) in Lausanne, Switzerland. He is a researcher at Microsoft Research in Redmond, Washington. His research interests include distributed server systems and database systems. His work on database replication received the best paper award at Eurosys 2007.

**Yuxiong He** received the PhD degree in computer science from the Singapore-MIT Alliance, in 2008. She is a researcher at Microsoft Research in Redmond, Washington. Her research interests include resource management, algorithms, modeling, and performance evaluation of parallel and distributed systems. Her research work has been selected among best papers in ICDE 2014.



**Suman Nath** received the MS degree and PhD degree from Carnegie Mellon University (CMU). He is a senior researcher at Microsoft Research in Redmond, Washington. His research interests include sensor/time-series data management, data privacy and security, and flash memory. His research work has been recognized by best paper awards at BaseNets Workshop 2004, NSDI 2006, ICDE 2008, SSTD 2011, Grace Hopper 2012, and MobiSys 2012.

**Walid G. Aref** is a professor of computer science at Purdue. His research interests are in extending the functionality of database systems in support of emerging applications, e.g., spatial, spatio-temporal, graph, biological, and sensor databases. He is also interested in query processing, indexing, data streaming, and geographic information systems (GIS). Walid's research has been supported by the National Science Foundation, the National Institute of Health, Purdue Research Foundation, Qatar National Research Foundation, CERIAS, Panasonic, and Microsoft Corp. In 2001, he received the CAREER Award from the National Science Foundation and in 2004, he received a Purdue University Faculty Scholar award. Walid is a member of Purdue's CERIAS. He is the Editor-in-Chief of the ACM Transactions of Spatial Algorithms and Systems (ACM TSAS), and an editorial board member of the Journal of Spatial Information Science (JOSIS), and has served as an editor of the VLDB Journal and the ACM Transactions of Database Systems (ACM TODS). Walid has won several best paper awards including the 2016 VLDB ten-year best paper award. He is a Fellow of the IEEE, and a member of the ACM. Between 2011 and 2014, Walid has served as the chair of the ACM Special Interest Group on Spatial Information (SIGSPATIAL).

## Affiliations

**Abdulaziz Almaslukh[1] · Amr Magdy[1] · Ahmed M. Aly[2] · Mohamed F. Mokbel[3] · Sameh Elnikety[4] · Yuxiong He[4] · Suman Nath[4] · Walid G. Aref[5]**

Amr Magdy
amr@cs.ucr.edu

Ahmed M. Aly
aaly@google.com

Mohamed F. Mokbel
mokbel@umn.edu

Sameh Elnikety
samehe@microsoft.com

Yuxiong He
yuxhe@microsoft.com

Suman Nath
sumann@microsoft.com

Walid G. Aref
aref@cs.purdue.edu

1 Department of Computer Science and Engineering, University of California, Riverside, CA, USA

2 Google Inc., Menlo Park, CA, USA

3 Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN, USA

4 Microsoft Research, Redmond, WA, USA

5 Department of Computer Science, Purdue University, West Lafayette, IN, USA