Top-K Spatial Term Queries on Streaming Data

by

Sara Farazi

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

© Sara Farazi, 2018

Abstract

With rapidly increasing user-generated geo-tagged content in social media, location-based queries has received more attention lately. There has been extensive work on finding top-K frequent terms in specific locations from social network data streams. However, the problem *reverse spatial term queries*, where given a term, one wants to find the top locations where the term is frequent, has not been much studied. We study the problem of efficiently answering two types of queries on streaming data: 1) *Top-k reverse frequent spatial queries*, where given a term, the goal is to find top K locations where the term is frequent, and 2) *Term frequency spatial queries*, which is finding the expected frequency of a term in a given location. We explore a probabilistic model of term distribution that allows us to estimate the frequency of locations that are not kept in a stream sketch or summary. We study the back-and-forth relationship between the efficiency of queries, the efficiency of updates and the accuracy of the results and identify some sweet spots where both efficient and effective algorithms can be developed. We demonstrate that our method can also be extended to support multi-term queries. Finally, we conduct experiments on a relatively large collection of both geo-tagged tweets and geo-tagged Flickr photos to evaluate our algorithms. The evaluation reveals that our proposed method achieves a high accuracy when only a limited amount of memory is given. Also the query time can be improved, compared to a baseline, by 2-3 orders of magnitude without much loss in accuracy and that the update time can further be improved by at least an order of magnitude under some term distributions or update strategies.

Preface

This is a collaborative work between me and my supervisor, Dr. Davood Rafiei. I implemented the code, performed the experiments, analyzed the results and made conclusions. Dr. Rafiei provided feedback on improving the approach, design of the experiments and presentation of the thesis. This work has been accepted as a poster for inclusion in the proceedings of the 35th IEEE International Conference on Data Engineering (ICDE 2019), titled *"Top-K Spatial Term Queries on Streaming Data"*.

To my parents

For supporting me in every step of my academic life.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Dr. Davood Rafiei whose continuous guidance and support along with constructive feedback and criticism made this work possible. I would also like to thank the members of the database research group at the computing science department of University of Alberta for their support. More importantly, I would like to thank my family for being a source of inspiration and motivating me to pursue the path that was right for me. Without their support and encouragement, I would not have been where I am today.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

With the increasing mobile use of the web from geo-positioned devices, a wide range of location-aware queries and applications have emerged. Web sites such as Twitter, Facebook, Instagram and Flickr harness data from such devices. People express their feelings, ideas, experiences or observations in these websites in the format of posts, tweets or photo captions. Hence, a large volume of data generated on day-to-day basis is in some form of streaming events, and those events are often associated or are attached to a location. For example, an event reported by a smart phone user on Twitter may be tagged with the location of the capturing device; queries received by a search engine may be tagged with the location of the user issuing those queries; and items added to the shopping cart of a customer may also be tagged by the location of the IP address of the user.

An important problem in such a streaming setting is detecting the attachment of events to locations. The large amounts of data available coupled with the increasing number of location-aware queries calls for efficient indexing and query processing techniques. One particular query that has been largely studied in this context is detecting for a given location (and time), the terms or events that are frequent [55] or trending [33], [39], [41]. Given a query in the form of a time interval and a location, these methods often retrieve the top-K frequent terms used in that location during the specified time interval.

In this thesis, we study the attachment between events and locations but in a reverse direction; more specifically, given a term denoting an event, we want to find locations where the term is frequent. Use case examples for this query include

detecting the location(s) of a "snow storm" from incoming tweets, identifying areas where a sport event such as "curling" has the largest number of viewers, and finding regions that purchase a larger-than expected quantity of a product.

A related but different query in the same context is given a term and a location, find the expected frequency of the term in that location. Examples of this type of query are, estimating the purchase rate of a product in a specific region, or estimating the number of mentions of a presidential candidate name in one of the U.S. states.

Problem Statement Let S be a stream of posts or events, each modeled as a set of terms. Suppose each post is associated to a geographical location, in terms of a latitude and a longitude. S can also be projected on terms, giving for each term a stream of locations where the term is observed. Unless explicitly stated otherwise, we use the terms *post*, *event* and *term* interchangeably in this thesis.

We are interested in two types of queries over such streams:

- Reverse Frequent Spatial (RFS) Query: Given a term t and a positive integer K, find top K locations on the geographical map where t is frequent. This query, referred to as RFS query, returns a list of locations. The frequency of t in each location may also be returned, in which case a confidence factor is assigned to the estimated top location. The confidence factor is a value between 0 and 1 and shows the level of certainty associated to the top-K location.

  For example querying $Q = \{\text{Christmas}, 3\}$ in a stream of tweets posted from within the United States will result in the top 3 locations where the term *Christmas* is frequent. The result may look like, {New York(529, 1), Los Angeles(487, 1), Texas(342, 0.88)}.

  The first number for each location is the term frequency in that location and the second number is the confidence factor. Since the confidence value for the top-2 locations is 1, the top-2 results are exact, but the certainty for the top-3rd location is 88%.

- Term Frequency Spatial (TFS) Query: Given a term t and a location l, find

the frequency of t in l. This query, referred to as TFS query, returns either the observed or the expected frequency of the term in the location.

Challenges There are a number of challenges in processing RFS and TFS queries. First, storing a stream of <term, location> pairs with T terms and L locations requires $O(TL)$ space and keeping track of the top-K frequent locations for each term at all times requires $O(logL)$ update time, for example, using a heap. This is costly for long streams and fast arrival rates. Also storing the data in hard disk will increase processing and query time, due to the high access time. We need to process the stream in a way that consumes minimum memory and also has a low query processing time. We also want a method that is scalable and works well with the increasing amount of data that we receive. Streams of social media text data can have thousands of posts per second. A useful method should be able to anticipate the rapid growth in geo-tagged data.

Streaming frequent item counting algorithms can reduce the space overhead and per-insert processing cost while introducing some bounded error. However, those algorithms are very limited when applied to TFS queries; for example, one cannot efficiently estimate the frequency of a term at a specific location for which the data is not collected.

Second, locations are not abstract terms with no relationships; one location can contain or be contained in another location. Also the boundary region of a location may or may not be known. Hence a simple counting of locations for each term will ignore the locality of locations and can have a large space overhead or time cost.

Related work has studied streaming algorithms for maintaining frequent terms for each location (e.g., [55]), and those algorithms may be employed for RFS queries, keeping for each term a set of top locations where the term is frequent. However, this solution will ignore the proximity relationships between locations. Consider the location stream of the term "election," for example, during a federal election in Canada. The stream may show most (if not all) locations in the country frequent; maintaining the frequency of every location in the country is not the best use of the limited space, while keeping the frequency of a subset of locations may not be sufficient for estimating the frequency in locations that are not kept.

Contributions This thesis proposes a method to efficiently answer RFS and TFS queries using a probabilistic spatial distribution model of terms. Our method stores a small footprint of the streaming locations for each term and uses a counter-based frequent item counting algorithm to count the top-k frequent locations for them. RFS queries can be answered by getting the top-K locations from the stored footprints, referred to as summaries. To answer TFS queries, we use a probabilistic spatial distribution model for each query term. The model estimates the probability of observing query terms in all locations, including those locations that are not saved in the summaries. Having the total frequency of query terms and the probability from the spatial model allows us find an upper bound for the expected frequency of terms in unsaved locations. We further show how the method can be extended to answer multi-term queries, based on the joint probability distribution of the query terms.

The contributions of this thesis can be summarized as follows:

1. We introduce reverse frequent spatial and term frequency spatial queries on streaming geo-tagged events, allowing detailed inquiries about the attachments of events to locations.

2. We propose a concise summary, based on a streaming frequent item counting algorithm, to index and update summaries for each term in order to answer RFS and TFS queries

3. We introduce an approach for efficiently calculating a probabilistic spatial distribution model of terms using the stored summaries. The model gives us the expected frequency of query terms in unstored locations.

4. We propose a method to answer multi-term queries using the probabilistic model.

5. We evaluate the proposed methods through extensive experiments on two large datasets from Flickr and Twitter.

Chapter 2

Background and Related Work

To set the stage for the topics we are presenting in this thesis, this chapter presents some of the background material referenced throughout the thesis followed by a review of the existing literature related to our work.

2.1 Background

The background material related to our work is divided into three parts: (1) frequent item counting algorithms, (2) geo-tagging of web content, and (3) spatial distribution models. These topics are reviewed here, providing the necessary background for our later discussions in the thesis.

2.1.1 Frequent Item Counting Algorithms

The frequent item counting problem is to process a stream of items and detect all items appearing more frequently than a certain threshold. Significant number of algorithms for frequent item counting have been developed to find top frequent items in a streaming setting [13], [32], [43], [55]. A challenge these algorithms try to address is achieving this goal using only a limited memory. Here we discuss some of the background necessary to understand these algorithms.

The exact ϕ-frequent items in a set S are $\{i \in S | f_i > \phi |S|\}$, where f_i is the frequency of an item i in S. Solving the ϕ-frequent problem requires linear space ($\Omega(|S|)$), which makes processing the frequent items in a stream setting inefficient, so an approximate version is often considered. The ε-approximate frequent items

in a set S are $\{i \in S | f_i > (\phi - \varepsilon)|S|\}$. We will use "frequent items" or "the frequent items problem" to refer to the ε-approximate frequent items problem.

There are two main approaches to approximate frequent item counting in streams: a) counter-based, and b) sketch-based. In sketch-based approaches, we estimate the frequency of all items in the stream through a hashing mechanism. Items are mapped to a small set of counters, and each counter is incremented with each occurrence of an item that is hashed to that counter. CountSketch [10], GroupTest [16], and Count-Min Sketch [15] are major sketch-based algorithms.

In counter-based approaches, items in a stream are counted using a fixed number of counters. When all the counters are used, depending on the algorithm, each new item is either dropped or is swapped with a currently monitored item in the set of counters. The frequency error for monitored items may be kept and be updated when one item is replaced with another. Frequent [18], [35], LossyCounting [44], and SpaceSaving [47] are major counter-based frequent item counting algorithms.

Sketch-based techniques require higher processing cost for each item in the stream [14], therefore we adopt a counter-based technique. As an example we discuss about the SpaceSaving algorithm.

The SpaceSaving algorithm starts with a fixed number of empty counters. Each counter keeps record of an item, its frequency, and the error in frequency. The frequency and error values are all zero at the beginning. Each incoming item on the stream is recorded in counters until there is no empty counter left. Then if a new incoming item does not exist in the current set of counters, an item with the lowest frequency is selected to be replaced with the new item. The value of the counter is incremented and the previous value of the counter is set as the current error value.

The idea behind this algorithm is that, frequent items are expected to reside in counters with large frequencies and they are less likely to be replaced by infrequent items that grow more slowly.

The following simple example shows how this algorithm works. Suppose we have a stream of items as follows: *<A, A, B, C, B, D>* and there are only 3 counters available. SpaceSaving algorithm counts the items until all three available counters are used. Before receiving item D, the counters are as follows: *(A: 2, err = 0), (B:*

2, err=0), (C: 1, err = 0). After D is received, there are no available counters left, so we find the counter with the lowest frequency and replace it with the new item. Then we update its error value. The resulting counters are as follows: *(A: 2, err = 0), (B: 2, err=0), (D: 2, err = 1).*

SpaceSaving provides a maximum error value as well as guarantees about the order of top-K items. Also it is experimentally shown that SpaceSaving outperforms similar approaches in terms of precision, recall and memory usage [14], [42].

2.1.2 Geo-tagging of Web content

There has been extensive studies to effectively geo-tag different web content such as, blogs [24], [38], web pages [3], [63], search engine query logs [4], [62], photos [17], [25] and even web users [31]. Geo-tagging posts or texts from social network websites such as Twitter and Facebook is useful for many applications. Some devices (e.g. smart phones) geo-tag the content with the location of the device. For example if Twitter users post tweets from their cell phones, their content may be tagged with the location of their cell phones.

One way of geo-locating user generated content is through the IP address of the user machine. Some applications may have access to the IP address and can locate the user. For example, Buyukkokten et al. [8] use geographic entities and network IP address information to geo-locate web pages.

Another approach that has been studied recently is, geo-tagging based on the content. Named entities used in web pages like news articles, have a geographic centre or focus where the entity is better known or associated with; an organization may be tagged with a location where it is headquartered in; an artist may be associated with his/her hometown; and a sport team may be identified by the location where it is based in.

Some studies extract names, addresses, postal codes, and other information from web content and match them to information listed in a geographical gazetteer to identify the geographical scope of web pages and blogs [3], [24], [65]. Gazetteers are open geo-resources, usually created by merging geographical databases that are gathered from governments, travel advisories and blogs. They contain information

about the geographical features and social statistics of a country, region, or continent [26].

Some study the problem of geo-tagging named entities in documents based on their mentions in relevant web pages, such as the work of Yu et al. [63]. These studied try to resolve the focus of a named entity at different location granularities (city, country, etc.). Studies of toponym resolution, the problem of assigning a location mention in a document to a geographic point with a latitude and a longitude on the map, is also related to this line of work. A location mention in a web page or a news article, can be assigned to different places on the map. For example the mention of the term "London" can be assigned either to a city in Canada or a city in England. Studies in this line of work try to find the most accurate referent for a location mention [34].

There are also studies of mapping the images in web pages to a location on the map. Recent work in this area use machine learning techniques for visual content matching. They map images to the location of similar images [25].

Generative probabilistic models and language models have been used for estimating the locations of users in social network websites. For example the work of Cheng et al. uses a probabilistic framework to find terms with a high local focus from a large dataset of tweets. Users that use a local term in their tweets are expected to be located in the central location of the local term. [11].

To show a simple example of geo-tagging, consider the following tweet from a user in Twitter: *"Having fun watching a game for the first time in Rogers Place! Go Oilers!"*

This tweet contains the name of a place *"Rogers Place"* that probably can be found in an external geographical source and from that we can find out that the user is probably in Edmonton. Another approach is to use the term *"Oilers"*. Knowing that *Oilers* is the Edmonton hockey team can help us find out that this user is from Edmonton. Even if we don't have this information, finding a generative model for the term *"Oilers"*, using the whole dataset of tweets (as in the work of Cheng et al. [11]) can show that the central location for this term is Edmonton and probably the user is also located in the same location.

2.1.3 Spatial Distribution Models

Finding models for the geographical distributions of different events and phenomena have been studied extensively in different sciences. These models have been used to describe the spatial distribution of species, people, stars and epidemic diseases. Our problem is similar to these studies in the sense that we also need to assign a geographical scope and a spatial distribution to each query term, in order to estimate the probability of the term appearing in locations for which we have not stored the term frequency. Distribution models have different shapes for different categories or events. One general spatial distribution model is an event with a center and spread around that center. There are usually some natural or artificial constraints, that may alter the shape of the spread.

In physics, the spread of light, and other electromagnetic waves are described with inverse square decrease. As we get further from the source, the intensity of the waves decreases [58].

In earth sciences, studies of earthquakes present a model for the energy of waves that starts from a center and decays logarithmically as the waves spread further away from the center. The density of aftershocks of an earthquake usually decreases linearly as the radiuses around its center increase [22]. In studies of hurricanes, probability maps are built that estimate the occurrence and the intensity of hurricanes in different geographical regions for cities and countries. The probability is often high in central areas and decreases proportional to the distance from the center, for the surrounding areas [49]. Modeling crustal structure of earth in volcanic areas uses the same concept [52].

Geographical distribution of species is modeled with exponential decay functions (such as Poisson, Gamma, etc), which show the decay in population as the distance from their home habitat increases [28]. The home habitat, however, may not always be in the center of the area. Centers can be in the coastal areas and as we get farther from the coasts, the population decreases. Climate, demographic and geographic features affect the distribution of different species [56].

In human and social studies, the amount of human mobility and movement patterns [37], population density of urban areas [19], economic growth [51], price of

land [59], etc. are all described with the same model, in which the population has a high density in downtown and commercial centers of cities and urban areas, and as we move away from those areas, the density decays exponentially. Different distribution functions are developed for traffic in urban areas. For example, double Gaussian and exponential spatial distributions are considered [2], where the density of traffic decreases as we get further away from the areas with high central busy streets.

Rivers, seas, mountains, human constructions and other natural or artificial constraints can affect the spread of these geographical models. In an urban area, a river in the city can change the spread of the population, making each side of the river a different center with its own spread of population. A mountain or a sea between two cities can change the spread of population of species in that area.

In social media, news about an important event are centrally focused in the location of the event. For example, news and talks about a provincial election in Canada are centrally focused in that province. Popularity of different baseball teams in the US is centered in their home states and decreases in other neighbouring cities or states [50].

This spatial distribution modeling has been recently used in the context of terms and events extracted from the web. We can assign spatial models to terms used in news articles, social network websites, or query logs, meaning that their usage is the highest in a central location and decreases in locations that are further away from the center.

Earlier investigations of this issue used machine learning techniques to classify search queries as either local or global [20], [27].

Language models [36], [54] and generative models [30], [63] have been used recently to assign a geo-center and a spread for terms used in web content.

Backstorm et. al. [4] propose a probabilistic framework for quantifying spatial variations for terms in Yahoo search engine query logs. Their model provides an estimate of a geographic center, as well as a measure of spatial dispersion for the terms, indicating whether it has highly local interest or broader regional or national appeal.

In this model, each term is assigned a center, a focus (C) and a spread (α). Using these attributes one can find the following probability model for the terms: $p = Cd^{-\alpha}$ which means that the probability of the term occurring in a place with a distance d from its center is approximately $Cd^{-\alpha}$. According to this model, a high value of α means that a term is more local and its usage drops rapidly as we move away from its center.

2.2 Related Work

We have divided the existing literature related to our work into three main categories: (1) Top-k spatial term queries (2) Sliding window top-k queries and (3) Spatio-temporal frequent term/trend queries.

Top-k Spatial Term Queries Given a location and a set of keywords, a top-k spatial term query returns the top-k spatio-textual objects ranked according to their proximity to the query location and relevance to the query keywords. This line of work is similar to ours, because it also runs queries of a term and a location in the spatio-temporal data to get the top locations where the terms are frequent. However, there is an additional condition, which is, the proximity of the returned locations to the queried location. Here we briefly review some of the existing work in this area.

Some earlier works [12], [53], [64] use index structures such as inverted file and R-tree, to improve the performance of top-k spatial keyword queries. Their frameworks encompass algorithms that utilize the proposed indexes for computing the top-k query, and are capable of taking into account both text relevancy and location proximity to prune the search space at query time.

Wang et al. [57] create a new index structure called APTree (adaptive spatio-textual partition tree) to solve the same problem. APTree groups queries using keyword and spatial partitions, guided by a cost model.

Cao et al. [9] aim to help explore a region containing multiple PoIs (points of interest), located in a spatial network, each of which is relevant to given query keywords. The query takes a set of keywords like, “restaurant” or “coffee”, a length constraint that indicates how large is the region that user is willing to explore, and a

general region of interest, e.g., Manhattan. It returns a region in the general region of interest that does not exceed length constraint and that contains most relevant objects. They show that answering this query is NP-hard. So they develop a greedy algorithm with an approximation bound.

Our work is different from the studies mentioned above, in terms of the query definition. Although they seem similar, their definitions and applications are different. We work on top-K spatial frequent term queries, meaning that given a term, we want the top K frequent locations where the term has appeared in the stream.

Sliding Window Top-k Queries There have been studies on monitoring top-k queries in streaming sliding windows. They, too, query for the top frequent terms or trends in spatio-temporal streaming data similar to our work, but only in the recent time interval. For example, BlogScope [5] aggregates documents from news feeds, mailing lists, blogs, etc. and sorts them temporally. Each document is also associated with the location of its author. The system supports the tracking of currently popular keywords [46], and the monitoring of temporal and/or spatial bursts [45].

Yang et al. [60] work on monitoring top-k queries in streaming windows. A continuous top-k query $Q(S, win, slide, F, k)$ returns the top-k objects having the highest scores from function F, within each query window *win* on the data stream S. Windows can be either time or count based. Budak et al. [6] use tweets from users and their social network graphs as the input stream and find keywords that are frequently used in tweets from clustered groups of users (coordinated trends) or keywords that are highly used among distributed users (uncoordinated trends).

In another work Budak et al. introduce Geo-Scope [7]. Geo-Scope proposes an approximate solution for the online detection of geo-correlated trends. It identifies trending (location, topic) pairs along a sliding window in a data stream using two inverted indexes and requires sub-linear memory and running time.

Spatio-Temporal Frequent Term/Trend Queries One of the queries that has been extensively studied recently is spatio-temporal frequent term or trend queries. First, we describe the difference between frequent terms and trends. Frequent terms are terms that appear most frequently in a specific time and space, in a spatio-temporal dataset. Trends are terms that appear most frequently in a specific time but have

less frequency in times before and after that. In order to find top trends, one needs to analyze the increase rate of the term frequency over time, whereas to find top frequent terms only considering the frequency is enough. For example a term like *"election"* may be trending during the time of the US election, but a term like *"love"* can be frequent in most locations independent of the time. We consider *"election"* a trend, but *"love"* is only a frequent term.

Various recent works have focused on efficient approaches to answering top-k frequent spatio-temporal trends, meaning that given a geographical region and/or a time interval, one wants to find the top-k trends associated to that geographical region and time. These types of queries are closely related to our work, because they search for the top frequent terms or trends in a given location, while we search for the top frequent locations for given terms. Here we review some existing work in this area. The difference between these works is mainly in the index structures they use, the memory optimization techniques and the scoring system they use to find trends.

AFIA [55] uses a multi layer grid of cells representing different geographical locations at different times and at different levels of granularity. In this work, an extended version of SpaceSaving is used to store a summary of the data stream in the cells. In addition, AFIA introduces a merging algorithm used to aggregate results at different granularities. However, it answers top-k spatio-temporal frequent terms, not trends.

STREAMCUBE [23] uses the same index structure as AFIA, It also uses hashtag clustering to create events. A group of hashtags together create an event. Events in each cell are ranked based on ranking function of popularity, localness and burstiness.

Mercury [40] searches individual micro-blogs and supports a real-time query response within a tight memory-constrained environment. Mercury confines its search space to certain spatial and temporal boundaries, and uses a dynamic in-memory index and an index size tuning module that dynamically finds and adjusts the minimum needed index size.

Geo-Trend [39] finds recent spatio-temporal trends in social media by storing

the terms in a pyramid index in memory, but discards terms that have been present in memory for a certain time interval in order to optimize the memory usage. GeoTrend uses two measures to find trends: the keyword rate of count increase over time and the weighted count over recent time period. Venus [41] is similar to GeoTrend, but the ranking function differs. The ranking function is a combination of a spatial score and a temporal score for each top-k term. The memory optimization techniques also differ.

GARNET [33] is a system for answering top-k trend queries in a special context. The context can be either the location of the trend, or the gender or the age of people who are creating the trend. It uses a multi layer grid cell for spatial indexes and a pyramid tree for temporal indexes, initially stored in main memory and periodically flushed to disk. In this system, different weights for different partitions of time are defined in order to find the trending terms, not only frequent terms. Garnet also partially supports reverse frequent/trend queries. To answer these queries, it scans through all possible grid cells and outputs, if exists, the context that is most popular among all locations for the requested keyword.

Ahmed et al. [1] propose a new approach in which they store all terms on disk. They operate on disk resident data and provide exact answers for top-k spatio-temporal range queries. Their index structure uses an R-tree [29] augmented by top-k sorted term lists, where they balance the size of the index to achieve faster execution and smaller space requirements.

The aforementioned systems all focus on queries where given a location, one wants to find top-k frequent or trending terms in that location. To the best of our knowledge, there has not been any work on efficient and scalable support for reverse queries where given a term, one wants to find locations where the term is frequent. Also most existing works provide no support or are not applicable to multi-term queries. We propose a method to approximately answer both top-k single-term and multi-term queries in a geo-tagged text stream.

Chapter 3

Methodology

To associate each event to a location, we divide the world map into a uniform grid of cells of pre-determined size. The size of a cell, which can be given as a parameter, indicates the resolution at which location information is kept and is related to the amount of available space or desired accuracy. Larger cells will result in less memory usage but smaller cells result in more accurate estimation of the appearance for the terms. Each cell in our case is a rectangle on the geographical map, represented by the coordinate of its center point. The length of the rectangle in our experiments is one degree latitude and its width is one degree longitude. This gives us 180 * 360 cells which we use to capture and save the term frequencies.

3.1 Exact Solution

A naive approach to process the stream is to record for each term the exact count of all grid cell locations where the term is observed, and update the cell counts as the data streams in. For a $L \times W$ grid over the world map and a stream of length N, the algorithm needs for each term $O(LW \times logN)$ space. Updates can be done in $O(1)$ time since each latitude and longitude of a location directly maps to a cell where it is located.

The algorithm is exact if the space cost can be ignored. TFS queries can be answered in $O(1)$ time by looking up the count of a given location l for a term t. For RFS queries, one needs to scan the set of all locations to find top locations, and this takes $O(LW)$ for top one and $O(LW \times logLW)$ for top k locations. Clearly

RFS queries suffer in the absence of a data structure that supports efficient access to sorted counts. It is possible to reduce the time cost to $O(1)$, for example, using a heap but to update the heap, one may have to go through all locations. This is not acceptable as the data is streaming in and a fast update time is expected.

This naive solution, although is exact, uses a high amount of memory space and has a high update time for each entry. We introduce a simple summary based method, which uses streaming algorithms to reduce the space usage and speeds up the update process.

3.2 Tracking Top Frequent Locations

To reduce the space cost of the exact solution, one approach is to keep for each term only a few top frequent locations. The problem of finding the top frequent items in a data stream is well-studied [47], [48]. In our case, we want to find not only the counts but also the locations where a term is frequent and one algorithm that is shown to perform better under this setting is what is known as SpaceSaving algorithm [14], [47].

In this approach, we use the SpaceSaving algorithm to count the top frequent locations for each term in the stream by using a small footprint that consists of a fixed number of counters in main memory. We name this footprint *summary*. We create a summary for each term in stream and update it as the stream goes on. To answer RFS queries, we use the saved summaries to find the top-K locations and the top-K frequencies of the term queries as well as the confidence factor. To answer TFS queries, we create a probabilistic spatial distribution model for the terms to approximate the probability of their appearance in every location. We will discuss this model in section 3.3.

3.2.1 Summary Structure

Our term summary (also referred to as $Tsum$) consists of a list of counters that count the top occurring locations for each term. The number of counters used in every summary is referred to as the summary size. A large summary size increases

memory usage although it makes the results more accurate. We use the SpaceSaving algorithm to update the summaries. Each counter in the summary has a cell Id (location), a frequency f representing the number of occurrences of the term in that cell, and a frequency error value Δ.

Suppose we have a location stream of size N for a term, meaning there are N occurrences of the term in the stream. Having a summary with size m, assume that we want to find all ε-approximate frequent locations in the stream. (The ε-approximate frequent locations in a stream S are $\{i \in S | f_i > (\phi - \varepsilon)N\}$, where $\phi \geq \varepsilon$ is a user-defined support threshold and ε is the acceptable error rate for estimated frequency.) According to SpaceSaving we have:

Theorem 1. *Given a stream of locations for a term, one needs a Tsum of size $m = \frac{1}{\varepsilon}$ to find all frequent locations with error ε. Any location with frequency $f_i > \varepsilon N$ is guaranteed to be monitored in the summary.*

Proof. Since $\phi \geq \varepsilon$, we can set $\phi = \varepsilon$. Using the smallest possible value for ϕ changes the problem to finding all the locations with frequency error less than εN. The worst space usage is when all terms have the same frequency, in which case the number of ε-frequent locations (i.e., locations that appear more than εN times in the stream) is $\frac{1}{\varepsilon}$. For a more detailed proof see Metwally et al. [47]. □

As an example, to find all locations that appear more than 10% in the stream with the error of 0.01, one will need at least 100 counters. Note that ε alone can define size of Tsum.

3.2.2 Stream Processing

In the beginning, there are no summaries stored in memory, given that no terms have been received. Upon receiving each term from the stream, a new summary with empty counters is created for the term. Starting with an empty term summary, each location in the stream is either added to the summary (if not there already) or triggers a counter update until the term summary becomes full. At this point, a new location cannot be added unless a location is kicked out. For evictions, we select the location with the least frequency. This location is replaced with the new location.

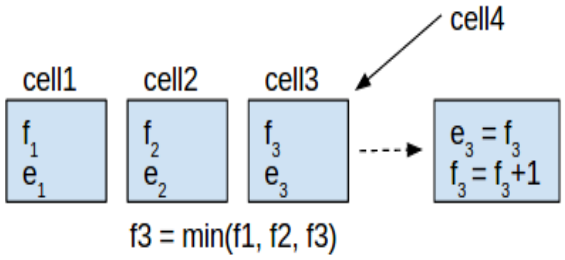

Figure 3.1: Updating the summaries with SpaceSaving algorithm

Its frequency is incremented and its value of error is set to the previous value of the frequency.

The following example shows how the process works for a stream of locations for a given term. For simplicity we use the name of cities instead of geographical coordinates in this example:

Example 1. Suppose we have only 3 counters in our summary and there is the following stream of locations for a term, say *"trump"*:

<columbus, dallas, dallas, new york, columbus, dallas, chicago>.

Before seeing "chicago", the counters will be: (dallas: f = 3, Δ = 0) (columbus: f = 2, Δ = 0) (new york: f = 1, Δ = 0).

Upon receiving "chicago" we know that there are no available counters. Therefore, we replace the location of the counter for "new york" with "chicago" and we update its error value. The new set of counters are: (dallas: f = 3, Δ = 0) (columbus: f = 2, Δ = 0) (chicago: f = 2, Δ = 1).

Figure 3.1 shows an illustration of the update process and Algorithm 1 gives the full detail. The time cost of a look-up in Step 1 is $O(1)$ if the locations are hashed; finding the minimum counter can take $O(m)$ but m is expected to be small which can be treated as a constant. Memory usage for each term depends on the summary size m which is an input of the system. The algorithm, stores only a list of m counters, and therefore has a memory usage of $O(m)$ per term.

Algorithm 1 Update(location l, Tsum S, size m)

```
1:  if l ∈ S then
2:      S[l].f ← S[l].f + 1
3:  else
4:      if S.size < m then
5:          S[l].f ← 1
6:          S[l].Δ = 0
7:      else
8:          i ← findMinCounter()                ▷ index of min counter
9:          S[l].Δ ← S[i].f                     ▷ max error
10:         S[l].f ← S[i].f + 1                 ▷ new count
11:         S.delete(i)
```

3.2.3 Confidence Factor

In order to find the confidence factor for the top-K locations, we use the value of frequency error (Δ). SpaceSaving guarantees an upper bound on the error.

Theorem 2. *Given a stream of locations of length N for a term and a Tsum of size m, the error* Δ *of each of the counters in the summary cannot exceed* $\frac{N}{m}$.

Proof. The worst case is when all locations are unique and each insert involves an eviction. In such a case, it takes m iterations before a new location is evicted, and the number of those evictions is bounded by $\frac{N}{m}$. For more details see Metwally et al. [47]. □

Given frequency f and error Δ for a location, a confidence factor is computed as

$$cf = \frac{f - \Delta}{f}. \tag{3.1}$$

So in a stream of locations with size 1000, and by using 100 counters, we can capture the 0.01-approximate frequent locations for the term with a maximum frequency error of $\frac{1000}{100} = 10$. For example if the saved frequency value for the term in one of its top frequent locations is 200, the maximum error can be 10 and hence the minimum confidence is $\frac{200-10}{200} = 0.95$.

In this model if the frequency error for a center is 0, it means that the frequency stored in that counter is exactly equal to the number of times the term has actually appeared in that location and the counter location has never been replaced with

another location from the beginning. So the saved frequency for that center is exact and $cf = 1$. As the center counters get replaced with new locations, Δ increases and hence cf will decrease, meaning that the saved frequency in the center is not exact and it has some error. Smaller values for cf show more error and less accuracy for the top-K locations.

3.2.4 Answering RFS Queries

It is straightforward to answer RFS queries for $K \leq m$ since top frequent locations for each term are kept in the term summary. The error in frequency can be bounded based on the value of Δ and as for Theorem 2. After returning the top-K frequent locations from the summary, the confidence factor is calculated for each top location and is returned as well as its estimated frequency. However, answering a TFS query (t,l) can be challenging if location l is not kept in the Tsum of term t.

Lemma 1. *Given a Tsum and a location l which is not in the Tsum, let $\check{f}$ be the smallest frequency in the Tsum. $\check{f}$ gives an upper bound on the frequency of l.*

The lemma provides some comfort in giving an upper bound estimate for TFS queries but the estimate can be too high especially if top locations in the summary are quite frequent. A question is if we can do better. To better answer TFS queries, we propose a method based on the probabilistic distribution of the terms.

3.3 A Probabilistic Model of Term Distribution

Given a term and a location, how can we find the expected frequency of the term in that location, if we have not stored that location in the summary? To estimate the frequency of locations that are not stored, we need a model of term distribution based on the information that is stored in the summary. In order to solve this challenge, we create a probabilistic spatial distribution model for each term, to approximate the probability of its occurrence in each location. Our approach is inspired by the method introduced in [4]. They propose a probabilistic framework for quantifying spatial variation for terms in Yahoo search engine query logs. Cheng et al. use the same method to estimate the user's location from the context of their tweets.[11]

Based on this model, each term t is characterized by its center (the location where it appears the most), a constant C which gives the frequency of t at its center and an exponent α which describes how quickly the frequency drops as we move away from the center. C and α are referred to as *term focus* and *term spread* respectively. Using this model, the probability that term t is observed at distance d from its center is

$$p = Cd^{-\alpha}. \tag{3.2}$$

A high value of α for a term means that the term is more local and its usage drops rapidly as we move away from its center whereas a low value of α signals a less local (or a more globally spread) term.

In the context of our work, which we store top-K frequent locations for every term, we chose the location with the highest term frequency (the first top location) as the center.

We can find focus (C) and spread (α) for each term at a given center *Cell* by deriving the optimal value for C and α that fits our data up to that point in the stream. For a term t, given a center, the focus of t at that center C, and the exponent α, we compute the maximum-likelihood value.

Let S be the bag of all occurrences of term t, and d_i be the distance of a location i from the term center. For every cell in our grid system, where t has appeared n times, we multiply the overall probability by $(Cd_i^{-\alpha})^n$. Also for every cell where the term has not appeared, we multiply the probability by $1 - Cd_i^{-\alpha}$. To avoid underflow, we calculate the logarithmic probability of the model. Then

$$f(C,\alpha) = \Sigma_{i \in S} \log Cd_i^{-\alpha} + \Sigma_{i \notin S} \log(1 - Cd_i^{-\alpha}) \tag{3.3}$$

is the log likelihood probability of observing the term in locations in S, as estimated by the model with parameters C and α. Backstrom et al. [4] prove that $f(C,\alpha)$ has exactly one local maximum over its parameter space meaning that we can find the optimum values of C and α by iterating over their possible values, maximizing the value of $f(C,\alpha)$.

To solve this problem we need to have all the locations in which the term has appeared as well as all locations where it hasn't appeared. Using the Tsum, we

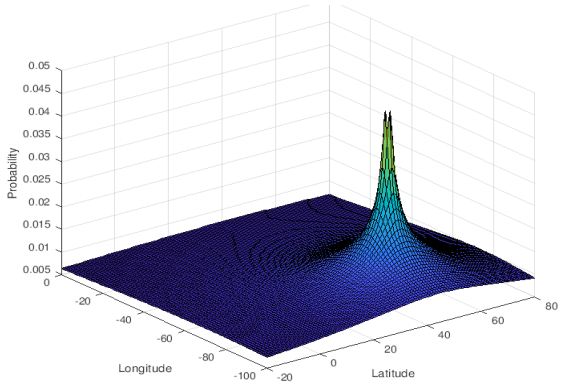

Figure 3.2: Optimized model for term *bitcoin* (Twitter dataset)

assume that each term has appeared only in the locations that we have saved in the summary counters and it hasn't appeared in the rest of the locations on the map.

Having all needed attributes, we can model the probability function for each term. Figure 3.2 shows an example of the optimized model for the term *"bitcoin"* on a dataset of tweets from December 1st 2017 to January 1st 2018. The central location is in the state of New York.

There are some challenges for using the model in a streaming setting. First, solving the optimization problem (as stated above) requires the knowledge of all locations in which the term has appeared as well as all locations where it has not appeared. A question is if a Tsum provides sufficient data to estimate the model parameters and if the model is accurate.

Second, as the data streams in, the parameters of the model can change. For example, the center can move and the degree of dispersion around the center can vary. For more accurate estimates, the model needs to be updated before answering a TFS query, but running the maximum likelihood for each query can be computationally intensive. A question is if this can be done efficiently in a streaming setting. To put the questions together, we want to find ways of balancing the accuracy of the estimates with the efficiency of the results. We address this in section 3.4.

3.3.1 Answering TFS Queries

Upon receiving a TFS query, we calculate the probabilistic model for the query term as explained above. We bound the search for optimal C and α for values

larger than 0. The probabilistic spatial model built for each query, can be treated as a probability density function, only if it satisfies the following two conditions:

1) for all x, $p(x) \geq 0$

2) $\int_{-\infty}^{\infty} p(x)dx = 1$

Here x is the distance between the given location and the center of a given term (which we named d in the previous section). The spatial model is in the format of $p(x) = Cx^{-\alpha}$. Since $C > 0$, and $x > 0$ the first condition holds. To satisfy the second condition, we find the sum of all the values of the probability function over its domain which is (0, maximum distance between two points on earth]. We have $x \neq 0$, since the center cell is not considered in the calculation of focus and spread. Since the sum may be larger than one, we divide the probability by the sum of all the values. This will ensure that the second condition is always true and the model can be treated as a probability density function. So for a given location l and a term t, the probability of occurrence for term t in location l, which we call $P(l)$, will be:

$$\begin{aligned} d &= getDistance(l, center_t), \\ P(l) &= p(d) \times \frac{1}{|\int_1^{\infty} C_t x^{-\alpha_t} dx|}. \end{aligned} \tag{3.4}$$

Note that the integral lower bound is 1, because in our system the smallest distance between two cells can be equal to a cell size, which is 1.

Having the probabilistic model for each term, we can find the probability of its appearance in each cell on the map. The product of this probability on a cell and the total frequency of the term gives the estimated frequency of the term in the given location. Suppose f_T is the total frequency of term t in all observed locations and $P(l)$ is the probability of term t being emitted from location l. Then the estimated frequency for t in l is:

$$f_l \simeq f_T \times P(l) \tag{3.5}$$

As a TFS query is received, we calculate the distribution model for the query term and approximate f_l to answer the query.

Note that f_T can be easily calculated by adding up the frequency value of all counters in the summary.

Theorem 3. *Given a Tsum for a term t in a stream, the sum of all frequency values in the Tsum will give the total frequency of t.*

Proof. For each occurrence of a term t in the stream, the SpaceSaving algorithm updates a counter in the term Tsum. This update is either in the form of adding one to the frequency of a location (if the new location already exists in the Tsum) or replacing the counter location with the new location before adding one to the frequency value and updating the error. In both cases, every occurrence of the term is counted, although the location of the occurrences may have some error. Thus the sum of frequencies gives the total frequency of the term. □

The accuracy of estimates based on Tsum is expected to be high when enough counters can be allocated (as shown in our experiments). However the running time of TFS queries may not be acceptable. This is because function $f(C, \alpha)$ in equation 3.3 includes a term for every location where the term appears in as well as a term for every location where the term does not appear in. The number of those distance calculations is equal to the number of cells on the map. Also, to find the parameters that optimize $f(C, \alpha)$ in Equation 3.3, one needs to computer the function multiple times.

We address this issue by introducing a new summary structure and a new approach to find focus and spread. In the new method, which we call *Ring Summary Method*, we maintain a set of potential centers as before and keep the previous summary structure, but we add another component to it. For every counter in the summary which we refer to as center counters, we store an additional set of counters, called *ring counters*. These counters store the occurrences of the term around the center they belong to.

3.4 Ring Summary Method

The idea behind the ring summary (also referred to as Ringsum) method is to speed up TFS queries by reducing the number of calculations for cells that are within the same distance of a center. A ring summary maintains for each term a fixed number of potential centers. Each center in the summary has a Cell Id denoting a location,

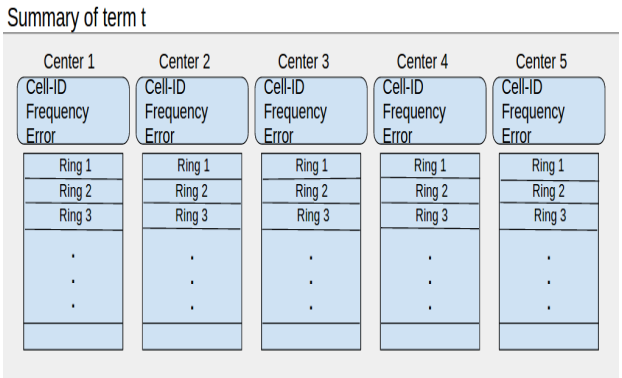

Figure 3.3: The structure of term summaries with ring counters for every center of the term.

a term frequency f in the center, an error value Δ, and a set of ring counters, each counting the occurrences of the term in a ring around the center. As in a Tsum, Cell Id, f and Δ keep track of the top-K frequent locations for each term.

We introduce the rings to accelerate the process of finding the spread of each term at each of its centers, as well as to increase the accuracy of spread while using smaller summary sizes. The assumption is that since the spread of a term is determined only based on the number of its occurrences at each distance (according to Equation 3.3), we can approximate the occurrence of the terms in every cell on the map except the center by a distance from the center and a frequency which shows the number of times the term has appeared in that distance from its center.

The stream processing in the presence of ring counters is slightly different though. In this new approach, we store an additional set of R counters for each of the center counters in our summary. Instead of storing all the occurrences of a term on the map, these new counters are used to store a summary of occurrences of the term around each of its centers. Figure 3.3 demonstrates the new summary structure.

The ring counters for a center are updated as follows: First, we select R distance intervals: $\{(0,d_1],(d_1,d_2],(d_2,d_3],\ldots(d_{R-1},d_R]\}$. We then count the number of occurrences of the term at these intervals from its center (see Section 3.4.1 for a discussion of how the distances are selected). These R counters represent hypothetical rings around each center of the term in the summary. Figure 3.4 demonstrates

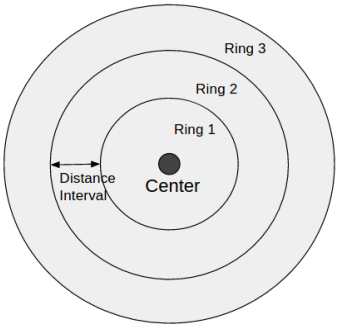

Figure 3.4: Rings around a central location. We save the number of times the term has appeared in each distance from its center in each ring counter.

the way rings are located around a center on the map.

We will have R counters all set to zero at the beginning. For every occurrence of the term, if the distance falls in the $(d_{i-1}, d_i]$ interval, the counter for $(d_{i-1}, d_i]$ is incremented. In other words, we save the number of times the term appears in each distance interval from its center. The ring counters provide an approximation of all the occurrences of the term across the geographical map. As the occurrence of a term falls further away from its center, its effect on the value of spread is lessened. Hence a good approximation of the spread may be obtained using a rather small R.

Example 2. Suppose we have a summary for term t, containing 10 center counters. Each center counter has 3 ring counters. What we have saved so far in ring counters of center c_i is: (ring 1: 100 , ring 2: 40, ring 3: 20). The ring distance intervals are all equal and 200 kilometers. This means that term t has appeared 100 times in locations with $(0, 200]$ kilometers distance from its center c_i. It has appeared 40 times in locations with $(200, 400]$ kilometers distance and also has appeared 20 times in locations with $(400, 600]$ kilometers distance from c_i.

There are some challenges when rings are used instead of cell counters. First, given R rings, a challenge here is, how do we choose distances $\{d_1, \ldots d_R\}$ so that we get the most accurate approximation of the real value of the spread? We will discuss the ways to optimize the distances between the rings in section 3.4.1.

Another challenge is how to update the rings around the changing centers? As long as the center counters in a summary are not being replaced by new locations, we can update the ring counters for each center as explained above. When a new

location is seen, it is counted by one of the center counters and is also added to other center counters' rings. The challenge is how to continue updating the rings for a center counter when it is being replaced by a new location during the top-K counting process?

Handling changes in potential centers: As the data streams in for a term, a top-K location (treated as a potential center) may be swapped with a new location. We need a mechanism to populate the list of ring counters for this new location.

One approach is to reset the rings, discarding all saved frequencies in each ring, and count only the future occurrences of the term around its new center. This approach, although straightforward, can result in a large error in the estimated spread α of a term. We discuss this alternative further in Section 5.

Another approach is to estimate the term frequency in each ring around a new center based on information that exists in the summary for other centers. This can be done by calculating the intersection area of the rings around the existing centers and the new center. Assuming that the distribution of the points in each ring around the center is uniform, we can say that a proportion of the frequency saved for an old set of rings can be transferred to the new set of rings. This proportion is equivalent to the ratio of the area of intersection between the new rings and the old rings to the whole area of the old rings. Having the intersection and the total area of the old rings, we can have an initial value for each ring counter for the new center. Example 3 shows a simple form of populating initial ring values for a center with 3 ring counters:

Example 3. Let L_0 be an old center and L_1 be a new center, and suppose the rings around these centers are as shown in Figure 3.5. The blue area is the intersection area between R3 from the new center and R3 from the previous center. The grey area is the intersection between R2 from the new center and R3 from the previous center and the green area is the intersection between R2 from the previous center and R3 from the new center. Let's call the grey area A, the blue area B, and the green area C. In this example the initial values for R1, R2 and R3 for the new center L_1 will be:

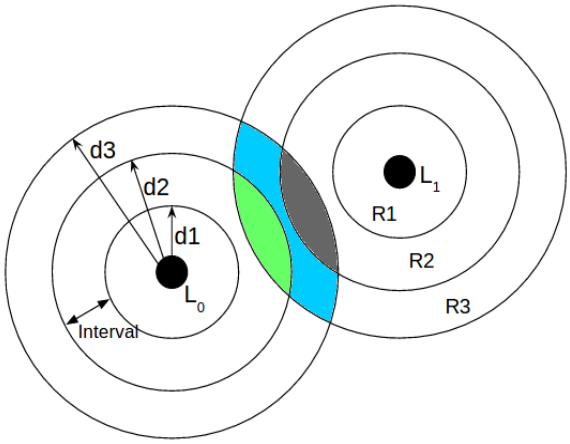

Figure 3.5: Finding the intersection between the new set of rings and the previous set of rings for a replaced center location to initiate the new ring values.

$$
\begin{aligned}
\phi_{R_{1,1}} &= 0 \times \phi_{R_{1,0}}, \\
\phi_{R_{2,1}} &= \frac{A}{area(R3)} \times \phi_{R_{3,0}}, \\
\phi_{R_{3,1}} &= \frac{B}{area(R3)} \times \phi_{R_{3,0}} + \frac{C}{area(R2)} \times \phi_{R_{2,0}}.
\end{aligned}
\tag{3.6}
$$

where $\phi_{R_{i,j}}$ is the value stored in the i-th ring counter of L_j. $area(R2)$ and $area(R3)$ are the area of respectively the second and the third rings around the old center. The first ring of the new center is initialized to 0 because there is no intersection between its area and any rings from the previous center.

Algorithm 2 gives the steps for updating a summary in the ring summary method.

The *addToRings()* function finds the distance between a newly seen location on the stream and the center locations in the summary. If the distance falls in any ring around a center, then the frequency count for that ring is incremented. The time cost of this function is $O(R)$ if the locations are sequentially scanned and $O(logR)$ using binary search.

The *initializeRings()* function finds the intersection between each previous ring and all new set of rings around a new center and calculates the portion of the value from the previous set of rings to be transferred as initial values to the new set of rings. The time cost of this function is $O(R^2)$. The time cost of the update algorithm in the ring summary method depends on the number of center counters, m, and the

Algorithm 2 Update(location l, summary S, size m)

```
1: if l ∈ S then
2:     S[l].f ← S[l].f + 1
3:     for i in {S − S[l]} do
4:         S[i].addToRings(l)
5: else
6:     if S.size < m then
7:         S[l].f ← 1
8:         S[l].Δ = 0
9:         for i in {S − S[l]} do
10:            S[i].addToRings(l)
11:    else
12:        j ← findMinCounter()
13:        S[l].Δ ← S[j].f
14:        S[l].f ← S[j].f + 1
15:        S.delete(j)
16:        S[l].initializeRings(l)
17:        for i in {S − S[l]} do
18:            S[i].addToRings(l)
```

number of rings per center counter, R. In the worst case the update time will be $O(mR+R^2)$ in which R is a very small number. Also for memory usage, the ring summary algorithm stores R ring counters per center and has a memory usage of $O(mR)$.

One caveat in estimating the focus and spread using Ringsum is that the summary keeps a count for all cells within distance d_{i-1} and d_i of a center whereas Equation 3.3 has the actual distance of each cell from the center. A solution is to use the mean distance, i.e. $d_{approx_i} = \frac{d_{i-1}+d_i}{2}$, for all cells at ring i. This reduces the distance calculation for ring i to a single term.

Another caveat in estimating the spread is that we also need the locations where the term does not appear. To reduce the overhead in processing those locations, we can compute an offline table which keeps the number of cells that fall inside each of the rings around a central cell on the world map. Using the Euclidean distance, the number of cells in each ring remains the same, independent of the center, and a pre-computed table can be used for all cells [1]. Given a center and its ring counters

[1] In order to be consistent with the probabilistic model, we map the sphere to a rectangular map in the Euclidean system and use the Euclidean distance between the points.

for a term, the number of cells at each distance interval from the center where the term does not appear can be obtained by subtracting the ring counter value for that interval from the number of cells in that distance interval (as recorded in the table described above).

Suppose our Ringsum has R rings for a center. Let d_{exp_i} denote the expected distance of cells in Ring i to the center and ϕ_{R_i} be the number of occurrences of the term in Ring i. If T_i denote the number of all cells in Ring i (as pre-computed and stored for easy look up), then $f(C,\alpha)$ can be written as:

$$\begin{aligned} f(C,\alpha) = \Sigma_{i=1}^{R} \log(\phi_{R_i} C d_{exp_i}^{-\alpha}) + \\ \Sigma_{i=1}^{R} \log((T_i - \phi_{R_i})(1 - C d_{exp_i}^{-\alpha})) \end{aligned} \tag{3.7}$$

All variables in Equation 3.7 are known and the equation has only one term for each ring. The number of terms to compute $f(C,\alpha)$ is now decreased to the number of rings, and this drastically reduces the time cost of TFS queries.

As we discussed in Section 3.3, the number of distance calculations using Tsum is equal to the number of grid cells on the map. Given a $L \times W$ grid system and assuming each center has R rings in the summary, the time cost of computing $f(C,\alpha)$ is $O(R)$ for Ringsum and $O(LW)$ for Tsum. It can be noted that the time cost for TFS queries only depends on the parameters R, L, W and the summary size m. Also, since $R << LW$, Ringsum is expected to be much faster.

3.4.1 Setting Ring Distance Intervals

In this section we discuss our approach to finding appropriate distance intervals for rings. A question that arises in the ring summary method is how the rings should be placed since both the number of rings and the distances between them can vary. The number of rings depends on the space constraint, and with a larger number of rings, the spread can be calculated more accurately because the distances saved for the terms in each ring will be closer to the real distance.

Selecting the sizes of ring intervals is not trivial. Ideally one needs to know the term distribution on the map to allocate the rings. However, the ring sizes are set before the terms are seen on the stream, and the term distribution is less likely to be known in advance. Let D be the radius of the largest possible ring around a center,

and $r \times D$ be the radius of the next possible ring around the same center where $0 < r < 1$. Assuming that the ratio r remains the same between two consecutive rings, the rings can be defined by fixing r and the number of rings.

One observation that can be made based on Equation 3.7 is that the occurrences in rings that are farther away from the center have little impact on the value of $f(C, \alpha)$, whereas the occurrences in rings that are closer to the center have a bigger impact. Therefore, an optimal ring setting is expected to keep more accurate distances for locations that are closer to the center. That means, smaller intervals should be assigned to the rings that are closer to the center and and larger intervals should be given to the rings that are farther away from the center.

Lemma 2. *Given a circular region with radius D and assuming that the distribution of points in the region is uniform, $\frac{2}{3} \times D$ is the average distance from the center to a point in the region.*

Proof. Consider a unit circle and a set of points that are uniformly distributed in the circle. The Probability Density Function (PDF) of the points at distance x is $2x.\mathbb{1}_{[0,1]}(x)$ and the expected value of the distance is given by

$$\int_0^1 2x^2 dx = \frac{2}{3}.$$

□

Suppose the distribution of locations for a term around its center is uniform. If we allocate only one ring for a center, $\frac{2}{3}D$ is the average distance between the points in the ring and the center. For more than one ring, we may divide the region at where the average falls and set the ratio $r = \frac{2}{3}$, giving the set of radiuses $\{\frac{2D}{3}, \frac{4D}{9}, \ldots \frac{(2^R)D}{3^R}\}$ for the rings. In our experiments, we evaluate this setting along with other settings of r in terms of their error in calculating the spread.

Chapter 4

Multi-Term Queries

Answering TFS and RFS queries for single-terms is described using the Tsum and the ring summary method. In this section we discuss how we can answer also multi-term queries.

A multi-term RFS query is denoted by $Q = (T, K)$, in which $T = \{t_1, t_2, ..., t_n\}$ and t_i is an individual term. We are interested in finding the top K locations where all the terms in T appear together frequently. Similarly, for multi-term TFS queries denoted by $Q = (T, l)$, we are interested in finding the estimated frequency in location l for all terms in T appearing together.

It is easy to evaluate RFS and TFS query for multi-terms in the exact approach. If f_1 and f_2 denote the frequencies of terms t_1 and t_2 respectively in a location, then $min(f_1, f_2)$ gives the frequency of both terms in the same location.

Answering these queries in Tsum and Ringsum is more challenging, because these methods do not keep the frequency of a term in all locations, and only the frequencies in top-K locations are kept. To estimate the joint frequency in a multi-term query, one may assume the term frequency in every location that is not kept in Tsum or ring summary is zero. This will clearly have error depending on the summary size. One approach is to intersect the summaries of the terms in the multi-term and if they have common cells, those can be the top-K frequent locations for the multi-term. This approach can answer TFS query if the queried location exists in the intersection of the summaries.

However, using the probabilistic model of terms can provide an alternative approach to answer RFS and TFS multi-term queries. Given a multi-term query, the

top-K centers for each term are stored in their Tsum or Ringsum. The focus (C) and spread (α) for each of these centers may be estimated, and this gives K probabilistic models $P = Cd^{-\alpha}$ per term. Each model will provide the probability of the term occurring in any cell on the map using $p = Cd^{-\alpha}$ where d is the distance from the center to the cell.

The join probability in every location on the map can be computed under some assumption. In particular, if we can assume that the appearance of the terms in a multi-term query is independent from each other, then the joint probability can be estimated as the product of the probabilities, i.e.

$$P(t_1, t_2, \ldots, t_n) = P(t_1) \times P(t_2) \times ... \times P(t_n). \tag{4.1}$$

If the occurrence of terms in a multi-term query is dependent on the rest of the terms, the lowest probability may give an upper bound estimate of the probability, i.e.

$$P(t_1, t_2, \ldots, t_n) \leq min(P(t_1), P(t_2), \ldots, P(t_n)). \tag{4.2}$$

With the probabilities of query terms estimated for all locations, an RFS query can return the top K locations with the highest probabilities. Algorithm 3 gives the steps of answering an RFS multi-term query with the assumption that the terms are independent.

Algorithm 3 QueryProcess(multi-term T, K)

```
1: for t ∈ T do
2:     for center_i in top-K centers do
3:         p_i ← C_i × d^{-α_i}
4:         for cell in Map do
5:             d_cell ← getDistance(center_i, cell)
6:             p(cell) ← C_i × d_cell^{-α_i}
7: for j = 1 to K do
8:     JointProbabilities_j ← ∏_{t,cell} p_t(cell)
9:     top-j result = Max(JointProbabilities_j)
```

Answering TFS queries is similar except that we find the joint probability of the terms only in the given location. The product of this probability with the minimum total frequency of the terms in the multi-term query will give an upper bound esti-

mate of the expected frequency of the multi-term. Algorithm 4 shows the process of answering TFS multi-term queries.

Algorithm 4 QueryProcess(multi-term T, location l)

1: **for** $t \in T$ **do**
2: $\quad d \leftarrow getDistance(l, center_t)$
3: $\quad p_t \leftarrow C_t \times d^{-\alpha_t}$
4: $TotalFrequency \leftarrow Min(TotalFrequency_t)$
5: $result \leftarrow \prod_t p_t \times TotalFrequency$

Chapter 5

Possible Improvements

The ring summary method, despite being very efficient for RFS queries in terms of the query time, has some drawbacks. First, the stream processing cost is higher in the worst case due to the overhead in replacing an existing center with a new one. Second, for a fixed space usage, the accuracy of focus and spread can be lower than that of Tsum especially if not enough center counters can be allocated because of the space allocated for rings. In this section, we develop three variations of the ring summary method to address these issues.

5.1 Fixed-Center Ring Summary

To avoid the overhead in stream processing when replacing a center, one strategy is to stop updating the centers after a certain volume of the stream passes. The intuition behind this is that for some terms the top central locations do not change after some amount of stream passes, and there is not much point in updating the centers that may not end up in top. Our experiments show that the average number of replacements in top-K counters of the summary decreases noticeably after about 20% of the stream is received (see section 6.8 for more details). Also the complex updating process of the rings in the ring summary method when their center counters are replaced with a new location may increase the error for spread. Fixing the center locations can result in lower error for calculated spread if we know that the centers are not going to change anymore.

The appropriate amount of stream must be processed for terms with different

distributions, in order to fix the centers afterwards, since we don't want to introduce error to the value of focus and spread. In chapter 6 we demonstrate that this approach results in significantly lower error for the calculated value of spread, but also lowers the center accuracy.

5.2 Light-Update Ring Summary

Another strategy to avoid the stream processing overhead due to the changes in center is to set the initial ring counters to zero when a new center is added. As explained in section 3.4, instead of using the intersection between the rings, we can just discard saved information about the occurrence of the term in each ring around its potential center and start filling the rings from scratch. The intuition here is that the existing rings cannot contribute much to the rings of a new center if the area of overlap between the rings is small. Also the initialization process is time consuming and ignoring it can speed up the update process. Under this consideration, the contribution of the existing rings may be ignored to save in processing time. This strategy can introduce more error in calculating the spread but it will decrease the summary update time drastically. We will demonstrate the reduction in update time in chapter 6.

5.3 Proximity-Aware Update

This variation of the ring summary method aims at avoiding unnecessary updates where a new center is not that far from an old center. The idea behind this strategy is that when a center moves back and forth between a set of adjacent grid cells, any of those cells can be a center. Also when two centers are nearby, the occurrence data maintained for these centers can be very similar. This will lead to the same or similar probabilistic models for the two centers, and any of such models can be as good as the other. Hence we may only keep track of centers that are within an acceptable distance from each other.

This strategy can be implemented by slightly changing our update algorithm. When all center counters in the summary are full, a new potential center is inserted

only if the new location does not fall within the first ring of other centers. Reducing the number of updates in centers can decrease the error for spread but the centers may not also be very accurate. This change in the update process changes the original SpaceSaving [47] algorithm and hence, there would be no guarantee about the summary size to catch all the ε-frequent top locations for the terms. We will discuss the result of this variation of ring summary method in chapter 6 too.

Chapter 6

Experimental Evaluations

In this section, we conduct experiments to evaluate our methods. First we describe our datasets and the queryset we use to test the methods. After that we discuss different experiments and their results.

6.1 Datasets and Queryset

To evaluate the efficiency and the effectiveness of our algorithms and their robustness to parameter setting, we conducted experiments on two datasets: (1) Flickr YFCC100m dataset [61] and (2) a collection of tweets collected from Twitter. The Flickr dataset included 100 million photos, of which roughly 60% had geo-tags. We used terms from the description field and user and machine tags of geo-tagged photos to create our first dataset. The second dataset included a set of 53 million geo-tagged tweets, posted from within the US and gathered from Twitter's stream API [21] from December 1st 2017 to January 1st 2018.

For each dataset, we used a random query set of 1000 terms that had a 0.02-frequent center (i.e. whose frequency at their center was more than 2% of their total frequency). The query terms were randomly chosen from the bag of all terms in the stream, meaning that more frequent terms were more likely to be picked. Unless stated otherwise, we use this query set for all the experiments in this work. All experiments are done on a machine running Ubuntu 16.04 with 8GB of memory and core i7 Intel CPU.

6.2 Ring Size Selection

Our first experiment is on the ring size selection and the robustness of our term distribution model to the ring size. It is expected that the more the number of the rings, the more accurate the count estimates and the less the error in estimating the spread. However, having a fixed number of rings, we wanted to know what the ring sizes should be. For this experiment, we varied the size ratio r from 0.1 to 0.9. This gave us a sequence of rings with the radius of each ring in the sequence reduced by a factor of r to give the radius of the next ring. Since the smallest ring included a 1°x1° latitude/longitude grid cell and the largest ring included all the cells, the number of rings varied for each r. For example, there were 3 rings for $r = 0.1$, 4 rings for $r = 0.2$ and 10 rings for $r = 0.6$; the number of rings were capped at 10 for larger values of r. For our ground truth, the parameters of the probabilistic model were estimated using all occurrences of the terms.

The relative error in spread α for our data sets are shown in Figure 6.2 and Figure 6.1. The figures show the mean and standard deviation of error for 1000 queries in each case. At $r = 0.6$, the Flickr dataset has its minimum mean error and the Twitter dataset is not that far from the minimum. This value of r also is not that far from the ratio $\frac{2}{3}$ discussed in section 3.4.1. Unless explicitly stated otherwise, we set $r = 0.6$ for the rest of our experiments.

6.3 Accuracy of RFS Queries

To find ε-approximate frequent locations for a term, one needs at least $\frac{1}{\varepsilon}$ center counters, as discussed in Section 3.2.1. Here we change ε from 0.02 to 0.002. The number of counters changes from 50 to 500. We wanted to find out how both the accuracy and the space usage changes with K. Figure 6.3 shows the accuracy of top-5 frequent locations for a query term in the Twitter dataset varying ε.

The top frequent location has an accuracy of nearly 100% since with $1/0.02 = 50$ center counters, our algorithms (both Tsum and Ringsum) find all locations that are at least 0.02-frequent (see Theorem 1). By using the same number of counters, top-K locations for $K > 1$ may not be exact since the location may not be 0.02-

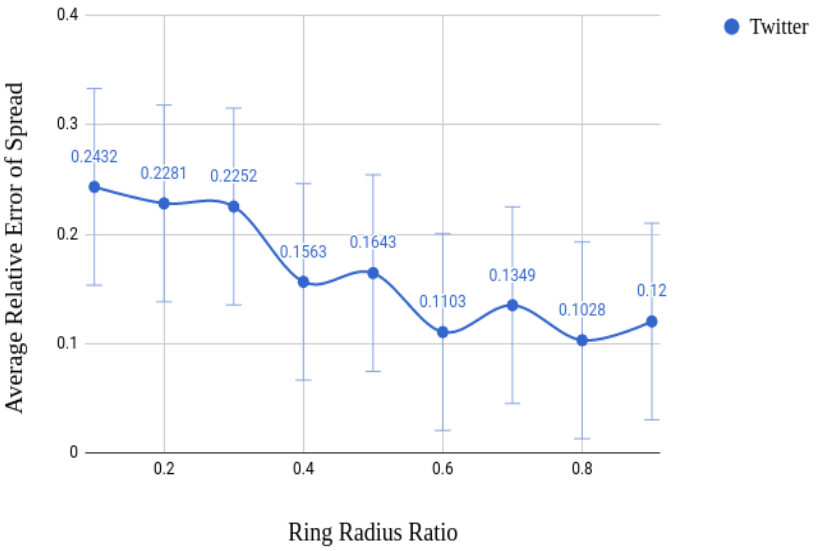

Figure 6.1: Error of spread by using different ring size ratios in ring summary method (Twitter dataset)

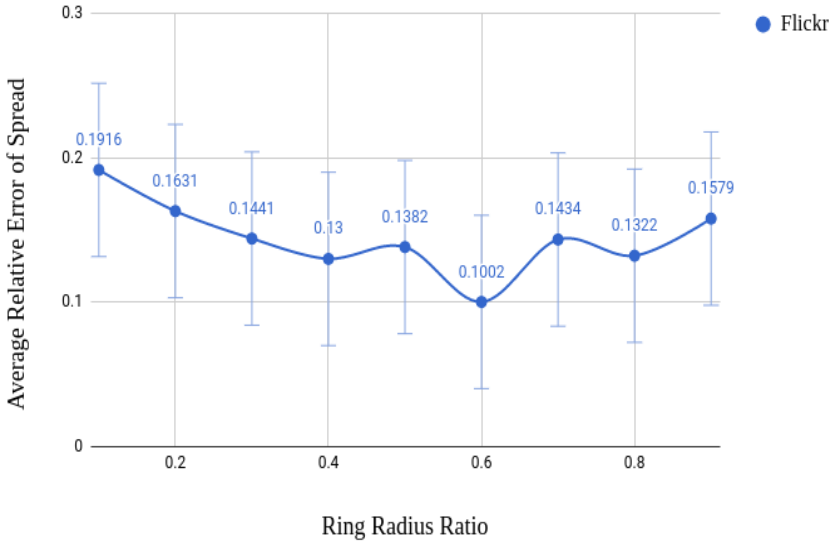

Figure 6.2: Error of spread by using different ring size ratios in ring summary method (Flickr dataset)

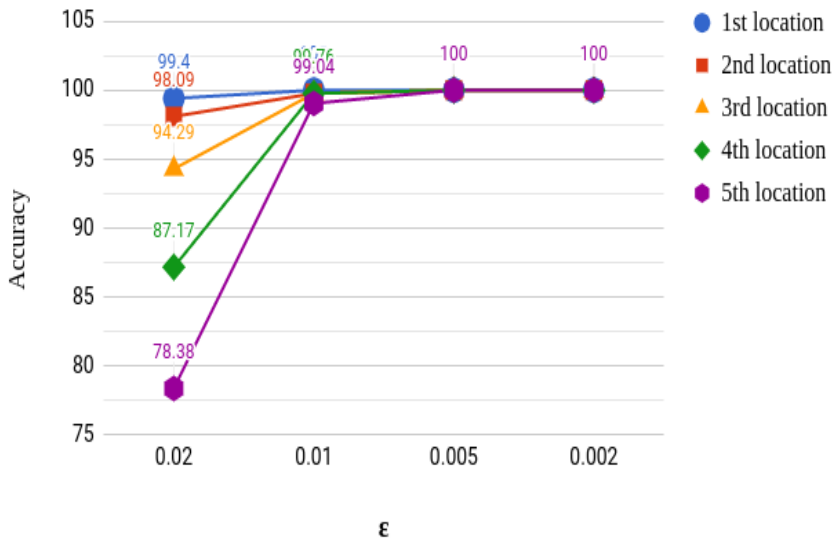

Figure 6.3: Average accuracy of top-5 frequent locations for query terms varying ε (Twitter dataset)

frequent. However, for smaller values of ε (which translates to a larger number of counters), the accuracy catches up for $K > 1$.

6.4 Probabilistic Model Evaluation

Both TFS and multi-term queries estimate the frequency of a term in an arbitrary location using our probabilistic model, hence their accuracies largely depend on the accuracy of the probabilistic model. In this section, we evaluate the probabilistic model under different parameter settings. We allocated the same space for both Tsum and Ringsum and compared their relative error of focus and spread and the accuracy of centers. For each center, a location id, a term frequency in the location and an error was kept whereas for each ring, a counter was sufficient. Both the error and the accuracy were measured compared to the case where the model parameters were estimated using all occurrences of the terms (as in the work of Backstrom et al. [4]).

A query set of 1000 random terms is given to both methods. We use 216 center counters in Tsum. In order to adjust the memory usage of both methods, we use 4 different counter settings for the ring summary method, which all use same

amount of memory as the 216 counters in the Tsum method. The counter settings are 81 center counters and 5 rings with with size ratio 0.3, 65 center counters and 7 rings with size ratio 0.4, 59 center counters and 8 rings with size ratio 0.5 and 50 center counters and 10 rings with size ratio 0.6. Table 6.2 shows the results of this experiment on Twitter data. The first line with zero rings represents Tsum and the other lines give different parameter settings for Ringsum. We can see that the errors of focus and spread are less using the Tsum method. The reason is that the larger summary size in Tsum captures the more correct points of occurrence for the query terms and allows us to calculate the maximum likelihood function more accurately. Increasing the number of rings reduces the error of focus and spread for Ringsum, and this comes at the cost of a reduced center accuracy (note that the memory size is fixed).

Figure 6.4 shows the changes in focus and spread error by increasing the number of rings, and as we see using more rings results in more accurately calculated focus and spread. The summary size is directly related to the center accuracy. The larger summary size results in higher accuracy for centers. The accuracy of centers is not 100% in any of the methods, since unlike Backstorm et al. [4] that maximizes the likelihood function by trying different possible centers, we are setting a fixed center for each term using the summary we save for the terms, which is the cell with highest frequency for each term.

The same experiment on Flickr dataset shows similar results. However the errors are relatively lower on Flickr dataset. That is mainly because terms in Flickr dataset are more focused around their centers and our method finds models for terms with higher central focus and less spread more accurately.

We used the machine tags in Flickr dataset to create models for different camera brands used to take the pictures of the Flickr data. Table 6.1 shows the center, focus and spread for each brand. We can see that brands like “Canon” and “Olympus” have higher spread and lower focus which means they are being used around the world widely, but brands like “Minolta”, “Leica” and “Fujifilm” have lower spread and higher focus. These brands are being used more locally around their centers.

Table 6.1: Center, Focus and Spread of different camera brands

Camera Brand	Center	Focus	Spread
Minolta	39/-95	0.146	0.741
Sony	42/-72	0.048	0.538
Canon	51/-1	0.033	0.246
Olympus	37/-122	0.023	0.283
Nikon	23/119	0.064	0.342
Pentax	43/-80	0.042	0.561
Samsung	51/-1	0.118	0.943
Leica	35/139	0.131	0.95
Kodak	51/-1	0.042	0.628
Fujifilm	24/121	0.143	0.967

Table 6.2: Accuracy of the probabilistic model for different settings of center counters and rings on the Twitter data

		center	relative error		center
r	rings	counters	C	α	accuracy
-	0	216	0.188	0.043	96.7
0.3	5	81	0.281	0.243	94.1
0.4	7	65	0.229	0.156	92.2
0.5	8	59	0.184	0.164	91.8
0.6	10	50	0.180	0.110	91.4

Table 6.3: Accuracy of the probabilistic model for different settings of center counters and rings on the Flickr data

		center	relative error		center
r	rings	counters	C	α	accuracy
-	0	216	0.152	0.038	97.3
0.3	5	81	0.247	0.252	92.5
0.4	7	65	0.198	0.147	89.6
0.5	8	59	0.173	0.148	88.7
0.6	10	50	0.154	0.100	88.3

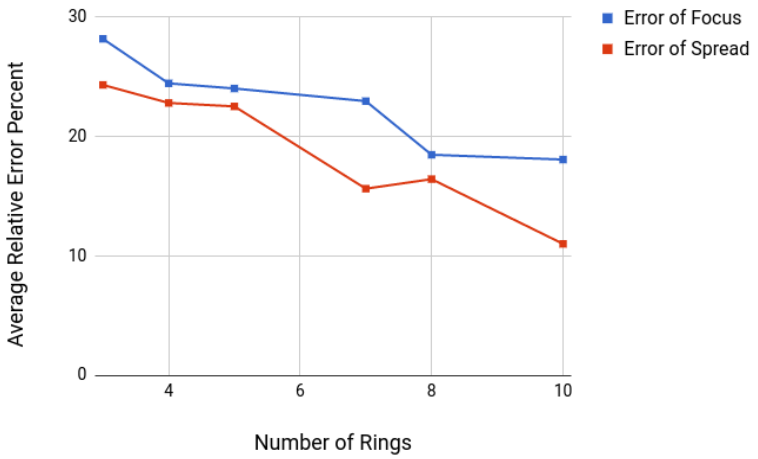

Figure 6.4: Changes in the error of focus and spread by increasing the number of rings in Ringsum

6.5 Accuracy of TFS Queries

In our next experiment, we employ our term distribution model to answer TFS queries. A question here is how well the model can predict the frequency of a term in an arbitrary location. It should be noted that this is a challenging task when the frequency is not kept for all locations. For this experiment, we selected 100 query terms randomly and for each term, we further randomly selected 10 locations where the term had a non-zero frequency. The result was a set of 1000 term-location pairs or queries. The ground truth for each pair was the real frequency of the query term in the location, which ranged between 1 and 2000. Our experiment shows that the log of frequency estimates are not that far from that of the actual frequencies, and the mean difference is around 0.50 for both Tsum and Ringsum on the Twitter dataset and 0.60 on the Flickr dataset (see Table 6.6 and 6.7). We measure the difference in the log scale to show the error in terms of the order of magnitude difference between estimates and actual values. For example, suppose the actual frequency of a term in a location is 30. Our method may estimate 50 for the frequency and we want this to be considered a good estimation, since the frequencies are in the same order of magnitude. The mean log distance error is calculated as follows:

$$\frac{|log(frequency_{estimated}) - log(frequency_{real})|}{1000}. \tag{6.1}$$

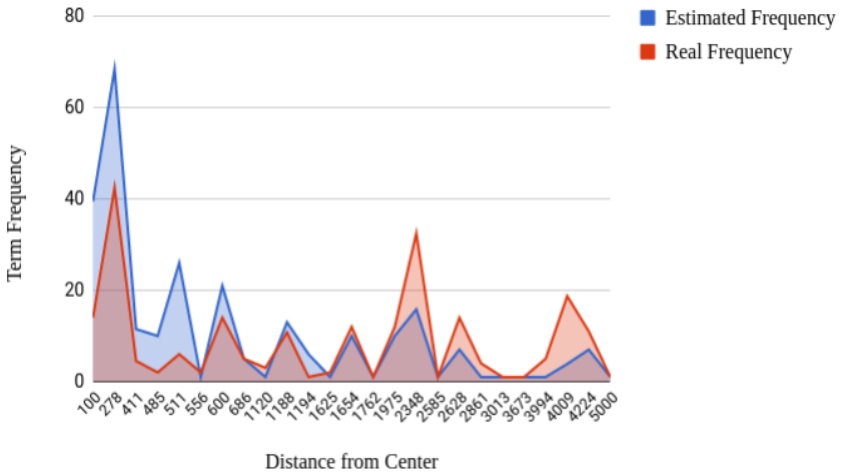

Figure 6.5: Frequency estimate and actual values varying the location distance from term center

Our next quest was to understand the types of errors the model was making. Figure 6.5 shows for 1000 random term-location pairs, both the mean frequency estimate as well as the mean real frequency of the term in the location. We can see that for locations that are closer to the center of queried terms, the frequency is overestimated and for locations that are farther, the frequency is underestimated. This stems from the fact that the probabilistic model assumes a high concentration of a term in its center, which gradually decreases as we move away from the center. This abstract model does not always hold especially when there are geographical and other constraints that affect the population around a center.

6.6 Multi-Term query Performance

One more area where the probabilistic model is expected to do better is multi-term queries. To evaluate this hypothesis and to assess the performance of our methods, we conducted an experiment to measure the accuracy of top frequent location for a set of multi-term queries. These queries were selected from the set of all two-terms that appeared at least once together in a location. We divided the multi-terms into four bins based on their frequencies:

- Rare: multi-terms that are rarely frequent in the data stream. Meaning that

they rarely appear together in a location. For example “fun girl” is a rare multi-term.

- Medium: multi-terms that are more likely to appear together in same locations. For example “metal music” is a medium multi-term.

- Frequent: multi-terms that appear together most of the times in the same locations. “High school” and “north carolina” are examples of this type of multi-term.

- Very frequent: multi-terms that appear mostly all the time together and are very frequently seen in the stream. Terms like “New York”, “National Park” and “Blue Jays” are examples of these multi-term queries.

To find the multi-term queries and their counts, we scan the whole stream and count the number of times different terms appear together. We divide the observed multi-terms to the above 4 groups based on their frequency and randomly pick 100 multi-terms from each group. We use 5 different methods to find the top first frequent location for the multi-terms in the queryset. The methods are:

1. Tsum Intersection: In this method, we just find the intersection between the summaries saved for each term in the multi-term query. The first common center is the result for the multi-term query.

2. Tsum Joint Probability: In this method, we find the joint probability of the occurrence of the multi-term in all locations and return the location with the highest probability as explained in chapter 4 (equation 4.1). We need the focus and the spread in order to find the probability model of the terms, which we find from the Tsum method.

3. Tsum Minimum Probability: In this method, we find the minimum probability of the occurrence of the multi-term in all locations and return the location with the highest probability as explained in chapter 4 (equation 4.2). We find the probabilities by using the focus and the spread that is calculated from the Tsum method.

4. Ringsum Joint Probability: This method is the same as Method 2, but the focus and the spread of each term in a multi-term query are calculated using the ring summary method.

5. Ringsum Minimum Probability: This method is the same as Method 3, but the focus and the spread of each term in the multi-term query are calculated using the ring summary method.

We know that query runtime is much higher in Tsum than Ringsum. We also know that the error of focus and spread is less in Tsum if we adjust the memory usage in both methods. So we expect to have a very high runtime for multi-term queries using methods 2 and 3 as well as more accurate results. We also expect to have low runtime for multi-term queries using methods 4 and 5, but with less accurate results. Figure 6.6 shows the accuracy of RFS queries for all 5 methods on the Twitter dataset, with queries selecting a top location for each multi-term. A few observations can be made here.

First, we can see that the method that does not use the probabilistic model (Tsum intersection) has the least accuracy among all our methods. This is expected since top locations for a multi-term does not necessarily appear in the summary of each of the terms. In other words, it pays off to use the probabilistic model since it allows us estimate the frequency in locations that are not kept in the summary. Second, Tsum has a slight edge over Ringsum in terms of the accuracy of the results and this is consistent with our result on estimating the frequency of single-terms in a location (as discussed in Section 6.5). Third, the independence assumption seems to work better for multi-terms that are rare or not frequent whereas a non-independent assumption gives more accurate results for multi-terms that are frequent or very frequent.

Finally the accuracy improves as queries become more frequent in a location. This is because such locations are more likely to be kept in individual term summaries, allowing more accurate estimates. An example of this is shown in Tables 6.4 and 6.5 for a set of camera brands and the term “lens” that appear in the machine tags of photos in the Flickr dataset. The table gives the actual and estimated cen-

Table 6.4: Top frequent location (RFS) for multi-terms on Flickr data

Multi-Term	Center	Estimated Center
Sony Lens (very frequent)	42/-72	42/-72
Olympus Lens (frequent)	37/-122	37/-122
Fujifilm Lens (medium)	24/121	25/121
Minolta Lens (rare)	40/-74	39/-95

Table 6.5: Estimated frequency in center (TFS) for multi-terms on Flickr data

Multi-Term	Center Freq.	Estimated Freq.
Sony Lens (very frequent)	1491	216
Olympus Lens (frequent)	511	48
Fujifilm Lens (medium)	126	14
Minolta Lens (rare)	21	1

ter where the term is most frequent as well as the actual and estimated frequency of the term in the center, using the ring summary method. The center is estimated correctly for frequent multi-terms "sony lens" and "olympus lens" and not correctly for non-frequent multi-terms "fujifilm lens" and " minolta lens".

6.7 Runtime

In our next experiment, we evaluate Tsum and Ringsum in terms of their running times. For each method, we measure the time it takes to evaluate queries and to update the summary as the data streams in. Our query time is the time to answer a TFS query. RFS queries are easier to evaluate and are not considered here. RFS query time is directly related to the summary size. For answering a top-K RFS query we need $O(mlogm+K)$ time to sort the summary (with size m) and return the first K items. Our update time is the time it takes to create and/or update the summary per one tweet (in Twitter dataset) or one photo tag (in Flickr dataset).

Tables 6.6 and 6.7 show the result of the experiment. We can see that summary update time is lower for Tsum. It is higher for Ringsum due to the complex computations for initializing ring counters when their center counter is replaced with a new location. Finding the intersection between the rings and initializing the rings by new values in this step of the ring summary method is a time consuming process that the Tsum method lacks. Tsum only updates the center counters during the

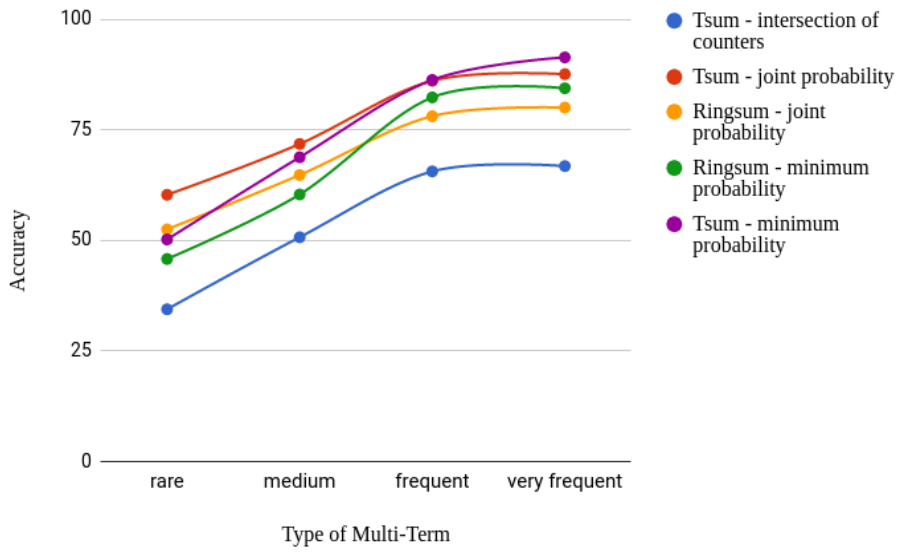

Figure 6.6: Average accuracy of the top location for multi-term queries (RFS) on Twitter data

Table 6.6: Performance on TFS queries on Twitter dataset

Method	Update Time (msec)	Query Time (sec)	Log Dist. Error
Tsum	0.068	38.15	0.47
Ring Summary	0.736	0.23	0.53
Light-Update	0.161	0.23	0.57
Fixed-Center	0.014	0.23	0.49
Proximity-Aware	0.804	0.23	0.57

stream processing since it has no rings.

TFS Query runtime is much higher in Tsum than Ringsum. TFS Query runtime in both Tsum and Ringsum is constant and does not depend to the size of incoming stream. This query runtime in Tsum is relative to the number of cells we have in the geographical grid system because for each center of a query term, the number of operations to calculate the focus and the spread is equal to the number of cells in the grid map. In another hand, the number of operations needed to calculate focus and spread for each center of a query term in Ringsum is equal to the number of rings around the center counter. Since the number of cells are always much larger than the number of rings, the query runtime will be much less for Ringsum than Tsum.

Table 6.7: Performance on TFS queries on Flickr dataset

Method	Update Time (msec)	Query Time (sec)	Log Dist. Error
Tsum	0.097	22.46	0.62
Ring Summary	0.853	0.12	0.68
Light-Update	0.242	0.12	0.78
Fixed-Center	0.018	0.12	0.60
Proximity-Aware	0.915	0.12	0.74

6.8 Improvements Analysis

In this set of experiments, we study the extent at which the improvements discussed in Section 5 can boost the performance of the Ringsum method.

Our first improvement looks at the possibility of fixing centers to save time during updates. Figure 6.7 shows the average number of replacements in the top-3 center counters in the summaries of the terms as different volumes of the stream passes. We can see that as more volume of the stream is received, the average number of replacements decreases. After passing about 40% of the stream, less 25% of the first center counters are getting replaced. This observation helps us define a variation of the ring summary method, which fixes the center counters after a specific amount of stream passes. This amount is different for terms with different focus and spread.

Figure 6.8 shows the needed percentage of the stream that we must process to be able to fix the centers for terms with different C and α. We fix the centers for terms if after that point more than half of the counters in the summary remain the same during the update process. Figure 6.8 shows that for terms with $C > 0.4$ and $\alpha > 0.6$ we can fix the centers after passing 20% of the stream. We show the error of C and α for these terms using the fixed-center ring summary method in Table 6.8. We can see that error for C and α in this experiment decreases significantly, but the center accuracy also decreases, which means that it cannot find some centers correctly, but for those centers that it finds, the probabilistic model will be more accurate.

We conduct an experiment to compare the stream processing runtime in light-update ring summary method with other methods. Tables 6.6 and 6.7 show the time for different methods and we can see that using the light-update we can achieve

Table 6.8: Error of focus, spread and center accuracy for improved alternative methods of ring summary

Alternate Method	Error of C	Error of α	Center Accuracy
Fixed-Center	18.32	7.54	81.3
Light-Update	20.03	14.42	91.4
Proximity-Aware	20.47	9.82	68.2

to a stream processing time close to Tsum. Table 6.8 shows the error of focus and spread for this variation of our method. We can see that the errors are more than the ring summary method with the same number of rings. So this variation, although creates a slightly less accurate model for terms, it is much faster than the ring summary method.

We also conduct another experiment for analyzing the error of focus, spread and center accuracy in the proximity-aware ring summary method. Table 6.8 shows that the errors of focus and spread decrease, which is due to the less number of replacements in center counters and hence, less complex initializing for new rings. Center accuracy decreases in this variation, because the update process is different than that in SpaceSaving and we can no longer guarantees to find the ε-frequent locations. This method, similar to the fixed-center variation, works well for cases that the center is correctly found.

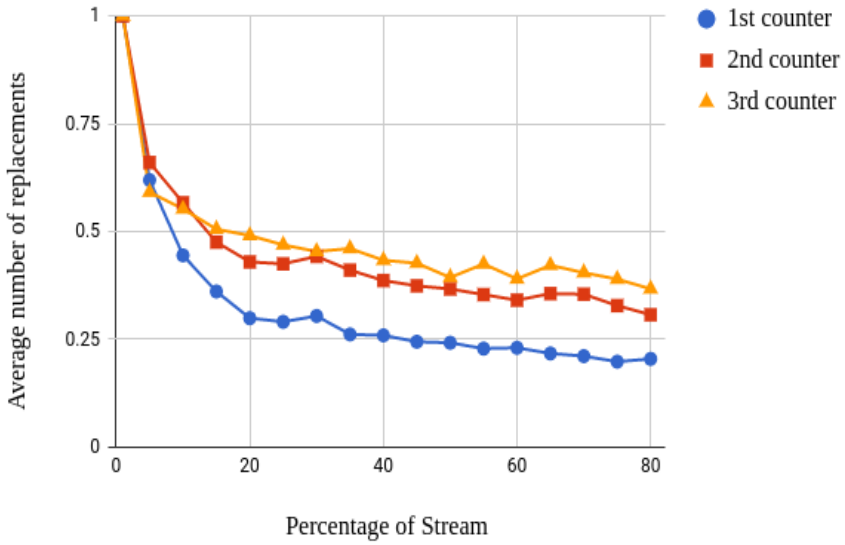

Figure 6.7: The average number of replacements in the first top-location (center counters) of the summaries as more data is received

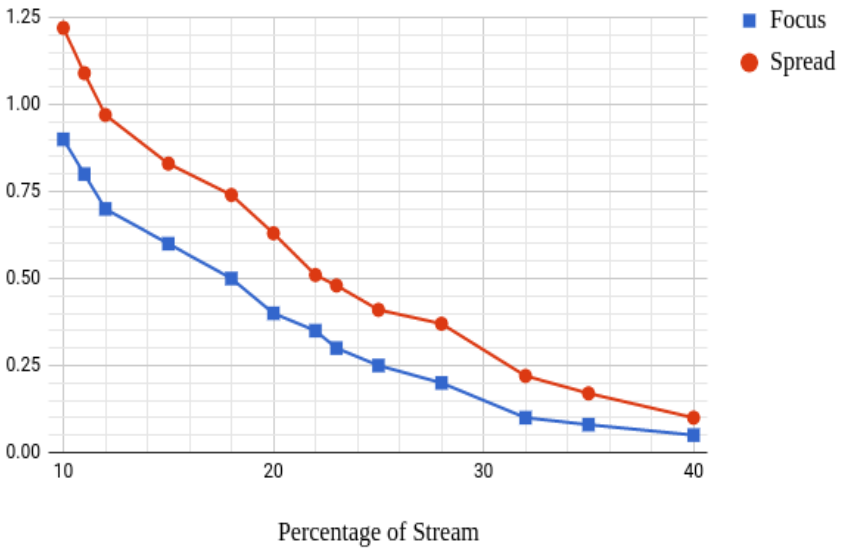

Figure 6.8: The needed percentage of the stream that must pass to fix the center counters for terms with different focus and spreads

Chapter 7

Conclusions and Future Work

We propose an efficient and scalable method for evaluating top-K spatial frequent term queries in a stream of geo-tagged events. More specifically, we introduce two new queries on streaming geo-tagged data as follows: (1) Reverse Frequent Spatial (RFS) queries: Given a term t and an integer K what are the top K locations where term t is frequently used (2) Term Frequent Spatial (TFS) queries: Given a term t and a location l, what is the estimated frequency of appearance for term t in location l. We develop two counter-based methods (referred to as Tsum and Ringsum) to keep a summary for each term, consisting of its top frequent locations and also a probabilistic distribution model to estimate the frequency of terms in different geographical locations on the map. Our method is general enough to estimate the frequency of a given term at any given location, even if the location is not saved in the term summary. We also use this model to extend our work and also support multi-term queries.

Our evaluation demonstrates that each of our methods shine in one or more areas of the accuracy of the results, the efficiency of queries, and the efficiency of updates, and our improvements can further boost the performance of Ringsum.

As a possible future direction, our algorithms may be studied over sliding windows with old data being removed as the time progresses, so it can support time window tracking and find the top recent locations for the trend of events. Another area for future research is managing the number of terms to be monitored. Maintaining a summary for every possible term or event is costly and selecting a subset of terms in advance for some domains may not be trivial. One may find some

groupings of the terms for example, political, social, natural, etc., and classify each query term to one of the groups, so that we only keep a summary for each group, not all the terms in the stream. Exploring different grouping strategies and the areas where our summary structure may be improved to better support those groups is an interesting future direction. Another research direction is to explore different term distributions in order to set more accurate ring distance intervals.

References

[1] P. Ahmed, M. Hasan, A. Kashyap, V. Hristidis, and V. J. Tsotras, "Efficient computation of top-k frequent terms over spatio-temporal ranges," in *Proceedings of the 2017 ACM International Conference on Management of Data*, ACM, 2017, pp. 1227–1241. 14

[2] S. Almeida, J. Queijo, and L. M. Correia, "Spatial and temporal traffic distribution models for gsm," in *Vehicular Technology Conference, 1999. VTC 1999-Fall. IEEE VTS 50th*, IEEE, vol. 1, 1999, pp. 131–135. 10

[3] E. Amitay, N. Har'El, R. Sivan, and A. Soffer, "Web-a-where: Geotagging web content," in *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, ACM, 2004, pp. 273–280. 7

[4] L. Backstrom, J. Kleinberg, R. Kumar, and J. Novak, "Spatial variation in search engine queries," in *Proceedings of the 17th international conference on World Wide Web*, ACM, 2008, pp. 357–366. 7, 10, 20, 21, 41, 42

[5] N. Bansal and N. Koudas, "Blogscope: A system for online analysis of high volume text streams," in *Proceedings of the 33rd international conference on Very large data bases*, VLDB Endowment, 2007, pp. 1410–1413. 12

[6] C. Budak, D. Agrawal, and A. El Abbadi, "Structural trend analysis for online social networks," *Proceedings of the VLDB Endowment*, vol. 4, no. 10, pp. 646–656, 2011. 12

[7] C. Budak, T. Georgiou, D. Agrawal, and A. El Abbadi, "Geoscope: Online detection of geo-correlated information trends in social networks," *Proceedings of the VLDB Endowment*, vol. 7, no. 4, pp. 229–240, 2013. 12

[8] O. Buyukokkten, J. Cho, H. Garcia-Molina, L. Gravano, and N. Shivakumar, "Exploiting geographical location information of web pages," 1999. 7

[9] X. Cao, G. Cong, C. S. Jensen, and M. L. Yiu, "Retrieving regions of interest for user exploration," *Proceedings of the VLDB Endowment*, vol. 7, no. 9, pp. 733–744, 2014. 11

[10] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," in *International Colloquium on Automata, Languages, and Programming*, Springer, 2002, pp. 693–703. 6

[11] Z. Cheng, J. Caverlee, and K. Lee, "You are where you tweet: A content-based approach to geo-locating twitter users," in *Proceedings of the 19th ACM international conference on Information and knowledge management*, ACM, 2010, pp. 759–768. 8, 20

[12] G. Cong, C. S. Jensen, and D. Wu, "Efficient retrieval of the top-k most relevant spatial web objects," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 337–348, 2009. 11

[13] G. Cormode and S. Muthukrishnan, "What's hot and what's not: Tracking most frequent items dynamically.," *TODS*, vol. 30, no. 1, pp. 249–278, 2005. 5

[14] G. Cormode and M. Hadjieleftheriou, "Finding frequent items in data streams," *Proceedings of the VLDB Endowment*, vol. 1, no. 2, pp. 1530–1541, 2008. 6, 7, 16

[15] G. Cormode and S. Muthukrishnan, "An improved data stream summary: The count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005. 6

[16] ——, "What's hot and what's not: Tracking most frequent items dynamically," *ACM Transactions on Database Systems (TODS)*, vol. 30, no. 1, pp. 249–278, 2005. 6

[17] D. J. Crandall, L. Backstrom, D. Huttenlocher, and J. Kleinberg, "Mapping the world's photos," in *Proceedings of the 18th international conference on World wide web*, ACM, 2009, pp. 761–770. 7

[18] E. D. Demaine, A. López-Ortiz, and J. I. Munro, "Frequency estimation of internet packet streams with limited space," in *European Symposium on Algorithms*, Springer, 2002, pp. 348–360. 6

[19] P. Deville, C. Linard, S. Martin, M. Gilbert, F. R. Stevens, A. E. Gaughan, V. D. Blondel, and A. J. Tatem, "Dynamic population mapping using mobile phone data," *Proceedings of the National Academy of Sciences*, vol. 111, no. 45, pp. 15 888–15 893, 2014. 9

[20] J. Ding, L. Gravano, and N. Shivakumar, "Computing geographical scopes of web resources," Stanford, Tech. Rep., 1999. 10

[21] *Docs — twitter developers*, https://developer.twitter.com/en/docs, Accessed: 2018-02-18. 38

[22] K. R. Felzer and E. E. Brodsky, "Decay of aftershock density with distance indicates triggering by dynamic stress," *Nature*, vol. 441, no. 7094, p. 735, 2006. 9

[23] W. Feng, C. Zhang, W. Zhang, J. Han, J. Wang, C. Aggarwal, and J. Huang, "Streamcube: Hierarchical spatio-temporal hashtag clustering for event exploration over the twitter stream," in *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*, IEEE, 2015, pp. 1561–1572. 13

[24] C. Fink, C. D. Piatko, J. Mayfield, T. Finin, J. Martineau, *et al.*, "Geolocating blogs from their textual content.," in *AAAI Spring Symposium: Social Semantic Web: Where Web 2.0 Meets Web 3.0*, 2009, pp. 25–26. 7

[25] A. Gallagher, D. Joshi, J. Yu, and J. Luo, "Geo-location inference from image content and user tags," in *Computer Vision and Pattern Recognition Workshops, 2009. CVPR Workshops 2009. IEEE Computer Society Conference on*, IEEE, 2009, pp. 55–62. 7, 8

[26] *Gazetteer*, `https://en.wikipedia.org/wiki/Gazetteer`, Accessed: 2018-07-13. 8

[27] L. Gravano, V. Hatzivassiloglou, and R. Lichtenstein, "Categorizing web queries according to geographical locality," in *Proceedings of the twelfth international conference on Information and knowledge management*, ACM, 2003, pp. 325–333. 10

[28] A. Guisan, T. C. Edwards Jr, and T. Hastie, "Generalized linear and generalized additive models in studies of species distributions: Setting the scene," *Ecological modelling*, vol. 157, no. 2-3, pp. 89–100, 2002. 9

[29] A. Guttman, *R-trees: A dynamic index structure for spatial searching*, 2. ACM, 1984, vol. 14. 14

[30] L. Hong, A. Ahmed, S. Gurumurthy, A. J. Smola, and K. Tsioutsiouliklis, "Discovering geographical topics in the twitter stream," in *Proceedings of the 21st international conference on World Wide Web*, ACM, 2012, pp. 769–778. 10

[31] M. Hurst, M. Siegler, and N. S. Glance, "On estimating the geographic distribution of social media.," in *ICWSM*, 2007. 7

[32] C. Jin, W. Qian, C. Sha, J. X. Yu, and A. Zhou, "Dynamically maintaining frequent items over a data stream," in *Proceedings of the twelfth international conference on Information and knowledge management*, ACM, 2003, pp. 287–294. 5

[33] C. Jonathan, A. Magdy, M. F. Mokbel, and A. Jonathan, "Garnet: A holistic system approach for trending queries in microblogs," in *Data Engineering (ICDE), 2016 IEEE 32nd International Conference on*, IEEE, 2016, pp. 1251–1262. 1, 14

[34] E. Kamalloo and D. Rafiei, "A coherent unsupervised model for toponym resolution," in *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, International World Wide Web Conferences Steering Committee, 2018, pp. 1287–1296. 8

[35] R. M. Karp, S. Shenker, and C. H. Papadimitriou, "A simple algorithm for finding frequent elements in streams and bags," *ACM Transactions on Database Systems (TODS)*, vol. 28, no. 1, pp. 51–55, 2003. 6

[36] S. Kinsella, V. Murdock, and N. O'Hare, "I'm eating a sandwich in glasgow: Modeling locations with tweets," in *Proceedings of the 3rd international workshop on Search and mining user-generated contents*, ACM, 2011, pp. 61–68. 10

[37] X. Liang, J. Zhao, L. Dong, and K. Xu, "Unraveling the origin of exponential law in intra-urban human mobility," *Scientific reports*, vol. 3, p. 2983, 2013. 9

[38] J. Lin and A. Halavais, "Mapping the blogosphere in america," in *Workshop on the Weblogging Ecosystem at the 13th International World Wide Web Conference*, vol. 18, 2004, pp. 1–7. 7

[39] A. Magdy, A. M. Aly, M. F. Mokbel, S. Elnikety, Y. He, S. Nath, and W. G. Aref, "Geotrend: Spatial trending queries on real-time microblogs," in *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, ACM, 2016, p. 7. 1, 13

[40] A. Magdy, M. F. Mokbel, S. Elnikety, S. Nath, and Y. He, "Mercury: A memory-constrained spatio-temporal real-time search on microblogs," in *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, IEEE, 2014, pp. 172–183. 13

[41] ——, "Venus: Scalable real-time spatial queries on microblogs with adaptive load shedding," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 2, pp. 356–370, 2016. 1, 14

[42] N. Manerikar and T. Palpanas, "Frequent items in streaming data: An experimental evaluation of the state-of-the-art," *Data & Knowledge Engineering*, vol. 68, no. 4, pp. 415–430, 2009. 7

[43] A. Manjhi, V. Shkapenyuk, K. Dhamdhere, and C. Olston, "Finding (recently) frequent items in distributed data streams," in *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*, IEEE, 2005, pp. 767–778. 5

[44] G. S. Manku and R. Motwani, "Approximate frequency counts over data streams," in *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*, Elsevier, 2002, pp. 346–357. 6

[45] M. Mathioudakis, N. Bansal, and N. Koudas, "Identifying, attributing and describing spatial bursts," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 1091–1102, 2010. 12

[46] M. Mathioudakis and N. Koudas, "Twittermonitor: Trend detection over the twitter stream," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, ACM, 2010, pp. 1155–1158. 12

[47] A. Metwally, D. Agrawal, and A. El Abbadi, "Efficient computation of frequent and top-k elements in data streams," in *International Conference on Database Theory*, Springer, 2005, pp. 398–412. 6, 16, 17, 19, 37

[48] J. Misra and D. Gries, "Finding repeated elements," *Science of computer programming*, vol. 2, no. 2, pp. 143–152, 1982. 16

[49] *National hurricane center*, `https://www.nhc.noaa.gov/aboutnhcprobs3.shtml`, Accessed: 2018-07-13. 9

[50] *New york times*, `https://www.nytimes.com/interactive/2014/04/23/upshot/24-upshot-baseball.html`, Accessed: 2018-07-13. 10

[51] W. D. Nordhaus, "Geography and macroeconomics: New data and new findings," *Proceedings of the National Academy of Sciences*, vol. 103, no. 10, pp. 3510–3517, 2006. 9

[52] D. Richarte, M. Lupari, A. Pesce, S. Nacif, and M. Gimenez, "3-d crustal-scale gravity model of the san rafael block and payenia volcanic province in mendoza, argentina," *Geoscience Frontiers*, vol. 9, no. 1, pp. 239–248, 2018. 9

[53] J. B. Rocha-Junior, O. Gkorgkas, S. Jonassen, and K. Nørvåg, "Efficient processing of top-k spatial keyword queries," in *International Symposium on Spatial and Temporal Databases*, Springer, 2011, pp. 205–222. 11

[54] P. Serdyukov, V. Murdock, and R. Van Zwol, "Placing flickr photos on a map," in *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, ACM, 2009, pp. 484–491. 10

[55] A. Skovsgaard, D. Sidlauskas, and C. S. Jensen, "Scalable top-k spatio-temporal term querying," in *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, IEEE, 2014, pp. 148–159. 1, 3, 5, 13

[56] *Species at risk public registry*, Accessed: 2018-07-13. [Online]. Available: `https://sararegistry.gc.ca/document/doc2481/consul_s5_e.cfm`. 9

[57] X. Wang, Y. Zhang, W. Zhang, X. Lin, and W. Wang, "Ap-tree: Efficiently support continuous spatial-keyword queries over stream," in *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*, IEEE, 2015, pp. 1107–1118. 11

[58] *Wikipedia*, `https://en.wikipedia.org/wiki/Inverse-square_law`, Accessed: 2018-07-13. 9

[59] *Wikipedia*, `https://en.wikipedia.org/wiki/Distance_decay`, Accessed: 2018-07-13. 10

[60] D. Yang, A. Shastri, E. A. Rundensteiner, and M. O. Ward, "An optimal strategy for monitoring top-k queries in streaming windows," in *Proceedings of the 14th International Conference on Extending Database Technology*, ACM, 2011, pp. 57–68. 12

[61] *Yfcc100m dataset browser*, http://yfcc100m.appspot.com/, Accessed: 2018-02-18. 38

[62] X. Yi, H. Raghavan, and C. Leggetter, "Discovering users' specific geo intention in web search," in *Proceedings of the 18th international conference on World wide web*, ACM, 2009, pp. 481–490. 7

[63] J. Yu and D. Rafiei, "Geotagging named entities in news and online documents," in *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management (CIKM'16). ACM, New York, NY, USA*, 2016. 7, 8, 10

[64] D. Zhang, Y. M. Chee, A. Mondal, A. K. Tung, and M. Kitsuregawa, "Keyword search in spatial databases: Towards searching by document," in *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on*, IEEE, 2009, pp. 688–699. 11

[65] W. Zong, D. Wu, A. Sun, E.-P. Lim, and D. H.-L. Goh, "On assigning place names to geography related web pages," in *Proceedings of the 5th ACM/IEEE-CS joint conference on Digital libraries*, ACM, 2005, pp. 354–362. 7