National Technical University of Athens

School of Electrical
and Computer Engineering

Division of Computer Science

Advanced techniques and algorithms to collect, analyze and visualize spatiotemporal data from social media feeds

PhD Thesis

of

Georgios Lamprianidis

MSc in Electrical and Computer Engineering,
National Technical University of Athens

Athens, July 2016

Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων Και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Προηγμένες τεχνικές και αλγόριθμοι για συλλογή, ανάλυση και οπτικοποίηση χωροχρονικών δεδομένων από κοινωνικά δίκτυα

Διδακτορική Διατριβή

του

Γεώργιου Λαμπριανίδη

Διπλωματούχου Ηλεκτρολόγου Μηχανικού και Μηχανικού Υπολογιστών ΕΜΠ

Συμβουλευτική Επιτροπή: Ι. Βασιλείου
Τ. Σελλής
D. Pfoser

Εγκρίθηκε από την επταμελή εξεταστική επιτροπή την 13η Ιουλίου 2016.

Ι. Βασιλείου Καθ. ΕΜΠ	Τ. Σελλής Καθ. Swinburne Univ. of Technology	D. Pfoser Αν. Καθ. George Mason Univ.
Α. Γ. Σταφυλοπάτης Καθ. ΕΜΠ	Κ. Κοντογιάννης Καθ. ΕΜΠ	Ι. Σταύρακας Ερευν. Β΄ Ερ. Κέντρου 'ΑΘΗΝΑ'
	Μ. Κάβουρας Καθ. ΣΑΤΜ ΕΜΠ	

Αθήνα, Ιούλιος 2016

...

Georgios Lamprianidis
Electrical and Computer Engineer, PhD, N.T.U.A.

...

Γεώργιος Λαμπριανίδης
Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών, PhD, Ε.Μ.Π.

Contents

List of Figures

List of Tables

PREFACE

This thesis is submitted in fulfillment of the requirements for the degree of Doctor of Philosophy, in the School of Electrical and Computer Engineering, National Technical University of Athens (NTUA), Greece. The presented work describes efficient methods and optimized algorithms to extract, process, analyze and finally visualize data from social media sources in order to solve several complex problems. The main volume of work has been carried out during the last six years in the Institute for the Management of Information Systems (IMIS) of Research Center "Athena" and the Geography and Geoinformation department of George Mason University.

I am very grateful to Prof. Yannis Vasilliou and Prof. Timos Sellis for supervising this thesis and giving me a chance to work with them and their teams during my studies, and all the valued members of my committee for all the helpful observations and remarks. I would like to thank Dr. Yannis Stavrakas for his advice and incentive to join the PhD programme. I would also like to thank all my colleagues and all the people that have in many ways contributed to this work, especially the Geoinformatics Group of IMIS, for all their ideas, support and those endless evenings working together. A very special and sincere thank you to Dr. Dieter Pfoser for all the encouragement, tenacity, patience and inspiration throughout all this time. Without his mentoring and guidance this effort would not be completed.

On a personal note, I would also like to thank my close friends for all the best times and my family to whom this work is dedicated: my parents Lazaros and Teresa, my sister Helen and my aunt Anastasia.

The work presented in this dissertation was also partially funded by the EU 7th Framework Programme under the following projects:
I.T.N. "GEOCROWD" (`http://www.geocrowd.eu`, GA: FP7-PEOPLE-2010-ITN-264994), "SimpleFleet" (`http://www.simplefleet.eu`, GA: FP7-ICT-2011-SME-DCL-296423), "TALOS Mobile Guides" (`http://www.tmguide.eu`, GA: FP7-SME-2012-315333) and "GEOSTREAM" (`http://geocontentstream.eu`, GA: FP7-SME-2012-315631).

Georgios Lamprianidis
Athens, July 2016

To my family.

ABSTRACT

During the past decade, the social media evolution has grown to become an essential part of our lives, producing an incredible amount of user-generated and voluntarily contributed content, about people's everyday activities, thoughts and interests. Besides the intent of the users and the platforms, we believe that the information residing within these streaming datasets can be useful in other contexts as well, such as enriching collections and gazetteers, providing real-time notifications and alerts, identifying trends and providing academia and industry with valuable on-the-ground knowledge. To this end, this thesis focuses on exploring the potentials of these datasets by designing and implementing advanced algorithms and data structures to extract, analyze and visualize data and information from the social media traffic.

Throughout this dissertation we describe several cases of using social media data to solve different problems, or enhance existing solutions. The cases studied include: (i) locating and describing geospatial features, (ii) integrating geospatial content from several sources, (iii) providing multidimensional interactive and interconnected summaries of social media traffic, (iv) enhancing event detection by suggesting a methodology to rank events based on their spatial footprint, and (v) linking together places using people's activities. Our extensive work is accompanied by a series of experiments and examples. The lessons learnt allow us to generalize and modularize the workflow pattern discovered in all our previous efforts, which benefits any future work.

Chapter 1

Introduction

In the era of *Big Data* and the connected world, social media platforms have become a commodity and as such have contributed to large pools of data feeds which are constantly increasing in size and evolving in complexity. This data becomes available in large volumes and high speeds, increasing the demand for efficient algorithms and systems in order to process it and extract meaningful information out of it. The data itself has several attributes, such as time and location tagging, optionally attached multimedia, plus social structure identifiers, giving us the opportunity to study these attributes not only separately but also in combination with one another for more comprehensive results. The goal of this thesis is both to present algorithms and data structures, and to describe fully implemented and functional systems for collecting, analyzing and visualizing social media feeds. This chapter describes the overall scientific focus of our efforts and highlights individual contributions.

1.1 Prologue

A very challenging task for most researchers, even before starting any kind of analysis, is the part of actually obtaining useful and relevant to the problem datasets that can be subsequently processed to give answers to the various research questions. Luckily, the huge increase in usage of social networking platforms over the past decade (cf. Figure 1.1), created an abundance of data as it allowed people of various demographics to voluntarily share moments, ideas, thoughts, moods and news in a very simple and efficient way. Tapping into this large pool of data might prove beneficial but can be challenging. As their origin might imply, these social media feeds provide data that reflects the public's opinion and thus cannot be considered as an authoritative source; in fact it is quite noisy and of high uncertainty, however easily accessible, voluntarily provided and usually instantaneous. As there are numerous scenarios where the public's perspective is important, social media streams become a good candidate for instant and up-to-date information. To increase the value and credibility of the analysis we can compare the results of the analysis with results extracted from official datasets to further assess the suitability of using social media feeds for specific categories of problems. Furthermore once suitability of usage has been confirmed for a specific category of problems, we can argue that using social media datasets for similar problems is sufficient and will produce acceptable results.

As explained earlier, social media data has embedded attributes describing several dimensions, however our focus in this work is primary the spatial and temporal dimensions. Geospatial content has seen a tremendous increase in use in applications and services over

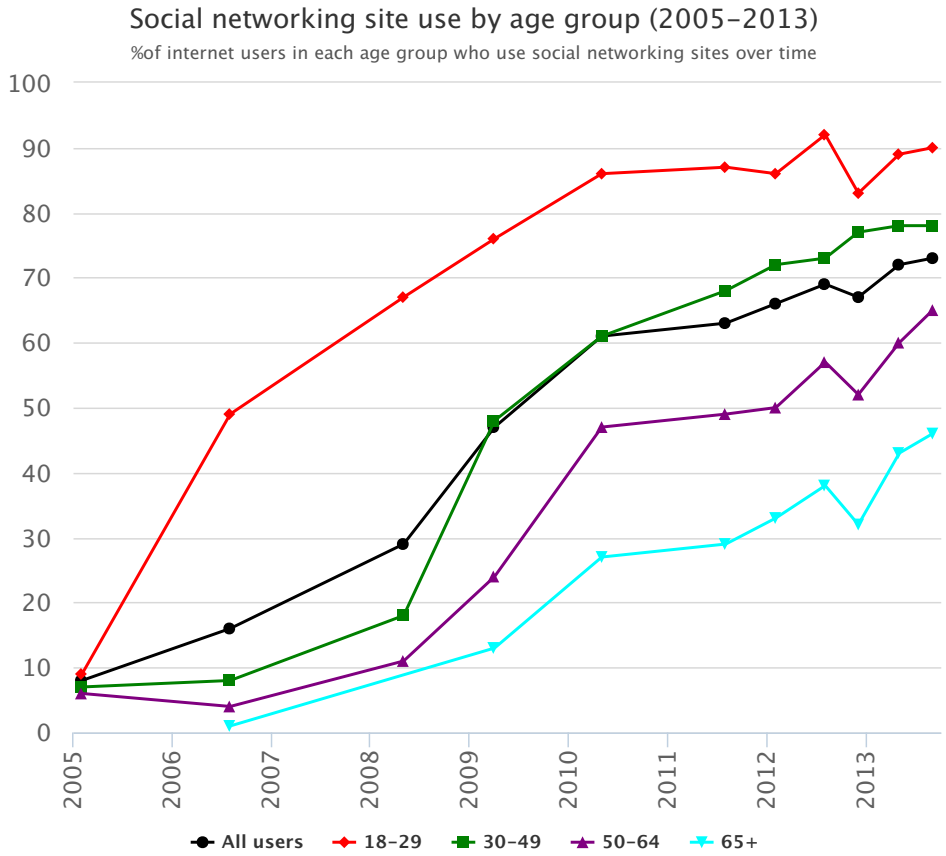


Figure 1.1: *Evolution of social platforms usage per age group.*

the past years. Even in cases where it is not used as content but only as metadata, the enrichment of the data with geographical information adds extra value both to the provided service and the dataset itself. When used in combination with timestamp metadata, it allows us to study the evolution of datasets over time, and to gain useful insight about how certain aspects of the data are changing as time elapses, or even identify patterns based on the different time intervals within a day, or during whole weeks, months, etc.

In this dissertation the focus is on techniques to extract, analyze and visualize spatiotemporal data from social media sources. A series of scenarios will be narrated, where in each one, a problem will be described and a solution will be given using one or multiple social media feeds. The techniques, algorithms and data structures presented, will be become the building blocks of the systems that implement the suggested solutions. While the research was initially problem-oriented, in the sense that a specific problem was discussed and subsequently a custom solution was designed and implemented, after a few cases it became obvious that the 3-fold pattern of extraction, analysis and visualization was emerging every time. In Chapter 8, we describe these commonalities and present a generalization of the specific methods described in the previous chapters.

1.2 Contribution

This dissertation contributed by means of robust data structures, efficient algorithms and architecture layouts, conceived while dealing with the different scenarios that will be described later on. In short the most important contributions are presented here:

1.2.1 Hierarchical grid/cube data structure

Since our dataset includes single social media posts that are assigned an exact timestamp (second or millisecond precision) and detailed geographic coordinates (accuracy at the 100m level or more), we need a way to aggregate this information in order to be able to make sense of it. Initially ([51]) we focused on the spatial dimension alone, overlaying the surface of the earth with a rectangular grid (mesh) of predefined cell size. By applying the grid on the dataset, individual social media posts get "bundled" into the grid cell that includes their geographic location, and the grid cell accumulates metadata for all the *bundled* posts. This is similar to the edge bundling techniques used to declutter network visualizations.

By design, the cell size is fixed to improve query performance and random access times. The structure is enriched by adding hierarchy in order to provide numerous views of the same dataset at different levels of detail. Multiple grid levels are introduced, by recursively combining grid cells starting from our base level grid to form bigger cells in the higher levels. Each cell in an upper level, combines not only the spatial characteristics of the cells it encloses, but also accumulates their metadata. Eventually the structure contains not just one grid, but a collection of grids, all displaying the same data at different detail levels. The construction of the upper levels of the hierarchical grid, is not a simple aggregation process, but it uses heuristics in each step such as cell adjacency, grid occupancy and coverage to eliminate *unimportant* cells as it moves to the higher levels, as described in more detail in Section 3.3.

Later on ([52]), we improved the data structure by supporting the temporal dimension as well, practically by converting the flat grid into a cube-like structure. The new structure stores information about each grid cell, separately per a predefined time interval, seemingly making each level look like a stacked pile of temporal versions of the grid. This way we can query the stored information to project data relevant only to a specific time period, rather that the entire time span of the dataset, plus observe and study the evolution of certain aspects of the data over time. The improved cube also implements the hierarchy concept in both dimensions (space and time), allowing different binning of the data in each one. The hierarchical cube is presented in detail in Section 5.7.

1.2.2 Browser based computing architecture

The remote nature of the datasets and their large size, indicate that they are a good fit for parallel processing on multiple nodes. As server farms infrastructures are relatively expensive and difficult to maintain, we experimented with the idea of publishing a system as a web application and distribute the workload to the browsers of its users. We introduce the notion of a web browser based computing architecture implementing a programming model similar to MapReduce, where users' browsers connected to our application are accepting to become computing nodes in this infrastructure. The architecture borrows several concepts from the original MapReduce model, such as fault tolerance, job reassignment, etc...

The web server plays the role of the master worker and is responsible for distributing the tasks to the clients (browsers) and handle concurrent access, erroneous responses, network connectivity and timeout issues. Since the availability of our computing nodes is not controlled or guaranteed, as the clients can connect and disconnect at random times, we change the job delivery mechanism, from a *push* operation — meaning the master worker sends the tasks to the available clients — to a *pull* operation, allowing the clients

to request for queued tasks whenever they connect to the system. The clients are then given a specific time allowance to finish their task and submit their results back to the server. Timeout events are simply ignored, and the task reenters the queue and becomes available to the next client. Using connected web clients as computing nodes increases the scalability of the architecture, as a potential high number of clients can be connected to the service to increase its level of parallelism, limited only by the amount of concurrent requests our web application can handle. Furthermore to maintain the throughput above a minimum, the system employs a small number of dedicated machines ready to step in when few or no actual web browsers are connected to the service.

1.2.3 Universal platform collector

As the opening statement of the previous subsection points out, social media feeds are remote datasources that first need to be collected locally before further processing can occur. The diversity of the available social media platforms and the different purposes for participation in each one makes the choice of one as a source of data for a research problem hard. Furthermore managing and maintaining implementations that tap into each of the respective APIs can prove to be challenging.

For the aforementioned reasons, we implement and utilize a universal collector platform, that is capable of connecting to each available source and easily download the required datasets. This universal solution has a common interface that allows it to be parameterized to allow queries based on keywords, geographic locations and given time periods. It is built in a modular way to allow adding new social media platforms with minimum effort, namely only changing the query interface to retrieve the data and optionally the response handler to save it into a persistent storage — if processing is needed before saving. This allows us to demonstrate how the enabled social media platforms can all be accessed in a similar workflow, just by tweaking a set of parameters. The concept for the universal collector is used at an early stage in [53, 54] and presented in more detail in Section 8.2.

1.2.4 Detecting duplicate records and regions of interest

Using a universal extraction solution, made getting data from multiple social media feeds a relatively easy task, however it resulted in the existence of multiple dataset that could potentially reference the same entities, and quite often contain supplementing or offending information about a given record. This fact made our efforts into providing a technique for duplicate detection essential, in order to get better search results on the combined pool of datasets from the different sources.

The suggested algorithm, uses a custom implementation of the DBSCAN [28] algorithm to create clusters of points of interest based on their spatial proximity and their labels textual similarity, provided that they all share a non empty set of common categories. The natural partitioning of the data records based on their geographic location, also allows the algorithms to be executed in parallel, even take advantage of our browser based MapReduce model to speed up the process while crowdsourcing computing power to external nodes.

The result is a list of clusters, each one containing the individual suspected duplicate entries as seen in their respective source. By combining and fusing this information we can create richer descriptions of POIs, since different social media platforms focus on

different aspects of a point of interest. The algorithms suggested are described in detail in Section 4.4. A more interesting and challenging task is to handle cases where the information provided by different sources is not supplemental but contrasting. One would have to examine use cases of the resulting dataset, and trustworthiness scores of each source among others. This approach is mainly a direction for future work.

With the clustering process already in place, we can the easily change the grouping parameters, to look not for duplicates, but for entities that are in close proximity and belong to the same category, namely *regions of interest* (ROIs). This way, by using just one clustering method we are able to produce two different types of clusters by adjusting the distance function.

1.2.5 Event detection and ranking

The streaming nature of information coming from some of the social media platforms, along with the fact that first-hand witnesses are able to report events as they unfold, indicates that an intelligent system monitoring these streams will be able to map specific patterns in the data as events. Of course it would be essential to have the ability to filter out noise and outliers from the overall traffic in order to be able to track events more reliably, especially since as mentioned earlier this particular type of data feeds is extremely noisy.

Our focus in this work is identifying events geographically bound to a specific area, thus we monitor the social media traffic originating from within our area of interest. The process of event detection involves a number of independent steps. Usually these steps are modular, so implementations can substitute any number of steps to provide different types of event detection.

Firstly, we need to choose a piece of information present in the data or metadata that will describe the event. As an example, we use the hashtags that are present in each message. Hashtags are often emerging community-coined terms that allow people to coordinate and spread their messages easily by providing increased visibility to the individual posts that include them. The next step is to extract this information from each individual post and in turn create time-series for each one. Subsequently we feed the time-series data to an algorithm that will analyze and decide if the particular time-series corresponds to an event based on some parameters we define. The algorithms used here can be as simple as enforcing a threshold or more sophisticated such as regression analysis, etc. Finally the time-series that correspond to events are analyzed further to provide descriptive metadata for the events, such as start and finish time, absolute and relative magnitude, etc.

Once events have been identified for an area, our goal is to rank each event's spatial footprint, based on their overall extent outside of our area of interest. To this end we query the social media platform again for the event identifier, but this time without enforcing any geographical restriction. Based on the received sample we then try to classify the event as local, if most of the traffic for this particular event is indeed within our original area, or global if there exists substantial coverage of locations outside our original area.

Besides comparing how similar the overall extents are, we also perform a progressive comparison of timely advancing snapshots of the spatial footprints to identify if the event can also be classified as "transitional", meaning that it started as local but eventually achieved global coverage over time. Detailed description of the algorithms and the process is available in Section 6.

1.2.6 Hierarchical graph data structure

The hierarchical grid/cube data structure — briefly described earlier in 1.2.1 — we use throughout our analysis in this work works really well for fixed spatial binning, but does not support irregular shapes such as predefined administrative borders or custom polygons usually produced by convex hulling point clusters. Furthermore, every level of detail is kept separately, not allowing mixing of grid cells of different sizes.

Inspired by our work in identifying strong links in user movements, described in Section 7.3, we design a new data structure capable of linking arbitrary locations to areas identified by cells of a hierarchical grid. This structure resembles a "hierarchical graph", since it uses attributed edges to link nodes of different types and levels, such as grid cells of any level/size with random polygons. Random polygons can be either any type of predefined administrative regions, such as neighborhoods, zip code boundaries or cities, or even computed polygons such as Voronoi cells or the regions of interest described in Section 4.5. The nodes with grid cell information embedded into them, can use the underlying grid configuration to obtain information such as grid spacing and fanout, and perform clustering on neighboring nodes to increase the abstraction of the view.

To give additional value to the results, the hierarchical graph supports a new "bundle" operation that acts upon the sets of nodes that belong to the same cluster and replaces them with one node while preserving all the links (edges) attached to the replaced nodes. All links from a node n to any of the replaced nodes automatically have their metadata merged appropriately, as they converge to a single link, e.g. the new link is assigned a weight equal to the sum of all the converged links weights. A generic implementation of this graph in the Ruby programming language has been published freely to the open source community.

1.3 Outline

The outline of this work is as follows. Chapter 2 describes previous work related to all our contributions. It is divided into three major categories: (i) the type of data that we analyze, (ii) existing works on methods and algorithms used to analyze the data and (iii) different types of processing techniques that we can use for efficient computations. The next chapters describe the individual scientific contribution of this dissertation. As mentioned earlier several scenarios will be narrated, one in each chapter, where a problem will be described and a solution will be given using one or multiple social media feeds. Chapter 3 describes how Flickr images can be used to locate and describe geospatial features. We focus on single entity features, although the described methods can work on multiple-entity features as well. The described entities can be well defined placemarks or administrative regions, or areas with fuzzy outlines defined by the local population. Chapter 4 gives details of an effort to build a single POI gazetteer by combining data from multiple social media platforms and other VGI projects. We add value to the unified collection by eliminating duplicate entries across sources and also identify dense regions of interest of POIs of the same type. In Chapter 5 we present algorithms to efficiently compute and visualize summaries over large social media datasets. These summaries are essential to make sense of these datasets that can radically increase over time. Subsequently, we use a subset of them in Chapter 6 to detect and classify events, based on their spatial extent. Chapter 7 describes ways to construct networks and discover patterns in user movements, as these are captured through social media publishing activity. Each

chapter is accompanied by an extensive experimental evaluation section that showcases the efficiency of the methods proposed. To summarize, in Chapter 8 we discuss the extraction, analysis and visualization pattern by providing a generalized approach that can be applicable to many similar scenarios as they ones previously discussed. Finally, in Chapter 9 we briefly expand on how our work was used in two different research projects to (i) help with getting a better understanding of the urban landscape and (ii) to narrate the story of the health related outbreak. The dissertation concludes with Chapter 10 on lessons learnt and directions for future work.

Chapter 2

Related Work

In this chapter, we will introduce some of the definitions and basic notations that are used later on throughout the following chapters of this dissertation. Moreover, we will summarize the necessary information in terms of related work, relevant to our overall efforts.

In general we can divide the related work in three main categories. First we describe the type of datasets we focus on analyzing in this thesis, which all fall under the umbrella of user-generated content. Next we move on to the outcomes of the different types of analysis, i.e. what is the knowledge gained as a result of the analysis throughout the different scenarios we examine. Lastly we look into the algorithmic and technical aspect of the processing methods described, and discuss existing works and expanding by mentioning our contributions.

2.1 User generated content

Content from blogs, wikis, forums, social networking platforms and many more similar sources, that is created by non-expert people who voluntarily contribute data is usually described as *user generated content* or *UGC* [47, 59]. This content is highly heterogeneous since it gets published on different platforms, and is mostly unmoderated (platform dependent). In our work we focus on two particular types of UGC, namely *Volunteered Geographic Information (VGI)* from OpenStreetMap, Wikimapia, etc. and content from popular *social networking platforms* such as Flickr, Twitter, Instagram, etc. which has been attributed the term *Ambient Geographic Information (AGI)*.

2.1.1 Volunteered Geographic Information

The collective act of people contributing and submitted geographic information to a repository, regardless of their qualifications or incentive is called *Volunteered Geographic Information* [33]. Some of the enablers of VGI were the advent of Web 2.0 and the fast-paced evolution of web standards, as well as the increase in broadband internet access, faster and more capable computing devices, and easier means to geo-reference features, such as the GPS. Since the people that are contributing to VGI repositories such as Wikimapia, OpenStreetMap and Google Map Maker, range from neophytes to expert authorities, there are no guarantees for the quality of their efforts [30]. Furthermore the above classification can refer to a variety of attributes of the contributors, which might not always match, for example an expert in GIS software might have limited knowledge

about a specific geographic feature, while an amateur in using GPS devices might be very familiar with a specific area being mapped [18]. This "localness" aspect, has been thoroughly studied, and it has been reported that while usually someone would expect that people local to an area would be the ones that contribute geographic information about it [37], this assumption is also affected by language and socioeconomic status obstacles [73]. However, it is also true that usually collective efforts produce better estimates than that of a single person, even that of an expert [81]. In this context, users working together by subsequently improving each previous work, will most likely produce an acceptable result [35]. In a sense most of the flaws and errors in the data can be corrected by the community, as it evolves and users grow their trust values, by implicit or explicit rating systems [7].

2.1.2 Content from social networking platforms

In this work, we use the term *social media* to describe content published in different types of social networking platforms, either general purpose ones (Facebook, Google+), or others with a more specific focus, such as multimedia sharing (Flickr, Panoramio, Instagram, YouTube), microblogging (Twitter, Tumblr), check-in services and place gazetteers (Foursquare), professional networking (LinkedIn), etc... Specifically we are interested in all query-able content that has thematic (text/multimedia), temporal (date/timestamp), geospatial (location/place), and social (user) attributes, that we can extract and use for further analysis.

The data collected from these sources when processed and analyzed, allows us to gain valuable insight on the evolving human landscape, and support situational awareness as it relates to human activities [79]. People use these platforms to talk about their activities, share thoughts and concerns and seek or offer information. Interesting analysis shows how users interact and connect with each other to discuss similar interests [43]. As mentioned earlier regarding VGI, the information published has no general quality guarantees and the same applies to content shared in social networking sites. There have been several studies looking into the credibility of the information flowing through the social media sphere, especially under crisis situations such as natural disasters [58] or acts of terrorism [78]. These works report that false information is present but the network usually automatically "heals" as incorrect rumors get disputed more than confirmed facts, however that can occur with significant delay in some case. Aggregation techniques can be used to detect and filter out these rumors. An important aspect to pay attention to, is the role that individual accounts play in the information flow, especially the fact that not all nodes (user accounts) affect the flow in the same way. A few people tend to inspire trust and the content they share gets redistributed on the network in significant larger volumes than the rest. These opinion leaders have a key role in information dissemination and bringing attention to the things they communicate [16, 39].

In the following section we present a number of research topics that showcase the use of social media datasets, the type of analysis possible and the information and knowledge that can be obtained in each scenario.

2.2 Types of analysis

Numerous studies have used user-generated content as primary or complementing datasets to extract useful information and reach interesting conclusions. The relative low effort of

acquiring (effort is mostly based on technical skills) and it's low or absent cost make it a great source of information for many of today's problems. The evolving nature of these datasets make them also useful for studies comparing how certain aspects change over the course of time. Some of the different types of analysis we came across during our work, include:

2.2.1 Identifying geospatial features

As people move and travel through space, they tend to capture their most significant moments on camera. This activity assigns meaning and value to the locations that the capturing took place. Using the location information embedded on photos and videos shared on the web, can help identify, locate and describe several geospatial features. Flickr is a popular photo sharing social platform, and has attracted a lot of attention, due its large volume of available content, which summed up to over 5 billion photos by the end of 2014. Existing methods to aggregate metadata from public photos, help us learn not only geospatial concepts but also relations among them [41]. Other methods try to extract place semantics by associating tags of photos with the location where the photo taken, in order to assess if certain tags have strong associative mappings to specific places [68]. Our work in Chapter 3 uses similar concepts such as aggregating locations to eliminate outliers and describe the boundaries and the usage of places.

2.2.2 Integrating entities from different sources

Entity resolution and integration from different sources, which involves schema mapping (also referred to as ontology alignment) and entity matching (also referred to as entity deduplication or record linkage), has attracted a lot of interest from both researchers and practitioners for decades. Several surveys provide overview, categorization, and comparison of existing approaches ([4, 75, 29, 27]). The basic underlying techniques employed include tokenization and string similarity calculations (edit distance, n-grams, Jaccard) and structural operations, dealing with ontology hierarchies, graph similarity, etc. Other semantic and extensional techniques, involving rule-based inference and background knowledge, have also been applied.

However, few approaches have dealt with geospatial entities in particular. The authors of [83] suggest applying the structure-preserving semantic matching approach to facilitate semantic interoperability of GIS web services. In another work, a survey of semantic similarity measures for geospatial data has been presented in [72]. These measures are classified in geometric, feature, network, alignment and transformational models, and are compared with respect to their suitability for different tasks. A more recent work [62], suggests a novel approach to efficiently compute the Hausdorff distance, as a metric for the distance between geospatial objects defined generally as polygons to deduplicate geospatial feature collections. Block processing strategies have been proposed to significantly increase the efficiency of the entity matching task by restricting the required number of pairwise comparisons [65]. We expand on this idea, and use a category splitting and a cell splitting strategy in our method for identifying duplicate POIs, where only the POIs of the same category within the same cell are checked for matching. An efficient algorithm for spatio-textual similarity joins has been presented in [12]. This algorithm can be used to efficiently identify duplicate POIs between two sources, based on their distance and name similarity.

Finally besides just detecting the duplicate entities, techniques to merge attributes from the duplicate entries, are very important in order to build a richer and more valuable dataset. In that direction a weighted multi-attribute strategy for matching POIs from different sources is presented in [57]. The matching takes into consideration place categories, textual user reviews, as well as geographic distance to conflate entities between Foursquare and Yelp.

2.2.3 Geocoding and geoparsing

As mentioned earlier, the three attributes we usually focus on when dealing with social media data, is time, space and thematic information (textual or multimedia). While time and themes are usually always present, location information is usually absent. When collecting tweets for example, one can either specify a bounding box to collect all the traffic from within the bounds of the box, or specify a set of keywords to collect tweets that include the keywords in their text. Geotagged tweets can either have precise location information (embedded coordinates), or more ambiguous location information (embedded place) such as at a neighborhood or city level. In the first case of a *geo* search, tweets returned will either have exact coordinates within the specified box, or will have their place's bounding box intersecting with the bounding box we specified in the original query. Averaging the percentage of geotagged tweets with precise coordinates for all the searches we executed throughout all of this work using a geographic bounding box search, gives a result of approximately 10% (sample size of 70 million tweets). Repeating this for all the searches that keywords were used to collect the tweets, we get a much lower percentage of 1-2% (sample size of 25 million tweets).

This sparsity of geospatial metadata creates a bias on any kind of geospatial analysis. Due to Twitter's popularity and high volumes of data, several methods have been proposed to complement and infer location information for tweets that do not include it. For example, in [46] the authors suggest two approaches: the first one attempts to find a user's location based on the social connection he forms with other users with known locations (similar to [22]); the second one attempts to geolocate a tweet based either on its text alone using the query likelihood model from [49], or on a user's habit of tweeting more often from the same locations. Machine learning techniques that weight the information available in tweets metadata, such as place name information, user's location description and preferred timezone, etc. to correctly pick candidates from place gazetteers have also been implemented [91]. Similarly machine learning techniques which focus on specifically geoparsing content from social media microblogs which is particular in nature (due to the limited length), achieve higher accuracy than general purpose geoparsers [31]. This is mainly attributed to the high use of place references that are abbreviated, misspelled, or highly localized.

2.2.4 Detecting events in the social media stream

Extensive work has been done in the field of event detection using social media data. In most of these works the prevailing social network of choice is Twitter, since its Streaming API service is easy to connect to and ingest traffic as soon as it becomes available. Some of the early works focused on a particular topic, such as *earthquakes* [70, 25, 20], the *stock market* [10], or *news* [71, 39]. One idea is to filter the traffic based on specific keywords and phrases and see how they evolve over space and time. Another idea is to focus on a

number of manually chosen accounts that publish information about a specific topic, such as news reporters, meteorologists, epidemiologists etc. The issue with such approaches is that they cannot handle the case where the event type is not known in advance.

A different approach is to investigate discovering clusters of related words in tweets which correspond to events in progress. Towards that direction [40, 3] primarily focus on context, specifically on methods to distinguish tweets to event and non-event related ones. They both use a semi-automatic approach, which involves manually annotating clusters in training datasets, as opposed to our framework which requires no manual manipulation of the data. [40] also relies on manually selecting event broadcasters (Twitter accounts) for establishing a ground truth. A similar work, based on clustering of wavelet-based signals is presented in [89]. The method groups words into topics, that are derived by the temporal patterns of individual lexical tokens using wavelets. A major problem with such methods is that the word clustering step can be expensive when the number of words that appear very frequently is large. The use of term burstiness has also been suggested [55, 24], so that keywords that frequently cooccur will be indicative of events. However, these techniques are sensitive to popular terms and spammers who often use them in their tweets, in order to obtain higher visibility in the network [34], only making matters worse.

Finally a rather interesting suggestion is using notions from emotional theories together with geospatial and temporal information emotion theories [85, 84] to detect deviations in different sentiment levels that are indicative of events. The main idea is that users experiencing an event will fluctuate their emotional state, and this fluctuation can be captured and attributed back to event by examining the data records responsible. These techniques however are limited to the languages supported by the NLP tools used to infer the emotions.

2.3 Processing techniques

Several of the processing techniques and frameworks we use during our work are described next. The sheer amount and high velocity of the social media traffic that we usually have to deal with, mandates the use of efficient algorithms that run on reliable frameworks that promote parallelism, such as MapReduce. Browser-based computing tries to emulate such a framework by crowdsourcing tasks to the users of the service. Clustering reduces the projected data in ways that can be comprehensive by aggregating data based on similar values on attributes, and assists in building multiple hierarchical views of the same dataset.

2.3.1 Parallel processing

One of the most widely used processing framework that allows for parallel processing of large datasets with relative small effort is MapReduce [23]. The programmer's responsibility is to specify a *map* function that iterates through the documents in a collection and *emits* intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Using this functional style programs can be automatically parallelized and executed efficiently on a large cluster of commodity machines. All the details of partitioning the input, scheduling the execution and coordinating the communication between machines, plus handling failures, are taken care by the MapReduce framework at run-time. This way programmers with little or no experience on parallelization can utilize the resources of large distributed systems.

The MapReduce model is a good candidate to use when working with spatial data, as shown in [14, 86] where it outperforms the traditional computational approaches. In [14] an approach is proposed on bulk-construction of R-Trees through MapReduce. In [86] the authors describe how to efficiently answer ANN (All nearest neighbors) queries and astronomical cross-certification using MapReduce.

In our work we use the MapReduce framework either via our browser-based version described in detail in Section 3.5, or the implementation used in the MongoDB database for fast in-database operations and data manipulation, e.g. the nanocubes construction described in Section 5.7.

2.3.2 Browser based computing

The Web has proven to be an excellent channel for broadcasting research topics and results, but the viewers are usually passive and not involved in the (computation and/or refinement) process. We build our browser based computing framework in order to address this issue, and employ the crowdsourcing concept to the fullest, by means of exploiting not only the public's knowledge, but also their computing resources, inspired by the SETI @ home project [45] and the MapReduce programming model [23] described earlier.

There has been little work around the topic of using browsers as a computing platform, although as the popularity of the JavaScript language is gaining ground, it is most likely that will landscape will change. A major concern is the single-threaded nature of the JavaScript runtime environment, which is primary due to the fact that the Domain Object Model, used to represent a webpage in the browser's memory was never designed for unsynchronized access. So a single thread, is running the main event loop, and listening and responding to events from user and network input as they occur. Because of this long running and intensive operations cannot not be carried out by JavaScript programs, as this would block the main thread and events would not be executed before the previous operation was complete, resulting in unresponsive web interfaces and degraded user experience. However this behavior has been amended with the introduction of web workers [42], which can run in separate threads that do not block the main event loop, but with the limitation that the workers cannot directly access the DOM, but have to post messages to the main thread in order to alter it and display information on the screen. In [92] the authors propose the concept of using this new technology to offload workload on mobile devices by transferring computation and storage from the cloud to the device and vice-versa as needed. Similar to our own work, the authors in [26] are using a browser based system to preprocess road networks, in order to answer shortest path queries.

2.3.3 Aggregation and clustering

When dealing with large in volume and granularity datasets, it is essential to group, bin, or cluster the data along the studied dimension in order to make sense of it. Consider for example a timeline, where each individual record in a dataset is added to at its default granularity. If for example the records have timestamps with second or millisecond precision, then the timeline will most likely be flat, with single marks at the separate points in time that the records where created. In order to make sense of the temporal evolution of the data, we would have to group records together based on their hourly or daily values for example. The same principal applies to several other dimensions, especially the

spatial dimension that is thoroughly prevalent throughout this work. Spatial clustering will reveal areas of high activity on a map, but will also indicate important and useful links between the spatial objects, depending on the distance metric used. There has been extensive research on generic clustering methods and algorithms in the literature. Some of the most notable are mentioned here.

An approach to identify clusters using PAM (Partitioning around Medoids) [44] suggests k-clustering on *medoids* of datasets. While it works efficiently on small collections, it is extremely inefficient for larger ones. This is amended in CLARA (Clustering Large Applications) [44] by creating multiple samples of the original large collection, and then applying the original PAM method to the samples. An evolution of the previous approach is CLARANS (Clustering Large Applications Based on RANdomized Search) [61]. It is based on PAM and CLARA, but uses a dynamic samples for nodes at each iteration instead of just the beginning of the search.

In a different approach, density based algorithms such as DBSCAN (Density Based Spatial Clustering of Applications with Noise) [28] and DBCLASD (Distribution Based Clustering of Large Spatial Databases) [90] are widely used. DBSCAN uses the notion of direct reachability to form clusters based on density. DBCLASD is also density-based but assumes uniform distribution of records within each cluster. ST-DBSCAN (Spatial-Temporal DBSCAN) [6] is an extension of DBSCAN that also has inherent support for spatial, temporal and spatiotemporal datasets.

Grid based algorithms have also been suggested such as CLIQUE (Clustering In QUEst) [2] which provides automatic sub-spacing that scales in high dimensional collections. MAFIA (Merging of Adaptive Finite Intervals (And is more than a CLIQUE)) [32] builds upon CLIQUE and provides faster runtimes by using adaptive grids (the interval size of the partitioning of the grid is dynamic). An interesting approach is taken by WaveCluster in [74] where the dataset initially goes through quantization process and subsequently a discrete wavelet transformation is applied to convert space to the frequency domain. Clusters are then identified as dense regions in the transformed domain.

Inspired by the grid based algorithms, in our work we employ a simpler approach with a fixed size grid and cluster spatiotemporal data simply based on containment. We find the use of this simple clustering technique efficient for the purposes of our analysis, but are keen on trying out more complex clustering techniques, as the ones previously mentioned, in order to assess if the offer improvement in our results. This is left as future work.

Chapter 3

Locating and describing geospatial features

Traditional place gazetteers tend to describe the locations of geospatial features with a pair of coordinates, that usually corresponds to their center or the most significant region within its bounds. Additionally, there might be information about the boundaries of the feature, more often in the form of a bounding box and less often as a more descriptive (multi)polygon. Little or no information though exists about the distribution of people's interest within the feature bounds. Our work focuses on providing not only the location and the boundaries, but also a more comprehensive description of landmarks, places, neighborhoods or cities, by using social media feeds to create a heatmap of popularity within the extent of each feature.

This chapter's content was initially included in [50] which was presented in the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems in 2011, and later on in [51] which was presented in the 20th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems in 2012.

3.1 Introduction

The ever-increasing number of Web and mobile applications producing geospatial data has led to a large number of user-contributed geospatial datasets. Users generate data by means of many different applications in which the geospatial aspect does not play a major role, but is simply used to index and access the collected data. Prominent examples that all include geotagging capabilities, are photo sharing sites such as Flickr, Panoramio and Instagram, micro blogging services such as Facebook, Twitter and Google+, and more recently check-in services, like Foursquare's Swarm.

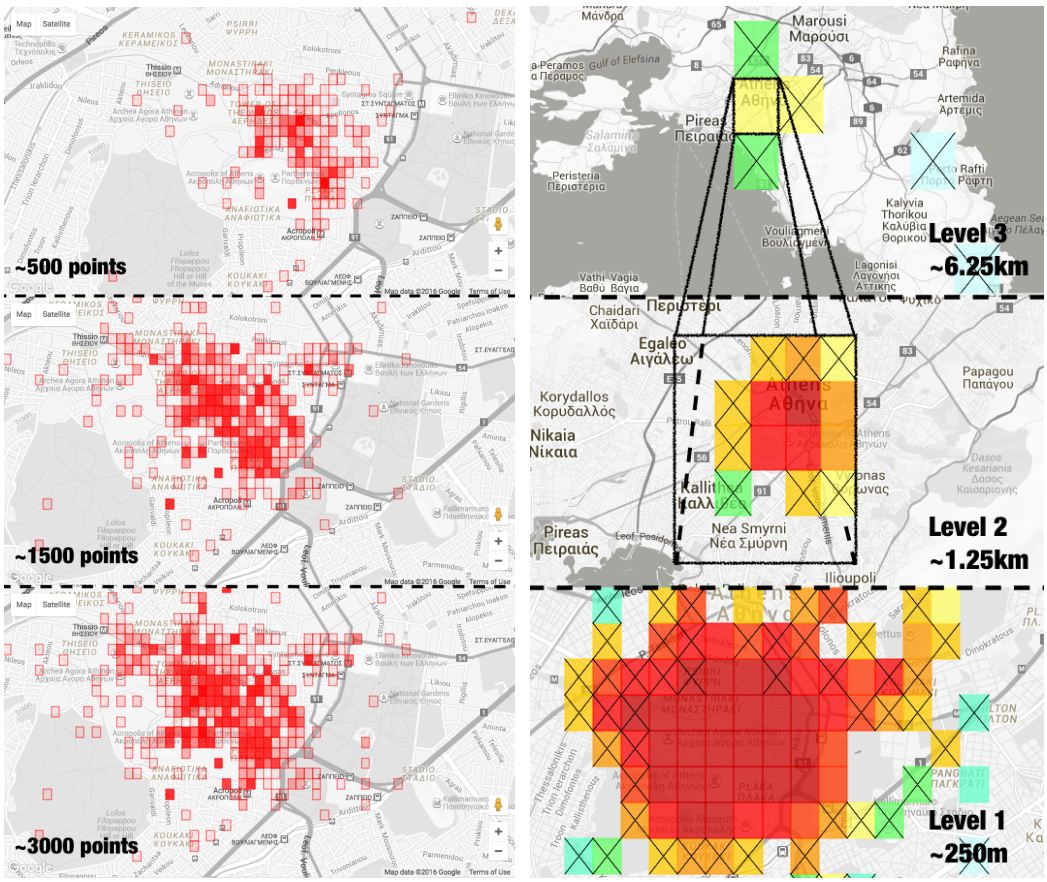
Our objective is to transform the available user generated geo-content into meaningful datasets, that can be efficiently queried to provide useful descriptions for the searched geospatial features. Specifically, we use the embedded location information in geotagged Flickr photos to derive the extent of certain places, such as landmarks (point features), neighborhoods or parks (small area features), but also cities and much larger areas. The particularity of our approach is that we involve the users that execute the queries in the actual query answering process, by exposing our computational framework as a web application. The core contributions of this chapter are (i) a collaborative spatial search algorithm, backed by (ii) a hierarchical grid data structure, embedded in a web application utilizing (iii) a browser-based MapReduce approach [23].

3.2 From estimating to refining locations

Flickr is a photo sharing web platform, where one can upload photos and attach useful metadata such as textual description or location information. If the uploaded photo already contains location EXIF metadata (e.g. it is taken using a camera equipped with a GPS sensor), the service will use that instead, since it's usually more accurate, at least for outdoor takes. A subset of these photos, are made publicly available by their owners, and the Flickr platform exposes them through their API.

In the following we describe "Jeocrowd", the algorithm and tool we use to collect, process and visualize the geospatial feature results. Our input dataset is obtained by querying the Flickr API, with a set of keywords describing a POI. We modify the request to include only geotagged photos. The response is a collection of photos referring to the specific POI, each one associated a latitude and a longitude value. Essentially, the geotagging of each photo represents an independent observation of the "true" location of the POI. The basic task at hand is how to derive a meaningful spatial description for the POI from this user-contributed point cloud data that have basically no quality guarantee. A typical approach would entail the *clustering of the points* (cf. [19]) in combination with outlier detection. The latter should eliminate points that have been assigned false coordinate information. This method would involve computing a distance matrix for all pairs of points in our input dataset, which can be computationally expensive ($O(n^2)$). A *grid-based clustering approach* (cf. [87], [61], [28]) will prove to be computationally more efficient. As a grid with fixed size cells overlays the entire area, each individual point is mapped to the grid cell that encloses it. This mapping significantly decreases the size of the dataset. The degree δ of a cell, tracks the number of points mapped to it. This initial list of grid cells forms the base level of the hierarchical grid data structure (described in more detail in Section 3.3) we use to store the point information. Additional upper (less detailed) levels can be built by recursively combining existing grid cells to form bigger ones. The parameters needed to construct the grid are the fanout n and the grid spacing at the lowest level a. For example, with $n = 5, a = 50m$, we generate a grid that at the lowest level (Level 0 or base level) consists of $50m \times 50m$ cells, at the level above (Level 1) of $250m \times 250m$ cells, etc. At every level (except the base one), a cell will enclose at most n^2 cells of the level below. By using fixed size grid cells, we can use simple grid arithmetics to traverse the grid and to compute various properties such as neighbor density and distances between cells (L_1 or Manhattan distance [8]). With the base grid in place, performing a keyword search for a POI is a matter of intelligently populating the cells with the location metadata embedded in the photos retrieved. Using various thresholds for their degree δ in connection with adjacency information to discard less important cells, we are left with a set of cells whose spatial extent will be the extent of the searched spatial feature (POI). The typical result will be a (multi)polygon with the accuracy of the shape being limited by the spacing of the grid. Due to the hierarchy of the grid, we can have several (multi)polygons, one for each available zoom level.

Since dealing with remote data sources, our goal is to *retrieve as little data as possible* in order to speed up the search and reduce network traffic. To this effect we split our search into an *exploratory* phase, which tries to quickly find a good estimate of the extent of the spatial feature and a *refinement* phase, which then tries to explore the shape of the spatial feature in greater detail. As shown in Figure 3.4, the hierarchical grid is used (i) to expand the search area after the exploratory phase and (ii) to eliminate unlikely areas during the refinement stage.



(a) Exploratory phase (b) Refinement phase

Figure 3.1: *Searching for Plaka, Athens*

Exploratory phase. During this phase the system retrieves some initial points matching the search terms to get an overview of the spatial extent of the spatial feature, i.e., is it a point (statue) or multi-point (coffee shops) location, or are we looking for an area feature (neighborhood, city, country). These initial points are used to construct the base grid. Each point is assigned to a grid cell according to its coordinates. The total count of points (photos) within a cell is called degree δ of the cell. During this phase we aim at collecting a representative but small in size point cloud relevant to our search terms. Hence when using Flickr as our point cloud provider, the data is requested and retrieved sorted first by relevance and then by descending interestingness. These two sorting criteria are parameters of the Flickr API that return the most relevant dataset in this case. In the current implementation, in this phase we collect up to 4000 unique points (photos). Figure 3.1(a) shows the evolution of the base level of the exploratory phase for the "Plaka" neighborhood in Athens, Greece, as it collects more and more point data. Specifically a density map is shown, where each tile's fill color opacity is indicative of the number of points it attracted.

Refinement phase. During the refinement phase, the search is forced into areas that are suspected of being part of the feature, but for which we have not retrieved (many) points during the exploratory phase. Users tend to take disproportionally many pictures at the most popular locations. The refinement phase uses the search term in connection with location estimates determined by the exploratory phase to retrieve additional photos for these "unpopular areas".

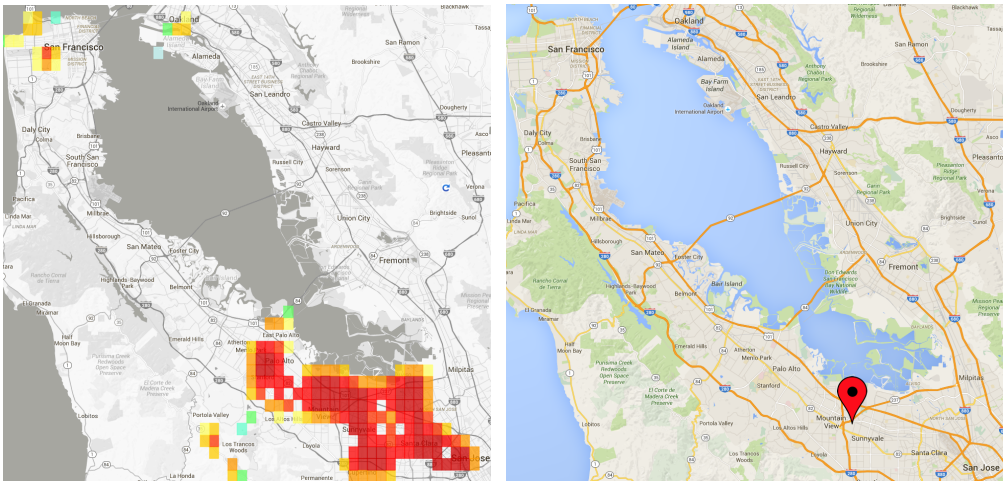
The phase begins by taking the base level of the grid as input, and starts building the upper levels by combining cells into bigger ones, until a "bounding (multi)polygon" enclosing the most relevant cells is created. This denotes the overall focus area that will be later *refined*, hence the name of this phase. Every time a grid at level L_i is built, not all the cells of level L_{i-1} are used. Intelligent heuristics based on the distribution of the degrees and the adjacency density can be used to each time eliminate less relevant cells as the aggregation recurses. Generally, this indicates that a cell at L_i that originated by just one cell at L_{i-1} might not be as important as a cell that was formed by combining multiple smaller cells. However, in this specific problem this is not as important, because each participating cell's validity is independently verified once the top level of the grid is built. The function that implements the logic to decide whether a cell at level L_i should exist, based on the cells it encloses at level L_{i-1} is called the *aggregation* criterion. Based on some heuristic formula that uses a threshold θ for the number of smaller enclosed cells and the cells degrees and adjacency density, it outputs a boolean value to denote whether L_i should include that cell or not. The recursion will stop when a grid at a certain level does not contain any cells. In cases where the *aggregation* criterion is weak, meaning there is no way for the recursion to stop producing higher and higher levels, a *termination* criterion is introduced to upper bound the top level and stop the recursion. So, essentially, the termination criterion will provide the L_{max} that will be the highest level of the hierarchical grid that will be built. For example, if the aggregation criterion allows each bigger cell to populate by having even at least one smaller cell enclosed, then each individual cell would grow indefinitely. Here a termination criterion is needed, for example to allow only a predefined or pre-calculated number of grid levels. In our experiments (Section 3.6) we use $\theta = 2$ and build the hierarchical grid up to the level L_{max} where the grid has at least one cell remaining. With $\theta = 2$ the aggregation criterion is not weak, and thus the recursion eventually stops at level L_{max}. We then use the value of L_{max} as the termination criterion, while we rerun the aggregation from the base level, having switched to $\theta = 1$ for the aggregation function.

When searching for *large-scale spatial objects*, the construction of the hierarchical grid might prematurely terminate, i.e., the estimated extent is too small due to sparse point clouds. In those cases, we want to allow the grid to grow beyond what is estimated. To accommodate for such scenarios, after computing the maximum level and grid, we use the following rule. We measure the distance between the centers of the h cells with the highest degrees. If their average distance is larger than s times the length of the edge of a cell for that maximum level then the grid is considered *sparse* and an additional level is added to the hierarchical grid. Parameters h and s can be adjusted by the user.

The top level of the hierarchical grid is a very rough estimate of the POI's location and spatial extent. In the case of the example where we searched for the "Plaka" neighborhood in Athens mentioned earlier, this is shown as the black rectangle in Figure 3.1(b). At this point we leverage the hierarchical grid structure again, to work our way back to the lower levels, verifying each cell's actual participation by querying the Flickr API to receive the total number of available photos within each individual cell's bounds. Each cell at level L_{max} is split (dissected) using the same spacing parameters, into $n \times n$ hollow cells at level L_{max-1}. These hollow cells are then sequentially searched to retrieve the relevant to our search keywords photos that were capture inside their bounds. Since we do not care about the actual photos but just the total count inside each bounding box, we only make one request to the Flickr API and read the total count value embedded in the response, saving both bandwidth and time. Figure 3.1(b) shows the transition from the upper levels to the

lower ones for our example search. Specifically a neighborhood density map is shown, in which colors from blue to red are assigned to cells based on how many neighboring cells have also collected point cloud data. Isolated cells are colored blue and surrounded cells are shaded red. This process naturally stops when reaching the base level L_0, but can also terminate earlier depending on the nature of the POI we are searching for, or the grid extent discover so far. For example, if we know we are searching for a landmark, or small neighborhood we would generally want to be as detailed as possible and continue the refinement until L_0. On the other hand, when we are looking for larger features such as cities, parks or national formations we might want to stop at an earlier acceptable level of detail. When the type of the POI searched is not known before hand, a threshold on the so far discovered extent (grid area coverage, number of cells, etc...) can be used.

Since each time we go down a level, the number of cells in the grid increases quadratically, it is useful to think of intelligent heuristics to reduce the actual number of these smaller cells that will be checked. Using a hierarchical grid, the goal is to rule out cells as early as possible, i.e., at high levels, since examining larger areas at lower levels is costly (many small cells). We propose a *filtering* process which includes two distinct steps that take place before the split of a level into smaller cells. Firstly, we discard the less important cells (outliers) based on the distribution of cells' connectedness and distribution of their degrees. Let $\overline{\delta(h)_{L=i}}$ be the average of the first h cells of a grid at level i in descending degree order. The *filtering* function uses the following formula to determine if a cell is discarded: (i) it is isolated (has 0 neighbors) and its degree is less than $0.5 \times \overline{\delta(h)_{L=i}}$, or, (ii) it is not isolated and its degree is less than $k \times \overline{\delta(h)_{L=i}}$. The remaining cells are then dissected to smaller cells, and form the level at L_{i-1} that is in turn searched as well. Secondly, we detect *core cells* that are completely surrounded by populated neighboring cells, i.e. all the 8 cells in their range 1 Moore neighborhood exist [88]. These *core cells* will not be dissected at lower levels as they represent significant areas of the spatial object discovered early on. Thus both filtering steps discover cells that need not checked further, the difference being the first process prunes these cells from the result, while the later integrates them in the result.



(a) Jeocrowd result (b) Google Maps result

Figure 3.2: *Comparing the "Silicon Valley" area result between Jeocrowd and Google Maps.*

Searching for the keywords "Silicon Valley" yields the results shown in Figure 3.2(a), in contrast with the result we get when we search on Google Maps in Figure 3.2(b) for the

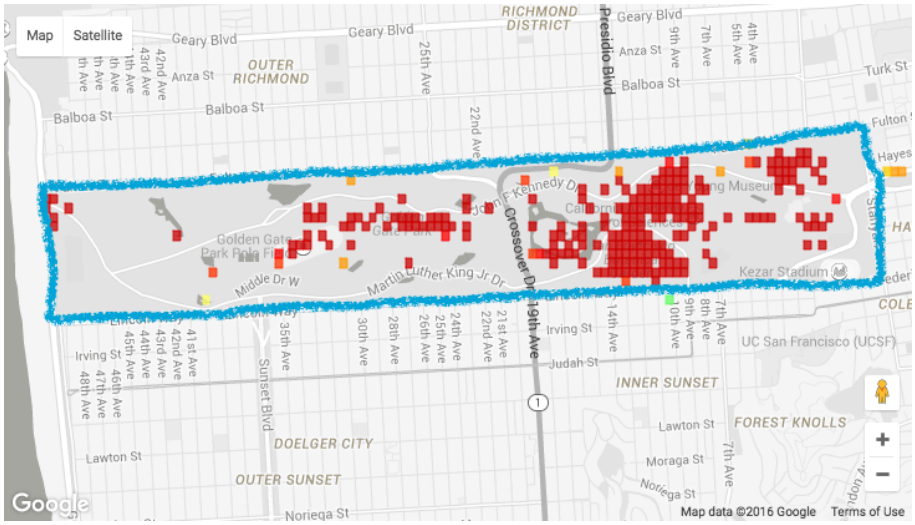


Figure 3.3: *Search result for Golden Gate park.*

same term. This comparison makes evident the value added by this work, as we not only get an estimate of the location, but also of the bounds of a feature. While the value added by describing the bounds is evident in this case, one might argue that there are other cases where the exact bounds are already well-known. In this cases our system also adds value by detecting the actual "hotspot" areas, as they get reported by the social media traffic (i.e. density of Flickr photos) at these locations. This representation is shown for the example of the Golden Gate park in San Francisco in Figure 3.3, where we see that while the park boundaries are well defined, the importance that people actually give to the certain areas inside the park is not uniform at all.

3.3 The hierarchical grid data structure

The data structure used to store the cells information is a hierarchical grid, where the hierarchical factor modifies the level of detail used to represent the same dataset. While the data is always available at the finest available granularity (base level), it is often useful to be able to aggregate it into larger bins to reduce the size of distinct values (higher levels), thus enable faster processing and visualization, especially at zoomed-out levels. The data structure is visually drawn in Figure 3.4.

This grid is a 2D matrix that stores info only for the spatial dimension, and it has fixed size rectangular cells. The cells size is measured in degrees of latitude and longitude, so even if their size is fixed in the measured degrees it actually varies in reality since the length of a degree of longitude greatly varies as we travel along a meridian and the length of a degree of latitude varies less when traveling across parallels. However around a small area and especially when using a small spacing the variation is quite small.

While each grid at level L_i can be represented by a 2D matrix, that poses several challenges with referencing individual cells with respect to their geographic coordinates and then positioning them on a map. For example, a 2D array where the element $c[0][0]$ correspond to a cell referenced at the geographic coordinates $(0°, 0°)$ makes sense but that would require extra manipulation for decimal degrees (i.e. using a multiplier to make all coordinate values integers so they can be mapped to integer array indices), or additional mapping in programming languages that do not support negative array indices to accom-

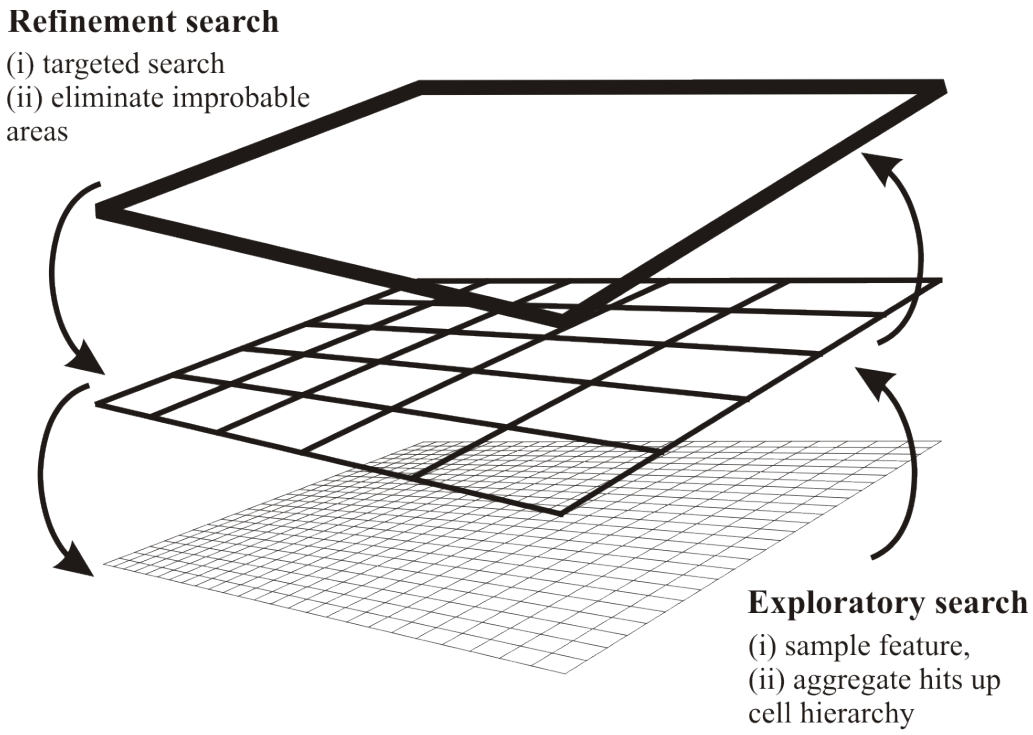


Figure 3.4: *The hierarchical grid data structure.*

modate for locations with negative latitude or longitude. Since in most cases our data is focused on a specific area, we could use a bounding box enclosing all the data points as a reference system and construct our array indices within these limits. But this solution also requires an extra mapping, since we are positioning $c[0][0]$ to a random coordinate pair, plus it would require reprocessing all the mappings if new data forces the bounding box to grow outside its original bounds. What is more, an efficient array structure would also require additional mapping functions for inter-level cell relationships, such as finding all cells indices of L_i that are enclosed by a specific cell at $c[x][y]$ in L_{i+1}.

To address this issues, and also save on space and memory we use a hash-map (dictionary) to store each level of the hierarchical grid, where the keys are the concatenated coordinates of the lower left corner of each cell, joined with a char separator, e.g an underscore '_'. For example a cell near the Acropolis in Athens at (37.97N, 23.73E) will have an id of `'23.73_37.97'`. Here we use the same format for coordinates as in the GeoJSON [13] specification, which places the longitude value first followed by the latitude value. This way, a cell's unique identifier also indicates its position on a map. To actually draw the rectangle on the map we also need to know the grid's spacing a parameter, which is given in degrees. Then by using the formula: `[x+`a`, y+`a`]` we compute the upper right coordinates and draw the rectangle.

The choice of the coordinates of the lower left corner of the cell as its unique identifier is important, as it allows us to compute a random point's cell id by adjusting its coordinates to the precision allowed by the grid spacing, using the *floor* function. A similar formula will be used to compute a cell's parent id, meaning the id of the cell in the level above that encloses a cell, by just adjusting its coordinates to the precision allowed by the upper grid spacing. As explained earlier if n is the fanout of the grid and $a = a_0$ is the spacing of the base level (L_0), a grid at level L_i will have a spacing of $a_i = a_0 \times n^i$. Thus the following formula will give the coordinates of the lower left corner of the cell at level L_i that encloses any point at (x, y) (or any other cell of level L_j, $j < i$ with its lower left

corner at (x, y)).

$$\left(\left\lfloor \frac{x}{a_i} \right\rfloor \cdot a_i, \left\lfloor \frac{y}{a_i} \right\rfloor \cdot a_i\right) = \left(\left\lfloor \frac{x}{a_0 \cdot n^i} \right\rfloor \cdot a_0 \cdot n^i, \left\lfloor \frac{y}{a_0 \cdot n^i} \right\rfloor \cdot a_0 \cdot n^i\right) \tag{3.1}$$

In a similar fashion we can create simple formulas to identify neighbor cells (Moore neighborhood [88] of range 1, i.e. each of the 8 adjacent cells in the same grid level), children cells (all the cell ids of all the enclosed cells on the level below), ancestor and descendants cells, or calculate distances between cells (L_1 or Manhattan distance [8]).

3.4 Outsourcing tasks to clients

Search is a typical task performed by web users. Our goal is to improve the task of retrieving the results of a search by combining efforts from many users that execute similar queries. To facilitate collaborative search, one not only needs to design a search strategy and respective data structures, but also an efficient computation framework. A framework around this notion can provide not just a query and retrieval service for users, but also involve them (and their resources) in the process as much as possible. The computational framework we built for this case, is exposed via a web application, transforming its users (clients) to computing nodes, as it uses their web browsers to perform calculations that help with formulating the result of the search they themselves initiate, following the MapReduce programming model.

Over time web technologies have evolved to give web developers the ability to create a new generation of useful and immersive web experiences. Today's web is a result of the ongoing efforts of an open web community that helps define these web technologies, like HTML5 and CSS3 for structuring and presenting content and APIs such as WebGL, localStorage, WebWorkers and indexedDB (former WebSQL) for interactivity. The web browser is the software that enables users to consume all these technologies, independently of the devices they are using. Of course not all browsers on all platforms support all the latest available APIs, but as the web moves to standardization the new capabilities are implemented in latest versions, and publishers ensure that they're supported in their web browsers. With that in mind, using the web browser as a computing unit yields enormous potential. There are several issues that should be dealt with of course, such as security, integrity, fault tolerance, etc. However if we are able to overcome these issues, or judge that for a specific application some of them are irrelevant, building a system based on browsers as computing units could achieve huge scalability. The key component that enabled browsers to display interactive and dynamic content is the use of a scripting language, JavaScript. The JavaScript programming language was first developed and used in the Netscape Navigator browser (which is now discontinued as of 2008). Although it was primarily designed for DOM manipulation inside web pages, it has since evolved, and with the addition of AJAX technologies [66] now allows the creation of interactive web applications that can communicate with external resources.

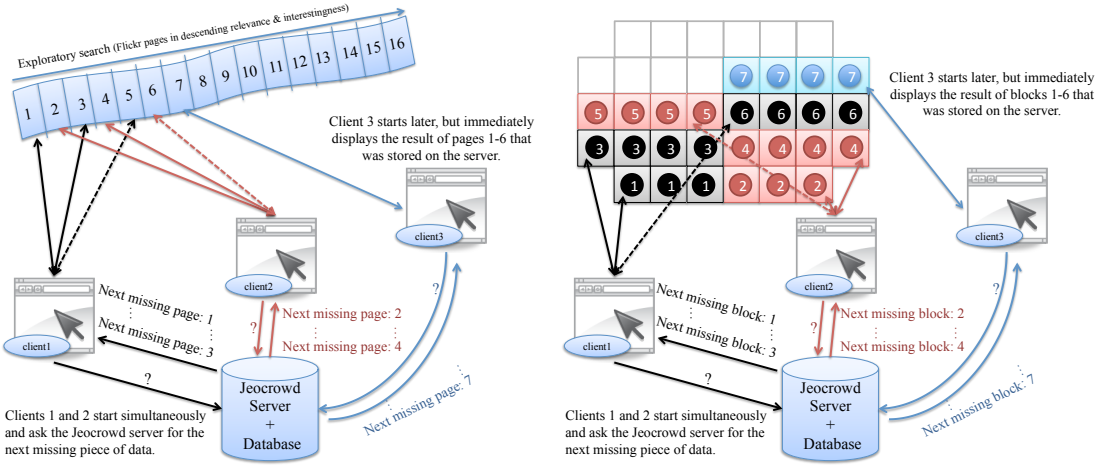
In this work we design, implement and test a browser-based computing platform that simulates the MapReduce workflow, using a central server as the master worker and users' browsers as computing nodes. A difference to the original MapReduce model is that due to the browser-based nature of the system, the task assignment uses a *pull* rather than a *push* mechanism. In conventional implementations a master worker assigns and pushes tasks to a list of available machines. Here each client (web browser) requests a new task

once it connects to the service. The server is only used for coordinating tasks and storing the results. All other computation takes place in the browser. This has many advantages, the main one being the increase in computational capacity, as it allows horizontal scaling out by just having more users to connect to the system. Since we are dealing with remote datasets that need to be obtained from an external source prior to processing, the specific problem we are trying to solve can benefit additionally from the outsourcing of the work to the clients, since each client can be responsible not only for the processing but also for the downloading of a portion of the dataset. This is extremely beneficial, since usually remote APIs will enforce max rate limits to their usage per IP or per client id. By allowing the users to download the data from their own machines using their own client id lifts this restriction and allows the system to obtain more data faster.

The basic work flow of the exploratory and the refinement search applying our browser-based MapReduce paradigm is shown in Figures 3.5(a) and 3.5(b) and is described in more detail in the following Section 3.5.

3.5 Browser-based MapReduce

The MapReduce programming model [23] can be used to allow parallel processing of large amounts of data among several computing nodes, by splitting the input dataset into non-overlapping parts and assigning each part to a specific node. The node is then responsible for *mapping* (applying the `map` function to) the part of the input dataset it received and *emit* (key, value) pairs. Each document in the input dataset can emit 0 or more pairs. These pairs are placed in an intermediate temporary storage and then sorted by key. The *reduce* part will then feed all pairs with the same key to a `reduce` function, usually responsible for accumulating all the different values a specific key has, into a combined object (for example sum up all the values).



(a) Exploratory phase

(b) Refinement phase

Figure 3.5: *Browser-based MapReduce during the two phases of the search.*

Now that we have a rough understanding of how the programming model works, we discuss the specific tasks we need to implement to solve our problem. The first step is to identify a way to parallelize the search process. Since as explained earlier we split our

search into an exploratory and a refinement phase, we will examine these two phases separately. For the *exploratory phase* the data is requested by an explicit page number after applying specific sorting criteria. Each page contains a fixed number of point locations. We do not need to retrieve actual images, but we need each photo's unique identifier in order to calculate accurate counts per grid cell when later building the grid. This separation into pages provides a way to partition the task of collecting the photo metadata from Flickr. Each page can be assigned to a client, i.e., a web browser connected to the service, which will query the Flickr API for any points matching the search term and then submit the results to the server. In the *refinement phase* the hierarchical grid provides the needed partitioning of the data, which in this case is a spatial partitioning. To parallelize this part of the process, each client is assigned a set of cells, then uses their bounding boxes as a filter to query the Flickr API and retrieve the total count of available photos inside each one. The count is reported back to the server. Figures 3.5(a) and 3.5(b) show how the tasks are split in each phase, and how they are assigned to the different clients.

The mechanism of task assignment and coordination is handled by the server that hosts the application. For persistent storage we use a no-SQL document database [15] that supports atomic in-place updates, e.g., MongoDB [17]. Our design aims at providing a scalable architecture with the ability to *resume tasks* from their last saved state by losing the minimum amount of work possible. To persist the state of a search, we record all the information needed to resume it, on the server. This includes which pages and blocks have already been retrieved and which need to be searched next. Once a new client connects to the service, regardless if it's the only one searching for a specific term, or if there are others as well, receives the latest snapshot of the search and is assigned the next available task. Depending on the phase of the search a different approach is used.

Task assignment. During the *exploratory* step we download photo ids with their coordinate information. We need to keep the id information to avoid counting duplicate points when calculating the counters to build the hierarchical grid, as it is possible to receive the same photo more than once, due to Flickr's architecture and cache-validation policies (different pages might be requested from different servers who might not be entirely in sync). The task assignment works as follows. There are two arrays with 16 slots each. The *scheduling array* is used for coordinating the clients and the *results array* is used for storing the actual results. Each array matches its index to the page number of the results fetched. To distinguish which client is responsible for which page we use millisecond timestamps. When a request arrives from a client, the server appends a new timestamp to the scheduling array and sends back the array and the timestamp to the client. The client reads the timestamp value, finds its index in the array and requests the page with the that index from the Flickr API. Once the result has been retrieved, the client sends the data back to server where it gets saved in the results array. At this point the timestamp that rests at the specified index in the scheduling array is replaced with the index value itself, thus marking that specific page as complete. In the exploratory phase only the base grid (at level 0) is stored in the database. Every other level can be computed and displayed on demand, e.g., for visualization purposes.

The *refinement* step just requests the total count for each bounding box specified by each cell, rather than actually downloading all the available metadata and counting it. For each level of the hierarchical grid, the search terminates after we have acquired the counts for all cells. The task assignment is this phase is accomplished as follows. Three arrays are used for each level. The first two are used to coordinate the client jobs; the *assignments array* is used to hold all the cell ids that are to be checked, and the *missing*

array contains all the cell ids for which results have not arrived at the server yet. The last one, the *results array*, holds the actual results as {cell id, count} tuples. This workflow helps in reducing (i) update racing conditions and (ii) the overlap in the data stored by the different clients. In terms of implementation detail, this is achieved by using atomic in-place modifiers that MongoDB offers when manipulating the arrays. Every time we want to assign a new block of cells to a client for processing, we first read all the cell identifiers, then explicitly pull designated identifiers from the *assignments array* and send these identifiers to the client. The atomicity offered by the database, guarantees that any request after that will not find the same identifiers in the array, thus it will be forced to assign a different set of cell identifiers to the next client. The client then queries the Flickr API for the total count of photos contained in each cell's bounding box. Once its assigned block of cells has all been checked, the client submits the results back to the server. The cells are appended to the result array, and the cells' ids are pulled from the *missing array*. When the *missing array* is empty, the search of this specific cell level is considered complete and the filtering and dissection process proceed to the level below.

Fault tolerance. To prevent the search from missing results due to clients failure to report back (browser closed or crashed, network connection interrupted, etc.) the system has a mechanism that allows for rescheduling a task to another client, if the original client did not report the results on time. During the exploratory phase, the system imposes a *timeout* on the clients in order to finish fetching, converting and finally submitting back the results. The time-out parameter is by default set to $t = 15sec$. This is why we use timestamps in the scheduling array. The time-out mechanism is only activated after the scheduling array is full, meaning that all 16 pages have at least one initiated corresponding task. Clients that will have to be assigned a task once the array is full have two options: if there is a timestamp in the array that is less than t seconds ago, that space in the array is re-assigned a new timestamp and a new task is scheduled; if there is no timestamp meeting the previous criterion then the clients just waits for t seconds and then requests the scheduling array again from the server in the hope that either all assigned exploratory tasks have completed and the search progressed to refinement phase, or that it can itself be assigned an overdue task.

In the refinement phase the fault-tolerance mechanism is integrated in the scheduling process described earlier in this section. When a level "runs out" of cell ids for assignment to clients, i.e. the *assigning array* is empty but the *missing array* is not, this means that some blocks were assigned to clients that have not yet report results back. In this scenario the *assignments array* is re-initialized with the data from the *missing array* and the search process resumes until the *missing array* is empty. The results table might be filled with the same result more than one time but that is insignificant because this duplication will be eliminated during the merging process, which collects all results in a hash-map; an entry with the same key (cell id) will always be considered only once (latest value).

Constraints and limitations. A couple of dependencies between the different steps of our algorithm can introduce delays and stall the process until all previous executions have completed. One constraint is that all exploratory subtasks for a search term have to be completed before proceeding to the refinement phase. Another one mentioned earlier as well, is that in the refinement phase all cells of a specific level have to be checked before progressing to the level below. This is because after each level is completely checked, a *filtering process* (core cells detection and outlier pruning) runs that does not allow some cells to be dissected and thus checked further. If a task were to start processing the lower level before the filtering process had completed, it might have started querying a cell that

would eventually be discarded, or marked as core, thus wasting much more time, since the number of cells increases quadratically each time we decrease a level. Hence, if all the tasks for a specific search step have been assigned, but not all completed, a new task can be created for an already assigned portion of the data with the hope that it might be able to finish earlier that the previously assigned task. We always use the latest value that has been reported to the system at a given time.

Overall, these techniques for coordinating, scheduling, and handling exceptions, are very common in various MapReduce implementations.

3.6 Experimental evaluation

The objective of the following experiments is to establish the performance of the collaborative spatial search algorithm in terms of (i) time it takes to compute a result and (ii) the data exchanged with the external provider(s). The performance will be evaluated for a varying number of search terms, representing four categories based on the anticipated spatial extent of the result:

1. point features such as “Eiffel tower”, “Statue of Liberty”, etc...
2. small areas such as “Central Park”, “Golden Gate Park”, etc..
3. cities such as “London”, “Athens”, etc...
4. large-scale areas such as “Alps”, “Grand Canyon”, etc...

Our experiments should establish the performance of the spatial search algorithm by measuring the *speedup* we can achieve when using multiple browsers to simultaneously search for the same term. In addition, we will measure the total *amount of data* exchanged between the browsers and the external provider and compare how this is reduced by applying the specific cell-based search strategy introduced in the refinement phase.

The experiments conducted will include six search terms for each of the aforementioned four spatial extent categories. Since the search is parametric, a set of configuration options can be provided to dictate its behavior. These settings are grouped in *profiles* and can then be re-used for similar searches. Each category will have its own profile (configuration settings), optimized to give good results for the selected spatial extent. The global parameters and the adjustable parameters assigned to each profile are summarized in Tables 3.1 and 3.2. The values were established through experimental evaluation or by making basic assumptions. In addition, the optimal refinement phase parameters as shown in Table 3.2 depend on the size of the spatial feature in question. For each one of these sets, we will then run the search 5 times, increasing the number of clients used by a factor of 2.

To obtain the following experimental results, the web application was installed on a 64-bit Ubuntu server virtual machine with QEMU Virtual CPU clocked at 2.3GHz with 4 cores, and with a total RAM capacity of 4GB. The web application is a Ruby on Rails 3.1 backed up application, using the jQuery 1.7 Javascript framework and the database used is a MongoDB 2.0 server. The use of multiple browsers is simulated by creating a script that would automatically launch the number of browsers required. For simplicity, all the browsers run on the same machine. The browser of choice was Google Chrome. What follows are the results of our performance experiments that detail the *cost attributed to the various searches in terms of transferred data and execution time*.

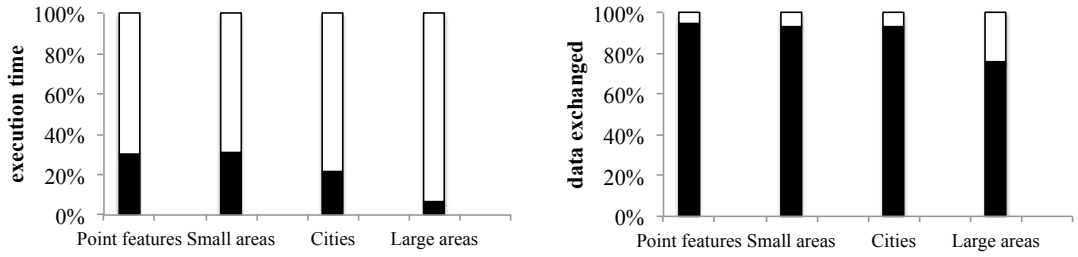
Parameter	Explanation
$n = 5$	grid fanout in each dimension
$a = 0.001° \approx 50m$	grid spacing at level 0
$\theta = 1$	threshold for aggregating to higher level
L_{max}	maximum level that survives when $\theta = 2$
h	number of cells with the highest degrees
s	minimum Manhattan distance for sparse cells
k	discarding isolated cell percentage threshold

Table 3.1: *Search parameter settings and overview*

Parameter	point	small area	city	large area
h	4	10	5	5
s	10	10	15	20
k	0.25	0.10	0.05	0.01

Table 3.2: *Values of adjustable search parameters*

In the exploratory phase a rough position estimate is established that is subsequently refined using a hierarchical grid-based search. What follows is an overview of the search cost for each case, respectively.



(a) Time distribution

(b) Data distribution

Figure 3.6: *Time and data distribution between each phase (black: exploratory, white: refinement) of the search split by category.*

Figure 3.6(a) shows the percentage of *time* spent for the varying types of spatial objects split by search type. As the time spent on the exploratory phase is the same for all searches (ignoring network delays and possible timeouts for requests to APIs of the data source), the *refinement step largely determines the overall search time*. The key parameter of the refinement phase is the hierarchical grid configuration for a specific search determined by (i) the starting level (top), (ii) the finishing level (bottom), and (iii) the spatial extent of the object(s) being searched. The starting level is determined by the outcome of the exploratory phase, i.e. the base grid that defines the rough estimate of the spatial extent of the object. The starting level of the refinement phase is calculated by the routine that aggregates the base grid into coarser higher level grid cells as explained in Section 3.2. The finishing level is selected based on pre-existing knowledge with respect to the size of the object. In terms of *search time* using one browser, the exploratory phase takes about 1min to complete, i.e., 16 requests taking approximately 4sec each. The refinement phase can take from a couple of minutes up to almost half an hour for some very large scale objects; an average refinement request request takes about 300ms.

The *data* retrieved for each search case is shown in Figure 3.6(b). While a considerably smaller search time is attributed to the exploratory phase, the respective data retrieved amounts to 95% of all data retrieved from the point-cloud data source. This is due to different nature of the data being fetched during these phases. In the exploratory phase we fetch 250 photo attributes per request, including their unique id, their location information, and the url to the actual photo stored on the servers, while during refinement this information is fetched essentially for only one photograph plus the total count for the specified bounding box. In terms of size, responses to exploratory request are usually $\approx$ 90KB, while a refinement request fetches only $\approx$ 20bytes of data from the Web data source. This results on average in data sizes of $\approx$ 1.4MB for the exploratory step and $\approx$ 25KB for the refinement steps. Table 3.3 summarizes the absolute times spent and data exchanged for four example cases. It is evident from the table that the costly part (time) of the algorithm is indeed the refinement phase. Nevertheless, this step retrieves very little data compared to the exploratory step.

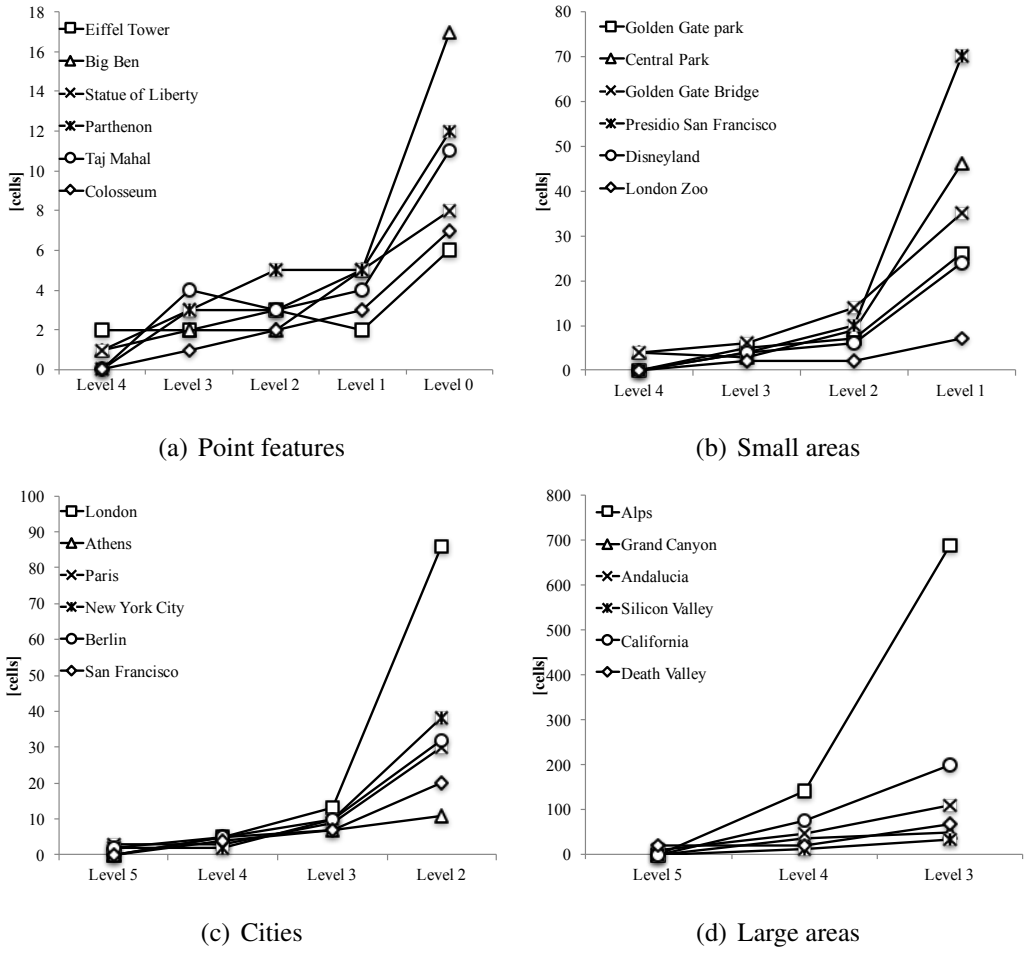
Search query	Time (mm:ss)		Data (bytes)	
	expl.	refin.	expl.	refin.
Eiffel Tower	00:56	01:39	1.40MB	9KB
Disneyland	01:05	02:13	1.42MB	10KB
London	01:14	05:51	1.42MB	36KB
California	01:02	23:51	1.43MB	101KB

Table 3.3: *Time and data values for example search queries, one browser*

Refining the positional estimate is the time-wise costly part of our geospatial search. Some intuition is given in Figure 3.7. The figure plots the number of visited cells per hierarchy level as the refinement phase progresses, ranging in each case from the starting level (the level the exploratory step identifies as the starting point for the refinement step) to the desired finishing level. Each subsequent level in the hierarchical grid grows in terms of number of cells visited. This is of course expected as we move from coarser levels to more fine-grained ones to better define the object's spatial extent. Using a 5×5 grid ($n = 5$), one might expect that the number of visited cells increases with each step by 25. We attempt to early reduce this number with the help of the *filtering function* and the concept of *core cells* as already described in Section 3.2. Using these two techniques, the increase in number of cells by progressing to the next levels is kept to an average of 2 - 4 (instead of 25) (cf. Figure 3.7).

An interesting metric is the percentage of cells that ends up being eliminated. Two variations are shown in Table 3.4. The first is simply computing the average percentage of the cells discarded at each level. The second approach calculates this percentage in a cumulative way by figuring out the number of cells discarded compared to the number of cells the grid would have if its highest level were to be dissected directly into base grid cells. This is cumulative in the sense that discarded cells of one level count also as n^2 discarded cells of the level below and so on, where n^2 is the amount of cells of level L_{i-1} enclosed in one cell of level L_i.

One question that still needs to be answered is whether our search is an actual improvement over the simplest way of computing spatial positions, namely retrieving all location data that is stored on the external providers for a specific search term. Our experiments clearly show that our algorithm performs not only faster, but also uses a lot less data when compared to the exhaustive approach. Figure 3.8 shows the results of that



(a) Point features (b) Small areas

(c) Cities (d) Large areas

Figure 3.7: *Line graphs that show how each hierarchical grid level's size tends to grow as the search progresses split by category.*

comparison. Both the speedup in the execution time and the decrease in data downloaded are very significant. Table 3.5 gives the actual percentages. The dominant case observed is the "Cities" category. Here, both the search time speedup and the decrease in downloaded data is very high. This abnormality can be explained primarily by the fact that "city-type" searches have a very high number of total available photos since city names are an expressive tag for photo descriptions. Thus, when just searching with a city name as a keyword one can expect that there will be a vast number of results and that most probably this search will include results from the first two categories - namely point features and small areas - that can be found in the specified city. What is more, cities do usually have a large enough spatial extent to allow for core cells to be identified, thus giving an additional performance advantage over searches in other categories. Large areas tend to include several isolated places of interest rather than being one large connected feature, i.e., disfavoring core cells.

The proposed method allows for the parallelization of search, i.e., the more users search for a spatial feature at the same time, the faster the spatial extent of the feature is computed. To measure the speedup in search, we performed experiments with a varying numbers of browsers searching concurrently. To conduct this experiment, a script runs each search term subsequently, increasing each time the number of browsers used by a factor of two. Since the script was running on one single machine we limited the number

	Point	Small areas	Cities	Large areas
Average	92.2%	88.3%	90.9%	71.7%
Cumulative average	99.6%	98.2%	98.8%	81.5%

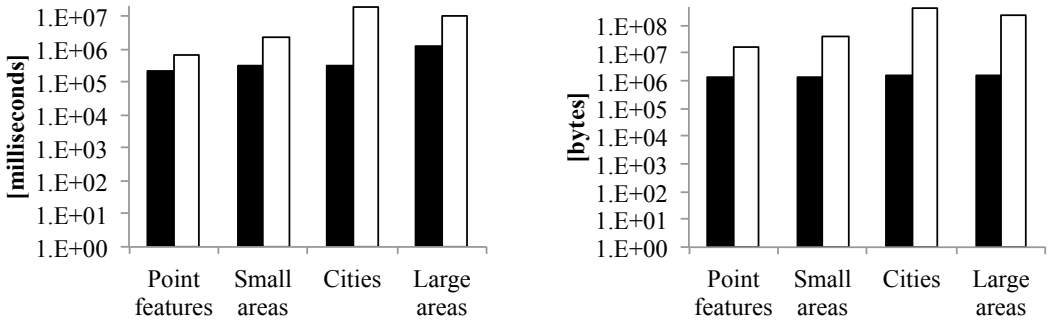
Table 3.4: *Percentage of cells that got discarded and eventually not searched thus improving the speed and reducing the data downloaded by the algorithm.*

	Point	Small areas	Cities	Large areas
Time	29.32%	12.26%	1.65%	14.45%
Data	8.46%	3.70%	0.33%	0.71%

Table 3.5: *Time taken and data downloaded as a percentage of the values the simplest way of fetching all data via exploratory search would achieve.*

of browsers (concurrent searches) to a maximum of 16. We expect the speedup to be more significant for cities and larger areas rather than smaller ones, because of the synchronization points in the algorithm (when switching from exploratory to refinement, and when descending each level during refinement) and because of the increased availability of assignable tasks. For the exploratory phase the speedup will be the same for all categories (same number of pages is fetched in all cases). The refinement phase performance however will be affected by the search category, and different speedups are expected since the spatial extent of the object plays an important role in the parallelization of the algorithm. For larger areas, a large number of blocks is available for assignment to clients for simultaneous processing before reaching the synchronization point of progressing to the next (lower) level. The average speedup measured per category can be seen in Figure 3.9. The speedup is not very significant for point features, since a lot of overlap exists between assigned search tasks. However in the other categories of larger scale objects, each browser will be assigned a fair share of the total area. Hence, all the execution runtimes for small area, city-scale, and large-area features decreases significantly with the number of browsers participating in the search achieving a speedup up to $5 - 8$ times faster when using 16 browsers.

Experimentation showed that the proposed method is a very efficient way for searching and discovering geospatial objects in user-contributed point cloud data. It provides a



(a) Time speedup

(b) Data gain

Figure 3.8: *Comparison of our algorithm (black) against the naive baseline (white) of fetching all data via exploratory search, both time-wise and data-wise.*

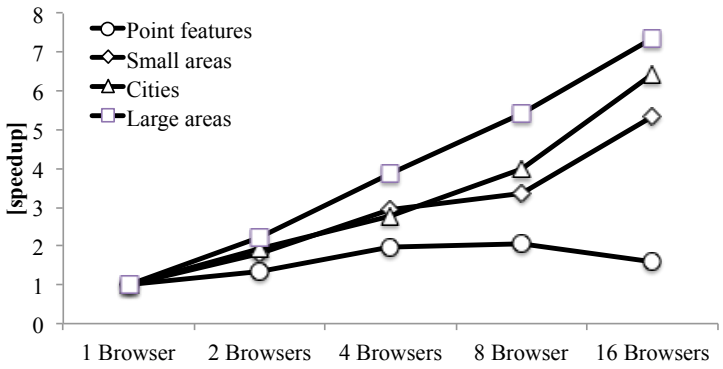


Figure 3.9: *Speedup when using multiple browsers simultaneously*

way to minimize, both, computation time and the data fetched from the Web sources. The ability to parallelize the search further decreases the amount of time required and further improves the search performance.

3.7 Conclusion and Future Work

In this chapter, we presented a complete system for locating and describing geospatial features using non-authoritative data sources, such as publicly available Flickr photos. We introduced the hierarchical grid data structure as a very efficient way to store point data and group their locations to grid cells to decrease processing time and improve visualization clarity. The hierarchical grid is also used as means to quickly establish a position estimate, and to direct the search in order to refine the geometry of the spatial feature in question. Experiments establish the efficiency of the approach, both, in terms of search time and the amount of data that needs to be retrieved to provide an accurate position estimate.

Directions for future work include designing a comprehensive mechanism to evaluate the quality of the derived data and to provide quality guarantees. The quality and coverage can be improved by including further data sources, such as different social media platforms (i.e Instagram), or user-contributed datasets such as OpenStreetMap. Additionally, we designed, implemented and tested a browser-based computing platform, that follows some aspects of the MapReduce programming model, in order to acquire and process the data. We intend to extract this component from the system and make it more modular and generic, so that it can be used in other applications as well.

These goals are partly covered in the next chapter, where we tackle algorithms to build a system that can integrate datasets of points of interest, derived from multiple sources. We also use our browser-based computing platform module, and load it with additional processing capabilities, such as cluster discovery, for this new combined dataset.

Chapter 4

Integration of geospatial content from multiple sources

In recent years we have seen a tremendous increase in location based services, primarily since the smartphone revolution. Many of these services often require auxiliary datasets, to enrich and enhance their offered geospatial capabilities, and so often turn to the constantly increasing pool of user-generated geo-content from social media platforms as it is available at very low or no cost at all. Examples include POI gazetteers used to suggest nearby places to visit (with included suggestions and ranking by friends), road network data used to provide navigation instructions (with included incident reporting by other users), or simple place name datasets used to allow direct or reverse geocoding.

However, this content is highly heterogeneous and diverse, varying significantly in quality and accuracy, often forcing application designers and system architects to chose and implicitly trust one of the available sources, even though their solution could benefit more by combining multiple sources. The goal of this work is to retrieve, then process and integrate crowdsourced Points of Interest (POIs) from multiple popular Web sources into a single unified dataset. To build this unified collection, we migrate the individual entries to a common schema, detect and fuse duplicates, and apply additional processing to provide added value such as identifying clusters of common types of points of interest. Eventually the goal is that the resulting fused dataset and the computed metadata will be exposed via a single endpoint allowing other users and systems to use the combined information.

This chapter's content is mostly based on the following two publications: [53] accepted in the 3rd ACM SIGSPATIAL International Workshop on Crowdsourced and Volunteered Geographic Information and [54] accepted in the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems both held in parallel in 2014.

4.1 Introduction

The constantly increasing participation of users in social networking sites and in efforts and initiatives for open and collaborative data publishing has led to an explosion of crowdsourced geospatial data on the Web. For instance, Wikipedia includes, among others, information about thousands of places and Points of Interest (POIs). Inspired by its success, OpenStreetMap and Wikimapia have emerged as collaborative mapping projects, both having a user base that exceeds a million users, resulting in a large pool of crowd-

sourced geospatial data. An abundance of geospatial entities can also be obtained from various other services (e.g. Google Places), from user check-ins in social networks (e.g. Foursquare), from Web sites providing information about events (e.g. Eventful), etc. This new wealth of sources and content opens up new opportunities and challenges for improving, enriching and enhancing applications and services in the geospatial domain, such as location-based services or trip planning.

However, harvesting, processing, and analyzing this content from multiple, disparate, and heterogeneous sources is a challenging process. Even acquiring the data in the first place is not trivial. Although many of these popular sources make it possible to access, query, and retrieve (part of) their data via some structured means, such as a RESTful API, each one provides its own methods, schemas, and formats. Moreover, apart from the different licensing issues that accompany the acquisition and use of these data, in most cases there are also technical restrictions imposed on the number of requests allowed within a time interval, as well as the maximum number of results that are returned per request. All these differences, restrictions, and peculiarities need to be taken into consideration and addressed, even before starting to synthesize such multi-source data.

Subsequently, once the raw data are retrieved, an integration process is needed to acquire a cleaner and usable dataset, and to gain further insights about the main locations of POIs of certain types within a larger area. Different schemas and category hierarchies are used by different sources, while the same real-world entity may often be found in multiple sources with different names and representations, and with overlapping or even conflicting information. Reducing heterogeneity and detecting duplicates, are both necessary steps before making data available for browsing, search, analysis, and visualization.

The contribution of this work is a methodology and detailed workflow, to collect geospatial content from multiple sources, map it into a meaningful common schema, clean the resulting dataset by fusing or removing duplicates, plus further analysis and processing to reveal areas of interest; then finally make the data available for browsing, querying, and visualization.

4.2 Collecting content from multiple sources

The first step towards our goal of building a unified dataset of geospatial content is actually obtaining the content from the many different sources that are available. Since our goal is to create a generic dataset that will be used in many applications, we wanted a rich set of different types of sources. We chose to collect data from 10 different platforms that fall into the following categories:

- photo sharing (Flickr, Panoramio)
- microblogging (Twitter)
- place gazetteers (OSM, Google Places, DBpedia, wikimapia, Foursquare)
- event information (Eventful, LastFM)

To make things easier we do not intent to download the entire content each platform provides. Instead we want to focus our effort in specific geographic areas of interest, such as cities that rank high as tourist attractions. This will guarantee a plethora of available user-generated content.

While studying each source in order to design an efficient way to access the respective datasets, we look into the available methods they offer for data retrieval. An important

Figure 4.1: *Our unified dataset is integrating content from all the sources above.*

thing to notice here are the terms of use specified by each platform for collecting and using its data, especially when the purpose of use can be commercialized. However this is outside of the scope of our work. In general, the following main cases for obtaining data can be distinguished:

- *Data dumps*. This refers to cases where a source provides links to file downloads that contain either entire or parts of the dataset. Based on the file format, the data can then be loaded or imported into another system for further processing. Such examples are file downloads provided by DBpedia[1] or OpenStreetMap[2].
- *Query endpoints*. This refers to cases where a source provides an interface for issuing queries to retrieve data. Most often the interface is an HTTP RESTful API, e.g. Wikimapia[3], Twitter[4], etc. and less often a SPARQL endpoint, e.g. LinkedGeoData[5].
- *Web scraping*. This refers to cases where the source does not provide a structured way to access the data; in such cases, one can resort to fetching and parsing the HTML code of each page and trying to extract the relevant information based on its contents and structure. This involves behavior similar to web spiders that follow links within pages to crawl entire websites. Clearly, this is more cumbersome and less reliable.

In the case of *data dumps* there is no complexity during data acquisition, except maybe if the files to be downloaded are very large and difficult to store or process, in which case we might have to resort to other services[6] that can produce smaller extracts of the original files, usually by including data bounded by a specified region and time period.

In the case of *Web scraping* there are two main considerations: the first is the access frequency and overall strategy for obtaining the web pages content from the source. Typically web services and servers will enforce limitations on how often a resource can be requested from the same IP address, or HTTP session. An automated script that downloads web pages would either need to simulate the speed of a human browsing the content

[1] `http://wiki.dbpedia.org/Downloads`
[2] `http://planet.openstreetmap.org`
[3] `http://wikimapia.org/api`
[4] `http://dev.twitter.com`
[5] `http://linkedgeodata.org/sparql`
[6] `http://download.geofabrik.de,http://extract.bbbike.org/`

by introducing random delays between requests, or use some sort of distributed architecture that would make the requests originate from multiple locations, such as use a pool of proxy servers, or even employ a browser based distributed system as the one we described in the previous Section 3.4 and Section 3.5. The second is of course the task of breaking down the structure of the HTML page itself and figuring out ways to extract relevant data out of it. This task almost always requires human involvement to examine the structure of a sample of web pages and infer where the relevant information is stored. Then by usually using HTML or XML parsers and referencing nodes with XPath (XML Path Language) or CSS (Cascading Style Sheets) selectors, scripts can extract information out of the web pages and store it in a more structured way.

Lastly, in the most common case where the data is available through a *query endpoint* we face similar limitations on the number of requests that we are allowed to issue per time window or account, plus a limit on the number of results for each request. However, in contrast to the case of *Web scraping*, SPARQL and REST APIs provide us with a structured way to retrieve content, meaning we can devise a much more elegant strategy to obtain the part of the dataset we need. Since our focus is on geospatial content, we can use the spatial dimension to direct our retrieval to locations we need data for. However, if a single query is issued to retrieve POIs for a very large region, it is very likely that the returned result will contain only a small portion of the actually available POIs in that source. Pagination can be used where available, but most times due to technical limitations, it will still not allow access to the entire dataset. The large size of the dataset prohibits efficient access when requesting fixed size batches of records with sequential offset. For an optimized solution, one must simulate pagination by applying filters that can be answered using indexed fields, such as *time* (creation or modification dates) or *space* (point coordinates or bounding boxes). For the latter, one possible way is to use a sufficiently small sliding spatial window to progressively cover the whole area. This effectively means dissecting an entire area by a rectangular grid with a fixed cell size. However, given that not all parts of an area are equally dense w.r.t. the number of POIs they contain, using cells of fixed size is not optimal. Instead, a more effective approach is to split the input region into a hierarchical grid (essentially, a quadtree), and then recursively retrieve data for each cell. For efficiency and simplicity, we use the same hierarchical grid described in the previous Section 3.3 and shown visually in Figure 4.2.

Depending on each data source's technical restrictions and limitations we need to define several configuration parameters for each external provider. $edge_{MAX}$ is defined as the maximum allowed edge of a bounding box that the external service will return valid results for. In case where the service issues radial and not bounding box queries the $edge_{MAX}$ parameter is transformed accordingly, to include the circle with the maximum possible radius. $edge_{MIN}$ is typically set by our system as a termination criterion for the hierarchical grid dissection process. $data_{MAX}$ is set to the maximum number of results the provider can return for a specific bounding box. Given a bounding box b, $|b|$ is defined as the length of the largest edge of the envelope specified by b.

The collection process repeats the following steps for each of the external sources (that provide a query endpoint) chosen: Initially, the given area is split into four quadrants. Each quadrant is inserted into a queue, to be processed by available worker processes. When a worker dequeues a bounding box b, if $|b| > edge_{MAX}$, b is again split into four quadrants which are re-inserted into the queue; otherwise, the corresponding download client is triggered to fetch data from the respective source. The amount of data gathered ($b.dataCount$) is then compared to the maximum number the provider can return

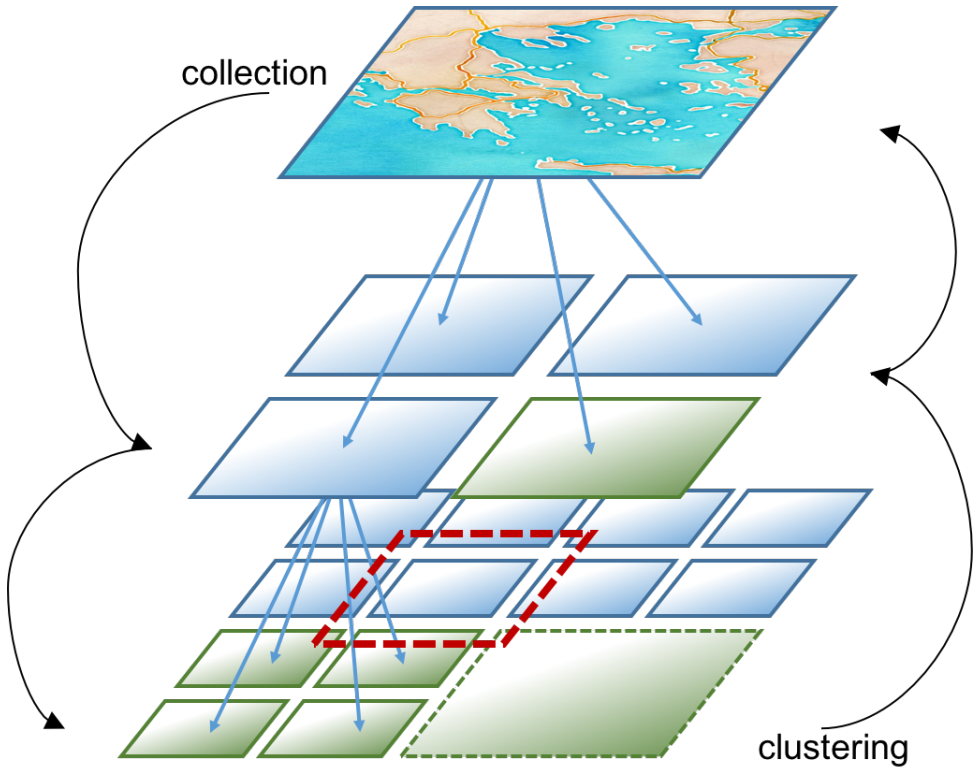


Figure 4.2: *Using the hierarchical grid to accommodate the collection and clustering process. During the clustering each node is expanded by ε (red dashed node) so as not to miss points in adjacent nodes.*

($data_{MAX}$) for that type of query. If $b.dataCount < data_{MAX}$ the bounding box is marked as "completed" (cf. the green rectangle on the lower right corner on the second level in Figure 4.2); otherwise, the splitting process is once more repeated. Sometimes multiple requests need to be made to collect all the data available within b. An improvement here is to use a separate "count" request if available by the provider to immediately get an estimate of the total available POIs within b, namely $data_{EST}$. If $data_{EST} > data_{MAX}$ then we do not need to start collecting all that data from b, as it is certain that we will collect more that the max $data_{MAX}$, but we immediately split b into its quadrants and perform the actual collection there. To avoid endlessly dissecting bounding boxes into extremely small quadrants, the split stops if $|b|$ drops below the specified threshold $edge_{MIN}$. The collection process completes when the queue is empty.

4.3 Homogenizing categories

Once the raw POI data have been collected from all the different sources, an integration process is needed to map all the data to a common schema, to make further analysis easier. Different sources may provide different kinds of information about a POI (e.g., ratings, comments, photos), based on the type and scope of each source. Naturally, this common schema will be the union of all the relevant fields we want to store for each POI entry in our unified dataset. This means that several fields will be empty for the data collected from a source that did not provide this set of fields, and other fields might be merged to provide similar functionality, e.g. a ratings field might be merged with a "favorite" or "likes" counter if we can map their values to a common scale. Nevertheless, one basic attribute

that is typically provided by all sources is the type or category of the POI, (e.g., restaurant, shop, monument). The type of the POI is particularly useful, since it determines for which applications or queries the POI is relevant. It can also be used, for example, to discover areas of interest with high density of POIs of a particular category. However, the problem when dealing with data from multiple, heterogeneous sources is that each source typically employs its own categorization schema, often without relying on a formal ontology but rather on user-created taxonomies that are constructed bottom-up. Hence, a challenge that arises is to map categories from various sources to a common category hierarchy, so that POIs from different sources can be compared and integrated.

Initially, we need to have the category hierarchy that we will use for our unified dataset throughout the rest of the analysis. This category hierarchy can be devised from scratch, chosen from one the sources, or modify an existing one as we did in this work. The schema we use is based on the category hierarchy employed by Foursquare, since this was found to have a broader scope and higher variety of POI types compared to the other sources. To make this categorization comprehensive, we have restricted it to up to three levels of depth. The top level categories are: *Entertainment* (e.g. music venue, nightlife spot, movie theater), *Culture* (e.g. opera house, art gallery, museum), *Education* (e.g. college, university, library), Food (e.g. restaurant, coffee shop, breakfast spot), *Places* (e.g. beach, forest, lake), *Shops* (e.g. souvenirs, bookstore, clothing store), *Services* (e.g. postbox, ATM, bike rental), *Professional* (e.g. company, office, convention center), *Travel and Transport* (e.g. airport, subway station, hotel), *Athletics and Sports* (e.g. gym, stadium, tennis court), and *Religion* (e.g. temple, shrine, church).

Given that a very extensive literature exists dealing with the problem of schema and ontology matching and alignment [75, 29, 4], we decided to leverage existing techniques to deal with the problem rather than to propose a new method for discovering schema mappings. In particular, we have used the S-Match semantic matching framework [76]. S-Match comes in the form of a Java library providing several semantic matching algorithms. Given any two graph-like structures, such as classifications, ontologies, database or XML schemas, the matching operator identifies those nodes in the two structures which semantically correspond to one another. S-Match identifies several pairs of candidate mappings based on various rules for matching and on external resources, such as WordNet[7].

Since we are only interested in selecting a single mapping for each source category to a category in the common categorization schema, we apply a post-process filtering step to rank the suggested mappings based on textual similarity. Specifically, we use the normalized Levenshtein distance [56] to compute the edit distance between the original category name and each of the candidate mappings, then choose the one that achieved the maximum score (smallest edit distance). The normalized Levenshtein distance is computed as:

$$\texttt{normalizedLevenshtein}(s_1, s_2) = 1 - \frac{\texttt{levenshtein}(s_1, s_2)}{\texttt{max}(|s_1|, |s_2|)} \tag{4.1}$$

These mappings are then used to assign each retrieved POI to one or more categories in our reference schema. From this point on, whenever a lookup for the category of a POI is performed the newly mapped category is returned instead of the original category description the POI had in its source's dataset.

[7] http://wordnet.princeton.edu/

4.4 Duplicate detection and POI fusion

Dealing with multiple sources that quite often try to create collections for the same entities, inevitably leads to duplicate results appearing in more than one sources, perhaps with different, complementary or sometimes even conflicting representations. Such problems have been widely studied in the literature, under the terms *entity matching*, *record linkage*, *data deduplication*, etc. [27, 65] as mentioned in Chapter 2.

Finding matching entities across sources, or even within the same source, is important for various reasons. First, it is often the case that different sources provide complementary information. Thus, identifying matching entities allows to build a richer and more complete entity profile. For example, one could collect photos of an entity from one source, while collecting comments and ratings from another. Second, it can help to detect, and potentially correct, errors in cases where conflicting information for the same entity is found in various sources. Automatic correction can be achieved when fusion rules are available, based on an assessment of the quality and trustworthiness of each source (e.g., "always prefer values obtained from Wikipedia over other sources") or based on a voting scheme (e.g., "keep the value provided by most sources"). Third, when other users or applications are searching and consuming the available content from our unified collection, it is desirable to avoid duplicates in the results, since such redundancy and repetition of information will likely deteriorate the user experience and might cause frustration.

The challenge in this task, similar to the case of reconciling different category hierarchies, arises from the fact that there are no unique, common identifiers for the entities appearing in the various sources. Moreover, the name of an entity may appear with slight variations or misspellings in different sources. Thus, the typical approach is to define some measure of similarity between entities and then consider as matches the cases where this similarity exceeds a specified threshold.

In our case, since we are dealing with geospatial entities, we can exploit both the spatial and the semantic information to identify duplicates. More specifically, we assume that two collected POIs P_i and P_j correspond to the same real-world POI if the following conditions hold:

- the coordinates of P_i and P_j are not more than a specified threshold ε apart
- the names of P_i and P_j have a similarity score higher than a specified threshold σ
- P_i and P_j have at least one common top level category.

Performing a pairwise comparison of all POIs in a large region has prohibitively large computation time. Instead, taking advantage of the first criterion mentioned above, we can exploit spatial proximity to significantly reduce the number of pairwise comparisons required. The process is outlined in Algorithm 1. The basic idea is to process POIs by splitting the bounding box of the original area recursively into smaller cells, similarly to the case of the POI retrieval process (cf. Figure 4.2), and checking for matches only within cells where the number of contained POIs does not exceed a specified threshold K (Alg. 1 lines 3 – 9). Each time a match is found, either a new set to store the duplicate entities is created or if one of the entities already belongs to set, that set is extended to include the other entity (Alg. 1 line 6). In order not to miss matches spanning across adjacent cells before loading the POIs of a cell, we expand its borders by ε (Alg. 1 line 2), to ensure that any matching POIs of the neighboring cells are included in the comparisons. For each processed cell, the conditions stated above are checked to identify matches (Alg. 1 lines 20 – 22). For computing the name similarity of POIs, we use the normalized Levenshtein

distance (edit distance) on their names. The advantage of the method outlined above is that it allows for the process to be parallelized in a rather straightforward manner. The processing of each cell can be assigned to a different process and be executed in parallel, with simply computing the union of the results at the end. It is also possible to optimize the processing in each cell; instead of performing pairwise comparisons (line 4), the efficient algorithm for spatio-textual similarity joins presented in [12] can be applied.

ALGORITHM 1: Outline of process `matchPOIs()`

Input: The bounding box B defining the area for which POIs are to be matched. The parameters ε, σ, K.

Output: A set $\mathcal{M}$ containing pairs of matched POIs.

```
1  Function matchPOIs(B, ε, σ, K)
2      P ← loadPOIs(expandBbox(B, ε))
3      if |P| ≤ K then
4          foreach P_i, P_j ∈ P do
5              if isDuplicate?(P_i, P_j, σ) then
6                  M_k ← extendOrCreateSetWith(P_i, P_j)
7                  M ← M ∪ M_k
8              end
9          end
10     else
11         [B_sw, B_se, B_ne, B_nw] ← quadSplit(B)
12         foreach B_i ∈ [B_sw, B_se, B_ne, B_nw] do
13             M_i ← matchPOIs(B_i, ε, σ, K)
14             M ← M ∪ M_i
15         end
16     end
17     return M
18 end

19 Function isDuplicate?(P_i, P_j, σ)
20     return |P_i.lon − P_j.lon| ≤ ε and |P_i.lat − P_j.lat| ≤ ε and
21            normalizedLevenshteinDistance(P_i.name, P_j.name) ≥ σ and
22            P_i.categories ∩ P_j.categories ≠ ∅
23 end
```

4.5 Discovering regions of interest

Now that we have obtained a cleaner uniform dataset, we can perform additional analysis to extract useful information out of it. One example is to identify emerging *regions of interest*, i.e. geographical areas with high density of POIs of certain categories. These ROIs can have various applications in many areas such as tourism, geo-marketing, business and area planning, etc... These regions are extracted from the unified point dataset by clustering related POIs based on their category and proximity to each other. The shape of each region is defined as the convex hull of the points in the cluster. Since we want to form clusters that indicate regions where a high concentration of POIs of a certain category is located, we need to set a threshold *minPts* for the minimum number of POIs a

candidate cluster should attract at least, in order for it to be considered permanent and not an outlier or noise. This parameter can be reconfigured depending on the needs of the application that will use the resulting clusters. More specifically, we assume that a set of points $\mathcal{P} = \cup_{i=1}^{n} P_i$ forms a ROI if the following conditions hold:

- $|\mathcal{P}| \geq minPts$ (cluster should contain at least $minPts$ points)
- all points in the cluster can be reached without having to travel more than ε distance units from a point to another at any time
- all points in the cluster have at least one top level category in common

Entities in our initial dataset are grouped by their top level category before initiating the clustering procedure for each group. This reduces the respective input size thus speeding up execution time and also ensures that only POIs that share a common category will end up in a cluster. In addition, similarly to the case of entity matching, to avoid a pairwise comparison of all the POIs in a large area, our implementation splits the input area recursively into smaller cells (cf. Figure 4.2), based on a threshold K on the number of enclosed POIs in each cell. The initial area is checked against the total number of enclosed POIs. If the number is bigger than K, the area is split into four quadrants, and each quadrant is then checked recursively. This eventually forms a quad tree, where only leaf nodes need to be examined.

In order to detect clusters within each leaf node, we use DBSCAN [28], one of the most known and used density-based clustering algorithms. DBSCAN fits well with our definition of a region of interest as it discovers clusters based on the notion of *density reachability*. It starts from a random point and advances by finding and adding *directly density-reachable* points to the same cluster. A point p_j is *directly density-reachable* from point p_i if their distance is less than ε, and p_i has at least $minPts$ neighbors. By definition, the *directly density-reachable* points from p_j are *density-reachable* from p_i, if p_j is *directly density-reachable* from p_i. If no *directly density-reachable* point can be found, the current point is marked as "noise" and the sequence continues with another not-checked random point. A point marked as "noise" can be eventually become "part of cluster" when found in another dense point's neighborhood. The algorithm finishes when all points have been visited. At that time each point will have a status of either "noise" of "part of cluster". Points that are part of the same cluster are mutually *density-connected*, i.e. there exists a point p_o in the cluster such as that for every pair $p_i - p_j$, both p_i and p_j are *density-reachable* from p_o.

Starting from the deeper leafs, once all four adjacent quadrants of the same parent are examined, the resulting clusters can be merged; following this reverse direction upwards to the root, eventually the clusters of the entire area are computed (Figure 4.3).

In order not to miss clusters that cross cell borders, when examining points within a cell we also consider points that are located in a buffer zone of width ε around its borders. This means that the resulting clusters may overflow the boundaries of each cell, and subsequently merge with clusters of adjacent cells. Inner and border points are kept in separate lists for each cluster, in order to reduce merging time, since during merging only border points need to be checked. That is, if two clusters of adjacent cells share a common border point, these two clusters are merged into one. A positive side-effect of the splitting, besides the speedup when cells are checked in parallel, is that entire cells that contain less than $minPts$ points need not be considered as starting points for the formation of clusters, and can be skipped. The cluster detection algorithm that is very similar to the duplicate detection algorithm is shown in Algorithm 2.

(a) Separate clusters (b) Merged clusters

Figure 4.3: *Independently calculate regions of interest for leaf nodes, then merge them into a single cluster.*

ALGORITHM 2: Outline of process `discoverROIs()`

Input: The bounding box B defining the area for which ROIs are to be computed. The parameters ε, $minPts$, K.

Output: A set $\mathcal{M}$ containing pairs of matched POIs.

```
1  Function discoverROIs(B, ε, minPts, K)
2      P ← loadPOIs(expandBbox(B, ε))
3      if |P| ≤ K then
4          foreach P_i, P_j ∈ P do
5              if belongToSameCluster?(P_i, P_j) then
6                  M_k ← extendOrCreateClusterWith(P_i, P_j, minPts)
7                  M ← M ∪ M_k
8              end
9          end
10     else
11         [B_sw, B_se, B_ne, B_nw] ← quadSplit(B)
12         foreach B_i ∈ [B_sw, B_se, B_ne, B_nw] do
13             M_i ← discoverROIs(B_i, ε, minPts, K)
14             M ← M ∪ M_i
15         end
16     end
17     return M
18 end

19 Function belongToSameCluster?(P_i, P_j)
20     return |P_i.lon − P_j.lon| ≤ ε and |P_i.lat − P_j.lat| ≤ ε and P_i.categories ∩ P_j.categories ≠ ∅
21 end
```

The gradual splitting of the area into smaller cells, allows us to easily parallelize the process to reduce execution time, by assigning each cell to a different thread, process or worker. In order to further evolve the browser-based computing platform we described Section 3.4 and Section 3.5, we also implement a variation of the process discussed in this section, that outsources the clustering process for each small bounding box to users browsers that are connected with our system. For example, each of the four cells shown in Figure 4.3(a) are being calculated by a different browser and the results are sent back to

our server. This concludes that our browser based computing platform is modular enough to be used for a variety of parallelizable problems from merely collecting large datasets to even performing more complex processing like running clustering algorithms on small datasets.

Independently of where the clustering process takes place (multithreaded single node implementation, browser based cluster, etc...), once cluster information for all four cells of the same parent is available, a merging process runs recursively to combine clusters residing at the borders as needed.

Modular design. If we take a close look at Algorithm 1 and Algorithm 2 we will find a lot of similarities. In fact both algorithms perform a very similar task and can be combined into a single function with tweaked parameters. We can think of duplicate detection as a clustering process where duplicate entities end up in the same cluster. In this special case though, essentially the *minPts* parameter will need to be set to 1, since clusters entities that have no duplicates will form a cluster on their own with size 1. On the other hand when checking the condition of whether two points should belong to the same cluster, we see that there is no need to check their labels textual similarity when we are detecting ROIs. In this case we can simply set $\sigma = 0$ to disable this check in the latter case.

4.6 Experimental evaluation

The following experimental results showcase the benefits of applying the suggested workflow on datasets collected from multiple sources. We focus on three large cities, London, Vienna, and Athens. For the purpose of the experiments, we collected crowdsourced POI data from a set of popular Web sources, in particular: DBpedia, OpenStreetMap, Wikimapia, Google places, Foursquare, and Eventful.

For DBpedia and OpenStreetMap, POI data were retrieved via a SPARQL query endpoint. The rest of the aforementioned sources provide REST APIs for retrieving data. The collection process follows a hierarchical grid partitioning, as mentioned in Section 4.2. Appropriate time delays were inserted between subsequent requests to the external APIs to ensure that the limits of each source are respected. Finally, since communication errors may also occur, we keep track of the progress so that we can resume the collection process after a failure instead of starting over. Table 4.1 (1st box) shows the number of POIs collected for each one of the selected cities from each of the selected Web sources.

After retrieving the POIs, the category mapping process described in Section 4.3 was applied to map the original POI categories to our reference schema. Based on the score

	London	*Vienna*	*Athens*
DBpedia	4,243	169	147
OpenStreetMap	20,538	10,278	1,291
Wikimapia	26,460	9,519	9,449
Google places	157,890	45,674	59,977
Foursquare	166,739	58,674	103,739
Eventful	17,785	730	351

Exact	*Medium*	*Low*	*None*
14%	26%	29%	31%
29%	34%	16%	21%
12%	17%	28%	43%
61%	18%	10%	11%
77%	16%	4%	3%
74%	4%	20%	2%

Table 4.1: *Number of POIs collected per data source and area. - Percentage of source categories mapped to the reference schema*

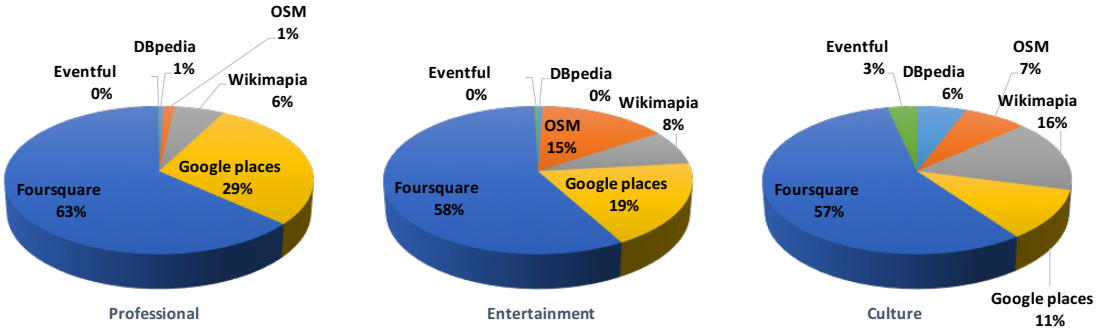
m computed by the matcher, we assign each automatically computed mapping to one of these cases: *exact* ($m = 1$), *medium* ($0.5 < m < 1$), *low* ($0.1 < m \leq 0.5$) or *none* ($m \leq 0.1$). For each source, the percentage of mappings found for each of these confidence levels, is shown in Table 4.1 (2nd box). Using these mappings, we manage to classify the collected POIs based on a common reference schema. The charts in Figure 4.4 show, for each category, how many POIs were retrieved from each source. Note that it is possible for a POI to belong to more than one category, either because it may have been tagged differently in different sources or because it may have multiple categories assigned to it even by the same source, e.g. POIs serving multiple purposes.

Subsequently, we try to detect duplicates in the entire unified dataset by using the methodology explained in Section 4.4. For this experiment, we have set the parameters of the duplicate detection algorithm to $\varepsilon = 0.002°$ and $\sigma = 0.7$. Note that POI coordinates are measured in degrees, thus setting the error threshold for the distance to $\varepsilon = 0.002°$ allows for detecting matches within distance of about 200 meters (in each axis). This value may be rather high for many cases, such as when matching cafés, restaurants, etc., but a stricter threshold might miss POIs of larger spatial extent, such as shopping malls, parks, parking areas, lakes, etc., where different sources may use different coordinates (i.e., marker positions) to represent the location of the entity. These parameters can be tuned to trade off between false positives and false negatives. Moreover, an interesting alternative would be to consider the exact polygon shape of the entity in the matching process. However, we have found that in most cases the POIs are represented by a single pair of coordinates, i.e. as points. In addition, we have set the threshold for deciding whether to split a cell before performing POI comparisons to $K = 500$. Once duplicates are identified, we compare them amongst all possible pairs of sources in Table 4.2. We notice that many matches are also detected between POIs from the same source. Besides actual duplicates, the matched POIs can potentially be related POIs that are located close to each other and have similar names (e.g. public transport stops for inbound and outbound routes). In fact, validating the results of the integration process can be done again via crowdsourcing and is part of ongoing work.

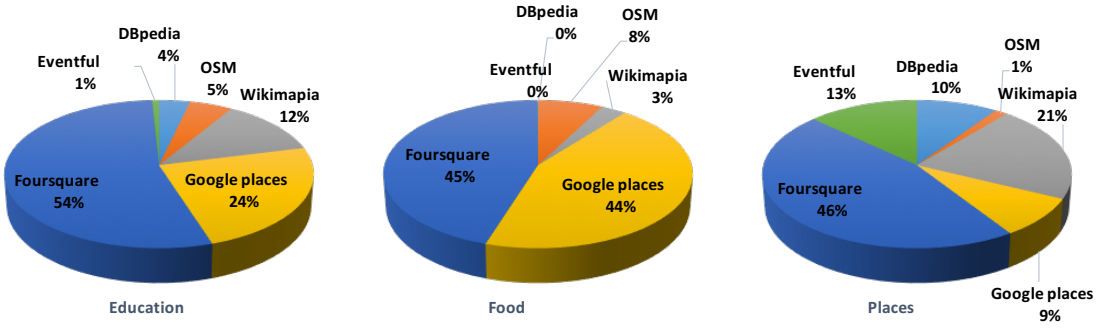
	DBpedia	OpenStreetMap	Wikimapia	Google places	Foursquare	Eventful
DBpedia	123	297	1,559	1,100	2,127	311
OpenStreetMap	297	1,094	4,281	6,114	8,687	136
Wikimapia	1,559	4,281	3,252	5,235	8,718	414
Google places	1,100	6,114	5,235	14,454	15,799	264
Foursquare	2,127	8,687	8,718	15,799	17,385	772
Eventful	311	136	414	264	772	587

Table 4.2: *Number of matched POIs between the sources.*

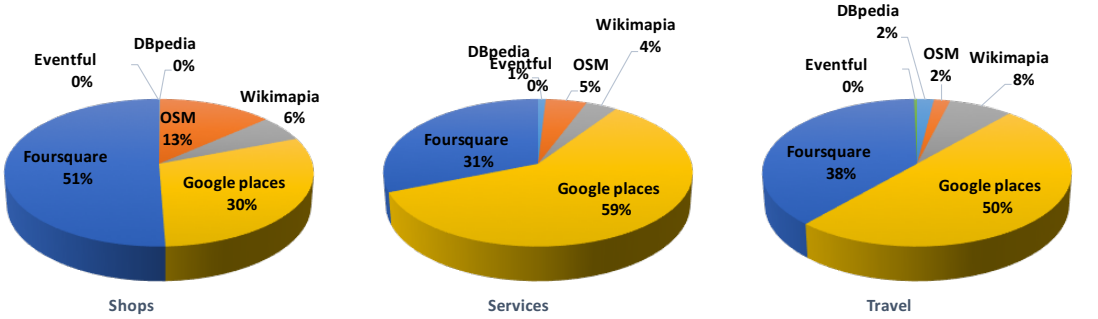
Finally, the clustering procedure described in Section 4.5 was applied to identify regions of interest. The modified version of the DBSCAN algorithm was executed for each of the eleven categories to produce eleven groups of clusters for every city. Figure 4.5 shows the raw point dataset and the computed clusters shapes for the "Entertainment" category for each of the three cities, Athens, London and Vienna. For this experiment, we have set the values of the clustering algorithm to $\varepsilon = 0.002°$ (approx. $\approx 200m$) and $minPts = 20$. With these parameters the computed clusters contain approximately 11% on average of their respective raw point dataset. These parameters can be tuned to control the extent of the clusters and their density. As is evident from the examples in Figure 4.5, this processing is extremely helpful, as it eliminates the noise found in the raw point dataset and reveals areas that have high density of POIs for the selected category.



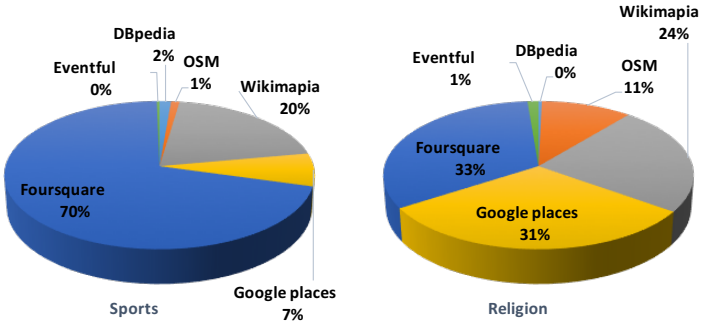
(a) Professional (43,889) (b) Entertainment (41,517) (c) Culture (9,601)



(d) Education (30,932) (e) Food (88,015) (f) Places (32,894)

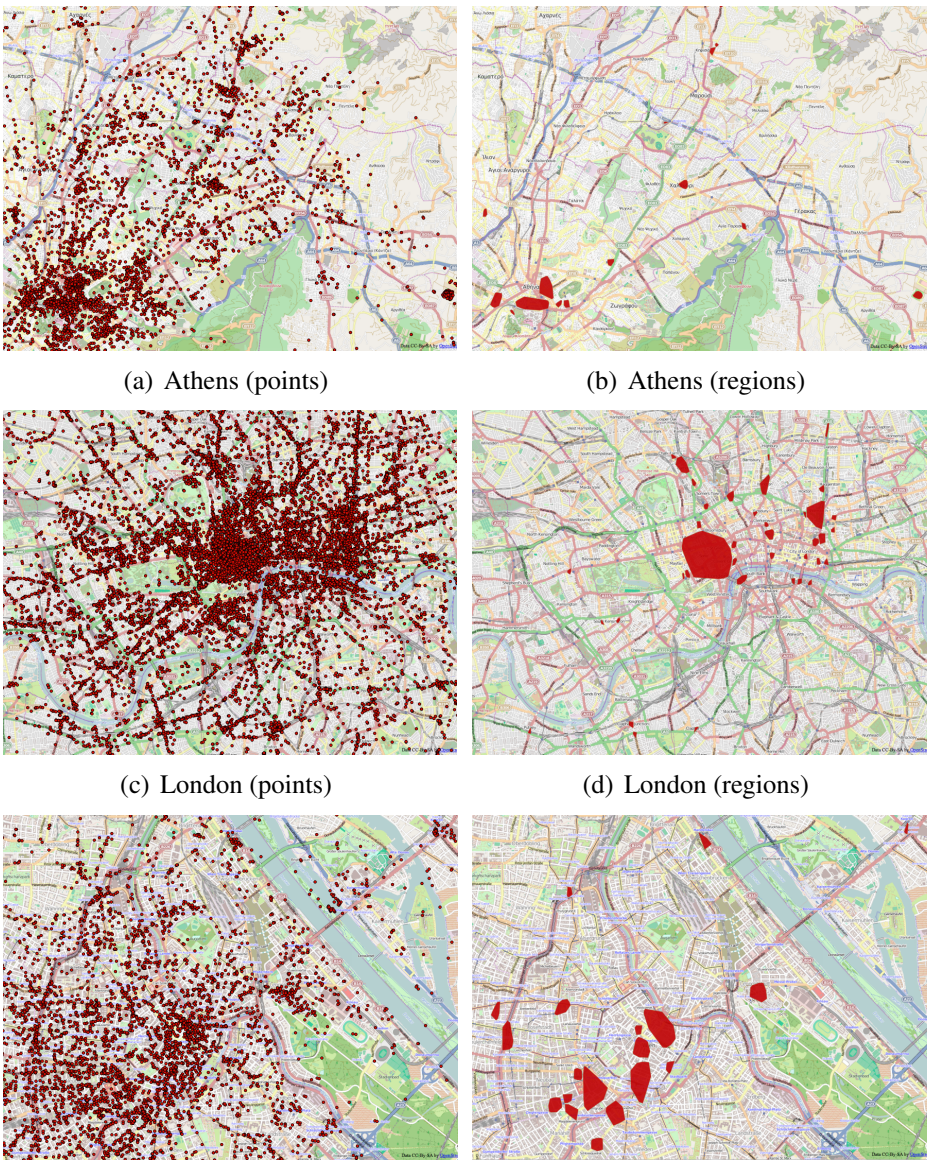


(g) Shops (76,971) (h) Services (104,189) (i) Travel (44,192)



(j) Sports (12,324) (k) Religion (11,952)

Figure 4.4: *Distribution of retrieved POIs per category per source.*



(a) Athens (points) (b) Athens (regions)

(c) London (points) (d) London (regions)

(e) Vienna (points) (f) Vienna (regions)

Figure 4.5: *Original locations of entertainment venues and extracted entertainment hotspots for Athens, London and Vienna.*

4.7 Conclusion and Future Work

This chapter introduced a workflow to derive a unified dataset comprised of data collected from multiple sources, each one having its own schema and purpose. We describe a process of mapping each external source's category schema to a common hierarchy, essential for unification and further processing. Subsequently using clustering techniques we detect and fuse duplicate entries, and discover regions of interest. As a direction for future work, we would like to do a comparative analysis by implementing and testing the performance and quality of various clustering algorithms. We extensively use the hierarchical grid structure that was introduced in the Chapter 3 to both store and parallelize the processing of our data to achieve maximum efficiency. The suggested workflow is also partly ported to use our browser based computing platform, also described in Chapter 3. This showcases the modularity and capabilities of our system and the potential of it being used

in many more scenarios. Finally, the goal of this chapter is not only to provide a unified dataset to other application and users, for them to utilize for their own cases, but also to offer more services on-top of the new dataset, such as ROI detection and additional analytics, such as the multi-dimensional summaries that are covered in the following chapter.

In the next chapter we shift our focus to *streaming datasets*, where the time dimension is strongly evident. We will present a series of analysis steps that provide summaries, vital to making sense of the large volume of these datasets, always keeping in mind their evolving nature. While our analysis mainly involves data coming from a single source, the results of our study can easily be applied to multi-sourced datasets such as the ones created with the methodology discussed in this chapter.

Chapter 5

Summarizing social media data streams

As the amount of user-generated geospatial content on the Web is constantly increasing, the sheer volume alone makes tracking and identifying important information challenging. Added to that, the noise and the lack of quality guarantees, makes the use of tools that provide quick overviews of the data really helpful. This chapter walks the reader through our efforts towards providing a set of different summaries of the available spatiotemporal content, with the added motivation of discovering the activities of the users that fuel the publishing of these content, thus enriching the locations information where possible. This content is mostly derived from streaming datasets, where the temporal dimension is prevalent, essentially capturing the evolution of all other dimensions, such as spatial, thematic and social.

This chapter's content was initially included in our [52] article under review for consideration of publication in the "Social Media Processing" special issue of the ACM Transactions on Intelligent Systems and Technology journal (submitted January 2016).

5.1 Introduction

The motivation for this work stems from the proliferation of citizen journalism, which enables the general public to collect, process, and publish descriptive location information through Web 2.0 platforms on a variety of topics. Harvesting this user-generated information presents us with a unique research opportunity to capture the *general public's perception of space*, which can then be used to enrich the digitized spatial world beyond mere collections of coordinates, maps, and images. In doing so we reach beyond simple geometry, to capture the online digital footprints of the population, understanding the trends and patterns of topics and events in space and time, and how they *link* diverse communities. This research will therefore lead to a better understanding of place and how it evolves over time.

This information can be used to complement and enrich the content of traditional gazetteers by introducing a perception gazetteer component, in the form of places and their complex descriptors. These descriptions are *geometrical* (e.g., place outlines), *contextual* (e.g., key terms associated with them), and *social* (e.g., identifying community members associated with them). They are also *temporal*, as they capture how the previous descriptors evolve, periodically (e.g., weekly) or intermittently (e.g., in response to events). The information required to generate such perception gazetteers can be harvested from open-source, user-generated geospatial content. Such content may range from map datasets (e.g., openstreetmap.org)[36], to geospatial data contained in narratives (e.g.,

travel blogs) [77], and geospatial content harvested from social media (e.g., tweets and flickr imagery) [79].

Our goal is to create a set of*interactive and interconnected summaries* that describe the same dataset. These interactive summaries will allow users to view the several descriptors in an interconnected fashion, where filters applied in one dimension would make all the other views adapt automatically to the relevant data. Each of the following summaries can be used both to display the data of the respective dimension, and as a filter to limit the data displayed in other dimensions.

5.2 Temporal summaries

Summaries on the evolution of metrics are the simplest and most widely used. Depending on the metric being reported, they are used primarily for two reasons:

- to detect bursts like sudden rises or drops during certain periods
- to select specific ranges as a filter for other dimensions to focus on

Based on the type of search that is used to collect the results, the peaks and drops in the chart can be interpreted and associated to specific events. For example, if the traffic is originating from a bounding box search, then peaks and drops are indicative of the social media volume produced within the specific area, and might signify a local event. On the other hand, when monitoring specific keywords, one can see increases and decreases in mentions of these keywords and associate these with specific times and events.

Depending on the temporal extent that the data covers and on the level of detail needed, the summary can group counts per different time periods. An important thing to notice here is the timezone used to group counts, especially when the time binning used in more than an hour. With an hourly temporal binning, different timezone can be represented simply by shifting the line chart left or right based on the UTC offset of the timezone selected. However when using a bigger temporal binning, e.g. counts per day, the actual timestamps of the data need to be shifted before the grouping takes place, to ensure correct containment within the timezone's bounds. Usually a timezone selection makes sense if the data extends within a single timezone's spatial bounds. In cases where the data extends to multiple timezones, Coordinated Universal Time is usually used.

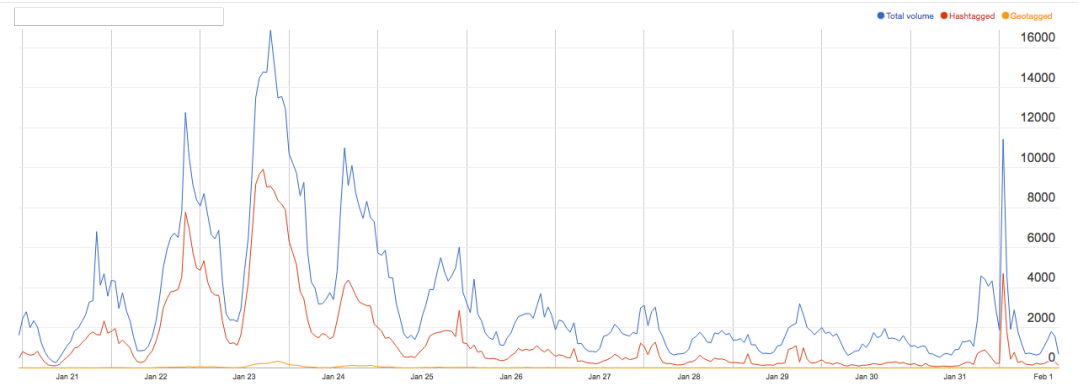


Figure 5.1: *Social media traffic (counts per hour) for keywords related to winter storms and blizzards. The traffic captured two spikes, one for the Category 5 winter storm "Jonas" that affected the US East Coast from January 22-24, 2016 and another one related to a less severe Category 2 winter storm during January 30 to February 1 that affected most of the US.*

Typically, temporal summaries are line or area charts that show the evolution of volume of a metric. Besides the overall volume of the captured social media traffic, temporal summaries are usually subviews of other type of summaries where they display the evolution of metrics meaningful to the respective summary. Our work focuses on the following type of temporal summaries:

Overall. The most typical temporal summary is the evolution of the overall traffic volume collected for a specific search. When dealing with content from social media streams, besides the total volume, there are two distinct categories of posts: posts containing at least one *hashtag* and post tagged with a geographic location (specific latitude and longitude, or a more generic place name). We call the first category *hashtagged posts*. These posts are used extensively in Chapter 6, to detect emerging events described by the included hashtags. The second category is call *geotagged posts* and is used for all the spatial related views. In this type of graph (example shown in Figure 5.1) one can observe the variations of the overall traffic (blue), compared to the traffic of *hashtagged posts* (red) and that of *geotagged posts* (orange).

Per geographic region (Spatial summary subview). Another temporal summary that is often useful, is the variation of the traffic originating from distinct administrative areas. In this type of summary, data is grouped twice. The first grouping key is the administrative area code, usually a country code, but depending on the focus and the level of detail needed can be any administrative area code (state, county, city, etc...). The second is the time granularity with all the considerations described previously. An example is shown in Figure 5.2.

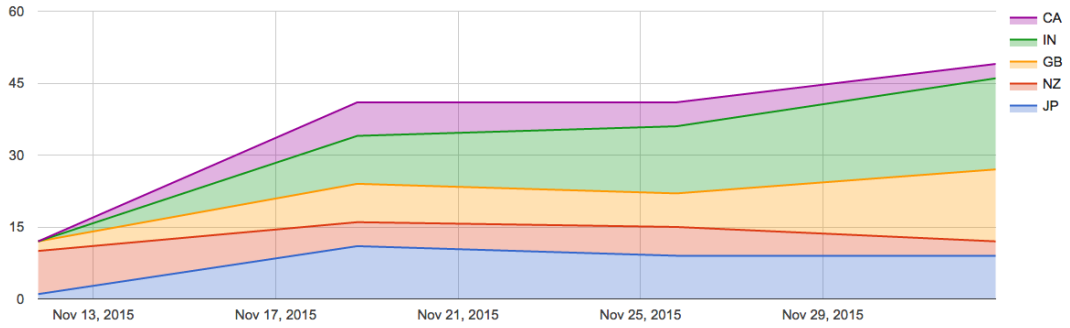
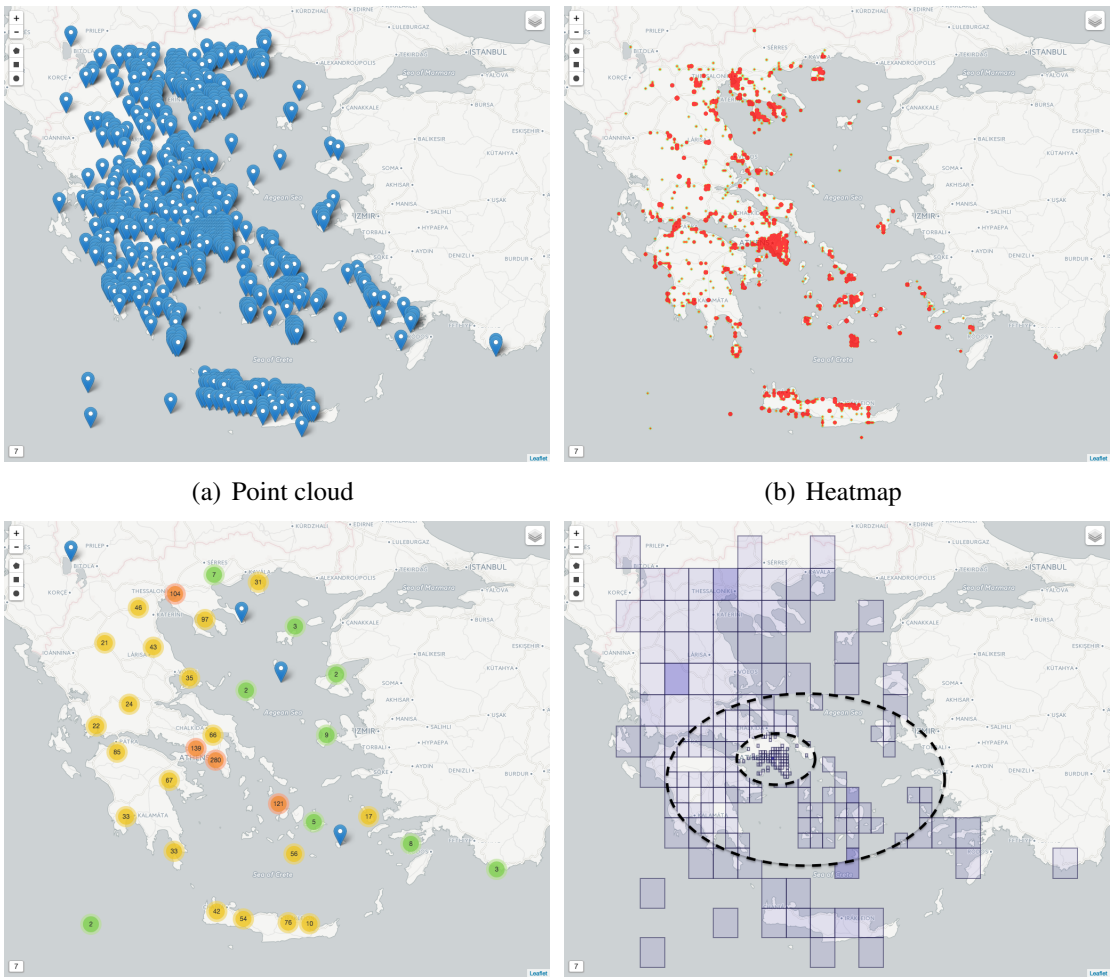


Figure 5.2: *Social media traffic (counts per week) of mentions to the* `earthquake` *keyword or hashtag, split by country (excluding the United States). For clarity only the top 5 countries (most volume produced) are shown.*

Lexical token evolution (Thematic summary subview). This summary displays the occurrences of a specific lexical token (hashtag, word, phrase) discovered in the text of the post. The evolution of certain hashtags is particularly important, as it an essential part of event discovery as explained in detail in Chapter 6.

Social presence evolution (Social summary subview). The variations of the social presence of a specific user in the social network can reveal valuable information and further provide insight to individuals behavior and influence. This summary shows how the number of posts that are somehow related to a specific user evolves over time for each specific category (e.g. published, quoted, retweeted, liked, favorited, etc...).

Entity mentions evolution (Entity summary subview). Entities discussed and discovered in the text of social media posts vary over time. This subview is similar to the lexical token evolution subview, with the added advantage that it can group together variants of words or phrases (abbreviations, different languages, different parts of speech, etc...).



(a) Point cloud (b) Heatmap

(c) Clusters (d) Hierarchical grid

Figure 5.3: *Different types of spatial summaries for the same dataset: Twitter traffic originating from Greece during the September 2015 elections.*

5.3 Spatial summaries

For the portion of the data that has attached location information, plus the portion of the data that location information can be derived for (by geoparsing and geocoding), we can create spatial summaries. These summaries are map visualizations that highlight active and inactive geographic regions, with respect to either the entire dataset, or a subset of it resulting by filtering one or more of its other dimensions.

The embedded location information in social media posts can come in two flavors: either a pair of coordinates usually derived from GPS-enabled devices such as smartphones that captures the exact point that the user authored the post or a more descriptive place name or region, such as a POI, neighborhood, city or country. To construct the map visualizations the two different flavors need to be homogenized. A geocoding process converts place information to actual coordinate pairs, while a reverse geocoding process converts coordinate to place information (primarily administrative region bounds).

Point based. The simplest spatial summary is placing markers on a map wherever a post occurs. However this can quickly lead to very cluttered visualizations as shown in Figure 5.3(a). An easy workaround is to cluster markers that are relatively close to each

other together to form clustered markers shown in Figure 5.3(c). This would require a clustering algorithm such as DBSCAN [28], or a grid based clustering algorithm such as the one described in Chapter 3, to group the coordinates of close-by points into clusters. Another very common spatial summary is a heatmap shown in Figure 5.3(b), which uses a gradually changing color range and diffusion to create beautiful gradients that easily identify hotspots. Finally, the hierarchical grid data structure we originally described in Section 3.3 can be used to provide spatial summaries, as it is able to display intensity as opacity of grid cells in a mesh, and is capable to adjust the level of detail shown depending on the spatial extent of the data.

Region based. In the cases where precise coordinates are not available, but place information is, we can use the later to build the spatial summary. In this case whole regions described by (multi)polygons are used instead of points to display locations on a map. Regions are then shaded more or less intensely based on the number of posts they contain. The subset of posts with actual coordinate information is reverse geocoded to match the granularity of the place information used in the rest of the dataset, e.g. country, state, prefecture or city.

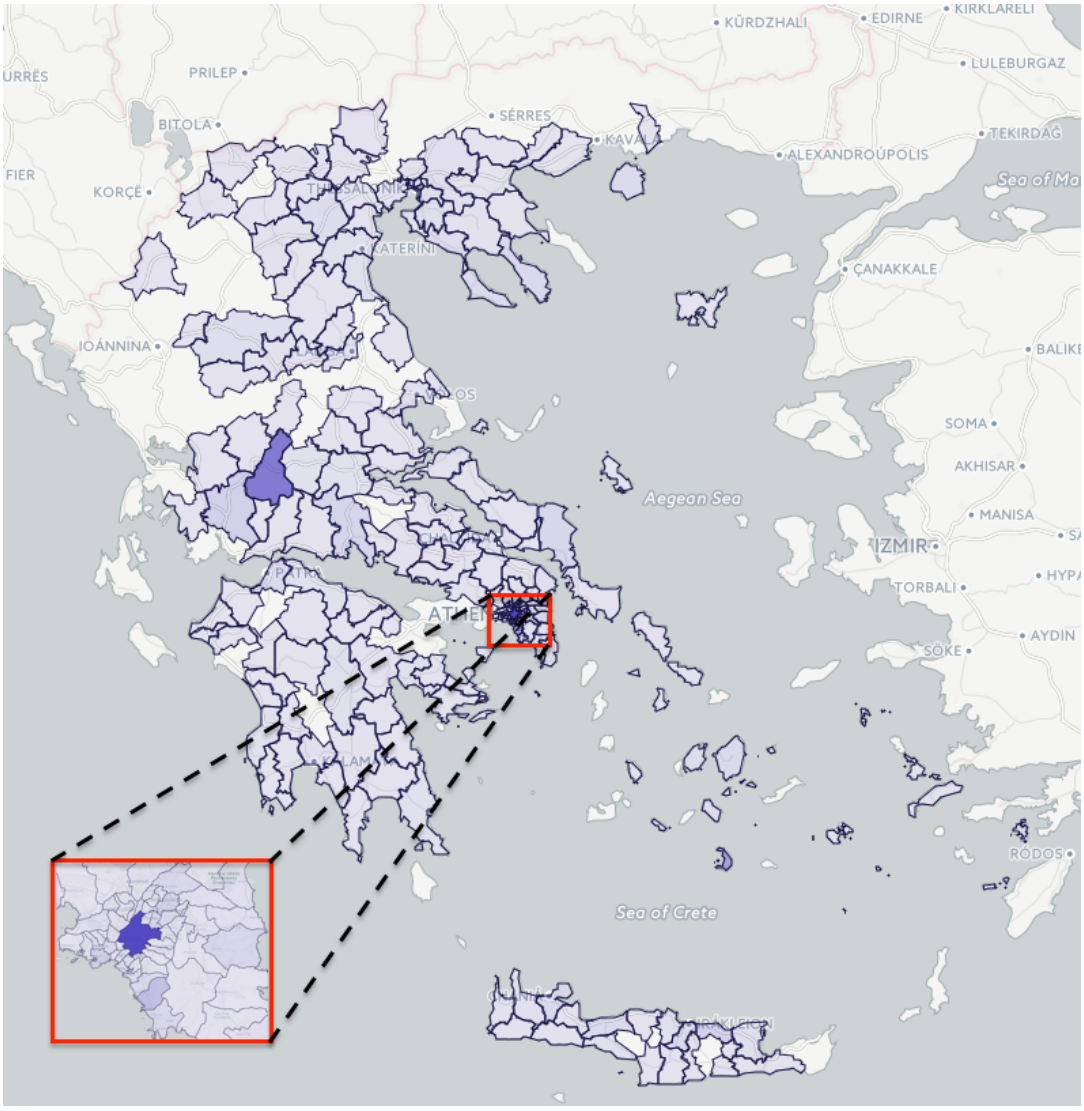
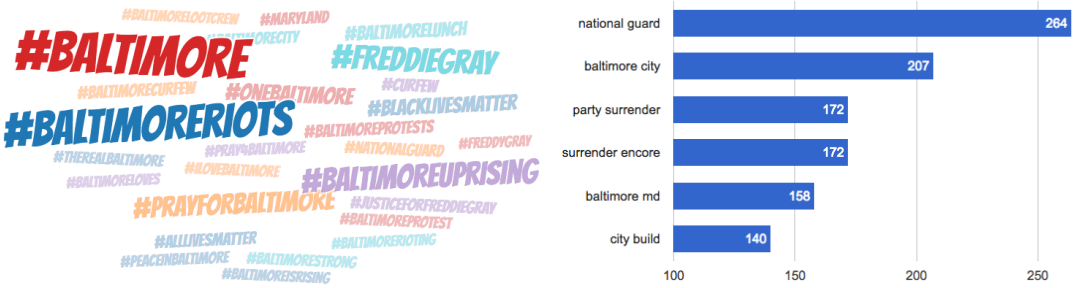


Figure 5.4: *Region based spatial summary of the same dataset as Figure 5.3: Twitter traffic originating from Greece during the September 2015 elections.*

5.4 Thematic summaries

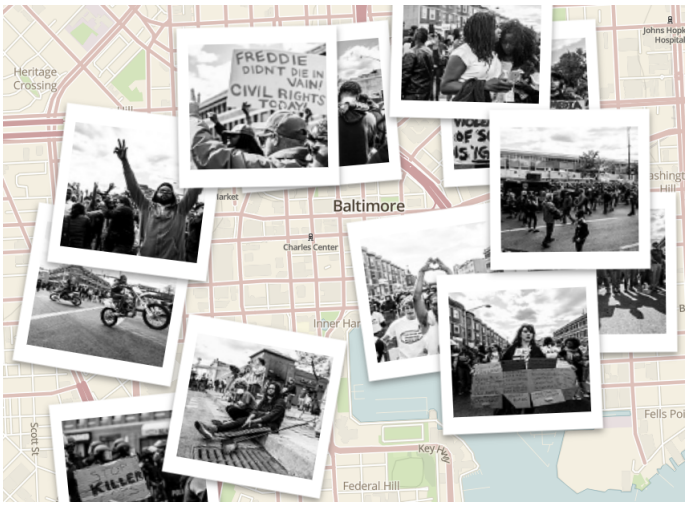
If space and time summaries answer to *where* and *when*, the thematic (or contextual) summaries answer to *what*, by looking into the textual (and less often the multimedia) information present in the social media posts. In this work we focus on tags (hashtags), single words (unigrams) and small phrases (bigrams and trigrams). These simple thematic summaries give a good indication of the topics discussed within a dataset. Combining this view with spatial and temporal filters, allows us to display more detailed summaries relevant to specific areas and time periods.

Tag summaries are the easiest to calculate since hashtags can be easily extracted from the text by means of a regular expression. In order to create the rest of the summaries we need to filter the textual content by removing stopwords (per language whenever possible), URLs, etc. and lowercase all text to achieve better grouping. Once we get a "clean" text we then tokenize it and then get all single, double, and triple tuples of adjacent tokens. For visualization purposes token thematic summaries are presented in either column/bar charts or word clouds, while multimedia thematic summaries are presented as a collage over a map.



(a) Hashtags cloud

(b) Bigrams frequency



(c) Multimedia (photos)

Figure 5.5: *Thematic summaries examples for a dataset collected from traffic originating from Baltimore during the April 2015 riots.*

Textual thematic summaries are closely related to entity summaries, which are constructed by using a named entity extraction process on the text of the posts. The main difference is that entity summaries identify established entities that already exist in the

knowledge bases, while thematic summaries are able to recognize newly surfacing and emerging keywords or phrases that might be indicative of events. This is why they are extremely important in event detection as described later on in Chapter 6.

5.5 Social summaries

When dealing with social media data, an important aspect to analyze is the underlying social network, and the connections its users form in it. There are two types of connections: connections based on user relationships and connections based on published posts. The former are built when two users directly connect using social media actions: befriending in Facebook, following in Twitter, subscribing in YouTube, etc... The latter are formed through the actual posts that the users publish and are also platform dependent: e.g. retweeting or mentioning in Twitter, liking or reacting in Facebook, tagging on Instagram, etc...

In this work we focus on the latter type of connections that can be easily derived from the consumed public stream, without having to query the network for extra information about its users. We introduce two different social summaries:

Social presence. In this type of summary we are concerned with a user's presence in the social network, either by initiating a discussion or by receiving exposure initiated by other users. To discover a user's social presence initially we create a list of all the types of communication that can be found in the retrieved posts: for example in the case of Twitter, lets imagine we want to build the social presence of imaginary user @GL. Their social presence is the union of all posts (tweets) that @GL either authored, or is referenced in. Furthermore the tweets that originate from @GL are broken down into *retweets* or *quotes* of other users, *replies to other users* and *original* posts. In a similar sense the tweets that are referencing but are not authored by @GL are also broken down depending on whether they are *retweets* or *quotes* of @GL by other users, *replies* of other users to @GL or simply *mentions* of @GL by other users. This partitioning of the messages is very helpful and can provide insights when studying the evolution of the volume of these categories. Essentially a user's social presence evolution will be shown as line charts for each of the aforementioned categories.

Social linkage. This summary is the natural evolution of the social presence summary where the focus is shifted from a specific user to the connections they form with other users through their messages. The linkage is represented with a graph (network) where nodes are users and edges are social media posts. An edge exists between $\texttt{user}_\texttt{i}$ and $\texttt{user}_\texttt{j}$ if there exists a post that is either authored by $\texttt{user}_\texttt{i}$ and references $\texttt{user}_\texttt{j}$ or vice-versa. Edges are also identified by their *kind*, which is the type of the reference between the users as mentioned earlier (retweet, quote, mention, etc.). Edges (posts) of the same kind between two user, are merged into one edge that has a weight equal to the count of all the edges that merged to form it. This also helps to keep the visualization less cluttered. The weight of an edge is not a fixed number but it varies depending on the temporal extent that is selected for the view. Since the individual edges that are merged to form the final edge all have attached timestamp information, every time the temporal window of the view changes, edges weights are recalculated based on that. This allows observing the changes in the weight of nodes and edges while animating the network over time, in order to pinpoint when an increase or decrease occurred.

Many of the graph theory concepts can be used when studying the social linkage graph. For example by running a BFS algorithm on the graph, it is easy to partition it

to distinct connected components, and decide to visualize subgraphs of size bigger that a certain threshold. This would eliminate the visual noise generated by groups of few users that only communicate with each other and never link to the "global" discussion. When further looking into the distinct connected components, one can also find interesting patterns, such as clusters formed by geographic barriers. People usually tend to engage into conversations and/or reference other users that live in close proximity as them, since they share the same problems or concerns; thus it often happens that regional connected components emerge, as shown in Figure 5.6.

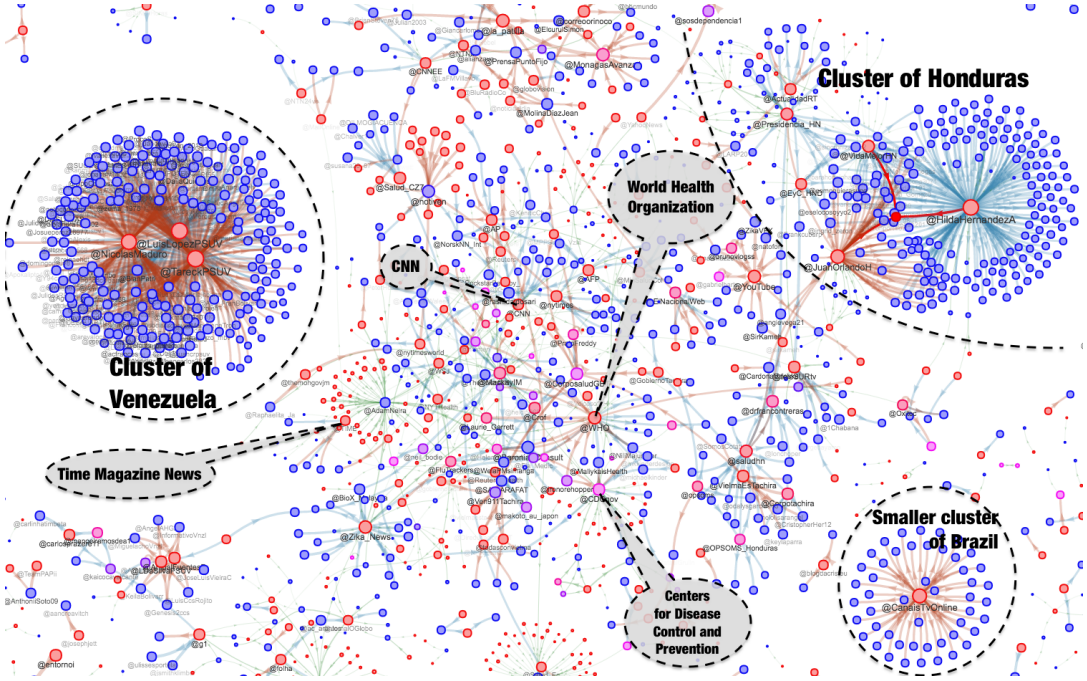


Figure 5.6: *Social linkage network for tweets mentioning the "Zika" virus outbreak. Regional clusters and some official major accounts have been annotated.*

It is worth mentioning that the social linkage graph is a directed graph, where direction indicates the flow of information, i.e. the author of a post (edge) is always at the destination end of the edge. The directionality of the graph allows us to easily point out producers and consumers of information, by simply calculating the in and out degrees of nodes to distinguish between "source-like" and "sink-like" vertices. To easily visually spot them, we use a gradient color from red to blue that is indicative to the in or out degree of the nodes, relative their overall degree. Information producing nodes ("source-like") are colored towards *red*, while information consuming nodes are colored towards *blue* as shown in Figure 5.6.

The resulting social summaries can be used in an analysis to assess the importance of a group of users within its social network by being able to:

- answer how does a user's publishing activity correlate with their messages being further spread?
- distinguish among users that merely produce information, users that consume it, and users that pass it through the network.
- measure the influence of user's message throughout the captured network (and optionally identify the penetration of a single message).
- check if the conversations a user engages in with other members are balanced

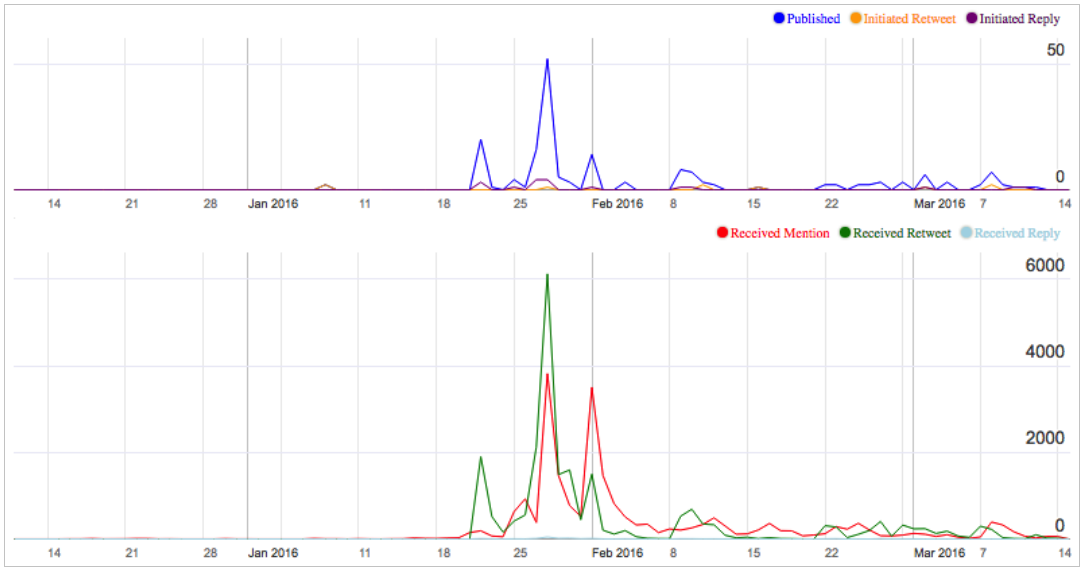


Figure 5.7: *Social presence of @WHO (World Health Organization) in the "Zika" virus dataset mentioned in Figure 5.6. Notably tweets published by @WHO (blue series at the top) are immediately getting picked up and retweeted (green series at the bottom).*

The aforementioned points make evident that social presence and linkage analysis reveals important information about the network and can be extremely beneficial in identifying opinion leaders and provide better insight on who to target in order to achieve better coverage and more efficiently communicate messages through the social sphere.

5.6 Named Entities summaries

Named entities summaries are an evolution of thematic summaries, where instead of using simple lexical tokens to understand the topics discussed, a more sophisticated named entity recognition process takes place. NE are more powerful that simple tokens since they can match variated typings, abbreviations, multiple languages (provided the NER algorithm supports the language detected), all into a single entity. This allows the summary to present a more complete view of how mentions of an entity evolve over time, and how entities link to each other (when referenced together).

Since named entity recognition algorithms rely on pre-existing knowledge bases, they are able to detect only established entities. Thematic entities complement NE summaries in this sense, by surfacing new terms as they emerge in the social media sphere, without having to look them up in a gazetteer. Luckily the evolving nature of collaborative knowledge bases such as Wikipedia due to crowd-sourcing and huge participation, allows NER algorithms to catch up and progressively recognize more and more entities.

An advantage that stems from using an existing knowledge base to match entities from, is that usually the entities recorded in the KBs, are subject to an ontology hierarchy. This allows an additional hierarchical tagging of the discovered entities in our social media feeds, that can yield interesting statistical information. By representing the ontology hierarchy with a tree structure, we can group together entities based on the ontologies they are assigned with, at a specific depth (level) in the tree. Color coding the nodes in the visualizations provides an easy way to spot connections between entities under the same and under different ontology types, as shown in Figure 5.8.

Originally NER algorithms were designed to work on full documents. When dealing with social media posts, where character usage might be limited or preferred to be kept low, these algorithms might need tweaking [69] to produce reliable results. However this is outside of the scope of our work. Our system is designed is such a modular way, where the actual NER tool or algorithm used is separated from the summary presentation and further analysis, so that anyone using it can plug in their preferred solution. Usually NER tools output matches with a confidence and optionally a relevance score. These values can be used to allow only a subset of the matches that obtained a score over a certain threshold to take part in the rest of the analysis.

Similarly to social summaries, our analysis of NE focuses on two aspects: the evolution of mentions of single NEs, and the evolution of the links among them.

NE mentions. This summary provides a listing of all the recognized NE in the corpus, and gives information about the categories they belong in different KBs, plus displays the number of times each one occurred. The occurrences can be traced back to the individual posts, so that the validity of the match can be verified. Since all posts are timestamped, all matches are timestamped as well. Because of that we can at anytime display not just the total number of matches, but also show how the match count evolved over time, by just grouping matches together per hour, day, etc. An example is shown in Figure 5.9 where we plot the evolution of mentions of the Apple Inc. products that were revealed at an event held in September 2015. As is highlighted by the annotations we can see that the actual announcements of the products, coincide with the respective peaks in the graph.

NE network. Once all NE have been discovered in the corpus, we can introduce links between them, i.e. by connecting for example NE that were mentioned in the same post. This leads to a graph, in which NE become vertices and posts become edges. Similarly to the social network view, to keep the visualization less cluttered, edges (posts) between the same NEs, are merged into one edge with increased weight (width). Thus, the size of the nodes becomes an indicator of the number of mentions of the NE, and the width of the edge an indicator of how strong the connection between the NEs is, i.e. how many times where they mentioned together in the viewed time window.

The visualized network provides a way to immediately spot prevalent entities, and

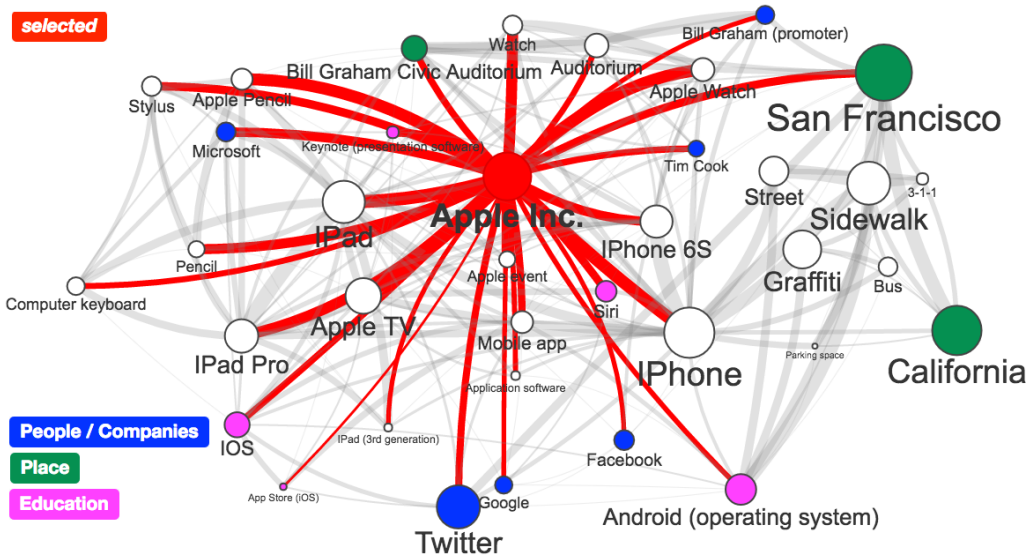


Figure 5.8: *NE network for data collected from the area of San Francisco during the event where Apple Inc. announced the iPhone 6S.*

their strong connections. Unlike the social network, the entities network doesn't form as many disconnected clusters, most likely due to the dataset originating by collecting data about specific terms or specific regions, that make all relevant posts somehow more often connected. Again by using simple graph theory techniques, such as BFS traversal, we can isolate an entity's immediate (1st order) or expanded (k-th order) neighborhood, or simplify the graph by rejecting nodes based on their size or degree. Unlike the social network graph, the entities graph is not directed.

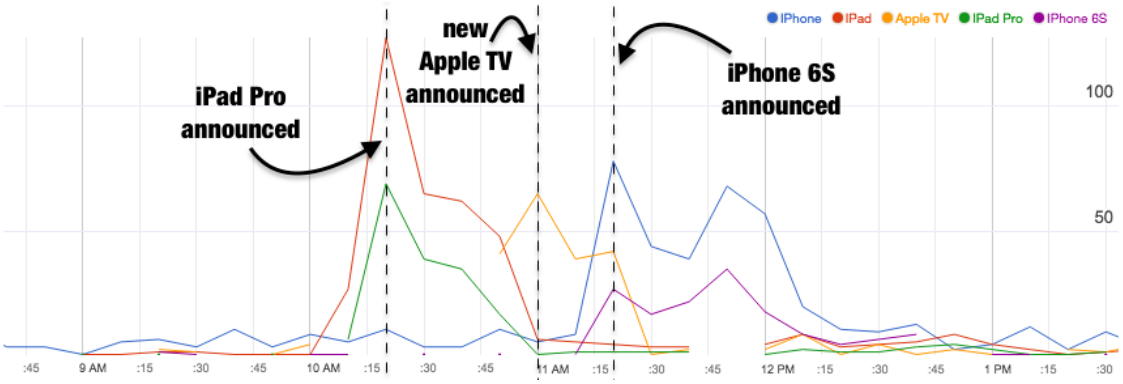


Figure 5.9: *Evolution of entities related to Apple Inc. products revealed at the Apple event on September 2015. Peaks in mentions occur at the time each product was revealed.*

5.7 Hierarchical cube

The previous section introduced various content summaries of social media. In the following, we describe the data structure used to efficiently compute some of these summaries. This so-called *nanocube* is an improvement of the hierarchical grid introduced in [51] that also handles the temporal dimension of the data. It is essentially a *hierarchical cube* (Figure 5.10) that can aggregate information in both space and time, while holding information about other dimensions, such as tokens, entities, etc.

The nanocube can be imagined as a collection of grids *stacked* on top of each other. Each grid "slice" contains data acquired within a predefined time interval. This temporal slicing allows the nanocube to answer queries given a specific time range, but is also very helpful to calculate and store statistics per single time interval (counts for tweets, n-grams, and hashtags). These cached values play an important role in event detection as described later in Chapter 6. Spatial queries (inclusion) are also supported.

The smallest building block of the nanocube is called a *cubicle* and is in fact a grid cell with a time dimension. As social media posts are received, they are in turn inserted into cubicles based on the attached time of publication and their attached geographical coordinates. In our approach, cubicles represent the most finite granularity that we capture information for, i.e. individual posts lose their identity once they are inserted into the cube.

Cubicles can be identified by a unique spatiotemporal id ($t_0x_0y_0$) and have multiple spatial ($x_1y_1, x_2y_2, ...$) and temporal ($t_1, t_2, ...$) ids depending on the granularity levels of the cube when data is aggregated. Besides the spatial and temporal ids, cubicles also store lexical tokens (tags and n-grams) from the corresponding posts, plus additional information such as languages detected or the country codes for each of the geotagged

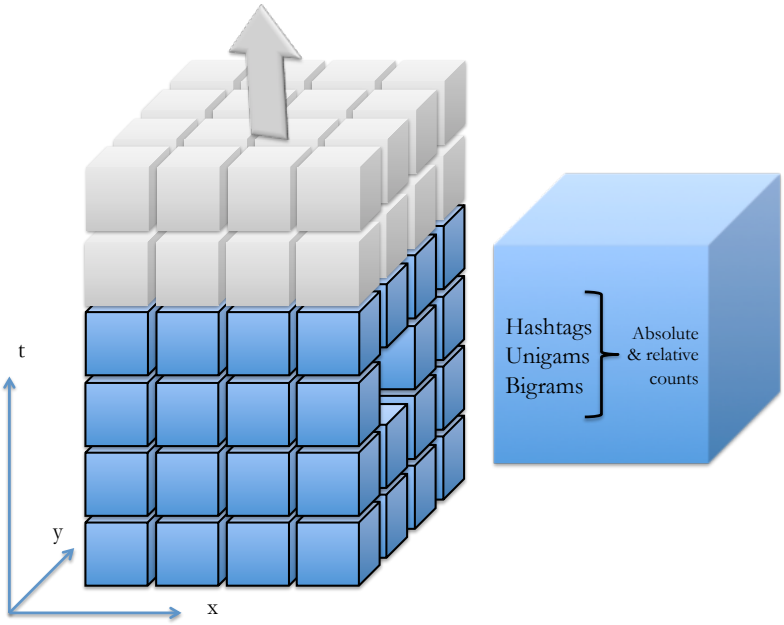


Figure 5.10: *Nanocube data structure*

posts. Each piece of information is stored in an embedded dictionary by means of key-count pairs where each count corresponds to the number of times each key's value was discovered in the spatiotemporally enclosed posts. Finally, each cubicle stores simple counts for the total number of posts (broken down in *hashtagged* and *geotagged*).

The structure was designed and implemented with two main goals in mind. It can:

1. aggregate information at multiple levels of granularity, with respect to, both, space and time
2. run queries that apply space/time/thematic filters on space, time, and the lexical tokens independently, or in combination with each other.

Being able to view the data at multiple levels of detail is very important. With large amounts of data being accumulated, it is essential for both comprehension and performance reasons, to move from finer to coarser levels of detail. Of course, whenever necessary, we are able to increase the level of detail by choosing the desired spacing of the cube.

This capability is provided for both, time and space. To do so, each cubicle is assigned multiple spatial and temporal ids, as described earlier. To compute the ids a truncation function is needed to "digitize" each tweet's attributes to match the properties of the cube. $\texttt{trunc}(v, step)$ returns the biggest multiple of $step$, which is less than or equal to v.

A cubicle might have a set of temporal ids:

$$\left\langle \begin{array}{c} t_0 = t_{1h} = \texttt{trunc}(t, 1_{\texttt{hour}}) \\ t_1 = t_{1d} = \texttt{trunc}(t, 1_{\texttt{day}}) \\ \vdots \end{array} \right\rangle \tag{5.1}$$

When increased temporal detail is needed or the dataset expands only a very short time the lowest t_0 is used, while for less detail or bigger extents we use t_{1d}, or even t_{1w}.

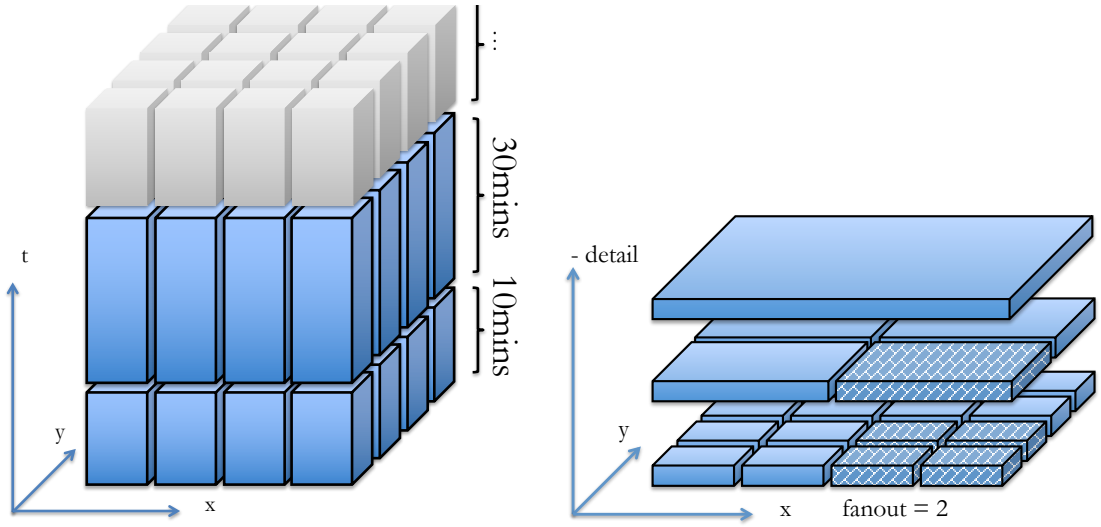
A similar approach is used for space as well. Here a spatial id $x_i y_i$ designates the position of the cubicle at level L_i of the hierarchical grid. The base level L_0 is built by

choosing an initial cell size c_{L_0} for a square spatial grid, of $c^{\circ}_{L_0} longitude \times c^{\circ}_{L_0} latitude$ sized cells and then insert each point inside their respective cubicle. Higher levels are then built by combining enclosing cubicles from the level below based on a grouping factor a. The cubicle's spatial id at level i that a point xy belongs to can be computed as follows: $x_i y_i = \texttt{trunc}(xy, c_{L_i} = c_{L_0} \times a^i)$, where c_{L_i} is the length of the edge of the cubicle at level L_i.

$$\left\langle \begin{array}{c} x_0 y_0 = \texttt{trunc}(xy, c_{L_0} = c_{L_0} \times a^0) \\ x_1 y_1 = \texttt{trunc}(xy, c_{L_1} = c_{L_0} \times a^1) \\ \vdots \\ x_i y_i = \texttt{trunc}(xy, c_{L_i} = c_{L_0} \times a^i) \end{array} \right\rangle \tag{5.2}$$

To monitor a small area at a high level of detail, we group cubicles using the $x_0 y_0$ spatial id, whereas as the area of interest grows, we switch to $x_1 y_1$, $x_2 y_2$, etc.

The level of detail used for the space dimension is independent of any granularity level applied to the time dimension. By manipulating each dimension separately, we can achieve multiple views of a dataset, as shown in Figure 5.11.



(a) Grouping by time

(b) Grouping by space

Figure 5.11: *Aggregation in time and space.*

Running queries on the nanocube is just a process of applying filters to the cubicles, and grouping together by either their temporal or spatial id, the ones that pass the filter. We call the result a *nanocube view*. We support searching on all three dimensions of the cube: space, time and lexical tokens.

To restrict space and time, we use range filters: A time range filter can be applied by specifying a minimum and/or a maximum timestamp. A space area filter can be applied by specifying a bounding box. Lexical token filters specify a set of keyword(s). The aggregation process uses only cubicles that fall within the specified range/area of the query. To optimize queries based on lexical tokens, the structure also manages separate (not embedded) dictionaries, which act as inverted lists and return the ids of the cubicles that contain them. Only the cubicle ids returned by the inverted lists are being fed to the nanocube view construction pipeline.

The spatial view of the nanocube is built by employing the aggregation techniques of the hierarchical grid described in [51]. The construction is based on three parameters: k_{top}, p_k and min_n. We use a combination of these parameters to decide which cells "survive" to be part of higher levels. A balanced threshold for δ is calculated as a percentage (p_k) of the average value of the k highest cell degrees (k_{top}) at each level. Initially at L_i we select only tiles that achieve a degree $\delta \geq k_{top} \cdot p_k$. These are aggregated to construct L_{i+1}. Later, depending on the desired neighborhood density, tiles with less than min_n adjacent cells are removed. The cube construction algorithm has been improved to allow different parameter values for each level (amount of detail) of the grid. Usually the lower levels (higher detail) get a min_n of zero to allow for better grouping of cells in close proximity.

5.8 Performance evaluation

To assess the performance and scalability of the nanocube data structure we perform a series of experiments on varying datasets. We call this version of the nanocube *Eventcube* to emphasize its usage in event detection, as described in the next chapter. As is evident from the description in Section 5.7, the Eventcube can answer three types of queries:

- Spatial distribution - projection of the cube on the $2D$ spatial plane.
- Temporal distribution - projection of the cube on the $1D$ temporal dimension.
- Token distribution - histograms of the counts of each token found in the dataset.

The queries also accept the following types of filters:

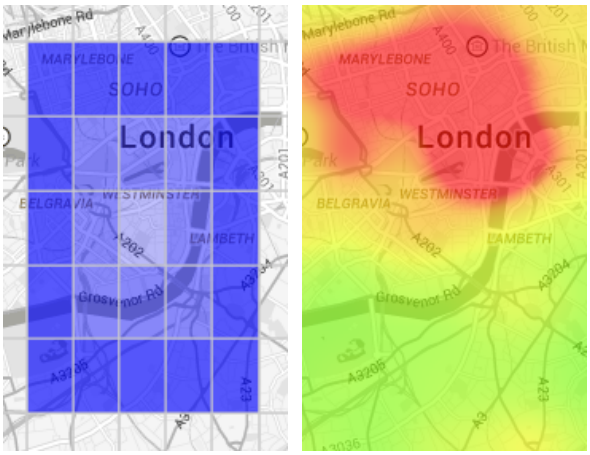
- Spatial filter - a bounding box, denoted by bottom left and upper right corner.
- Temporal filter - a time range, denoted by a minimum and a maximum timestamp.
- Token filter - a specific lexical token and its type, e.g., hashtag, unigram, or bigram.

In our experiments, we compare the runtime performance of the Eventcube with a baseline algorithm that operates directly on the tweets dataset and transforms it into the same output the Eventcube query produces. Each metric (runtime in seconds) is extracted by running each specific procedure 10 times and computing an average. Our experiments test each of the three types of output and each of the three types of filters. More specifically, for each query type (space, time, token) we apply spatial, temporal, spatiotemporal and token filters. The applied filters used in the test scenario are designed to sequentially increase the input size of the dataset.

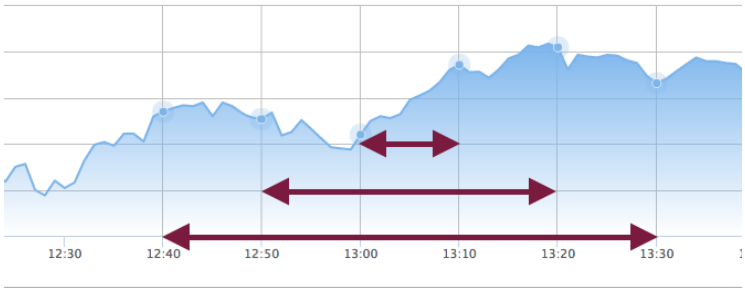
The baseline algorithm operates directly on tweets that can have any value for their locations and timestamps, while the Eventcube uses cubicles which are assigned values from a discretized domain (based on the spacing of the cube). To make the comparison as fair as possible, we make sure that the spatiotemporal filters used do not "slice" cubicles, but rather adhere to the granularity of the cube. Spatiotemporal filters are selected in such a way that they contain only whole cubicles. Spatial filters start as a neighborhood of $k_s \times k_s$ cells and expand e_s cells in each direction. Figure 5.12(a) gives an example for $k_s = 1$ and $e_s = 1$. If the initial test case uses a 1×1 filter, the next one uses a 3×3 and so on. Temporal filters start at a time interval of length k_t and expand e_t time units in both past and future. The red arrows in Figure 5.12(c) represent temporal filters with $k_t = 10min$ and $e_t = 10min$. Token filters are independent of the granularity of the cube. Essentially they contain lexical tokens (hashtags, unigrams or bigrams) that need to be

present in the input data. In the experiments, we select the two most frequent hashtags for each input dataset.

Irregularities can of course arise from a skewed distribution of the data. This has to be taken into consideration in the sense that increasing a filter's dimensional extent does not necessarily increase the input data size in the same way for all datasets as is evident from comparing Figure 5.12(a) and Figure 5.12(b) for the space dimension, and Figure 5.12(c) for the time dimension.



(a) Spatial filter (b) Data intensity



(c) Temporal filter and data distribution

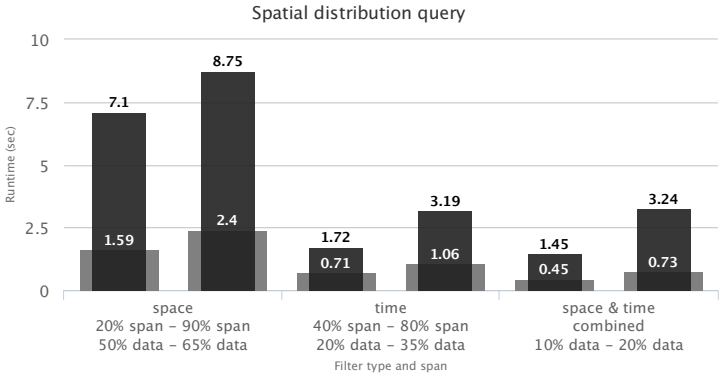
Figure 5.12: *Space and time filters are based on the granularity of the cube.*

The basic metrics extracted are:

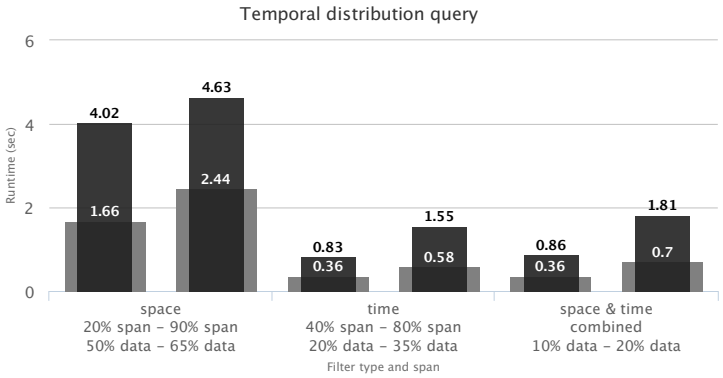
- The performance of the Eventcube compared to the baseline method.
- The scalability of the Eventcube as the input size grows.
- The behavior of the algorithm as the spatial and temporal granularity is adjusted.

All the data is stored in a MongoDB v3 database server. Cubicles are stored at the highest level of detail only, each one having a size of $0.01° \times 0.01° \times 10min$. To manipulate the cubicles in order to answer the queries mentioned in the beginning of this section, we use the MongoDB Aggregation Pipeline Framework, which declares a series of steps that the data flows through and gets transformed to eventually produce the final output. A typical scenario includes a [match → group → sort → limit] pipeline. A limited set of functions allows for data transformation between these steps if needed.

The first experiment involves benchmarking a spatial and a temporal query. The dataset extends approx. $1.5° \times 0.75°$ in space and spans approx. 5 days in time. Figure 5.13(a) compares the running times for a spatial query, while Figure 5.13(b) displays



(a) Space query



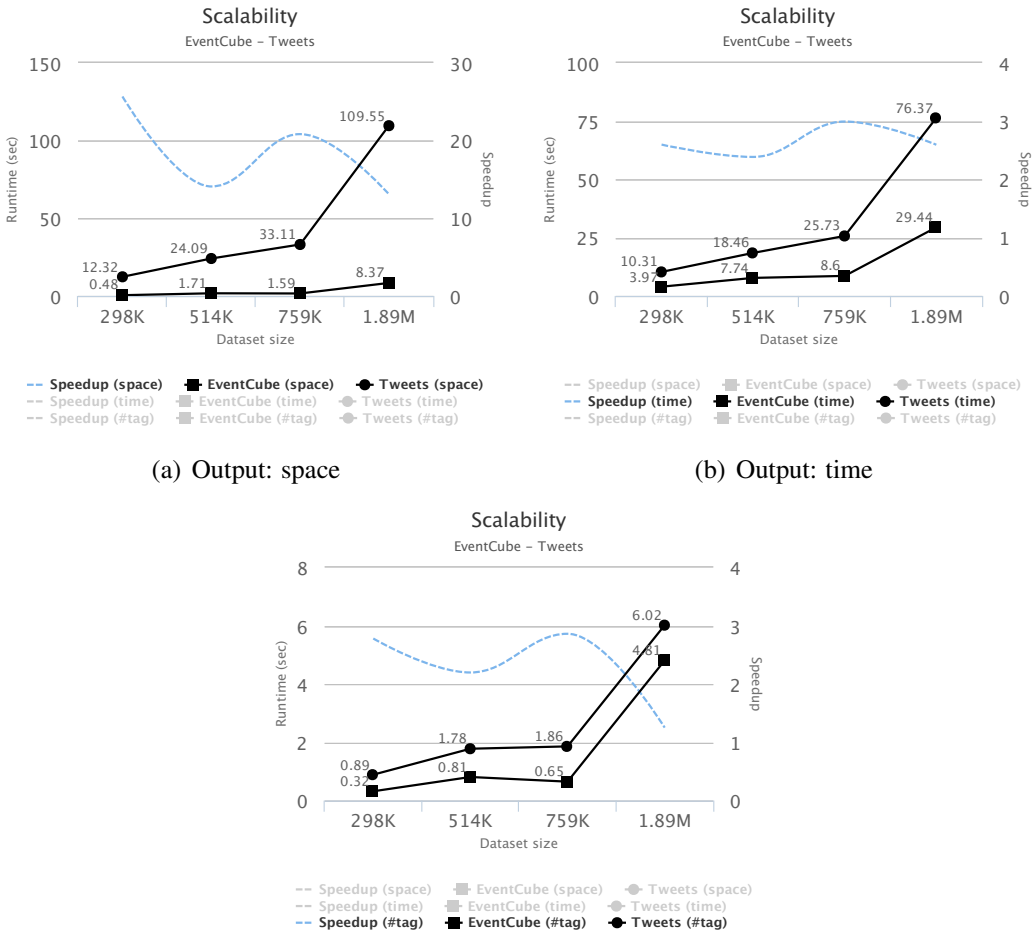
(b) Time query

Figure 5.13: *Performance of the Eventcube (gray) compared to the baseline method (black).*

similar benchmarks for compiling the temporal view. The Eventcube is shown in gray and the baseline method is shown in black. In both query cases, we apply a series of different filters shown as labels on the x-axis. The "data" label displays the actual percentage of the data volume each filter allowed to pass through it. Space filter examples (two leftmost columns) use a bounding box to limit the initial input data. The "span" label displays the area percentage of the bounding box of the filter compared to the bounding box used to collect the data. Temporal filter examples (two middle columns) use a time range to limit the input data. The "span" label displays the duration percentage of the range of the filter compared to the overall duration the data was collected for.

Overall, spatial queries have a higher running time and the advantage of using the Eventcube is more evident. We do not show any experiments for token filters since the Eventcube uses an external inverted list to apply this type of filter; all our experiments showed that this combination has a better performance than the baseline method. It is evident from the figures that the Eventcube outperforms the baseline method in all cases. The speedup can be attributed to the fact that the cubicles collection is a pre-processed/aggregated version of the raw tweet dataset. Essentially during the queries on the raw tweets, additional computational effort is needed to perform the processing and aggregation on-the-fly. What is more, typically the cubicles collection has a document count between 40% - 60% of the total tweets count. As a consequence, the database indexes for the cubicles collection are much smaller in size. Lastly, the indexed fields (timestamps and coordinates) of the cubicles have values within the range allowed by the

granularity of the spatiotemporal grid, while the same fields in the raw tweets dataset can have any value. This leads to a lower cardinality for the cubicles collection indexes, which further increases the query performance.



(a) Output: space

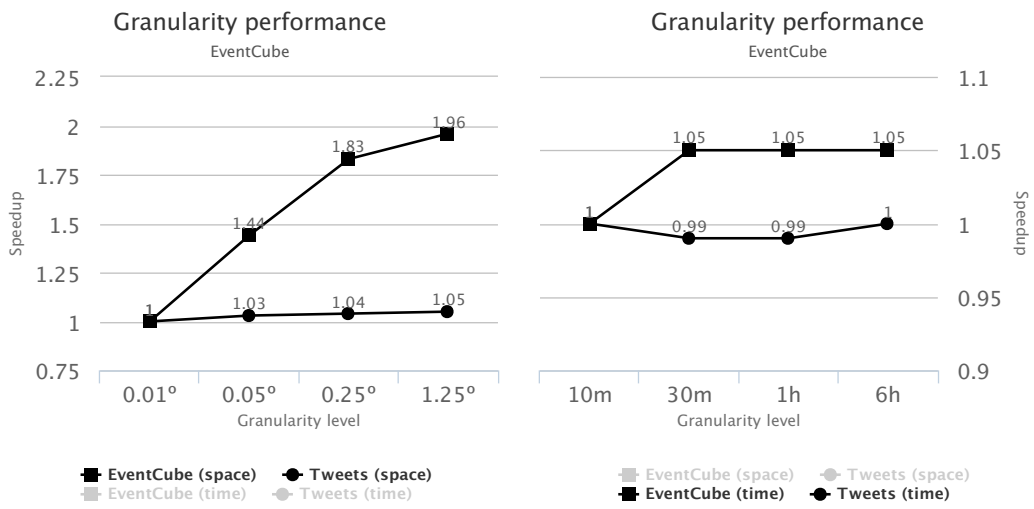
(b) Output: time

(c) Output: hashtag

Figure 5.14: *Scalability.*

Moving on to the scalability tests, we study the behavior of the Eventcube when increasing the size of the input dataset. In these tests we apply no filters and specify no output constraint. The results shown in Figure 5.14 indicate that for spatial queries the Eventcube shows quite a big speedup that gradually declines with a growing input size. The speedup is still > 10 in the case of a 2M tweets dataset. The variance can be attributed to the irregularities of the spatial distribution of the input datasets. The temporal queries achieve a lower speedup of about 2.5 on average, however this speedup is maintained as the input size grows. The hashtag queries get an initial speedup of 2.8 that drops to about 1.3 as the input size increases.

Finally, Figure 5.15 compares the performance gain of the Eventcube when adjusting the spatial and temporal granularity of the output. These results relate to a 2M tweets dataset, spanning $24.6° \times 23.2°$ in space and about 10.5 days in time. We observe a noticeable speedup for spatial queries that approaches 2 as the level of detail decreases. The baseline method does not improve much as the speedup remains very close to 1. For temporal queries, none of the approaches benefits considerably when decreasing the level of detail. The Eventcube shows a slightly improved performance. Still, adjusting the

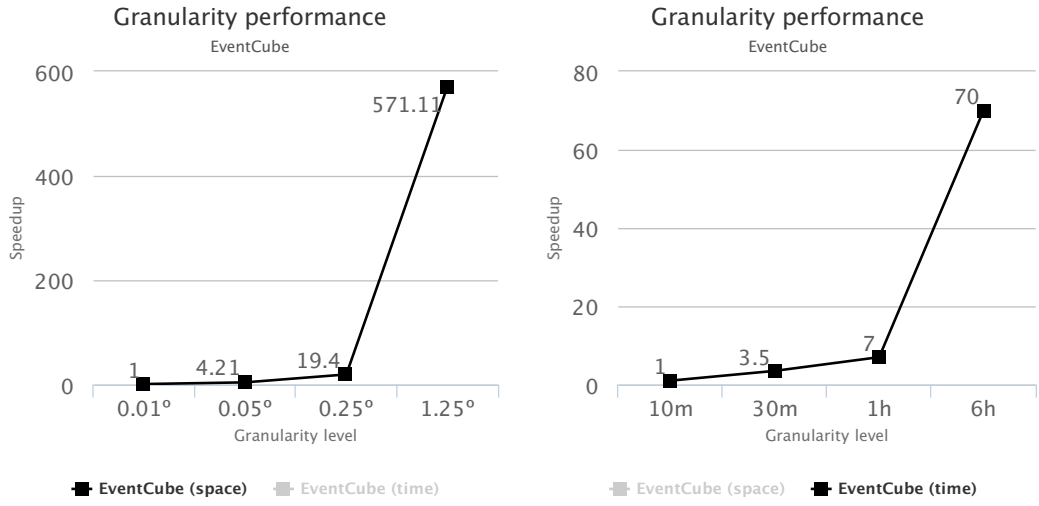


(a) Spatial granularity

(b) Temporal granularity

Figure 5.15: *Speedup as a function of output granularity.*

granularity based on the size of the data is beneficial as it allows for a better representation of the results. For example, as more data is retrieved, a time series that initially shows counts per 10*min* intervals, switches to counts per hour, per day, and so on. What is more important is that higher granularity levels contain less data, thus are faster to serialize and transmit over the network (to browsers, etc.). Figure 5.16 shows a significant runtime reduction and an increase in speedup for the serialization step of the response as the level of detail decreases.



(a) Spatial granularity

(b) Temporal granularity

Figure 5.16: *Speedup as a function of output granularity in rendering output to JSON.*

5.9 Conclusion and Future Work

This chapter introduced a great variety of social media content summary that provide gist views on several of the dimensions that exist in these rich datasets, such as time, space, context, multimedia and social presence. We design these interactive summaries in an interconnected way, to allow better exploration of the dataset. When applying a filter on a dimension, all other views automatically adapt to show relevant data. Two main focuses on all of the summaries are the temporal element (how the dimension values change over time), and the ability to link back to the individual posts, to verify what the summary is displaying.

We also present an efficient data structure used to store aggregated information in spatiotemporal cubes. We benchmark this data structure to demonstrate that it allows efficient querying of the data and that it responds well while the size of the dataset grows, due its adjustable granularity. We plan on improving this concept, by investigate into additional attributes that can be stored in the nanocube, and offer an extra caching layer that would be able to store query results for immediate access.

In terms of future work, we will focus on expanding the gallery of content summaries, improving the connections between them and also showcasing how they can be used for further analysis of the dataset. As such in the following chapter we discuss the problem of event detection by monitoring social media feeds, which is an extensive analysis on the evolution of thematic and spatial views described in this chapter.

Chapter 6

Online event detection

The summaries described in the previous chapter give a quick overview of the content of a streaming social media dataset. They act as a starting point to ignite further analysis on a number of different scenarios. In this chapter we focus on analyzing the spatiotemporal evolution of the thematic summaries described previously in an effort to identify "geo-events", that is events with a discrete spatial footprint. By monitoring the social media traffic that originates from a specific geographic area, we essentially capture the fluctuations in the local terms discussed within it. By quantifying these fluctuations we are able to identify events. The temporal evolution will identify an event's start and finish, and its intensity in-between. The spatial evolution will then rank it as local or global compared to a base line observation area.

This chapter's content was also initially included in our [52] article under review for consideration of publication in the "Social Media Processing" special issue of the ACM TIST journal (submitted January 2016).

6.1 Introduction

As users constantly post snippets of information regarding their thoughts, actions, interests or perception of their surroundings on microblogging web platforms, these streams are becoming an excellent source of up-to-date on-the-ground information. Having millions of sensors on the ground, as this is what essentially the social media users act as, has allowed early reporting of recreational activities (concerts, sports games, ...), breaking news (major accidents, attacks, ...), hazardous natural phenomena (floods, tornadoes, ...), etc. In this work we use Twitter as a source of information, primarily due to its streaming and voluminous nature. In Twitter, hashtags are a means to organize and link discussions. Accordingly, they are a natural choice to act as a *basic set of labels for events*. As mentioned earlier, we use the term *geoevents* to refer to events with a discrete spatial footprint. The assumption is that all events of interest will be denoted and discussed by using hashtags, but not all hashtags correspond to actual geoevents. The focus in the overall approach will therefore be on *identifying the hashtags and the set of locations that represent geoevents*.

Our goal is to identify events that originate from traffic local to a particular geographic area. Once we have identified a hashtag as an event from this local stream, we then compare the local traffic with the overall available traffic produced by this hashtag to rank the event as local or global. The essential elements of our algorithm are:

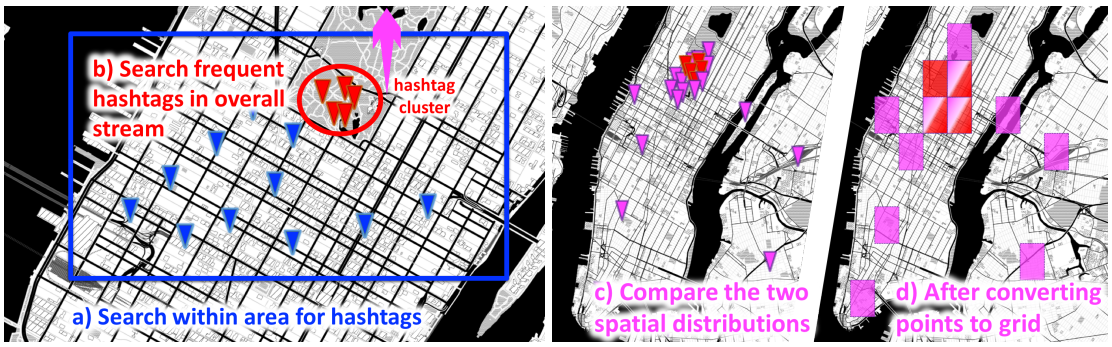


Figure 6.1: *Geoevent search workflow.*

1. to monitor a certain geographic area in order to establish a baseline of hashtags that are typically associated with it
2. to detect new emerging hashtags and examine their overall spatial distribution to assess whether they communicate events that are local to the monitored area (e.g., #U2concert vs. #PresidentialElections)
3. to explore local geoevents further, monitoring their spatial footprint, content, and associated social networks over time

Figure 6.1 illustrates this approach. The blue bounding box represents the Twitter Streaming API query to get posts within an area of interest (e.g., midtown Manhattan). The red cluster consists of geotagged tweets that have a common hashtag. Using this hashtag we now query the Twitter Search API, without any geographic restrictions imposed, to estimate how spatially distributed the hashtag actually is. If it appears only in the close vicinity of the original area then we have found a local geoevent whose location and other semantic parameters we track.

6.2 Identifying events

The concept of geoevent detection, is based on monitoring a specific area of interest and identifying bursty terms [55] [1] from the social traffic within it. In our approach, the first step is to *identify candidate geoevents*. These are essentially hashtags observed at a high frequency in the data stream for a specific amount of time. To monitor hashtag frequency and assign candidacy, we use two parameters, a threshold frequency $\theta_{\#}$ and a duration T of a sliding time window, where $T = n_t T_{min}$. For simplicity we assume that the sliding window actually snaps to T_{min} intervals, i.e., it starts from a multiple of T_{min} and moves in a discretized way, at steps of T_{min}. For example, if the first window is $W_1^t = [t_1, t_1 + T)$ then the next one is $W_2^t = [t_1 + T_{min}, t_1 + T_{min} + T)$. If $n_t > 1$, the windows overlap. Each time an interval of duration T elapses, we calculate the number of occurrences c_h for each hashtag that was observed. The hashtags that achieve a $c_h \geq \theta_{\#}$ are then marked as geoevents. The process continues as time elapses and events may continue into the next time interval or not, depending on the calculated value of c_h for each hashtag for each interval.

By using the thematic content summaries described in the previous Chapter 5, specifically the tags evolution graphs, we can easily identify the hashtags that correspond to geoevents. For each hashtag found in the stream we plot its evolution graph based on the temporal granularity T selected earlier. Then by drawing a horizontal line at $\theta_{\#}$, only the charts that have parts drawn above the line, correspond to geoevents, as long as they also

have a discrete spatial footprint. For simplicity, events that resurface in non consecutive intervals are merged and are considered as one, although the evolution analysis reveals the time periods the event was active. We can then extract starting and finishing times and create a new content summary, called *event timeline summary*, as shown in Figure 6.2.

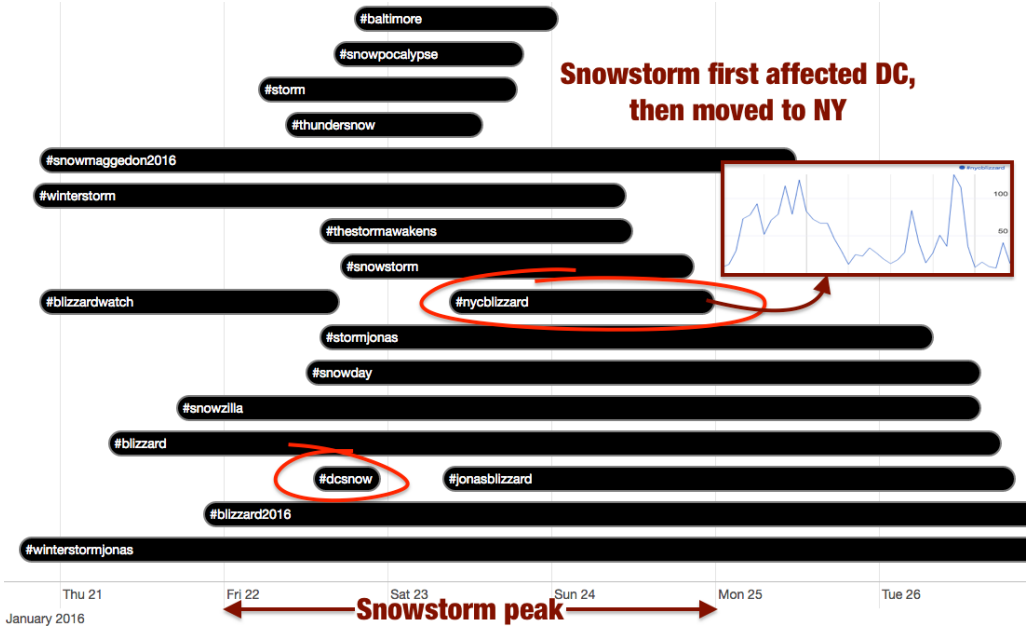


Figure 6.2: *Events identified for the winter storm dataset originally mentioned in Figure 5.1. Event detection parameters:* $T = 1h$ *and* $\theta_{\#} = 30$*. Interestingly we can also observe the time precedence of events, e.g. the storm first affected D.C. and then moved to N.Y.*

Although our detection algorithm maybe simple enough, our main goal is not the actual detection, but the ranking of the events described later on in Section 6.4. The algorithms and methodology used later on for ranking, are independent of the algorithms used for detection, hence, various existing research efforts that tackle the problem of event detection from spatiotemporal streams can be used to obtain different sets of events. The ranking procedure can then rank any of the possibly different event sets.

6.3 Describing events

Once an event have been identified, its description includes its hashtag (label), a start and end time, plus the evolution of the hashtag's mentions over that time window. By taking advantage of additional content summaries from the previous Chapter 5 we can obtain a richer description for the discovered events. By using the hashtag (thematic) and the start and finish time (temporal) as filters, we can inspect how the rest of the summaries adapt to provide information only about the specific event.

The spatial summary will highlight the areas on the map where the event is being discussed. Animating the spatial view over time can reveal the spatial spread of the discussion for the event. The NE summaries will be narrowed down to display only the named entities that were matched in the social media post discussing the event. The NEs are a good indication of what the discussion is about, in case where the hashtag is not self-explanatory. Finally the social summaries will display just the part of the social graph that is relevant to the event. This subgraph, which may or may not have a different topology that the overall graph, will emphasize the most important users in the scope of the event.

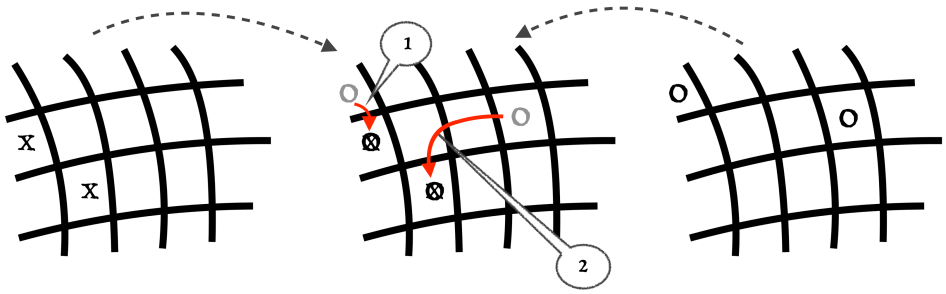


Figure 6.3: *Grid matching concept. Moving the event cells ∘ to overlap the stream cells ×. Calculating the grid match score:* $d_\Sigma = 1 + 2 = 3$, $\overline{d_\Sigma} = \frac{3}{2} = 1.5$.

6.4 Evaluating and ranking events

Geoevents provide us with an understanding of the most frequent terms (hashtags) that are discussed within the monitored area. Next, we want to *discover their spatial extent* and derive whether they are of significance only to this area or appeal to a more global audience. We leverage the Twitter Search API to initiate separate searches for each geoevent (hashtag) without enforcing any geographic restriction. From the geotagged portion of the retrieved data we get an initial estimate of the spatial extent of the geoevent. Running a geocoding process based on the authors profile location for the remaining non geotagged portion of the data further improves this estimate.

To assess if a geoevent has a local or a global spatial footprint, we have to compare the spatial distribution of the data retrieved from the Search API (we refer to it as the *event search*), with the portion of data already acquired from the Streaming API that correspond to this particular geoevent (referred to as the *stream search*) — namely tweets that contain the hashtag that characterizes this event and were published within the same time period the event was detected.

In the case of a local geoevent, the two datasets would have a very similar spatial footprint. If the *event search* retrieves data outside the monitored location, then this is an indication that the event might be global and we have to assess how spatially different the two searches are. The two spatial distributions are point datasets. Point-wise comparing them can prove a complex and time consuming task. To improve and simplify the comparison process we map each point in the dataset to a hierarchical grid data structure (cf. Section 5.7 and Section 3.3), giving each grid cell a degree δ equal to the count of tweets inside its bounds. As such we only need to compare the two grids and not the actual point datasets. Figure 6.1 illustrates the process.

The goal of the comparison is to extract a score for each geoevent that will signify how much the two spatial distributions differ, thus *categorizing the event as local or global*. We initially calculate the minimum sum of the distances required for all cells of the event grid to be moved to completely overlap the positions of the cells of the stream grid. We then divide this with the number of event grid cells to get the average distance traveled by each cell; we call this *grid match score* ($\overline{d_\Sigma}$). Using the average distance instead of just the sum allows for a more fair ranking between events covering few vs. events covering many cells.

Let $M_{g=s}$ be the grid with the stream search data, $M_{g=e}$ be the grid with the event search data, and $|M_g|$ be the count of cells in grid g. If $|M_s| \geq |M_e|$, we will just compute the distances needed for all cells of M_e to overlap the first $|M_e|$ cells of M_s in δ descending order. In case of $|M_s| < |M_e|$, we have to duplicate M_s as many times such that $|M_s| = |M_e|$.

Again, since cells with higher δ are more important, we sort the cells of the stream grid in δ descending order and keep duplicating the sequence until $|M_s| = |M_e|$. Let M'_s denote the possibly modified stream grid. Once the two grids have the same number of cells, our diff score computation problem has been transformed to an optimal assignment problem of cells between M'_s and M_e.

Using the Hungarian method (a.k.a. Kuhn-Munkres algorithm [48, 60]) we can easily compute the optimal assignment cost by just providing a movement cost matrix. In our case the cost matrix is a $|M'_s| \times |M_e| = |M_e|^2$ matrix with each element $(i, j) = dist(M'_s[i], M_e[j])$. Since the distance is computed between the cells of a grid, the most fitting metric is the chessboard distance (a.k.a. Chebyshev distance) $D(i, j) = max(|i_x - j_x|, |i_y - j_y|)$. The Hungarian method produces an output in the form of pairs $(M_e[j] \rightarrow M'_s[i])$ signifying that the optimal movement was to move tile j from M_e to the position of tile i of M'_s, which will eventually cost $D(i, j)$. The cost can be obtained from the initial input cost matrix and does not have to be recalculated. Summing up all the costs for each tile gives us the overall movement cost. Finally, we divide it by $|M_e|$ to retrieve the average travel distance of the cells that is needed for the match. An example of the concept is show in Figure 6.3.

In the hierarchical grid, upper levels combine cells from lower levels to present zoomed-out views of the spatial distribution. Thus, the upper grid cell count is much smaller making computing the diff score much more efficient. This helps in situations where performance is critical as it provides an inexpensive computation of the grid diff score for the higher levels and on-demand drilling down to increase detail as needed.

A more interesting comparison approach progressively compares snapshots of the two grids for predefined time intervals so to observe changes in the diff scores. Local events are expected to have a low variance. However, events that initially had a limited spatial footprint, but later attracted global awareness will display a notable variance; we call these events *transitional* (transitioned from local to global). When studied further these events can indicate how topics propagate through space over time.

The ranking of geoevents as local or global is complemented here with several examples. The following three types of geoevents were observed:

- geoevents with a local footprint throughout the event *(local)*
- geoevents with a global footprint throughout the event *(global)*
- geoevents that started as local and eventually turned global *(transitional)*

		grid match score					
event	(area)	level 0		level 1		level 2	
#GoKingsGo	(Los Angeles, CA)	0	34×23	0	7×5	0	2×2
#closeFCPS	(Washington D.C.)	1.71	42×41	0.24	9×7	0	2×1
#ElectionDay	(London, UK)	246.06	66×75	49.27	11×5	10.43	2×2
#SuperBowl	(Phoenix, AZ)	2909.76	20×9	584.12	1×1	117.29	1×1
#SydneySiege	(Sydney, AU)	4909.22	49×33	1471.00	10×5	294.17	2×2
#BaltimoreRiots	(Baltimore, MD)	7015.94	62×27	1403.25	4×2	280.69	1×1

Table 6.1: *Grid match scores of the discovered geoevents.*

The nanocube used in the following experiments has a base grid with $0.01^\circ \times 0.01^\circ$ spacing and a grouping factor of $a = 5$ to aggregate cells to a higher level (5×5 cells are combined to one larger cell). We calculate grid match scores for levels 0, 1 and 2. In the following maps, where we visualize some of the example geoevents discovered, we use blue markers or cyan rectangles for the *stream* search data, and red markers or magenta circles for the *event* search data. The markers are used when events span large extents, since they maintain their size regardless of the map's zoom level.

6.4.1 Local events

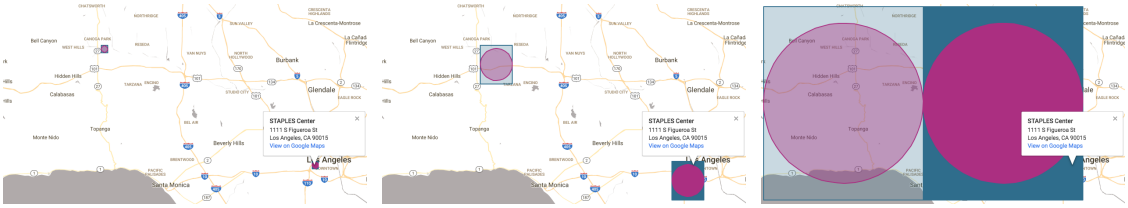
Typically *local events* are of importance only to a small community, area or city. Examples include sport events, music concerts, contained weather phenomena, etc. Our first example is a purely local event, as the event search data spatially coincides with the stream search data. The event is #GoKingsGo and refers to the hockey game between the L.A. Kings and the Edmonton Oilers held in the Staples Center in Los Angeles (Nov. 14, 2015). The grid match score for this event is 0 for all levels of the hierarchical grid. Visual confirmation is given in Figure 6.4(a)-6.4(c) as cyan cells coincide with magenta circles. Another example of a local event observed in Fairfax County, VA is described by #closeFCPS, the closing of public schools due to heavy snowfall (Jan. 6-7, 2015). The spatial distributions of the stream and event search data is shown in Figure 6.4(d)-6.4(f) at different levels of detail. As shown in Table 6.1 the match score for this event is low, compared to the spatial extent of the stream search data.

6.4.2 Global events

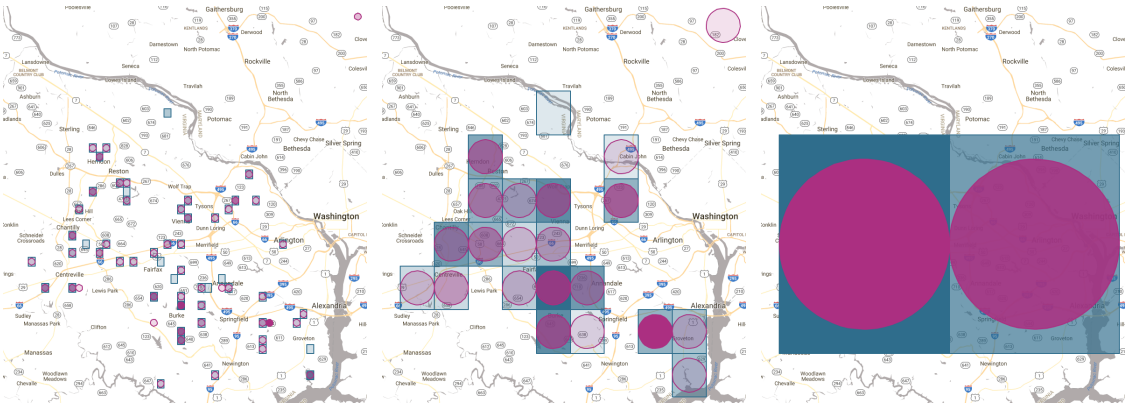
Global events are on the other hand relevant to a larger audience and area, such as countries or even continents. For example the UK Elections in May 2015 had an appeal beyond the monitored area of London (cf. Figure 6.5(a)). However, when assessing locality it is important to consider the comparison area. For example, the election event was global in relation to the monitored London area, since the event was discussed all over the United Kingdom. It was however local to the UK as not many people outside the UK posted relevant content. Similar conclusions can be drawn from studying the Super Bowl XLIX game that took place in Feb. 2015 at the University of Phoenix Stadium in Arizona. We monitored the area around Phoenix, but the event appears to be discussed all over the US (Figure 6.5(b)). Thus, it can be categorized as global, given that our monitored area is the city of Phoenix. Lastly, Figure 6.5(c) shows a global event with worldwide coverage, namely the World A.I.D.S. day on Dec. 1, 2015.

6.4.3 Transitional events

A rather interesting category are events that are considered global, but when studying the temporal evolution of the grid match score, we can see that *initially the event had a relatively local footprint* and as time elapses it reaches more coverage. These events are usually non-scheduled and unpredictable, e.g., natural disasters. They can be identified as events with a high variance in grid match score values. To identify this type of events we need to compute the grid match score using a progressing time window that filters the input data as it gradually moves to cover the entire period the event was active. It is always interesting to spot steep increases or decreases in the grid match score values, and try to

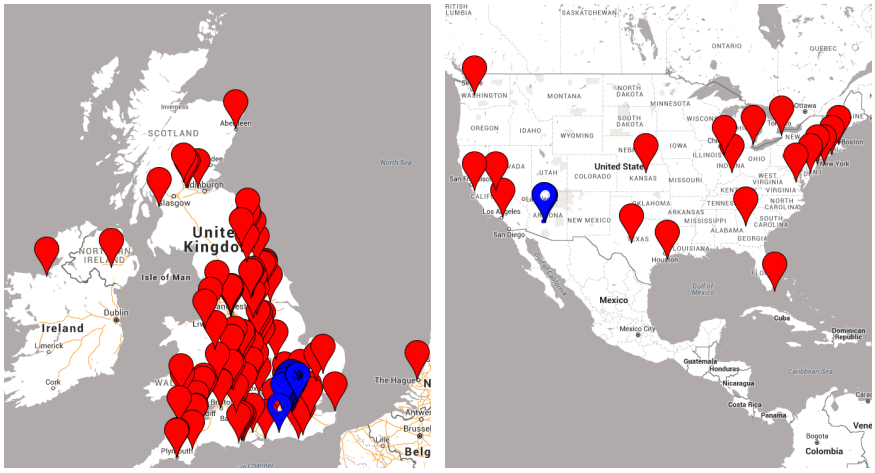


(a) #GoKingsGo (level 0) (b) #GoKingsGo (level 1) (c) #GoKingsGo (level 2)



(d) #closeFCPS (level 0) (e) #closeFCPS (level 1) (f) #closeFCPS (level 2)

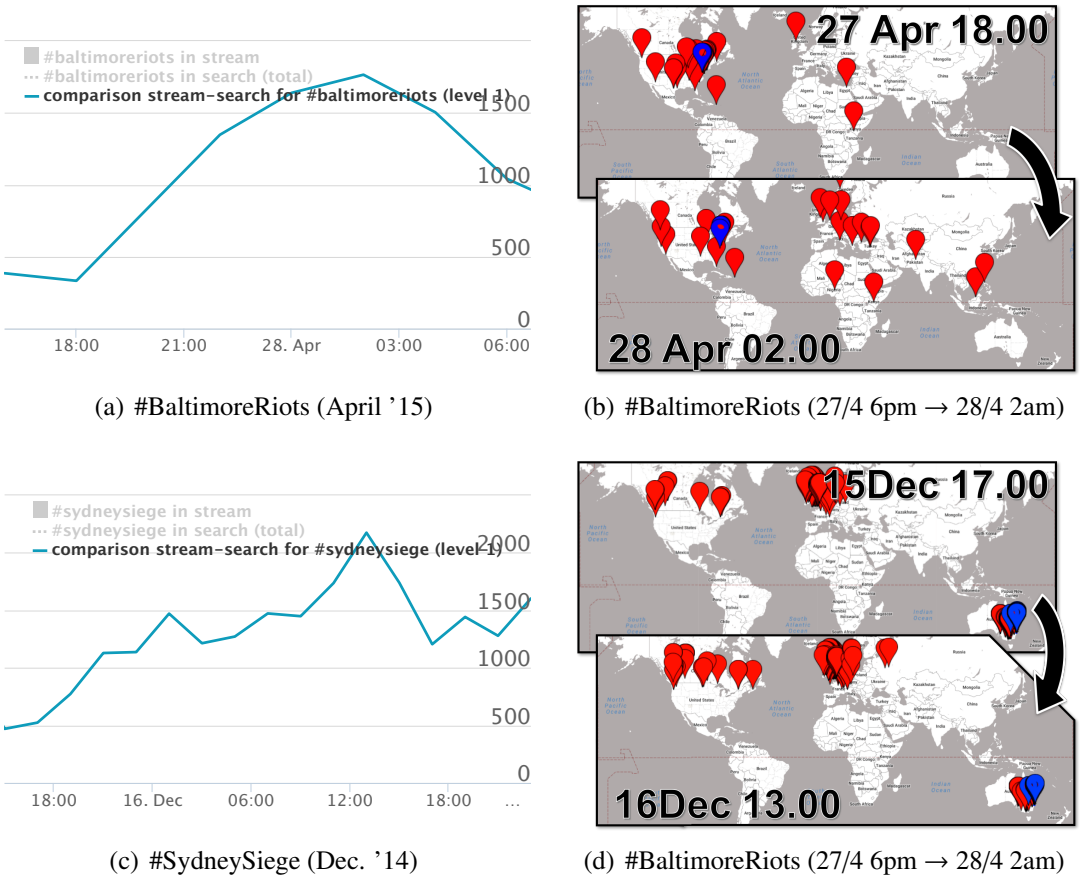
Figure 6.4: *Local geoevents*



(a) #ElectionDay (May '15) (b) #SuperBowl (Feb. '15)

(c) #WorldAIDSDay (Dec. '15)

Figure 6.5: *Examples of **global** events*



(a) #BaltimoreRiots (April '15)

(b) #BaltimoreRiots (27/4 6pm → 28/4 2am)

(c) #SydneySiege (Dec. '14)

(d) #BaltimoreRiots (27/4 6pm → 28/4 2am)

Figure 6.6: *Temporal evolution of the grid match score. Times are local to the event.*

understand the cause of these spikes by resorting to the rest of the content summaries described in Chapter 5.

One example, for which the evolution of the match score is shown in Figure 6.6(a) are the riots in the city of Baltimore (#BaltimoreRiots) in April 2015. Early in the evening the match score was in the $\approx$ 500 range and tweets appeared mostly near Baltimore, a few in the rest of the United States, and very few abroad. Later that night, as awareness grew, the event reached a score of $>$ 1500 and tweets appeared all over Europe as well (Figure 6.6(b)).

Similarly, in case of the Sydney hostage crisis on December 2015, for which the match cost evolution graph is plotted in Figure 6.6(c), the match score jumped from $\approx$ 500 in the evening to nearly $>$ 1500 during night time as events progressed towards a resolution of the siege (Figure 6.6(d)), and even more during the next day, as the world started reporting the updated news.

Both charts show the evolution of the grid match score using a $4h$ sliding window progressing $2h$ at each step. The diff scores are computed at level 2 of the hierarchical grid, which corresponds to 0.25° per distance unit.

Table 6.1 summarizes the events discovered and their overall grid match scores, for each of the first three levels of the hierarchical grid. Next to each score is the spatial extent of the stream grid measured in grid cells at that level. Local events are defined as events that the grid match score is less than the maximum dimension of the stream grid extent, while global events capture all the rest. Transitional events can be discovered by studying

the temporal evolution of the grid match score, as demonstrated earlier in Figure 6.6(a) and Figure 6.6(c).

6.5 Conclusion and Future Work

With respect to the established field of event detection, the contribution of this work is the presentation of a methodology that allows one to deduce the spatial footprint of the event and classify it as local or global, compared to a base area. Throughout our study we assumed that a geoevent is described by a specific hashtag. However, experimentation revealed that typos and/or term coining by various people, results in several hashtags relating to the same event. In ongoing and future work, we will try to identify all the alternative hashtags describing an event and to combine/fuse the attached data to study each event with a larger compiled corpus of posts.

With the introduction of transitional events, we got an indication of how information spreads through space. That is essentially the temporal evolution of location indicators when grouping the social media traffic by the tags that form the events. In the next chapter, we shift the focus from tags to users, and try to infer patterns in the users movements. Initially we group location by user to obtain paths, then we group by path legs to check if patterns emerge.

Chapter 7

Paths and links of interest

Grouping together social media posts by the tags included in them allowed us to identify events as described in the previous chapter. In this chapter we explore another interesting grouping using the social aspect. The embedded user information within the social media traffic, allows for inferring single user paths whenever geospatial metadata is also available. By snapping these paths to the spacing of the hierarchical grid introduced previously we try to convert single user paths to aggregated movement patterns between places, and assign meaning to them. Finally we interchange the hierarchical grid spacing with the regions of interest described in Chapter 4.5 and attempt to link places and regions, in an effort to discover patterns in users movements.

This chapter's content was initially included in our [52] article under review for consideration of publication in the "Social Media Processing" special issue of the ACM TIST journal (submitted January 2016).

7.1 Introduction

By tapping into the richness of UGC, ranging from web blogs and forums to social media, we have the opportunity to transform geospatial data collections by *capturing meaning in addition to geometry*. So far we have tried to develop methods to extract geospatial content in order to enrich the descriptions of places. These places-of-interest (PLOIs) are a natural evolution of traditional point-of-interest (POI), and reflect the assignment of meaning to locations by means of human activities and events. As these activities and events are dynamic, so too should be their respective representation in a place database. With that in mind, to provide an even richer description, we want to capture and model the linkage among places by generating links-of-interest (LOIs). These links capture the semantics of relationships among places given by the people that travel along them. Combined, places and links comprise a *time-parameterized Web of Places* that captures both the meaning assigned to locations as well as links among them. In a sense the outcome of this work, will enhance our content summaries described in Chapter 5 by adding a "Movement" dimension to the summaries. Additional analysis can reveal patterns in movements and provide the reasoning behind these people flows. Links of interest can connect different types of places, such as highly dense (with respect to the amount of traffic they generate) arbitrary areas, to other less dense arbitrary areas, to well-defined places such as public transport stations, to custom regions of interest (cf. Section 4.5). The rest of this chapter describes how to extract user paths from the social media traffic, and how to combine them to discover links of interest between places.

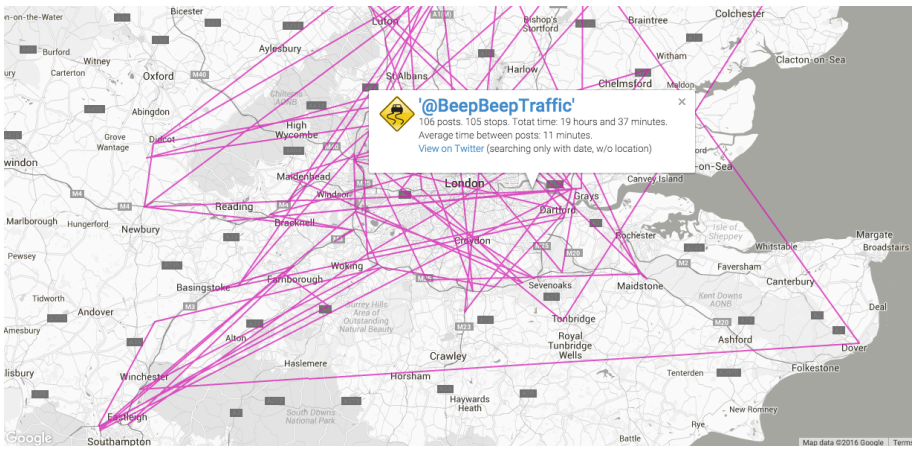


Figure 7.1: *"Path" of traffic incidents in London.*

7.2 Paths summaries

As people move from place to place and share geotagged posts on social media networks, they give a semantic context to their route, which is a link between their origin and destination. Part of this work is to identify meaningful connections between places based on users activities in each place during their journey. Since posts are tagged with author information, grouping them by their author id and accumulating the locations and timestamps in time ascending order is a way of constructing *candidate paths* (Algorithm 3).

Quite often automated systems (bots) and sensor networks use social media to post updates about incidents in various locations. A common example is an earthquake monitoring system reporting seismic activity by posting a status update with information about the phenomenon. The post usually gets geotagged with the epicenter of the earthquake. Other examples include traffic incident reports (cf. Figure 7.1), and news agencies posting geotagged story headlines at different places. These cases can add significant noise to our analysis, since the locations joined by these accounts are not relevant to actual people's movements. An attempt to exclude such paths is made by adding an additional step to filter path segments based on the average travel speed of each segment. We assign a speed value to each segment based on travel time (difference between the timestamps of the posts in each location) and distance (Euclidean, Haversine, etc...) between the locations. If a higher accuracy is required, a more precise speed can be calculated by obtaining travel times and distances via a routing/navigation service. The actual plausibility decision is abstracted in a `isValidTravelTime(distance, time)` function, so that different cases can implement a methodology suitable for their needs. Once we have calculated travel speeds for all path segments we have to possible choices: we can either choose to eliminate only the segments that fall outside of the valid speed bounds that we set; or we can calculate the count p_v of segments with valid speeds (within the specified range) per user account and given a threshold θ_p we will completely discard all of the user's paths if the percentage of invalid segments is higher than θ_p. The latter method tries to identify and eliminate bot accounts by counting the number of times they "travelled" at invalid speeds. Once we have identified these accounts, we can completely discard all of their path segments, as our goal is to filter our path data to only actual people accounts. After cleaning the path dataset, the remaining paths correspond to single user movements over time.

7.3 Identifying links of interest

Single user paths are great for precision, but cannot be easily interpreted as links that connect places, because of the finite granularity level of the coordinates in their origin and destination. Essentially each path segment connects two pairs of coordinates and not two places. To complement the movement summaries and derive links-of-interests we first have to identify important locations, i.e., places of interest from the trajectories and their points.

In the following we present various different ways to build stronger links of interest, by combining similar path segments. The similarity of the path segments is derived by spatial restrictions enforced on the paths origins and destinations. We present three types of groupings that lead to three types of links.

7.3.1 Flat links

In the simplest form we can derive places from a spatial grid overlaying the area of interest, by mapping places to grid cells. Cells that gather enough traffic can be considered as places in the broader sense, given meaning by the increased people's activity within them.

We employ a trajectory generalization technique that utilizes a spatial grid segmenting the space occupied by the paths. Each path origin and destination is "moved" to coincide with the center of the grid cell it falls into. This way path segments that have a common origin cell and destination cell can be merged, increasing the (aggregated) path segment's importance. We refer to such merged segments as links between two locations. The importance of a link is judged by the number of segments it aggregates. This means that more users have traveled along it, and thus the connection between the places it connects must be more important. However, in doing so we run the risk of eliminating path segments that have their origin and destination within the same cell. This can be minimized by choosing a small initial cell size for the grid. To implement this movement summary we use a graph structure, in which edges model links, and nodes model places, i.e., the cell centroids (cf. Algorithm 4). The `SnapToGridXY(coord)` function converts the initial point coordinates (individual post location) to the center of the respective grid cell by truncating each coordinate to the nearest multiple of the grid's spacing (cells are squares). Assuming a grid has a spacing equal to c_L, then $coord' = \lfloor \frac{coord}{c_L} \rfloor \cdot c_L + \frac{c_L}{2}$. Each time a segment with the same origin and destination is added, the weight of an edge is increased by 1.

(a) Single paths (b) Flat links (c) Hierarchical links

Figure 7.2: *Single and grouped paths during the April 2015 Baltimore protests.*

ALGORITHM 3: Build User Paths from Social Media Posts Collection

Input: The geotagged social media posts collection (*posts*) and optional filters (*criteria*).

Output: A list of timestamp-location pairs associated with each user id found.

$criteria = criteria \triangleleft \{coordinates \neq NULL\}$

$$pipelineSteps = \left[\begin{array}{c|c|c} \textbf{match} \rightarrow & \textbf{sort} \rightarrow & \textbf{group} \rightarrow \\ \hline criteria & \left.\begin{array}{r} \$id \\ \textbf{or} \\ \$timestamp \end{array}\right\} \Rightarrow \texttt{ASC} & \begin{array}{r} id = \$userId \\ length = \{\$\texttt{SUM} \Rightarrow 1\} \\ time = \{\$\texttt{PUSH} \Rightarrow \$timestamp\} \\ location = \{\$\texttt{PUSH} \Rightarrow \$coordinates\} \end{array} \end{array}\right]$$

$paths = \texttt{Aggregate}(posts, pipelineSteps)$

ALGORITHM 4: Build paths graph

Input: A list of user paths (*paths*).

Output: A graph representing flat paths.

$graph = \{nodes : \emptyset, edges : \emptyset\}$
foreach *path* **in** *paths* **do**
  for $i = 1$ **to** $|path| - 1$ **do**

$$edge = \begin{cases} origin : \texttt{SnapToGridXY}(path.location[i-1],\ level = 0) \\ destination : \texttt{SnapToGridXY}(path.location[i],\ level = 0) \\ timeStart : path.time[i-1] \\ timeEnd : path.time[i] \end{cases}$$

    $edge' = \texttt{LookupEdge}(graph, edge)$
    if $edge' \neq NULL$ **then** $edge.weight = edge'.weight + 1$ **else** $edge.weight = 1$
    $graph.edges \triangleleft edge$
    $graph.nodes \triangleleft [edge.origin, edge.destination]$
  end
end

ALGORITHM 5: Hierarchical Paths

Input: The paths graph (*graph*), and a threshold for combining path segment endpoints (θ_o).

Output: A graph representing hierarchical paths.

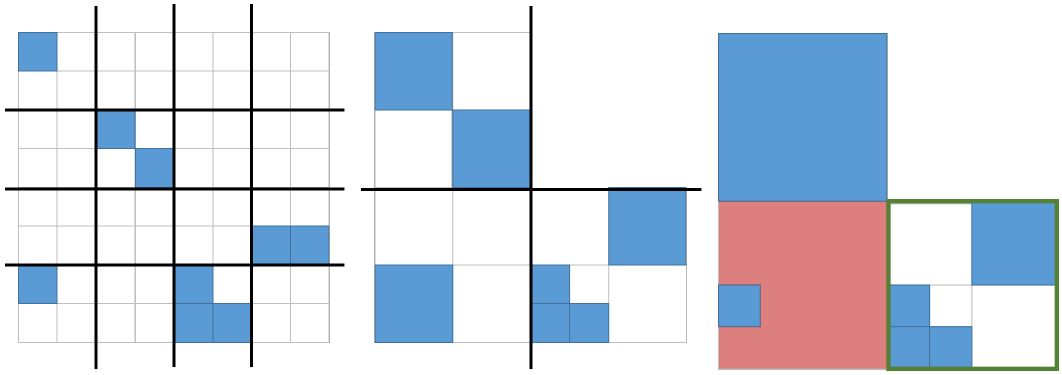
Function `BuildHierarchicalPaths`(*graph, level,* θ_o)
  $nodeSet = \texttt{GetNodesByAttributes}(\{\texttt{level} : level - 1\})$
  foreach *node* **in** *nodeSet* **do**
    $parentNodeId = \texttt{SnapToGridXY}(node.id, level)$
    $parentBbox = \texttt{GetBoundsOfGridCell}(parentNodeId, level)$
    $loopNodeSet = \texttt{GetNodesWithin}(graph, parentBbox)$
    $cluttered$ = FALSE
    foreach *node* **in** *loopNodeSet* **do**
      if $node.level \neq level - 1$ **then** { $cluttered$ = TRUE ; break }
    end
    if not *cluttered* **and** $|loopNodeSet| < \theta_o$ **then**
      $parentNode = \texttt{MergeNodesToSingleNode}(loopNodeSet)$
      if $parentNode.edges == \emptyset$ **then** `RemoveNode`(*graph, parentNode*)
    end
  end
  if $nodeSet \neq \emptyset$ **and** $level < level_{max}$ **then** `BuildHierarchicalPaths`(*graph,* $level + 1$, θ_o)
end

7.3.2 Hierarchical links

In densely populated areas, e.g., city centers, it makes sense to examine paths using a finer granularity due to the higher volume of distinct PLOIs, and respective transportation modes (walking, dense public transportation network, slow vehicle speeds). In less dense areas however, we can trade origin/destination precision for a higher degree of aggregation, i.e., more significant links. To accomplish this we need to combine the grid cells that are the origins and destinations of the flat LOIs, wherever the enclosed traffic is below a certain threshold, in an attempt to grouped together sparsely distributed less dense areas. Borrowing concepts from the ideas in [38] we use the following steps. Initially all grid cells are of equal size c_0 (cells are $c_0 \times c_0$ sized squares). We refer to this as the base or zero level L_0 (flat LOIs).

Given an integer expansion factor a, the grid cells of the mesh, can be combined to form larger cells using $a \times a$ cells from the original mesh, recursively building higher (more abstract) levels with larger cells. The decision of whether to combine the cells of a lower level into a bigger cell of the level above is taken based on an occupancy threshold θ_o, where if a neighborhood of $a \times a$ cells has θ_o or more cells occupied it is left ungrouped to preserve the needed detail, while if it includes less than θ_o cells it is grouped and all links with at least one endpoint connected to any of the merged cells, have that endpoint moved to the center of the new supercell. Besides the occupancy threshold θ_o, we apply another rule that preserves necessary detailed regions and also helps with decluttering the visual representation of the graph. Neighboring cells of level L_i will not be combined in a bigger cell of level L_{i+1} if that bigger cells also encloses any cells of L_{i-1}. An example is shown in Figure 7.3, with $a = 2$ and $\theta_o = 2$. The lower right quadrant of cells of level 1, does not get combined to form the bigger green cell in level 2, since that green cell would also contain cells from level 0. This process executes recursively until cells of defined maximum level are built.

As an additional improvement, we also track the amount of cells that get aggregated each time to a bigger cell. This helps in the case where a cell of level L_i managed to reach level L_j ($i < j$), while being the only one in its neighborhood. This can happen quite often since the occupancy threshold θ_o only imposes a max limit. By tracking this information, we can revert these bigger cells of L_j back to L_i, since they didn't manage to attract and absorb any additional cells, to justify their existence in the upper levels. Once the maximum level is built, every cell that has been marked as originating from



(a) Level 0 (b) Level 1 (c) Level 2

Figure 7.3: *Combining grid cells to form upper levels.*

only one cell of a lower level, gets reverted to that lower level and smaller size. This is shown in Figure 7.3, where the cell in the lower left corner grows from level 0 to level 2 and then reverts to level 0 again, since it never managed to absorb more than one cells in its neighborhood. This procedure is described in pseudocode in Algorithm 5. An actual example concerning the April 2015 Baltimore protests dataset is shown in Figure 7.2.

7.3.3 Connecting places and regions of interest

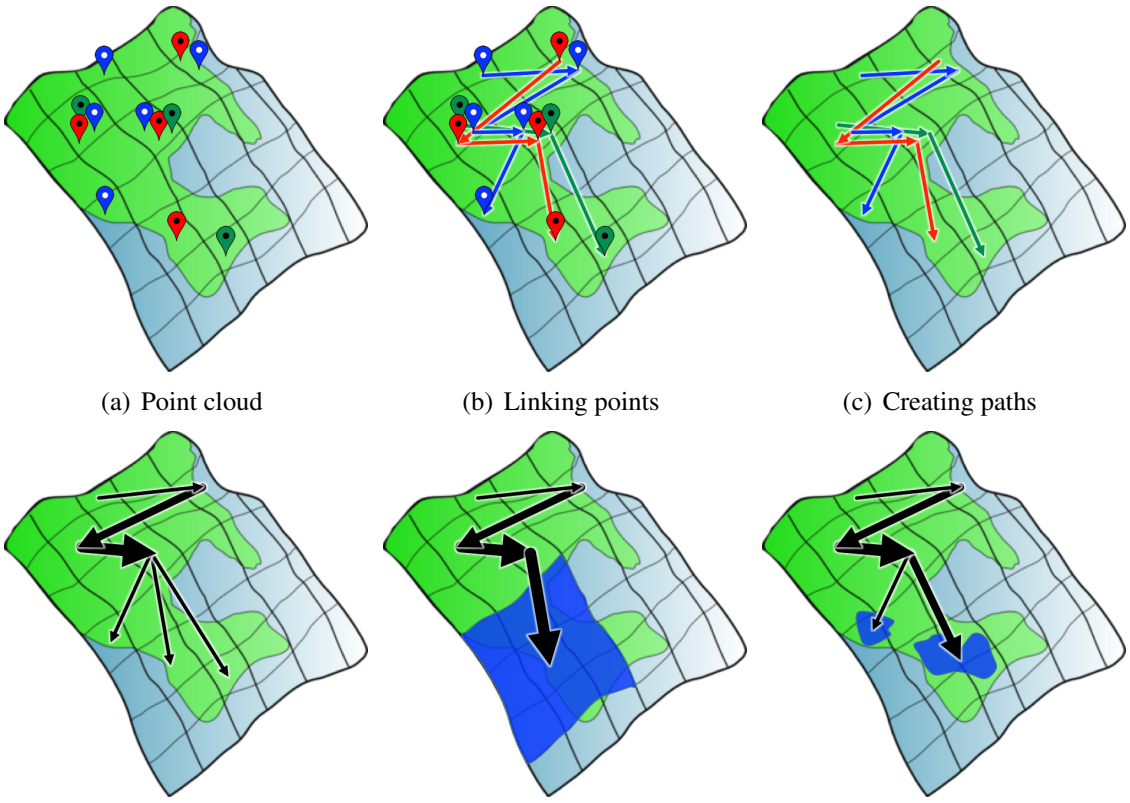
In Section 4.5 (originally in [53]) we presented a way to discover regions of interest by clustering together POIs based on their spatial proximity and textual similarity of associated tags. This extension describes an approach to connect these regions either with each other or with other types of places using links of interest. Essentially our effort is to build a graph, where the edges are derived from users publishing posts while moving in space, same as previously, while the nodes (places) include not only arbitrary locations (cells specified by the partitioning of the grid) but also different types of regions, such as administrative regions from authoritative sources or, more specifically in this case, the regions of interest computed in Section 4.5.

The construction of the graph is building upon the *hierarchical links approach* presented in the previous Section 7.3.2. This new approach uses different datasets for the nodes and the edges. The nodes are enriched with the regions that were found by processing the combined POI datasets collected from DBpedia, OpenStreetMap, Wikimapia, Google places, Foursquare, and Eventful, in addition to the standard cells of a grid overlaying the spatial plane. The edges which are sourced from the social media traffic, are in fact a network of user paths over time. Deriving the links from the social media point dataset can be viewed as map construction task, in the sense that the result is a network of "roads" the social media users use to navigate through the greater area of interest. The process involves four steps:

1. Associate the precomputed regions of interest with geo-tagged posts within them
2. Create the single user paths as described earlier
3. Split each user path into distinct segments
4. Aggregate the segments based on temporal and spatial features such as:
   - which time interval within a day the origin and destination timestamps belong to
   - which of the precomputed regions contains the origin (or the destination)
   - which of the grid cells contains the destination (or the origin)

The process is visualized in Figure 7.4. The different marker colors in Figure 7.4(a) correspond to different users. Figure 7.4(b) draws a line to connect sequential markers (posts) of the same color to build the paths in Figure 7.4(c). The links are then aggregated based on their origin and destination, which can be either one of the precomputed regions of interest (Figure 7.4(f)) or a grid cell (Figure 7.4(e)).

The regions have been computed for each of the following categories of POIs: Culture, Education, Entertainment, Food, Places, Professional, Services, Shops, Travel Transport, Athletics Sport. The goal is to identify movement patterns for specific region categories and specific time intervals during a day. To this end we split the day into four 6-hour intervals: 4am - 9am, 10am - 3pm, 4pm - 9pm and 10pm to 3am. Then for each category and each interval we compute the links that originate from a region of that category and the links that end within a region of that category. This way we break down the incoming and outgoing traffic for each category and study them individually.



(a) Point cloud (b) Linking points (c) Creating paths

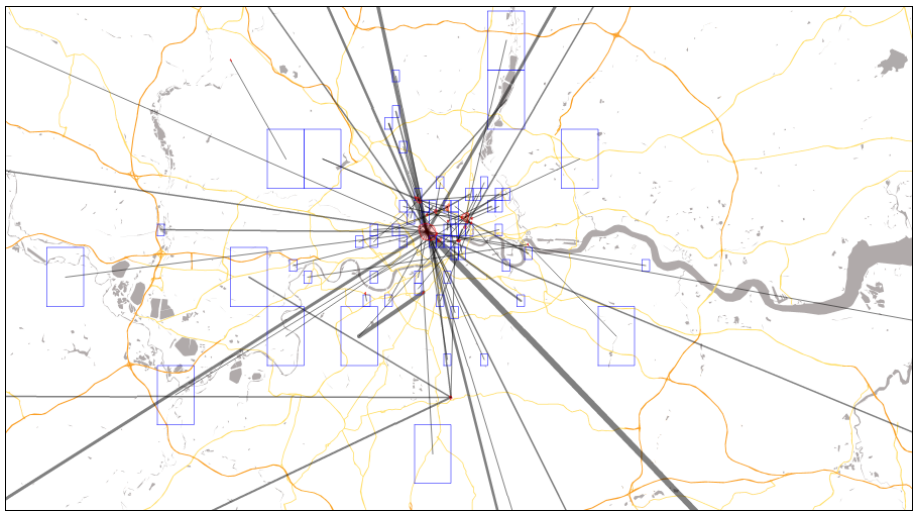
(d) Aggregating path segments (e) Grid cell endpoints (f) Custom region endpoints

Figure 7.4: *Constructing links between regions of interest and arbitrary locations.*

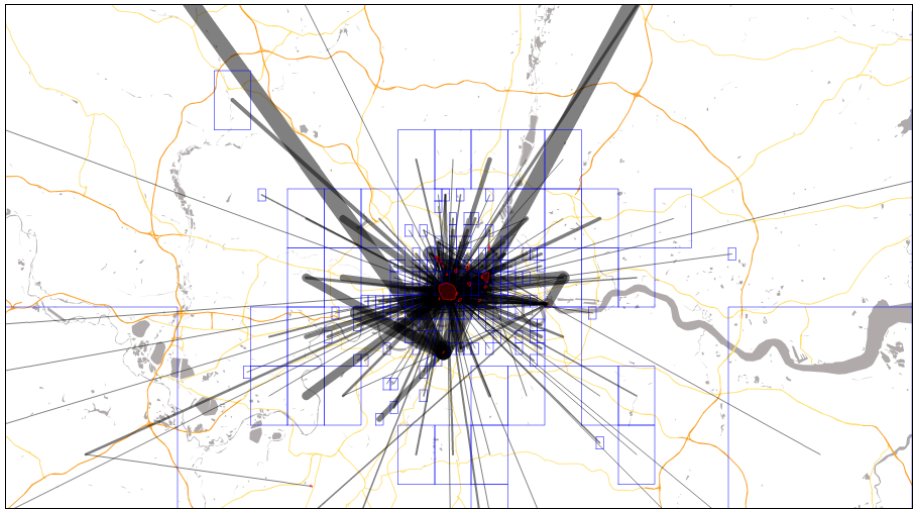
Borrowing from the original links of interest concept (Section 7.3.1) we initially build a similar graph using the social media posts collected over the area we are interested in. Geotagged posts are grouped by their user id, and the attached locations are accumulated in a timestamp ascending order to form each user's path. Subsequently we break each path into its separate segments by choosing each consecutive coordinate origin-destination pair, preserving all the embedded post information for each endpoint.

The next step involves associating all path segments endpoints in the dataset with available regions of interest. As a pre-processing step, for all regions of interests we run containment queries on the social media posts dataset. The unique ids of the posts that are contained within a region are attached to the region object. Reversely, the posts and thereafter the path segments are assigned the category of the region. Since regions of different categories can overlap, a post or a path segment endpoint can have more than one category assigned to it.

Once the path segments have been assigned categories we can start building the graphs. Each graph is identified by three parameters: the category of the regions of interest, the time interval within the day, and whether it includes incoming or outgoing traffic with respect to the regions. Depending on whether we are looking at incoming or outgoing traffic, we select all path segments that either have destinations or origins with the specific category we are working with. For example, in the case of incoming traffic for the "Entertainment" category, all path segments with their destination tagged with that category are selected. For each segment in this subset it is a fact that the destination is contained within a region. What is left to be found is if the origin is also contained within a region or not. If both the origin and the destination are contained within regions then the



(a) Day (10am - 3pm)



(b) Night (10pm - 3am)

Figure 7.5: *Traffic incoming to the "Entertainment" regions in London (from late February 2016 until early April 2016).*

regions are added in the graph as nodes that are connected with a single edge. As further paths segments connecting the same regions are found in the subset, the edge connecting them increases its weight accordingly. On the other hand, if the origin of the segment is not contained within a region of the same category, then that endpoint is assigned the cell that encloses its coordinates, chosen from a rectangular mesh that covers the greater area of study. Thus, this time an edge connecting a grid cell with a region is added to the graph. This technique allows us to group segments with origin endpoints outside of the regions but in close proximity to each other. Again the more segments that connect the cell with the region that get discovered, the more the weight of the edge increases.

To get better results, we use the selected time interval to further restrict the path segments, by splitting each day to four 6-hour intervals as described earlier. This filtering also helps eliminating unimportant links, where the time difference between the origin post and the destination post is too long apart. Here depending on the direction of the traffic analyzed we use the following filter rule: for analyzing traffic incoming to the regions of interest of a category during a time interval T_w, we select all path segments that

the destination post was published within $[T_w, T_{w+1})$ and the origin post was published within $[T_{w-1}, T_{w+1})$. Similarly, when analyzing outgoing traffic during T_w, the filter allows all path segments with an origin post published within $[T_w, T_{w+1})$ and a destination post published within $[T_w, T_{w+2})$. Continuing with the previous example, when building the graph of incoming traffic for the "Entertainment" category and for the 4pm - 9pm interval, we use all segments that have a destination post published between 4pm - 9pm and an origin post published anywhere between 10am - 9pm.

At this point, we use the *hierarchical movement summaries concept* from the previous Section 7.3.2, originally mentioned in [52], and add hierarchy to the links, by combining endpoints attached to neighboring cells in the mesh, based on the occupancy percentage of the possible neighborhoods. This process takes into account only the nodes of the graph that are bound to a grid cell, and doesn't affect the nodes that have been bound to regions. For example, if we are building the graph for of incoming traffic for the "Entertainment" category, all destination endpoints would be attached to the "Entertainment" regions, while most of the origin endpoints would be associated with a cell in the grid (a small percentage of origin endpoints might also be attached to "Entertainment" regions). Algorithm 5 as previously can be used to add the hierarchy to the links. Figure 7.5 shows incoming traffic to the "Entertainment" regions for the area London and compares day (Figure 7.5(a)) versus night (Figure 7.5(b)).

7.4 Conclusion and Future Work

In this chapter, we have described a methodology for building links-of-interest (LOIs) which are connections between locations (points, places, or regions of interest) extracted from the social media traffic. This concludes the series of scenarios used to demonstrate the usability, suitability and potential of social media feeds. An interesting direction for future work is to attempt to build links between other types of regions (e.g. administrative regions, or Voronoi cells), and test different link bundling techniques, such as hierarchical and gravity-based, that have already been suggested in literature.

In all of the scenarios narrated so far, we notice a very distinguishable pattern emerging: extraction → analysis → visualization. Extraction is necessary to obtain the data. It usually entails one of the three approaches described in Section 4.2. Analysis is essential to obtain useful information from the data. All the previous chapters present a series of analysis steps to explore different aspects of the datasets and solve various problems. Visualization is the way to efficiently and easily communicate the result of the analysis. Interactive visualizations provide the ability to explore the results of the analysis and drill-down certain views to discover even more interesting data. In the next chapter we describe this three-fold pattern as a general methodology.

Chapter 8

The extraction → analysis → visualization pattern

As we are trying to identify the challenges and design solutions for the problems described in the previous chapters, we usually list out the requirements and implement custom components from scratch (or borrow concepts from previous works). After tackling a fair amount of such scenarios, it became evident that there was a pattern repeating in our workflow, and that this repetition could be extracted, modularized and reused, to improve the quality and reduce the time spent in future similar tasks. Once a module has been created, improvements to or enhancements on it benefit all of the solutions that integrate it, not just the specific work that happened to trigger the change.

We can separate the components into three distinct categories: extraction (collection), analysis (processing) and visualization. We hope that this separation and the ideas described in this chapter will benefit not only us but also other researchers as well, by providing guidelines and methodology that can applied towards ongoing of future efforts, to help with the design of solutions to problems similar to the ones we have previously described.

8.1 Introduction

Having distinct separation in design, architecture and implementation is a fundamental concept in building large and scalable systems, as it allows for individual components to evolve independently as long as they respect and adhere to the common interfaces used to communicate with the rest of the setup. The separation also helps with testing and ensuring normal operation as this can be done per component. In the event of a malfunction, we need only to investigate the part that is responsible and not the entire architecture.

In the separation and pattern suggested he have three distinct parts: extraction, analysis and visualization. The extraction part is responsible for the acquisition of the data when it is available in a remote location, for its conversion to a predefined common format, and for its storage considerations. It is also responsible for keeping the algorithms and general workflow of the acquisition of the data, up-to-date and in compliance with all the external sources used.

The analysis module contains all the processing algorithms that are applied on the transformed dataset to extract information. This module will need to access the data as it was prepared from the extraction module, so either an interface is needed, or the use

of pre-agreed format so that both modules can just communicate by just accessing the storage component. Especially in our case, where we usually deal with a large amount of data in the processing steps, we found that being able to directly access the data is faster and less computationally expensive. Once access to the data has been secured, the module can they apply a series of processing steps to transform the raw datasets to meaningful content summaries or more complex datasets that provide additional information, as in the examples mentioned in the previous chapters.

Finally the visualization module's tasks include accessing the results of the analysis, or even the raw datasets is some cases, and being able to present the information in an appealing way that will each time highlight the aspects that the analyst wants to explore. As mentioned earlier the visualizations are interactive and interconnected. The interactivity allows analysts and domain experts to apply filters on the views and drill down to a specific subset of the data in order to look for answers or interesting pieces of information. This makes them not only informative summaries, but essentially search interfaces that anyone can use to browse the data. The visualizations are backed by a query engine, that is able to provide more or less detail, depending on the amount of data requested, plus linking back to the analysis module when a specific query requires additional processing that cannot be answered by cached results.

In the following sections we present each module in more detail.

8.2 Extraction

The collection of the data is the first part of the workflow. This step includes not only the downloading, fetching or copying of the dataset from a remote source, but also the storage aspect. Storing the data into a compatible common format that will allow efficient processing is an important design decision; the process usually entails extracting the parts of the data that are useful and making them easily accessible. Since our knowledge is limited to existing previous uses of the data we can only assume what part of the data will be used later in the process. This is why we suggest also keeping the original format of the data intact, for future possible usage or reference. This redundancy will increase storage space requirements, but can prove very beneficial since it allows exploration of more aspects of the data which can lead to additional analysis directions; furthermore it allows reusing previously collected datasets in new scenarios, since the old datasets have all the original metadata intact. In an optional distributed setup, the extraction module would also be responsible for distributing the data to the cluster of persistent storage submodules. As an example consider the following: in Chapter 3 we obtain data from Flickr in order to identity geospatial features. For this we need only to keep just the id and the location of each downloaded photo object. However later in Chapter 4 we want to gather photos from different sources including Flickr, from within a specific area in order to build a gazetteer; in the later case we also need the title and description of the photo object obtained from the API. This difference in requirements forces us not only to re-implement a new downloader module for Flickr that will request and preserve richer information, but also to re-run the collection process for the several areas we need. In such a case the use of a modular architecture like the one described here would be extremely beneficial, as we would be able to reuse the already existing general purpose implementation for accessing the Flickr API, plus we might already have available datasets for several of the areas we are interested that we can use without needing to download it again.

The common format we use to handle social media data has three main entities: posts

(or documents), users and urls. A post may embed these additional models: a location, a list of user mentions, a list of url mentions, a list of document mentions and finally a list of multimedia objects. Based on this common format, every dataset that has a spatiotemporal aspect, plus any thematic or any of the previously mentioned components can be converted to this format and be later available for analysis. Even datasets with just some but not all of the components mentioned above can be used, but that would limit the types of analysis that can be applied on them.

Another task that is usually assigned to the extraction module is data migration. Regardless of the decision to use an schema powered SQL or no-schema NoSQL database, after obtaining the social media traffic, a homogenization process converts all of the acquired data in a common format as described above. The use of a common format benefits the architecture because:

- analysis algorithms need to know how to use only one specific format as input
- existing algorithms can be applied to new datasets from potentially different sources
- combining multiple dataset (either from the same or different sources) is simpler
- available analysis tasks on a dataset can be inferred by the presence or absence of specific fields

Besides converting the data from the originating platform's format to a common one, from time to time changes in our common schema might also be required; for example we might add a new processing step that requires extra fields to be present in the input data. In such scenarios a two-step process takes place: first, the extraction module is updated to copy or calculate the newly required attributes and store them in the common format collections; second, an additional migration process runs on the already collected datasets to make sure that their mappings conform to the new schema, and that the new algorithms can run on them without any issues.

The main focus of the extraction module is, as mentioned briefly in the introduction, to bridge the gap between the remote sources and the analysis module, by providing a simple interface for allowing the acquisition of datasets from the available platforms. This might not be as trivial as it sounds, since social media platforms have different capabilities, limitations and quotas, purposes, terms, and ways to provide access to their data. The module must ensure that it is as up to date as possible with any changes that each individual source might enforce.

As mentioned briefly in Section 4.2 there are usually three different ways to collect data from these platforms:

- *Data dumps* where links to file downloads that contain the whole or parts of the dataset are provided.
- *Query endpoints* where a source provides an interface for issuing queries to retrieve data.
- *Web scraping* where the source does not provide a structured way to access the data and thus we must resort to fetching and parsing the HTML code of each page and trying to extract the relevant information based on its contents and structure.

8.2.1 Data dumps

This is the case where the remote source provides file downloads for the whole set or subsets of the data. It is often used by services that employ a proprietary format but still

want to publish their data in a way that is disconnected from the rest of the service. If the datasets do not change very often, then this is a very practical way of distribution. By offering data dumps they are making available snapshots of their data, and in some cases the snapshots get regularly updated to reflect the live datasets. A very famous example is the OpenStreetMap initiative where each week, a new and complete copy of all data is made available. Since the complete copy is quite large[1] diff files are also made available that contain the patches that need to be applied to the previous version to bring it up to date. When focusing on a specific area, hence data for the entire planet are not needed, there are available extractor services that can provide download links to only the portion of the data that is relevant to a research. In the case of geospatial data, such as OSM, this usually means sub-selecting based on one or more administrative boundaries or custom bounding boxes. Another example of public datasets available as data dumps are collections published by government agencies; they might be available in the country, state, county, city or municipality level. While these datasets can be extremely useful, not all of them are closely related to the social media philosophy. This is only mentioned here to clarify that not all datasets can be converted to the common format, but the common format used in our methodology is closely related to social networking data. These datasets can however be used as complementing or auxiliary sources for further analysis and verification, since they have guaranteed validity as they originate from official sources.

8.2.2 Query endpoints

High velocity social media traffic is made available to third parties through query endpoints. The most common case is an HTTP RESTful API, and less often SPARQL endpoints are used. The query engine has direct access to the social media data as it is published on each respective platform, and thus this kind of access gives real on near real time access to the live datasets available on the platforms. Since the HTTP RESTful APIs are the most widely used we primarily focus on this type communication interface. The goal which was originally motivated by our efforts described in Chapter 4 is to create a generic workflow to obtain the remote content, that is platform independent; then only overwrite the absolutely necessary parts each separate source requires for specific access. Of course this might not be always entirely possible, since not all platforms offer the same capabilities, and many of them have different limitations.

With the focus being on *spatiotemporal* data we primarily use space and time to devise our collection strategy. We also — at least initially — only support spatially bound and/or keyword based queries. The two filters are joined with either and `OR` or an `AND` predicate depending on what each platform supports and what the objective is. Our generic collection workflow is similar to the one described in Section 4.2, but has adopted a more modular design to allow different services to inherit the general workflow and only provide specific actions at the steps required to prepare the query, access the service and manipulate the retrieved data. The process starts by performing any initial configuration required, such as initialization and state setup. It then starts by creating progress tracking records, that describe the overall search area in case of a geographically bound search, or the time/id range in case of a keyword based search. Immediately after, the main loop starts; every time the progress tracking records indicate that a pending request exists the algorithm enters the loop and starts exploring the first available pending request. For each request we try to exhaustively collect all the available data that can be fetched, then mark

[1] OSM compressed planet XML file: approx. 50GB on June 2016

it as completed and move to the next. The procedure is briefly described in Section 4.2 and is repeated here.

When performing a keyword search we usually collect the entire dataset available by using id or timestamp ranges. This simulates paging by is more efficient, since it uses indexed fields to traverse through the dataset, rather than a progressively increasing offset. It is also more efficient on streaming datasets with high velocity, since page offset frequently change as new data becomes available, while id range filtering always ensure that we are requesting new data. By using a *max* and a *min* id we are able to request more data until we have covered the entire available dataset. In a regular search where we are collecting historic data, the usual workflow follows these steps:

- K1 starting at the current point in time t_0, collect all past posts in time decreasing order
- K2 request a batch of available data
- K3 from the data received extract the minimum available *id*
- K4 assign the minimum available *id* to the *maxId* variable and use it in the next request to obtain all available data with an $id < maxId$
- K5 repeat steps K2 - K4 until no more data can be received, or until receiving data with a timestamp that falls outside the time period of interest

If we want that search to switch to a streaming mode, once the first batch of historic data is complete (previous steps), we initiate a new search that starts again at the current time $t'_0 > t_0$ and search for data using the same filters and same process up until it reaches the id or timestamp at t_0. These subsequent batch of requests are using a *minId* variable which is assigned the value of the maximum id obtain from the first batch. The use of the *minId* variable can become obsolete if we always check the received data to check if we received a record with an id equal or smaller that the maximum id of the first batch, but can still help in reducing the data exchanged.

When performing a bounding box b search, before iterating over the ids or timestamps of available data, we wrap around an extra step of iterations which sets the correct spatial filter. The process includes:

- B1 if the area that is to be examined has an extent that the provider allows us to examine, i.e. if $|b| \leq edge_{MAX}$
  - (a) *estimate* the total number $data_{EST}$ of records available in that area (if supported)
  - (b) if $data_{EST} \leq data_{MAX}$ or if an estimate cannot be obtained
    - i. collect all data within b by using id or timestamp ranges as explained earlier
    - ii. if the received data count is less than the max we can get ($b.dataCount < data_{MAX}$) mark the bounding box as complete and continue the process from B1 by dequeuing the next bounding box
- B2 split the current bounding box b into four quadrants, and enqueue these smaller bounding boxes so that they in turn can be searched, provided that the new quadrants are not smaller that a specified threshold $edge_{MIN}$

Using the workflow described above we can collect data from many different compatible sources by following the same steps and only change specific parameters and the calls to access each specific platform. Specifically for each source only steps K2 and B1.a would need to be changed. This way we can easily add support for more compatible sources and enrich our datasets.

8.2.3 Web scraping

Lastly when none of the previous options are available, we resort to web scraping and HTML/XML parsing in order to extract the relevant information from individual web-pages. Data is obtained by starting with an entry-point URL, then downloading, parsing and saving the content, and finally following all or a subset of the hyperlinks found in each page, until all the accessible pages have been visited. Web scraping is a method used often on traditional platforms such as travel blogs and forums where the focus is not so much in the social aspect of the data but rather in the content itself. This is why usually dedicated forums or blogs contain much richer and more to-the-point information about a specific topic. They are however, more difficult to harvest as there is no one-solution that can parse every source and extract information from it. Once one has been implemented though, similar analysis tools and algorithms as described in the previous chapters can be used to discover interesting pieces and information both in the actual content and in optionally available social structure in these platforms.

Web scraping is one of the most challenging types of collection since it requires access to content that is far more unreliable, less structured and mush more diverse. Usually a manual investigation is required for both the collection and the parsing of the content, each time a new source is considered. Typically content extracted from blogs or forums is organized by a computer program; thus patterns for the extraction can usually be discovered. For example, when collecting user posts from a forum, all the pages that contain relevant content to our search might have the same prefix. In that case, when scanning the HTML code of the downloaded web pages we would want to follow only the URLs that share this common prefix. If the source also provides listing pages, e.g. pages that list the available content, then they can be used as a directory to facilitate and help with the exploration, even make it resumable by allowing to easily identify all the content that has been already visiting in previous crawl jobs, and only focus on new content. If such auxiliary content is not available, the crawl jobs would have to re-extract the whole website just to make sure that all new content has been covered in each pass.

There are two types of information extraction taking place: one is *during* crawling and is necessary in order to obtain the links pointing to the rest of the content; the other usually happens *after* crawling is finished (or in parallel if the computing resources permit it) to actually extract the relevant to the analysis information from the raw HTML content of the downloaded pages. Usually there is a manual pre-processing step which annotates the structure of the page and tags where specific information can be extracted from. The most commonly used reference techniques are XPath [5] and CSS selectors [11]. Using either notation an algorithm can parse every individual document in the downloaded collection and pull out the referenced values.

In the case of evolving datasets, extreme care is needed to install alert mechanisms in place that can notify the manual annotators in the event of a structural change in the HTML code due to new deployed code in the source platform. For example, a website might change its design and information about the published timestamp of an article that was recovered using an XPath expression, might require a manual annotator to update the expression to obtain the value from its new position.

Another important decision is how to handle content that has already been downloaded and exists in the data-stores, but when subsequently accessed is found to have changed. While this decision is crucial in all the three types of collection described earlier, it is certainly more challenging in the case of *web scraping* because it is not always easy to identify what has changed since that last obtained version of a document. In addition to

an actual change in content which is the case we want to capture, it could be that only the structure of the page has changed, or that we are served an error page claiming that the site is under maintenance or that we have exceeded our quota in visits, etc... Tackling these cases is an interesting and challenging task, but is outside of the scope of our work so far and is left as potential future work.

8.3 Analysis

A generic analysis framework is definitely not a panacea to address every possible case, but it can provide a basic set of processing methods and algorithms to get a good estimate about the extent of the overall content and usability of the data. Using a common format for all the UGC we collect from various sources, makes it easy to apply these generic analysis algorithms and methodologies, without having to re-implement them each time we want to add a new source, or study a new scenario. The data arrives in our repository from the extraction module, and is asserted to be compatible with the analysis algorithms.

For example, building and studying a simple temporal summary as described in Section 5.2 can immediately answer questions such as:

- what are the most and less active hours, days, months?
- is the traffic high enough per time period to be able to deduce valid conclusions?
- is there a high or low variance over time in traffic volume?

Similarly a spatial summary (cf. Section 5.3) can be a way to easily answer typical questions such as:

- what are the most and less active places, regions, states, countries, continents?
- is the coverage of my area of interest enough to lead to valid conclusions?
- how does the normalized traffic based on the local population of an area behave?

Thematic, lexical token summaries pick out the most frequently observed n-grams discussed within a collection, plus answer questions such as:

- what are the most discussed topics in an area during a time period?
- are specific tokens always present or do they suddenly emerge in the stream?
- which other keywords does a specific keyword most often link to (cooccur)?

Especially the last question is of extreme important to the collection/extraction process, as it can improve the original query set to the external source by adding relevant keywords to further improve coverage.

As a final example in this non-exhaustive list, since we are dealing with user generated content that has social information attached, a social summary would quickly reveal information such as

- who are the users producing most of the original content?
- how do users react and engage interaction based on their connections?
- how densely connected is the network?

The preliminary results from the observations of the previously mentioned summaries can be the motivation to initiate further and more focused type of analysis and research. Maintaining a core of analysis methods and algorithms as a separate module, is beneficial not only for current but also for future (and sometimes past) research. A specific research

topic might result in an improvement of an existing algorithm, or in a completely new one. Incorporating this improvement to the analysis module allows it be capable to process all of the existing datasets previously collected, plus any new dataset extracted in the future with minimum or no effort at all. This is valuable to both the datasets and the algorithm. The algorithm is tested with a more diverse set of input collections and becomes more robust. Scalability issues can be spotted easily and optimization efforts are pursued to tackle them efficiently. On the other hand, all the existing datasets can be processed with the new algorithm and obtain new and interesting data with minimum effort. In the case where the algorithm needs to access additional attributes of the data, than the existing ones in the common schema, we need only to plan a migration step to transfer the required attributes from the original datasets. If no new attributes are needed, then the algorithm can be executed without any modifications on the datasets, and results are available immediately (depending on algorithm runtime complexity and dataset size).

Since we are dealing with UGC from multiple source, although there is a best effort to map this content to a common schema, it is inevitable that not all sources will provide all the information needed for all the attributes specified by the schema. We can use the available and missing fields information as an indicator of what types of analysis can be performed to each dataset. Each analysis task will be tagged with the attributes that it requires to be present in the input data for the output results to make sense. Then depending on the availability of these fields in the collected datasets different types of analysis will be available. For example, content scraped from a blog will usually not have geographic information attached to it. Spatial coverage, spatial clustering and the rest of the algorithms that require location information to be present in the input data will not be available for this and similar datasets. Similarly, potentially anonymized datasets might be completely lacking user account information which is essential for social network and social presence analysis. Thus, applying these algorithms to these datasets will not be possible. This essentially helps to distinguish between which sources are good candidates for several types of research and which are not. It may also motivate research for methods to infer the missing attributes from the overall data, such as with geoparsing and geocoding (Subsection 2.2.3).

8.3.1 Supporting framework

The benefits of having an integrated supporting framework that hosts all the separate analysis methods are substantial. Such as with all well designed frameworks, all generic functionality and house-keeping is managed by the framework, which allows the research and implementation to focus on the actual specific analysis, rather than worrying about getting access to input data and managing output, capturing benchmarks, synchronizing analysis tasks access to the data and many more. In the following we describe some of the desired properties such a system should have, which are also implemented in our own architecture.

One of the desired functionalities, is automatic benchmarking of runtimes during each of the processing steps. Especially when testing new algorithms or trying alterations or optimizations to existing algorithms, a very important aspect is to ensure that the latest version always outperforms previous versions, but also being able to pinpoint what caused the decrease or increase in execution runtime. A nice functionality is to keep a history of benchmarks of previous versions as the development of the algorithm progresses, and being able to compare benchmarks for datasets of different types and sizes.

Integrating the system with a web application allows a large audience (with proper credentials to authenticate against the system) to request processing on the available datasets. This makes the need for locking mechanisms and progress reporting even more important. We implement a per dataset and per operation locking mechanism to ensure only one processing job of a specific type per dataset runs on our platform at any given time. This reduces the multiple access attempts on the same dataset, which would in turn reduce the throughput of database operations on it, as more database level read/write locks would need to be acquired; however the operation does not need to run, since a similar process is already running. Moreover, this eliminates the risk of corrupt or incorrect output data, as multiple processes might attempt to write to the same output collection. An improvement to the locking mechanism is to add process dependency validation and not allow depending jobs to run until all the necessary jobs have finished executing, but also prevent jobs to start if they will alter datasets that currently running jobs use as either as input or as output. An important requirement that should be carefully implemented, is that locking mechanisms should be immune to crashes of the processing jobs. Since a lock is acquired at the beginning of the execution of the algorithm and released when it ends, a lock might never be released if the algorithm crashes or does not exit normally for whatever reason. Bad design can lead to dataset remain unprocessable indefinitely or until an administrator can manually unlock the crashed job.

Another functionality often useful to such systems is progress reporting. As it is one of the functionalities that is almost always implemented when designing an algorithm, because it is important to know that there is progress during an execution, one can imagine how beneficial is to have this functionality implemented once, and the provide an interface for the algorithm to use in order to report progress. Moreover once implemented, we will not have to worry about the presentation of the progress report as this functionality would have already been taken care of by the framework. Progress reporting is strictly tied to benchmarking, so these two functionalities are closely implemented together. Understandably each processing job might have different ways to calculate and report progress. In our framework though, we use a simple approach for all jobs, by calculating the progress as the number of input documents already processed over the total count.

Finally an important feature is result and query caching, in order to provide fast responses and make the visualizations discussed in the next section extremely responsive. Caching is done in a hierarchical way, similar to how we approached most of the research scenarios described in the previous chapters. It uses variable granularity to cache results at different detail levels. Usually the less detail answer is originally reported, and the as the user interacts with the visualization more detailed answers can be returned for the entire, or usually for a subset of the dataset. We distinguish the cached results between permanent (or semi-permanent) and temporary. Caching results that explore specific dimensions of a dataset without any additional filtering are cached in a permanent way, because they are the ones more often used. For example, when requesting to see the spatial distribution of an entire dataset, without applying any additional filters, that information is retrieved from the permanent cache. It is semi-permanent in the sense that it expires and gets updates only once the collection is updated with new data (or updates or deletions). We expect that queries without filters will be requested more often than others since they are the default. On the other hand when the user applies a filter to the query, for example when requesting to see the spatial distribution limited to the time window of last month, then we need to calculate the result and store it in a temporary cache. The reason is that

the filters applied are user specific, and subsequent users or queries might chose slightly different ones, so it makes no sense to store them for a very long time. This information is saved in a cache with an expiration date (TTL), which obviously gets updated each time the cached value is accessed, but gets deleted when the TTL elapses.

8.4 Visualization

> Above all else show the data.
>
> *Edward R. Tufte*

In his book [82], Edward R. Tufte describes the properties of excellent graphics that are able to communicate complex ideas with efficiency, clarity and precision. The first point he makes is that graphics should "show the data". Some other properties include:

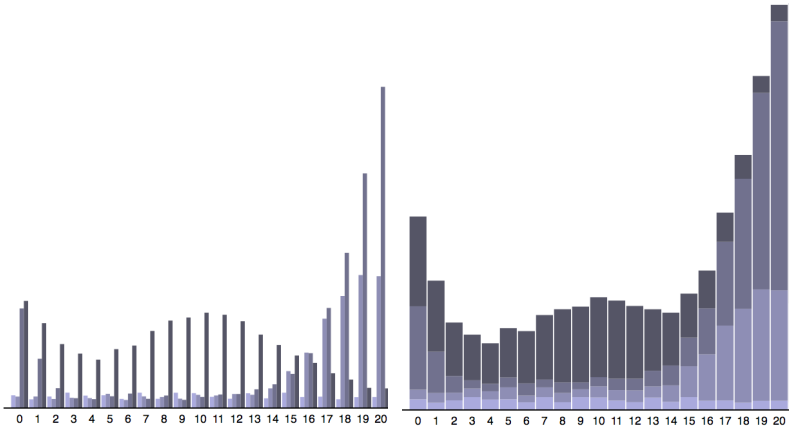
- avoid distorting
- stimulate the viewer to compare differences in the data
- make sense of large datasets
- reveal several levels of detail (from broad overview to finer form)
- making the viewer think about what the data has to say rather than how the data or the graphic were produced

A visualization should focus on a few aspects of the dataset and do not try to interpret every aspect of it, as that usually leads to cluttered interfaces. Unlike printed visualizations, interactive on-screen visualizations can better handle this case, as their dynamic nature allows the graphic to adapt to the current viewport and detail level.

Throughout our work, we try to construct informative graphics and transform them into interactive and interconnected visualizations wherever possible. The data itself is separated from the visualization. This is a key aspect into building scalable and upgradable visualization components. Once a new visualization is designed, data can be fed to it from all the available datasets with minimum effort. Most of the basic visualization components, such as time series, maps, and graphs have already briefly described in Chapter 5. In this chapter we describe enhancements and complement the list with a few more interesting visualizations.

Time series. We have already presented temporal summaries in Section 5.2. We often use the line graphs when the horizontal axis is continuous, or the step is small enough to produce a large number of bins to aggregate on. As the bins grow in size, it is usually preferable to switch to column charts that better display the discrete time intervals that the temporal extent is divided into. For example a line time chart that shows frequency per hour or day, can be converted to a column charts when displaying data per week or month. Additionally when multiple series are present we can either group them together (more cluttered) or stack them on top of each other (less cluttered) as shown in Figure 8.1.

When the dimension measure can fluctuate between a min and max value during the time intervals of the horizontal axis, we can use *candlestick* charts that show the marginal difference between the highest and lowest value, but also display information about the starting and ending value during the time period. These charts are primarily used to depict stock market prices, and they are also often called open-high-low-close (OHLC) charts. They use color coding to highlight the direction of the change, e.g. white for increase and blue for drop. Since our work focuses on user-generated content particularly from



(a) Grouped columns (b) Stacked columns

Figure 8.1: *Converting group columns to stacked to reduce clutter.*

social media, we usually do not have to deal with stock price data. However these charts can be helpful for comparison and correlation, where for example we study the variance in traffic in a blog compared to a particular stock or set of stocks. Specifically what is usually of interest, is to identify patterns that might indicate that variances in price drive the fluctuation in traffic or vise versa. An example is shown in Figure 8.2.

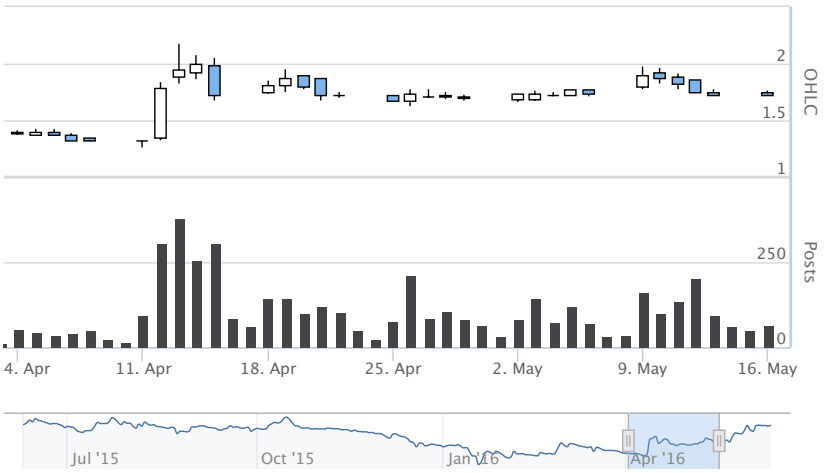
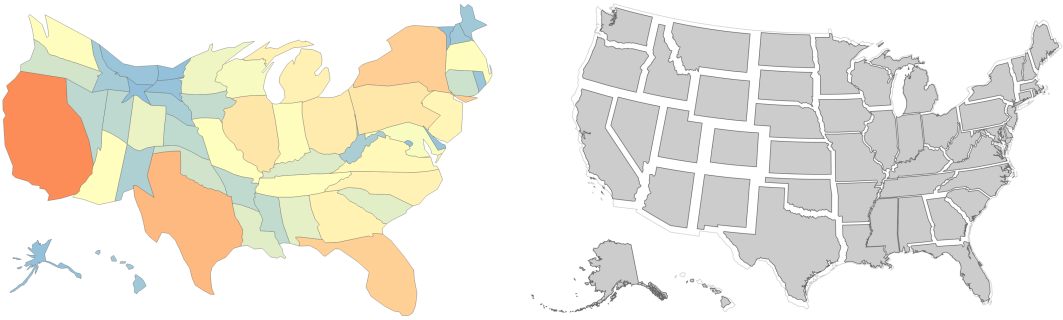


Figure 8.2: *Comparing stock prices with traffic of blog posts.*

Maps. We have explored several map visualization in Section 5.3. Depending on the available granularity in the data we can chose between point based and region based visualizations. Point based data on the map can only be shown when the records embed precise coordinate information. Region based visualizations, also called *choropleth maps* can be used in both cases as long as the boundaries of the regions we are interested in can be retrieved or computed. Records with precise location information can be spatially joined to match their enclosing region. An interesting variation of the choropleth maps, are *cartograms* [67] which encode data using area rather than color, resulting in distorted geographic boundaries. They can use a variety of scaling functions to produce different visual results. Two major types are contiguous and noncontiguous cartograms [64] as shown if Figure 8.3.



(a) Population estimates (US Census 2010) (b) Ratio of obese adults (CDC 2008).

Figure 8.3: *Example of contiguous 8.3(a) and non-contiguous 8.3(b) cartograms.*

Besides the choropleth maps where the regions are based on administrative bounds, we can calculate arbitrary regions based on the data itself. Spatial clustering is one of the available calculations that can produce regions-of-interest as described in Section 4.5 (cf. Figure 4.5). Another interesting type of computed clusters are Voronoi diagrams [63], a partitioning of space into regions based on distance to specific locations (seeds), where for each location there is a corresponding region consisting of all points closer to that location than to any other. For example, when trying to infer user movements in a transport network, we can use a Voronoi partitioning of the area where the seeds are the public transport stations, e.g. metro stations or airports. Then the Voronoi cells can be used as boundaries to aggregate social media posts within them. All the posts that have been geotagged with a location within each Voronoi cell will be mapped to the cell's station. With this location bundling strategy we can then compute the movement network between the stations with a similar process as the one explained in Section 7.3.2. An example is shown in Figure 8.4 where the seeds are the location of all major airports in the United States.

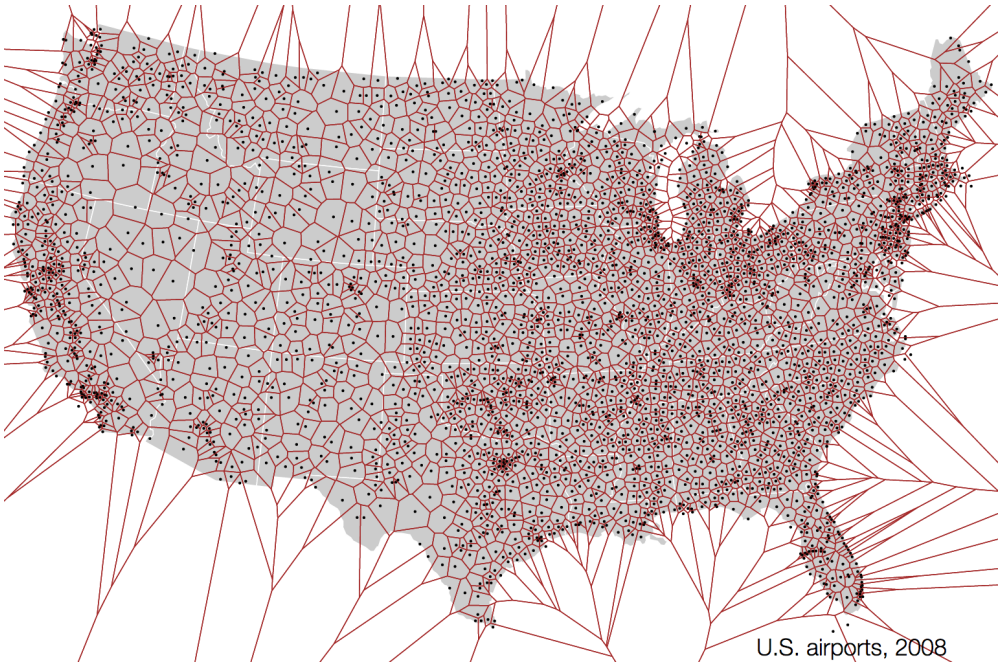
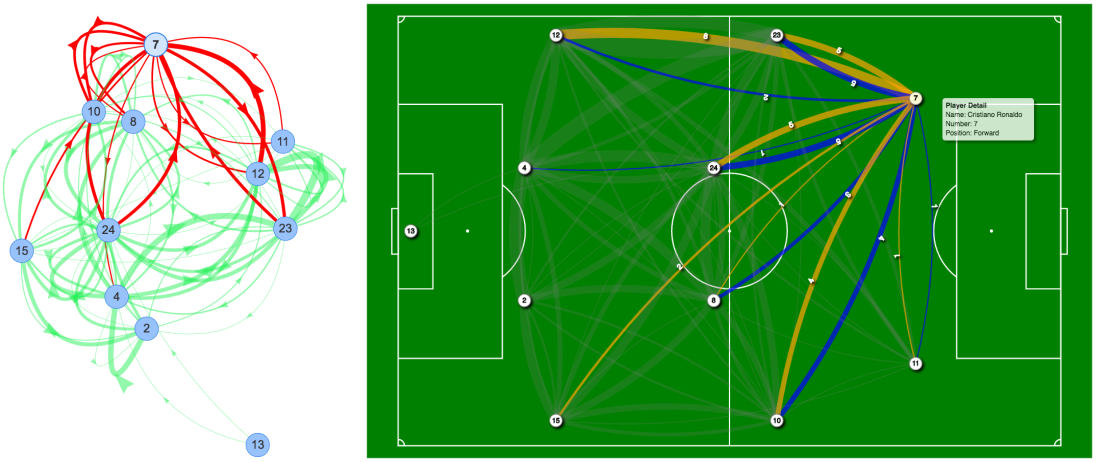


Figure 8.4: *Voronoi tessellation using location of airports as seeds.*

Networks. We used network visualizations (graphs) to display connections between users (Section 5.5) and concepts (Section 5.6). Networks are a great way to display these connection and the adaptive sizing of the nodes and edges represents the significance of either the entity or the link. An improvement to network visualizations is to give context to the positions of the nodes. In our visualizations so far we use a force-directed layout algorithm that works towards reducing overlap and clutter. However sometimes we can position the nodes in fixed locations based on one of their attributes. While this attribute is usually geographic, this is not always the case. For example in Figure 8.5, we use a graph to display passes exchanged between players of Real Madrid in the 1^{st} half of the 23 Sept. 2014 game vs. Elche. We can see immediately that the positioning based on the player's position in the field, immediately gives context to the visualization and makes it more comprehensive.

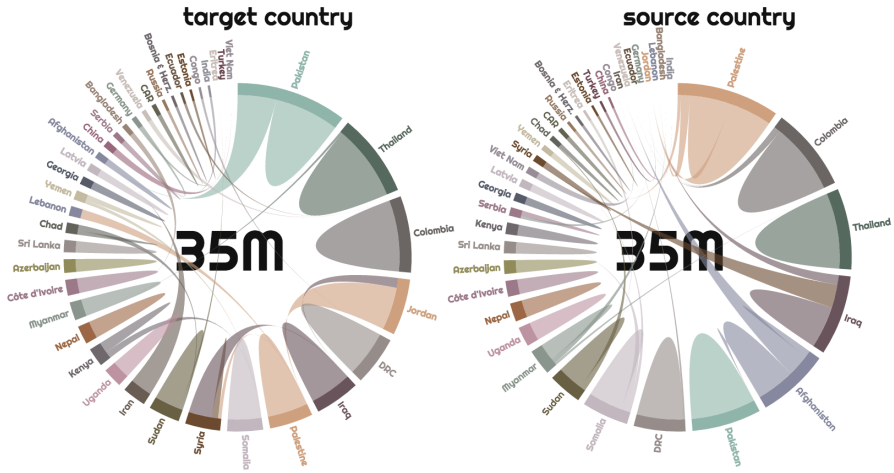


(a) Force directed layout. (b) Fixed locations based on player's position in field.

Figure 8.5: *Network with positional context. Passes exchanged between players of Real Madrid during the 1^{st} half of the 23 Sept. 2014 game vs. Elche. Cristiano Ronaldo is the highlighted node.*

The main problem is that network visualization can easily become cluttered as the network size grows, and that the relative size among nodes and links is not easily identified in a dense network. When the nodes have fixed positions there is not much we can do to declutter the graphic. When using force-directed layout algorithms, we can try to solve this problem by allowing edges as well as nodes to be part of the physics engine, so that they start repelling each other. An alternative way to display connection between a limited amount of entities that manages to display the relative size of the nodes and connection in a better way is *chord* diagrams. These diagrams use a technique called hierarchical edge bundling [38] which similar to our hierarchical links in Section 7.3.2 is designed to reduce visual clutter. An example is shown in Figure 8.6. This is a dataset about refugee migration flows in 2008. Initially the graph that connects flows between countries is overlaid on a world map. A detail is shown in Figure 8.6(a). The graphic becomes too cluttered due to the large amount of data. As an alternative we draw two separate chord diagrams for the countries that either received or sent more than 100k refugees. The result is shown in Figure 8.6(b). The chord diagram performs a better job at conveying the message, which is to easily identify the countries that sent and received the largest number of people. The network visual, forces the position of the node into the location of the countries on the map. This might help to identify flows between neighboring countries, but when just the volume is of importance the chord diagram is much more explanatory.

(a) Network with map background.



(b) Chord diagram

Figure 8.6: *Refugee migration flows in 2008. The same dataset shown as a directed network overlaid on a map and as two separate chord diagrams.*

Additionally the chord diagram can display volume of flow between the same source and target, information that is only available in the network visualization by using self-connecting edges, which tend to usually get omitted.

Thematic summaries. In Section 5.4 we described several thematic summaries by calculating frequency on hashtags and n-grams. Another interesting aspect to study besides volume, is the cooccurrence of certain tokens. This is already achieved with the existing architecture by setting keyword filters; selecting a specific keyword as a filter, all other views including the thematic summary view gets update to include only relevant data, thus showing only the tokens that cooccur with the selected one. However, the bar chart visualization does not communicated very well the relationship between the terms that cooccur. A chord diagram can communicate this relationship in a better way.

Events. In Chapter 6 we described a methodology to discover events as dominant and frequent hashtags found in the Twitter stream. Figure 6.2 display the discovered events in a timeline but it does only give information about the start and end time. A similar visualization shown in Figure 8.7 displays information about events but also shows how

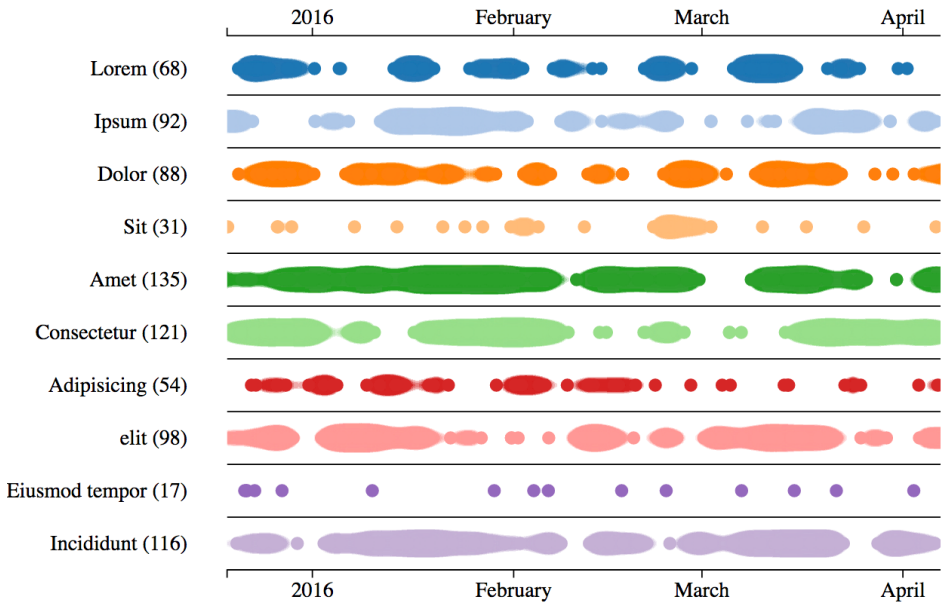


Figure 8.7: *Timeline of hashtags showing their fluctuation over time.*

the corresponding event label fluctuates over time. This new visualization is more suited when the number of observations is fixed; as this is not always the case, we continue to use the simple start-end visual for the events timeline in Chapter 6, since when we monitor the stream we have no a-priori knowledge about the number of events that will simultaneously emerge. However the fluctuation graphic can be used to show the variations of a predefined set of labels we want to monitor. A good use case is to use this type of visualization to monitor the keywords that we base our search on, whenever we perform a keyword-based search rather than a geographic bounding box search. Another case might be to monitor the top k keywords based on overall volume or volume over a progressing time window.

Chapter 9

A tale of two studies

To further illustrate the importance and validity of our work, we briefly present two separate studies that benefit from using UGC content from social media and other sources in order to (i) study the form and function of urban space and (ii) narrate a health topic related to the Zika virus outbreak. These studies are making use of the concepts of extraction, analysis and visualization of social media feeds that we described in Chapter 8 plus specific methodologies and algorithms from the previous chapters.

This chapter summarizes our work in [21] published in the International Journal of Geographical Information Science in 2015 and [80] which is at the time of writing under review (submitted June 2016).

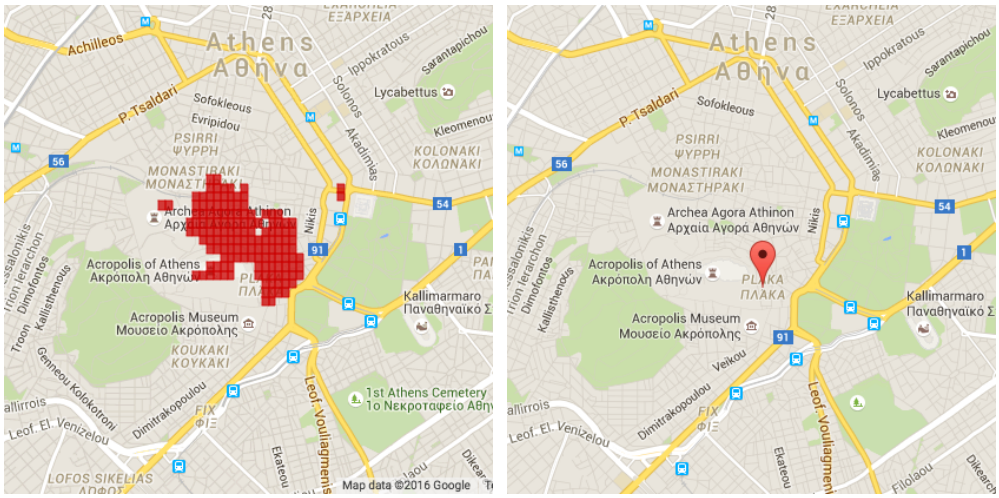
9.1 Understanding urban landscapes

In [21], we study the benefits provided by the concepts of crowdsourcing and user generated content in the study of urban space with respect to primarily understanding their form and function. The term *urban form* refers to the geometry of the city, i.e. the layout of the buildings, the streets, the parks, and all other elements that make up the urban space. By *urban function* we mean all the activities that take place within the landscape of the city and the semantics that are attached to it. Using authoritative datasets to capture the form and function of urban space has been studied thoroughly in the past; however these studies can be as granular as the available spatiotemporal and thematic data, which present noticeable limitations. Volunteered and ambient geographic information that have emerged in the recent years, are helping to overcome these limitations and are gradually presenting new opportunities to study urban form and function. The contribution of these new rich data sources for both form and function is both explicit and implicit in nature.

The proliferation of crowdsourced geographical information has introduced new ways to collect urban form data. While traditionally this has been in the care of regional or national mapping agencies, recently we have seen an abundance of data coming from crowdsourced online platforms such as OSM, Wikimapia and Google Map Maker. The non-experts who use this services perform basic cartographic operations and despite having little or no knowledge of mapping standards, quality control procedures, and metadata in general, it has been found that their contributions are accurate enough, when compared with authoritative or commercial mapping products. Similarly to form, urban function is also being captured by the crowd. In the past, function content was extracted from authoritative sources such as travel-to-work patterns, tax assessment records for land-use and census for demographic information. And while these datasets provided data

at various levels of aggregation primarily due to privacy reasons (which did not allow explorative analysis in great detail over large metropolitan areas), recently finer resolutions are becoming available. However, by just using these datasets it is still not possible to derive function through aggregating various socioeconomic data. Direct and explicit function information would greatly benefit studies related to urban morphology. Towards that direction, content made available by the crowd, can be used to extract knowledge that can help. Some examples include travel logs, guides, or direct interviews, but transforming that content into geographical information is still challenging. Other examples such Foursquare and Wikimapia not only provide explicit land-use information, but also allow the contributors to add function content to the geospatial entities.

These crowdsourced datasets allow exploring form and function without having to lose information through aggregation. Analysis reveals the semantic context given to space from people's activities, and shows that extracting even implicit form and function content can be extremely beneficial. As an example of implicitly crowdsourcing urban form is using GPS traces from floating car data for traffic assessment and forecasting. The tracking data can be coming from fleet management systems, or simply general purpose positioning or navigation software. On top of estimating and predicting traffic conditions, the tracking data can also be used to construct actual road network geometries. These crowdsourced maps, although they do not provide complete coverage of the network, they are very indicative of traffic volume, and can also capture the traffic's evolving nature.



(a) Jeocrowd result (b) Google Maps result

Figure 9.1: *Comparing the "Plaka" neighborhood result between Jeocrowd and Google Maps.*

Another example of implicitly crowdsourcing urban form and function has been already discussed in Chapter 3. User generated point clouds (collected from Flickr photos using specific keywords as input) are mined by the crowd in a collaborative fashion, to derive urban form and function given to the area that comprises the result. Such non-authoritative content is used to transform implicit user references into meaningful information about the function of an area (e.g. a polygon representing the location of a spatial object). For example, Figure 9.1 compares the result of the search for the "Plaka" neighborhood in Athens, Greece. The UGC content derived result is contrasted with the result we get when we search Google Maps for the same term. The benefits are evident; the boundaries of the neighborhood are known to locals, but do not exist in most authoritative sources.

While searching for the spatial footprint of certain keywords can reveal these colloquial areas, following the opposite direction, i.e. from space to thematic context, we can attribute context to specific places by monitoring the originating social media traffic from these locations. An example has already been covered in Chapter 6. Similarly in this work, we collect social media traffic from a specific area, and then analyze the discovered topics in three different granularity levels: individual buildings, streets, and neighborhoods. To discern topics from the raw textual descriptions of the social media data we use Latent Dirichlet Allocation (LDA) [9]. The analysis at the building level reveals two themes, *entertainment* and *food*, but they are not consistent throughout the time period the dataset covers. In the street level example, the difference between day and night is very distinct. While the *food* topic is prevalent throughout the entire period, the *labor* topic appears mostly during the day and the *entertainment* topic mostly during the night. This transition between day and night themes highlights the transformation between different social functions of the same urban area over time. Finally, in the neighborhood example, due to the bigger spatial extent of the studied areas, the topics are more diverse and better describe the individual neighborhoods. *Education* related topics appeared in the neighborhood where the universities are located, *entertainment* was mostly observed in the area where most of the bars and night clubs are located, while *politics* where discussed in the neighborhoods close to the business districts.

As a final example, we consider harvesting people's opinions about specific places and their functions, to assign sentiment to space. By using a large corpus of travel blogs, we can parse the text to extract place names and derive sentiment information. Then each paragraph's sentiment, is mapped to a bounding box that contains all the coordinates of the toponyms that were extracted from it.

Through all the previous example, it becomes evident that by harvesting directly from the crowd we get data at a much finer temporal, spatial and social scale. This allows to study the conceptualization of how a city operates in more detail, using a bottom-up instead of a top-down approach. Cities are found to be *smart*, *adaptive* and *reconfigurable* systems and we get the opportunity to observe the mechanisms that drive their operations, by complementing the traditional authoritative datasets with user generated content in the form of both *volunteered* and *ambient* geographic information. As this data comes at high frequency and volume, its combination with traditional sources, can form a model of urban morphology. This model is not static but dynamic, as its gets constantly updated with new records, and can be used to detect hidden patterns through space and time. These patterns can be used to enable the city to learn and respond efficiently to similar changes that will occur in the future. This new research agenda has been found to revolve on three key areas so far: our ability to collect and curate user generated content; our ability to analyze and visualize these rich and heterogeneous datasets; and adapting the existing analysis techniques to the finer scale of the new data. The demand for improving this new science of better understanding the form and function of city space, will only increase in the future, as the world shifts to becoming increasingly urban.

9.2 Narratives through data-driven journalism

In [80], we present an example of storytelling which is based on information extracted from both authoritative and user generated datasets. Specifically, we narrate the events related to the 2015-2016 Zika virus outbreak, as it was captured on Twitter. Our efforts explore three distinct dimensions present in the data: the *geographical*, the *linguistic* and

the *social*. The temporal dimension is used as the weaving fabric that ties all the other previously mentioned dimensions together by providing evolutional views that highlight the changes. In a sense our narrative is being built by the studying of the interplay of locations, concepts and actors over time, and combining it all together into a coherent story. In particular, with respect to the story about the Zika virus outbreak the analysis reveals three different types of events:

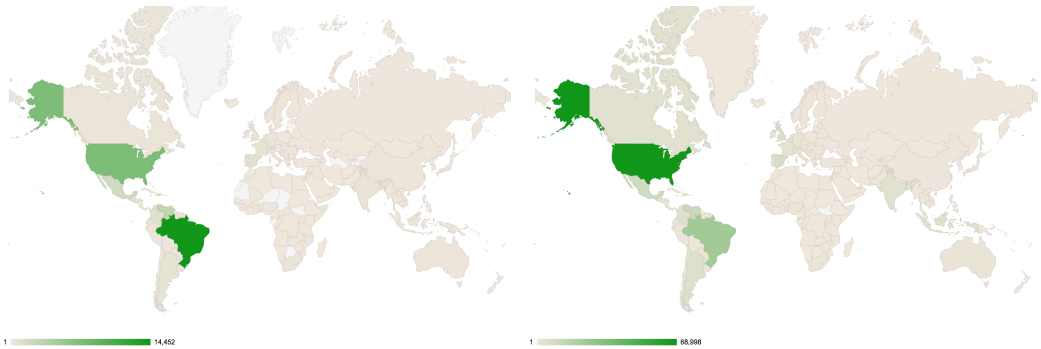
- along the spatial dimension, we are able to detect *geographical events* that capture the progression of the talk about the epidemic, as it begins its path from South and Central America to reach global coverage.
- along the linguistic dimension, we capture *concept events* as they are formed from references to certain related health concepts — such as e.g. pregnancy and microcephaly — which make bursty appearances in the data feed.
- along the social dimension, *social media presence events* are revealed as certain people and organizations participate in the discussion and influence the flow of information; key players in this case are the Center for Disease Control and Prevention (CDC) in the United States and the World Health Organization (WHO).

Our narrative is driven by a corpus of approximately 6 million tweets, collected from December 11, 2015 through March 4, 2016, based on variations of the keyword "zika". Approximately half of those tweets were geotagged with precise coordinates which makes our spatial analysis achieve adequate coverage. The overall traffic experiences a peak around the first week of February, with approximate one quarter of the entire dataset being published during the first seven days of February. After that peak, the traffic gradually dropped to normal volumes.

Many different types of narratives are present in social media, the most dominant being culturally common stories that reflect people's perceptions of the physical and social environment. However, as both users and authorities contribute views, ideas and thoughts, other types such as official authoritative stories and firsthand stories about individual experiences are also present. This diversity requires a novel analysis framework to process and make sense of the narratives hidden in the data.

In our analysis we use the three previously mentioned dimensions in an interconnected way to layout the narrative. Locations, concepts and actors emerge out of the discussion and we build the story along these three axes by detecting events as temporal variations in each direction. In a sense, this *health narrative triangle* is used to answer to the threefold of where-what-who questions that help with telling the story based on the available data. The geographical analysis answers "where" the narrative is taking place and represents the spatial footprint of the communities that participate in the discussion. Events along space would be indicative of the geospatial flow of information over time. The concepts that are detected within the data answer the question of "what" is the discussion about, in the form of keywords that map to ontologies defined in established knowledge bases. Events along the linguistic dimension emerge as bursty terms that expand and redefine the scope of the narrative. Finally, analyzing the social network structure and flow of information, answers the question of "who" are the main contributors to the discourse either in terms of content or influence. Events along the social dimension are detected by analyzing each node's contribution and impact. In the following we present the results of the analysis using the described *health narrative triangle*, by presenting the events described in each of the three dimensions.

The spatiotemporal patterns discovered in our corpus are indicative of certain events



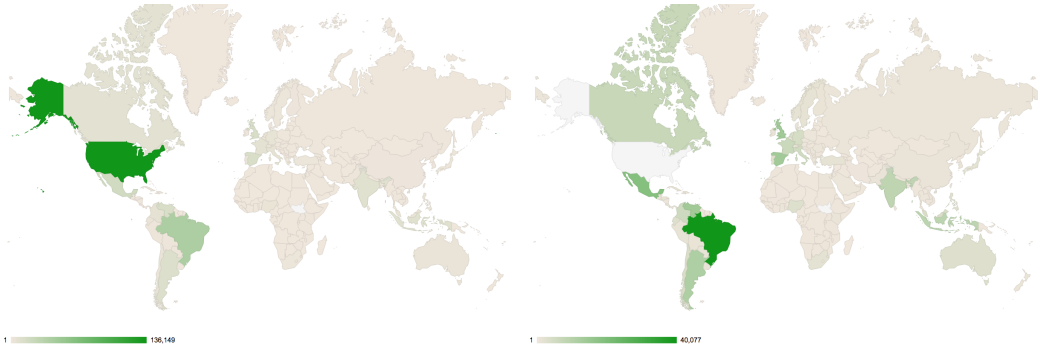
(a) December 2014 (b) January 2015

Figure 9.2: *Spatiotemporal evolution of the discussion about "Zika" per country.*

that resulted in either new area participating in the discourse, or intensifying the discussion in existing regions. By using clustering techniques such as DBSCAN [28] we discover strongly connected geographic clusters which reveal a high level of local participation in the communities within them. The temporal emergence of these clusters shows when each region joined the discussion about the Zika virus outbreak, and how the contributions transitioned from local to global within a few weeks. Early in December there were only two clusters in Brazil and the Venezuela/Colombia region. This is in line with the reported cases during that period. In the coming weeks, new cases are reported in Haiti and Puerto Rico while there is an increase in cases from Guatemala and Mexico. Again, this can be seen in the data as new clusters are formed in Central America. Subsequently, we see clusters appearing first in Europe and then in Africa as these continents join the Zika discourse and finally almost global coverage is reached.

A simple geochart with traffic per country show similar results. Early in January 2015 most of the traffic originates from Brazil, while in February the US takes the lead, as is shown in Figure 9.2. Because the traffic produced by the United States is by far more than any other country, it makes it difficult to see the variations in the rest of the world. To amend that we can plot two charts as in Figure 9.3, one showing the traffic produced by all countries and a second one with the top country (USA) removed, so that the participation of the rest of the world becomes more noticeable.

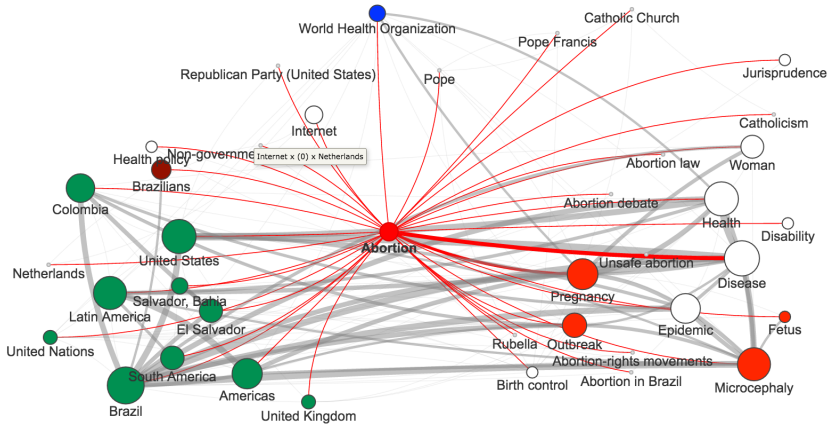
Key concepts related to the Zika virus outbreak are detectable in our dataset. As an ex-



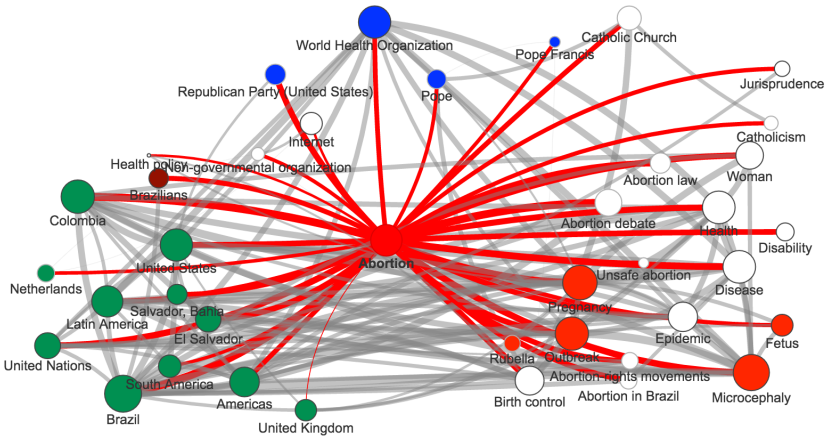
(a) February 2015 overall (b) February 2015 (USA removed)

Figure 9.3: *In February the discussion becomes global, however the USA dominates the discussion. Contrasting geo-charts with USA present and removed.*

ample, we consider the concepts of "pregnancy" and "abortion". Both terms were rarely mentioned during the first weeks in our dataset, but references to pregnancy dramatically increased after January 15^{th}, when the CDC issued a travel guidance for pregnant women, because there was growing evidence that the children born to infected mothers had an increased risk of born with microcephaly. After this event the pregnancy concept stays associated with the virus throughout our dataset, and reaches a peak on February 5, when CDC offered Zika tests for all pregnant women who had potentially been exposed to the virus. The abortion term, peak just after this event, and had similar traffic to pregnancy with the exception of a burst in February 17^{th} which coincided with Pope Francis commenting about the use of contraception and abortion in the context of the Zika outbreak. Besides observing just the volume of mentions of concepts throughout the dataset we also study the links between them (cooccurrences), with an analysis similar to the one described in Section 5.6.



(a) Early January 2015



(b) Mid February 2015

Figure 9.4: *Concepts connected to the "Abortion" topic over time. Circle size is logarithmically scaled to the number of mentions of each term.*

The social network analysis indicated two key organizations that played an important role in the narrative as briefly mentioned earlier: the CDC and WHO. We study their contributions and impact on the network by calculating an evolving time series of their publishing activity and the other participants reaction to that (re-sharing or starting a communication). Figure 5.7 shows the analysis for WHO. The results show that while CDC

has a higher level of publishing activity than WHO, messages from WHO initiate approximately double the traffic that those of CDC. This can be attributed to the fact that WHO is a global organization while CDC is a U.S. agency. Besides the two key organizations, the network analysis also discovers dense connected components that are mapped to specific geographic locations. In particular we notice distinct subgraphs, such as the one around Nicolas Maduro, president of Venezuela, and another one around Hilda Hernandez, the Honduran Minister of Communication, as shown in Figure 5.6.

The workflow we followed in our study lays the ground for developing a framework toward improving the understanding of how people and organizations engage in discussions about health related topics, specifically in this case the Zika virus outbreak in early 2016. Following a threefold analysis along the geographical, linguistic and social dimensions, allows us to detect temporal bursts in traffic and connect these to real life events related to the topic. Additional detail about the locations, concepts and actors will prove of great value for future examinations on the evolution of health narratives. While the importance of each dimension may differ depending on the nature of the topic that the narrative is about, they are all valuable and together provide a methodology to evaluate and better formulate data-driven storytelling.

Chapter 10

Overall conclusions and future work

In this dissertation we have covered a lot of ground. We presented our work on designing and testing efficient algorithms and techniques to work with user-generated content datasets retrieved from a variety of sources. We introduced scenarios where this data might be helpful, and walked through the entire process of solving each of the proposed problems, from obtaining the data, to processing it, to finally visualizing it to show our results. We significantly increased the scope of our results by creating fully functional real-world systems and services based on our proposed methodologies. The results derived from theses systems will motivate future work in terms of both efficiency and processing context. In the following, we give a brief summary of our contributions and a potential list of extensions of our results for future work.

10.1 Summary

In this section we will summarize the results produced by this dissertation. The main contributions of this dissertation can be classified into three categories.

1. First, we presented a set of methodologies designed to improve and enrich the UGC collections that become available from social media platforms.
   In our first work, we retrieved data from Flickr photos in order to locate and describe places identified by a set of search keywords. We demonstrated how this can be helpful in enriching place information gazetteers by adding a spatial popularity heatmap for geospatial features, besides just their location and respective bounds. We also showcased an efficient way of calculating the shape of each searched feature, by exposing our infrastructure as a web application and utilizing a browser based computing framework that outsources the collection and processing tasks to the connect web browsers.
   In later work, we tried to complement the datasets of individual VGI platforms by attempting to collect and bundle together information from a variety of sources, in order to provide a more complete, accurate and informative dataset. We further improved the unified dataset by running a clustering algorithm to detect duplicate entries, and we added value by calculating clusters of close-by POIs of the same type, that we call *regions of interest* or ROIs.
2. Second, we implemented a comprehensive list of inter-connected content summary views, designed to provide overviews of these rich datasets, with efficiency and ease. We covered many of the available dimensions of the data, such as the temporal, spatial, thematic, social, etc. and presented a series of visualizations for each

dimension. Our goal was to connect the dimensions via the interactive graphics, to build a graphical query interface where users and domain experts can study the data and drill down to more detailed views to discover interesting results. This is accomplished by narrowing down the viewport with respect to one of the dimensions (e.g. a specific time period, or a particular region on a map) and having all other views adapt automatically to the relevant data. We also described the details about the data structure we used to efficiently store aggregated information about the dataset, to be able to answer the different queries fast and reliably. The efficiency of the structure was evaluated through extensive experimentation.

3. Third, we investigated the association between the digital and the real world, by attempting to map patterns in the data to real life activities.
   Initially we tried to observe fluctuations in the mentions of certain hashtags in the social media traffic over a specific region, as indicators for events. Once a noticeable variation has been detected for a particular hashtag, we track the spatial extent of that hashtag outside of the monitored region, and rank the derived event as local or global depending on whether there is too few traffic outside of the region or not.
   Lastly we tried to extract the trajectories of social media users using their geotagged posts, in an effort to discern paths and general user movement flows. We then bundled these trajectories together and identified high traffic volume paths in space, that we linked with actual user activity on the ground, by using information from the social media content itself. Specifically the flows were categorized based on their temporal presence, and then analyzed in order to extract the thematic context present along a route. As an improvement we evolved the algorithm used to construct the paths, to not only bundle trajectory segments together based on the spatial clustering of their origin / destination end-points, but to also accept arbitrary polygons as cluster region identifiers. Using the regions of interest computed earlier as the cluster regions, we identified incoming and outgoing traffic patterns related to each ROI category type. Similarly we could also use administrative or other type of regions, to produce different results.

Finally we wrapped up by presenting the workflow followed through all our case studies, in a more generalized way. We tried to formulate a general purpose framework that will become the basis of similar and future research endeavors, as it lays out the fundamental blocks that are the backbone of our work. By using separate modules for collection, processing and visualization of the data, we built a robust and extensible solution for efficiently working with UGC from the web and similar datasets.

10.2 Future Work

During the course of this dissertation, we have identified several interesting research directions to follow for future work.

One direction would be to compare the results of the Jeocrowd tool, presented in Chapter 3, with other authoritative sources or crowdsourced dataset and assess how similar or different the results are. This could use the universal collector system that is available to facilitate the process. For example, when looking for place names, we could compare results against other VGI datasets (such as OSM and wikimapia), or against authoritative sources such as government issued administrative boundaries datasets. Although we do not expect to have an exact match, since for example on large area features

it is practically impossible to take photos at all the locations needed to capture the entire area, it would be interesting to calculate similarity metrics. Besides comparing the resulting shapes with other VGI datasets, we can also compare Flickr to other similar photo sharing platforms such as Instagram and Panoramio.

Our work with the fusion of datasets retrieved from multiple sources in Chapter 4, provided a way to detect duplicate records and fill in information missing from one dataset but being present in others, to build more complete POIs. In future work we can further examine how we can use both merge and conflict resolution rules to better handle the fusion process, in cases where the values of certain attributes are not complementing but rather offending each other. Some sources can be ranked more trustworthy for specific attributes than others, or other aggregation techniques can be used to extract the most widely used values. Furthermore, we used a custom implementation of the DBSCAN algorithm to compute clusters, either the ones comprised of duplicate records, or the ones that include POIs of the same category in close proximity (ROIs). Different clustering and grouping algorithms can be used to contrast and compare the results.

Another future effort would be to work on improving the browser based computing system we introduced in Chapter 3 and reused in Chapter 4. We have successfully used it in two different scenarios so far, and have proven it can provide great performance and scalability. We would like it to take it to the next level, by extracting the basic components and build a framework around it, so that other researchers and developers can benefit by it.

Moreover in Chapter 5 we presented a series of social media content summaries to assist in giving an overview of large datasets. On future work we plan on expanding our gallery of content summaries, improving the connections between them and also showcasing how they can be used for further analysis of these datasets. Some of the additional views that can be added to the gallery have already been discussed in Section 8.4. Furthermore we want to increase the interconnectedness of the views by allowing more complex filters on more dimensions.

During our work in event detection in Chapter 6 we used a single hashtag to identify a candidate event. However, as the experiments indicate, typos and/or term coining by different people, can result in several similar hashtags relating to the same event. In ongoing and future work, we try to identify all the alternative hashtags describing an event and try to combine/fuse the attached data to study each event with a larger compiled corpus.

Finally, our work towards building links of interest that connect locations of increased social media traffic can be expanded to connect more types of arbitrary locations. Besides the ROIs described in Chapter 4, by slightly variating the proposed algorithm we can build links between administrative regions, Voronoi cells, etc. Plus different link bundling techniques can be studied and used, such as hierarchical and gravity-based, that have already been suggested in the literature.

In conclusion, there are multiple interesting and novel topics relevant to this dissertation and we really hope that this thesis will be a starting point for igniting further research in the suggested directions.

Bibliography

[1] H. Abdelhaq, M. Gertz, and C. Sengstock. Spatio-temporal characteristics of bursty words in Twitter streams. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 194–203. ACM, 2013.

[2] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. *Automatic subspace clustering of high dimensional data for data mining applications*, volume 27. ACM, 1998.

[3] H. Becker, M. Naaman, and L. Gravano. Beyond trending topics: Real-world event identification on twitter. 2011.

[4] Z. Bellahsene, A. Bonifati, and E. Rahm, editors. *Schema Matching and Mapping*. Springer, 2011.

[5] A. Berglund, S. Boag, D. Chamberlin, M. F. Fernández, M. Kay, J. Robie, and J. Siméon. Xml path language (xpath). *World Wide Web Consortium (W3C)*, 2003.

[6] D. Birant and A. Kut. St-dbscan: An algorithm for clustering spatial–temporal data. *Data & Knowledge Engineering*, 60(1):208–221, 2007.

[7] M. Bishr and W. Kuhn. Geospatial information bottom-up: A matter of trust and semantics. In S. I. Fabrikant and M. Wachowicz, editors, *The European Information Society*, Lecture Notes in Geoinformation and Cartography, pages 365–387. Springer Berlin Heidelberg, 2007. DOI: 10.1007/978-3-540-72385-1_22.

[8] P. E. Black. Manhattan distance. *Dictionary of Algorithms and Data Structures*, 18:2012, 2006.

[9] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

[10] J. Bollen, H. Mao, and X. Zeng. Twitter mood predicts the stock market. *Journal of Computational Science*, 2(1):1–8, 2011.

[11] B. Bos, H. W. Lie, C. Lilley, and I. Jacobs. Cascading style sheets, level 2 css2 specification, 1998.

[12] P. Bouros, S. Ge, and N. Mamoulis. Spatio-textual similarity joins. *PVLDB*, 6(1):1–12, 2012.

[13] H. Butler, M. Daly, G. S. Doyle, Allan and, T. Schaub, and C. Schmidt. The geojson format specification, 2008.

[14] A. Cary, Z. Sun, V. Hristidis, and N. Rishe. Experiences on processing spatial data with mapreduce. In *Proc. 21st SSDBM conf.*, pages 302–319, 2009.

[15] R. Cattell. Scalable sql and nosql data stores. *ACM SIGMOD Record*, 39(4):12–27, 2011.

[16] M. Cha, H. Haddadi, F. Benevenuto, and P. K. Gummadi. Measuring user influence in twitter: The million follower fallacy. *ICWSM*, 10(10-17):30, 2010.

[17] K. Chodorow and M. Dirolf. *MongoDB: the definitive guide*. O'Reilly Media, Inc., 2010.

[18] D. Coleman, Y. Georgiadou, and J. Labonte. Volunteered geographic information: the nature and motivation of produsers. *Int'l Journal of Spatial Data Infrastructures Research*, 4:332–358, 2009.

[19] A. Cope. The Shape of Alpha. Where 2.0 conf. presentation, http://where2conf.com/where2009/public/schedule/detail/7212, 2009.

[20] A. Crooks, A. Croitoru, A. Stefanidis, and J. Radzikowski. # earthquake: Twitter as a distributed sensor system. *Transactions in GIS*, 17(1):124–147, 2013.

[21] A. Crooks, D. Pfoser, A. Jenkins, A. Croitoru, A. Stefanidis, D. Smith, S. Karagiorgou, A. Efentakis, and G. Lamprianidis. Crowdsourcing urban form and function. *International Journal of Geographical Information Science*, 29(5):720–741, 2015.

[22] C. A. Davis Jr, G. L. Pappa, D. R. R. de Oliveira, and F. de L Arcanjo. Inferring the location of twitter messages based on user relationships. *Transactions in GIS*, 15(6):735–751, 2011.

[23] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[24] Q. Diao, J. Jiang, F. Zhu, and E.-P. Lim. Finding bursty topics from microblogs. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 536–544. Association for Computational Linguistics, 2012.

[25] P. S. Earle, D. C. Bowden, and M. Guy. Twitter earthquake detection: earthquake monitoring in a social world. *Annals of Geophysics*, 54(6), 2012.

[26] A. Efentakis, D. Theodorakis, and D. Pfoser. Crowdsourcing computing resources for shortest-path computation. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, pages 434–437. ACM, 2012.

[27] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng.*, 19(1):1–16, 2007.

[28] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, pages 226–231, 1996.

[29] J. Euzenat, C. Meilicke, H. Stuckenschmidt, P. Shvaiko, and C. T. dos Santos. Ontology alignment evaluation initiative: Six years of experience. *J. Data Semantics*, 15:158–192, 2011.

[30] A. Flanagin and M. Metzger. The credibility of volunteered geographic information. *GeoJournal*, 72(3):137–148, 2008.

[31] J. Gelernter and S. Balaji. An algorithm for local geoparsing of microtext. *GeoInformatica*, 17(4):635–667, 2013.

[32] S. Goil, H. Nagesh, and A. Choudhary. Mafia: Efficient and scalable subspace clustering for very large data sets. In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 443–452. ACM, 1999.

[33] M. F. Goodchild. Citizens as sensors: the world of volunteered geography. *GeoJournal*, 69(4):211–221, Nov. 2007.

[34] C. Grier, K. Thomas, V. Paxson, and M. Zhang. @ spam: the underground on 140 characters or less. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 27–37. ACM, 2010.

[35] M. Haklay. How Good is Volunteered Geographical Information? A Comparative Study of OpenStreetMap and Ordnance Survey Datasets. *Environment and Planning B: Planning and Design*, 37(4):682–703, Aug 2010.

[36] M. Haklay and P. Weber. Openstreetmap: User-generated street maps. *Pervasive Computing, IEEE*, 7(4):12–18, 2008.

[37] M. Helft. Online maps: Everyman offers new directions. *The New York Times (Nov. 2009)*, 2009.

[38] D. Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on visualization and computer graphics*, 12(5):741–748, 2006.

[39] M. Hu, S. Liu, F. Wei, Y. Wu, J. Stasko, and K.-L. Ma. Breaking news on twitter. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pages 2751–2754, New York, NY, USA, 2012. ACM.

[40] E. Ilina, C. Hauff, I. Celik, F. Abel, and G.-J. Houben. Social event detection on twitter. In *Web Engineering*, pages 169–176. Springer, 2012.

[41] S. Intagorn and K. Lerman. Harvesting geospatial knowledge from social metadata. *Knowledge Creation Diffusion Utilization*, (May):1–10, 2010.

[42] H. Järvinen. Html5 web workers. In *T-111.5502 Seminar on Media Technology BP, Final Report*, page 27, 2011.

[43] A. Java, X. Song, T. Finin, and B. Tseng. Why We Twitter: Understanding Microblogging Usage and Communities. In *Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 Workshop on Web Mining and Social Network Analysis*, WebKDD/SNA-KDD '07, pages 56–65, New York, NY, USA, 2007. ACM.

[44] L. Kaufman and P. J. Rousseeuw. Finding groups in data. an introduction to cluster analysis. *Wiley Series in Probability and Mathematical Statistics. Applied Probability and Statistics, New York: Wiley, 1990*, 1, 1990.

[45] E. Korpela, D. Werthimer, D. Anderson, J. Cobb, and M. Lebofsky. Seti @ home - massively distributed computing for seti. *Computing in Science & Engineering*, 3(1):78–83, Jan 2001.

[46] D. Kotzias, T. Lappas, and D. Gunopulos. Addressing the sparsity of location information on twitter. In *EDBT/ICDT Workshops*, pages 339–346, 2014.

[47] J. Krumm, N. Davies, and C. Narayanaswami. User-generated content. *IEEE Pervasive Computing*, 7(4):10–11, Oct. 2008.

[48] H. W. Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[49] J. Lafferty and C. Zhai. Document language models, query models, and risk minimization for information retrieval. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 111–119. ACM, 2001.

[50] G. Lamprianidis and D. Pfoser. Jeocrowd: collaborative searching of user-generated point datasets. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 509–512. ACM, 2011.

[51] G. Lamprianidis and D. Pfoser. Collaborative geospatial feature search. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, pages 169–178. ACM, 2012.

[52] G. Lamprianidis, D. Pfoser, and T. Sellis. Spatiotemporal summaries of social media content. *Intelligent Systems and Technology - Special Issue on Social Media Processing*, ?(?):?–?, 2016.

[53] G. Lamprianidis, D. Skoutas, G. Papatheodorou, and D. Pfoser. Extraction, integration and analysis of crowdsourced points of interest from multiple web sources. In *Proceedings of the 3rd ACM SIGSPATIAL International Workshop on Crowdsourced and Volunteered Geographic Information*, pages 16–23. ACM, 2014.

[54] G. Lamprianidis, D. Skoutas, G. Papatheodorou, and D. Pfoser. Extraction, integration and exploration of crowdsourced geospatial content from multiple web sources. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 553–556. ACM, 2014.

[55] T. Lappas, M. R. Vieira, D. Gunopulos, and V. J. Tsotras. On the spatiotemporal burstiness of terms. *Proceedings of the VLDB Endowment*, 5(9), 2012.

[56] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. In *Soviet Physics Doklady*, volume 10, pages 707–710, 1966.

[57] G. McKenzie, K. Janowicz, and B. Adams. A weighted multi-attribute method for matching user-generated points of interest. *Cartography and Geographic Information Science*, 41(2):125–137, 2014.

[58] M. Mendoza, B. Poblete, and C. Castillo. Twitter under crisis: Can we trust what we rt? In *Proceedings of the first workshop on social media analytics*, pages 71–79. ACM, 2010.

[59] M.-F. Moens, J. Li, and T.-S. Chua. *Mining User Generated Content*. Chapman & Hall/CRC, 2014.

[60] J. Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial & Applied Mathematics*, 5(1):32–38, 1957.

[61] R. T. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. In *Proceedings of the 20th International Conference on Very Large Data Bases*, pages 144–155, 1994.

[62] A.-C. N. Ngomo. Orchid - reduction-ratio-optimal computation of geo-spatial distances for link discovery. In *International Semantic Web Conference (1)*, pages 395–410, 2013.

[63] A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu. *Spatial tessellations: concepts and applications of Voronoi diagrams*, volume 501. John Wiley & Sons, 2009.

[64] J. M. Olson. Noncontiguous area cartograms. *The Professional Geographer*, 28(4):371–380, 1976.

[65] G. Papadakis, E. Ioannou, T. Palpanas, C. Niederée, and W. Nejdl. A blocking framework for entity resolution in highly heterogeneous information spaces. *IEEE Trans. Knowl. Data Eng.*, 25(12):2665–2682, 2013.

[66] T. Powell. *Ajax: the complete reference*. McGraw-Hill, Inc., 2008.

[67] E. Raisz. Rectangular statistical cartograms of the world. *Journal of Geography*, 35(1):8–10, 1936.

[68] T. Rattenbury and M. Naaman. Methods for extracting place semantics from flickr tags. *ACM Trans. Web*, 3:1:1–1:30, January 2009.

[69] A. Ritter, S. Clark, Mausam, and O. Etzioni. Named entity recognition in tweets: An experimental study. In *EMNLP*, 2011.

[70] T. Sakaki, M. Okazaki, and Y. Matsuo. Earthquake shakes twitter users: real-time event detection by social sensors. In *Proceedings of the 19th international conference on World wide web*, pages 851–860. ACM, 2010.

[71] J. Sankaranarayanan, H. Samet, B. E. Teitler, M. D. Lieberman, and J. Sperling. Twitterstand: news in tweets. In *Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems*, pages 42–51. ACM, 2009.

[72] A. Schwering. Approaches to semantic similarity measurement for geo-spatial data: A survey. *T. GIS*, 12(1):5–29, 2008.

[73] S. W. Sen, H. Ford, D. R. Musicant, M. Graham, O. S. Keyes, and B. Hecht. Barriers to the Localness of Volunteered Geographic Information. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 197–206, New York, NY, USA, 2015. ACM.

[74] G. Sheikholeslami, S. Chatterjee, and A. Zhang. Wavecluster: A multi-resolution clustering approach for very large spatial databases. In *VLDB*, volume 98, pages 428–439, 1998.

[75] P. Shvaiko and J. Euzenat. Ontology matching: State of the art and future challenges. *IEEE Trans. Knowl. Data Eng.*, 25(1):158–176, 2013.

[76] P. Shvaiko, F. Giunchiglia, and M. Yatskevich. Semantic matching with s-match. In *Semantic Web Information Management*, pages 183–202. 2009.

[77] G. Skoumas, D. Pfoser, and A. Kyrillidis. Location estimation using crowdsourced geospatial narratives. In *Arxiv Preprint*, 2014.

[78] K. Starbird, J. Maddock, M. Orand, P. Achterman, and R. M. Mason. Rumors, false flags, and digital vigilantes: Misinformation on twitter after the 2013 boston marathon bombing. *iConference 2014 Proceedings*, 2014.

[79] A. Stefanidis, A. Crooks, and J. Radzikowski. Harvesting ambient geospatial information from social media feeds. *GeoJournal*, 78(2):319–338, Dec 2011.

[80] A. Stefanidis, E. Vraga, G. Lamprianidis, J. Radzikowski, P. L. Delamater, K. H. Jacobsen, D. Pfoser, A. Croitoru, and A. T. Crooks. Health narratives in twitter: Locations, actors, and concepts in a case study of the zika virus. ?(?):?–?, 2016.

[81] J. Surowiecki. *The Wisdom of Crowds: Why the Many Are Smarter Than the Few and How Collective Wisdom Shapes Business, Economies, Societies and Nations*. Doubleday, May 2004.

[82] E. R. Tufte and P. Graves-Morris. *The visual display of quantitative information*, volume 2. Graphics press Cheshire, CT, 1983.

[83] L. Vaccari, P. Shvaiko, J. Pane, P. Besana, and M. Marchese. An evaluation of ontology matching in geo-service applications. *GeoInformatica*, 16(1):31–66, 2012.

[84] G. Valkanas and D. Gunopulos. Event detection from social media data. *IEEE Data Eng. Bull.*, 36(3):51–58, 2013.

[85] G. Valkanas and D. Gunopulos. How the live web feels about events. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 639–648. ACM, 2013.

[86] K. Wang, J. Han, B. Tu, J. Dai, W. Zhou, and X. Song. Accelerating spatial data processing with mapreduce. In *Proc. 16th ICPADS conf.*, pages 229–236, 2010.

[87] W. Wang, J. Yang, and R. R. Muntz. Sting: A statistical information grid approach to spatial data mining. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, VLDB '97, pages 186–195, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[88] E. W. Weisstein. Moore neighborhood. *From MathWorld–A Wolfram Web Resource. http://mathworld. wolfram. com/MooreNeighborhood. html*, 2005.

[89] J. Weng and B.-S. Lee. Event detection in twitter. *ICWSM*, 11:401–408, 2011.

[90] X. Xu, M. Ester, H.-P. Kriegel, and J. Sander. A distribution-based clustering algorithm for mining in large spatial databases. In *Data Engineering, 1998. Proceedings., 14th International Conference on*, pages 324–331. IEEE, 1998.

[91] W. Zhang and J. Gelernter. Geocoding location expressions in twitter messages: A preference learning method. *Journal of Spatial Information Science*, 0(9):37–70, Dec. 2014.

[92] X. Zhang, W. Jeon, S. Gibbs, and A. Kunjithapatham. Elastic html5: workload offloading using cloud-based web workers and storages for mobile devices. In *International Conference on Mobile Computing, Applications, and Services*, pages 373–381. Springer, 2010.

Curriculum Vitae

Contact Information

George Mason University,
Department of Geography and GeoInformation Science (GGS)
4400 University Drive, Fairfax
Virginia 22030, USA
Tel: (+1) 703-993-1000
E-mail: glampria@gmu.edu
Web: http://about.giorgos.me

Education

- Nov 2009 - Jul. 2016: PhD from the School of Electrical and Computer Engineering, National Technical University of Athens, Greece. Thesis title: "Advanced techniques and algorithms to collect, analyze and visualize spatiotemporal data from social media feeds". Supervisor: Prof. Y. Vassiliou
- Sept. 2002 - Sept. 2009: MSc, Electrical and Computer Engineering, National Technical University of Athens. Greece

Work Experience

- Oct. 2014 - present: Research scholar / Research associate. George Mason University, Fairfax VA USA.
- Mar. 2011 - Sept. 2014: Software engineer / PhD candidate. Institute for the Management of Information Systems (IMIS). Research and Innovation Centre "Athena", Athens Greece.
- Sept. 2012 - Dec. 2012: Software engineer internship. Google UK Ltd. - YouTube, London UK.
- Feb. 2012 - May 2010: Web developer. Daily Secret Inc., Athens Greece.
- May. 2011 - Aug. 2011: Corporate Operations Engineer. Google Ireland Ltd., Dublin Ireland.
- Mar. 2009 - Sept. 2010: Web and AS developer. Grebooca / Gameyola @ fbFund 2009, Athens Greece & Palo Alto CA USA.

Volunteering

- 2014: Volunteering at the Internation Olympiad in Informatics (IOI 2014) held in Athens Greece, after the Olympic Games.

Publications

Journals

- A. Crooks, D. Pfoser, A. Jenkins, A. Croitoru, A. Stefanidis, D. Smith, S. Karagiorgou, A. Efentakis and **G. Lamprianidis**. Crowdsourcing urban form and function. In International Journal of Geographical Information Science, 2015, 1-22

Conferences

- **G. Lamprianidis**, D. Skoutas, G. Papatheodorou, and D. Pfoser. Extraction, integration and analysis of crowdsourced points of interest from multiple web sources. In Proceedings of the 3rd ACM SIGSPATIAL International Workshop on Crowdsourced and Volunteered Geographic Information, pages 16-23. ACM, 2014
- **G. Lamprianidis**, D. Skoutas, G. Papatheodorou, and D. Pfoser. Extraction, integration and exploration of crowdsourced geospatial content from multiple web sources. In Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pages 553-556. ACM, 2014
- A. Efentakis, S. Brakatsoulas, N. Grivas, **G. Lamprianidis**, K. Patroumpas and D. Pfoser. Towards a flexible and scalable fleet management service. In Proceedings of the 6th ACM SIGSPATIAL International Workshop on Computational Transportation Science, IWCTS 2013, pages 79-84, New York, NY, USA, 2013
- A. Efentakis, N. Grivas, **G. Lamprianidis**, G. Magenschab and D. Pfoser. Isochrones, traffic and demographics. In Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPATIAL 2013, pages 538-541, New York, NY, USA, 2013. ACM
- **G. Lamprianidis** and D. Pfoser. Collaborative geospatial feature search. In Proceedings of the 20th International Conference on Advances in Geographic Information Systems, pages 169-178. ACM, 2012
- **G. Lamprianidis** and D. Pfoser. Jeocrowd: collaborative searching of user-generated point datasets. In Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pages 509-512. ACM, 2011

Preprints - Under submission

- **G. Lamprianidis** and D. Pfoser and T. Sellis. Spatiotemporal Summaries of Social Media Content. "Social Media Processing" special issue of ACM Transactions on Intelligent Systems and Technology, 2016. *Submitted January 2016*
- A. Stefanidis, E. Vraga, **G. Lamprianidis**, J. Radzikowski, P. L. Delamater, K. H. Jacobsen, D. Pfoser, A. Croitoru, and A. T. Crooks. Health narratives in twitter: Locations, actors, and concepts in a case study of the zika virus. *Submitted June 2016*

Awards and Honors

- 2012: Best poster award. 3rd ACM SIGSPATIAL International Workshop on Crowdsourced and Volunteered Geographic Information. Extraction, integration and analysis of crowdsourced points of interest from multiple web sources. **G. Lamprianidis**, D. Skoutas, G. Papatheodorou, and D. Pfoser.

Research Projects

- 2013 - 2015: Software engineer. European Union Seventh Framework Programme "GEOSTREAM" - Exploiting User-Generated Geospatial Content Streams.
- 2012 - 2014: Software engineer. European Union Seventh Framework Programme "TMGuide" - TALOS Mobile Guides.
- 2012 - 2014: Software engineer. European Union Seventh Framework Programme "SimpleFleet" - Democratizing Fleet Management Solutions.
- 2011 - 2012: Software engineer. European Union Seventh Framework Programme, Initial training network "Geocrowd" - Creating a geospatial knowledge world.

Research Interests

- Spatial Databases.
- Crowdsourcing.
- Cloud computing.
- Parallel algorithms and distributed systems.
- Visualization techniques.