ONLINE ANALYSIS OF HIGH-VOLUME SOCIAL TEXT STEAMS

by

Nilesh Bansal

A thesis submitted in conformity with the requirements for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

© Copyright 2013 by Nilesh Bansal
Abstract

Online Analysis of High-Volume Social Text Streams

Nilesh Bansal
Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto
2013

Social media is one of the most disruptive developments of the past decade. The impact of this information revolution has been fundamental on our society. Information dissemination has never been cheaper and users are increasingly connected with each other. The line between content producers and consumers is blurred, leaving us with abundance of data produced in real-time by users around the world on multitude of topics.

In this thesis we study techniques to aid an analyst in uncovering insights from this new media form which is modeled as a high volume social text stream. The aim is to develop practical algorithms with focus on the ability to scale, amenability to reliable operation, usability, and ease of implementation. Our work lies at the intersection of building large scale real world systems and developing theoretical foundation to support the same.

We identify three key predicates to enable online methods for analysis of social data, namely:

- Persistent Chatter Discovery to explore topics discussed over a period of time,
- Cross-referencing Media Sources to initiate analysis using a document as the query, and
- Contributor Understanding to create aggregate expertise and topic summaries of authors contributing online.

The thesis defines each of the predicates in detail and covers proposed techniques, their practical applicability, and detailed experimental results to establish accuracy and scalability for each of the three predicates.

We present BlogScope, the core data aggregation and management platform, developed as part of the thesis to enable implementation of the key predicates in real world setting. The system provides a web based user interface for searching social media conversations and analyzing the results in multitude of ways. BlogScope, and its modified versions, index tens to hundreds of billions of text documents while providing interactive query times. Specifically, BlogScope has been crawling 50 million active blogs with 3.25 billion blog posts. Same techniques have also been successfully tested on a Twitter stream of data, adding thousands of new Tweets every second and archiving over 30 billion documents. The social graph part of our database consists of 26 million Twitter user nodes with 17 billion follower edges. The BlogScope system has been used by over 10,000 unique visitors a day, and the commercial version of the system is used by thousands of enterprise clients globally.

As social media continues to evolve at an exponential pace, there is a lot that still needs to be studied. The thesis concludes by outlining some of possible future research directions.
## Contents

1 **Introduction**  
1.1 Overview ................................................. 3

2 **BlogScope: System Description**  
2.1 Related Work ............................................... 6
2.2 Analyzing The Blogosphere  
2.2.1 Popularity Curve .......................................... 7
2.2.2 Keyword Bursts ........................................... 9
2.2.3 Keyword Correlations ..................................... 10
2.2.4 Hot Keywords ............................................. 11
2.2.5 Spatio-Temporal Search ................................... 12
2.2.6 Authoritative Blog Ranking  
2.2.7 Interface .................................................. 13
2.3 System Architecture  
2.3.1 The BlogScope Crawler .................................... 14
2.3.2 Spam Removal ............................................ 15
2.3.3 Searching and Indexing .................................... 16
2.3.4 Spatial Component .......................................... 16
2.3.5 Popularity and Bursts ..................................... 16
2.3.6 Keyword Correlations ..................................... 17
2.3.7 Authoritative Ranking ..................................... 18
2.3.8 Hot Keywords ............................................. 19

3 **Persistent Chatter Discovery by Identifying Stable Clusters**  
3.1 Related Work ............................................... 22
3.2 Cluster Generation ........................................... 23
3.3 Stable Clusters .............................................. 26  
3.3.1 The Cluster Graph ........................................ 28
3.3.2 Breadth First Search ....................................... 28
3.3.3 Depth First Search ......................................... 30
3.3.4 Adapting the Threshold Algorithm  
3.3.5 Normalized Stable Clusters  
3.3.6 Online Version ........................................... 36
3.4 Experiments .................................................. 36
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.4.1</td>
<td>Cluster Generation</td>
<td>37</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Stable Clusters</td>
<td>37</td>
</tr>
<tr>
<td>3.4.3</td>
<td>Qualitative Results</td>
<td>41</td>
</tr>
</tbody>
</table>

4 Persistent Chatter Discovery in Presence of Topic Hierarchies | 44 |
| 4.1 | Related Work | 45 |
| 4.2 | Problem Definition | 46 |
| 4.3 | Aggregation in Presence of Hierarchies | 47 |
| 4.4 | Aggregation with Probabilistic Guarantees | 48 |
| 4.4.1 | Probabilistic TA Assuming Geometric Distribution | 48 |
| 4.4.2 | Probabilistic TA Using Histograms | 50 |
| 4.5 | Deterministic Aggregation with Precision Guarantees | 52 |
| 4.6 | Preprocessing for Aggregation | 54 |
| 4.6.1 | Offline Preprocessing of Hierarchies | 56 |
| 4.6.2 | Online Processing of Hierarchies | 56 |
| 4.7 | Experiments | 56 |
| 4.7.1 | Evaluating $pH - RA$ | 56 |
| 4.7.2 | Sparse Interval Set | 60 |
| 4.7.3 | Real Data | 61 |

5 Cross-referencing Media Sources by Querying by Document | 63 |
| 5.1 | Related Work | 64 |
| 5.2 | Problem Definition | 66 |
| 5.3 | Phrase Extraction: QBD | 68 |
| 5.3.1 | Extracting Candidate Phrases | 68 |
| 5.3.2 | Scoring Candidate Phrases | 69 |
| 5.4 | Using Wikipedia: QBD-W | 71 |
| 5.5 | Experiments | 76 |
| 5.5.1 | Data Sets and Alternative Techniques | 76 |
| 5.5.2 | Quality of Phrase Extraction | 77 |
| 5.5.3 | Quality of Document Retrieval | 78 |

6 Contributor Understanding with Social Author Profiles | 82 |
| 6.1 | Peckalytics | 84 |
| 6.1.1 | Technology | 85 |
| 6.1.2 | Usage Experience | 86 |
| 6.2 | Topic Signatures | 87 |
| 6.2.1 | Constructing Topic Signatures | 90 |
| 6.3 | Related Work | 91 |
| 6.4 | Problem Formulation | 91 |
| 6.5 | Aggregate Signatures | 93 |
| 6.6 | Temporal Evolution of Aggregate Signatures | 95 |
| 6.6.1 | Minimizing RMSE | 96 |
| 6.6.2 | Minimizing Hamming Distance | 96 |
## List of Tables

3.1 Sizes of resulting keyword graphs .................................................. 23  
3.2 Example execution of DFS. ................................................................. 34  
3.3 Comparing BFS, DFS and TA based algorithms for different values of $m$. .................. 38  
4.1 Example snapshot of the buffer ............................................................ 53  
4.2 Sparse interval set system ................................................................. 55  
5.1 Example of Noun Phrase Patterns and Instances .......................................... 68  
5.2 Transition matrix for graph in Figure 5.3 ................................................. 74  
5.3 RelevanceRank scores after 1-5 iterations and at convergence ................................ 76  
5.4 $QBD-W$ run times for different $l_{\text{max}}$ .................................................. 79  
6.1 Top accounts for Toronto ....................................................................... 100  
6.2 Topics from AS for Toronto ................................................................. 101  
6.3 Topics from AS for Lamborghini and Ferrari .............................................. 102  
6.4 Topics for the query Lamborghini on Nov 08-09 2012 .................................... 102  
6.5 Overall topics for various Canadian banks .................................................. 105  
6.6 Running $TEMPEVOL$ for mobile handset manufacturers .............................. 106  
6.7 $TEMPEVOL$ for HSBC ........................................................................ 106  
6.8 $TEMPEVOL$ on machine learning ............................................................ 106  
6.9 Qualitative comparison of RMSE vs Hamming score function in $TEMPEVOL$ .......... 108  
6.10 Qualitative comparison of RMSE vs Hamming score function in $TEMPEVOL$ ............. 108
List of Figures

2.1 Popularity curves for keywords *Pixar* and *Abu Musab al-Zarqawi* ........................................... 8
2.2 Popularity comparison curves for keywords *soccer vs zidane.* .................................................. 9
2.3 Correlations for keywords *Philip Seymour Hoffman* ................................................................. 10
2.4 Example *hot keywords* cloud tag for 30th July 2006. ............................................................... 11
2.5 High level system architecture of BlogScope. ............................................................................... 14
2.6 Various components of the query execution engine and their interaction ..................................... 15

3.1 Example cluster for discovery of stem cell in amniotic fluid ....................................................... 21
3.2 Example cluster for soccer ............................................................................................................. 21
3.3 Example keyword graph ............................................................................................................... 26
3.4 Example of stable clusters with gaps ............................................................................................. 27
3.5 Example of cluster graph for three temporal intervals ............................................................... 28
3.6 Running time of the Art algorithm. ............................................................................................... 37
3.7 Running times for BFS based algorithm 1 .................................................................................. 38
3.8 Running times for BFS based algorithm 2 .................................................................................. 38
3.9 Running times for BFS based algorithm 3 .................................................................................. 39
3.10 Running times for BFS based algorithm 4 ................................................................................ 39
3.11 Running times for DFS based algorithm 1 ................................................................................ 40
3.12 Running times for DFS based algorithm 2 ................................................................................ 40
3.13 Running times for DFS based algorithm 3 ................................................................................ 40
3.14 Running times for BFS based algorithm 4 ................................................................................ 41
3.15 Stable clusters with path of length 3 without gaps ...................................................................... 42
3.16 Stable clusters with path of full length ...................................................................................... 43

4.1 Aggregating two lists in presence of a hierarchy. ......................................................................... 45
4.2 Example sparse system over 10 initial points ............................................................................ 55
4.3 Observed running times for the algorithm 1 .............................................................................. 57
4.4 Observed running times for the algorithm 2 .............................................................................. 57
4.5 Observed running times for the algorithm 3 .............................................................................. 58
4.6 Performance comparison .............................................................................................................. 58
4.7 Improvement factor ...................................................................................................................... 59
4.8 Time required for merging 100 lists ............................................................................................ 60
4.9 Merging 90 lists obtained from BlogScope, with correlated hierarchy. .................................... 60
4.10 Merging 90 lists obtained from BlogScope, with uncorrelated hierarchy. .............................. 61
4.11 Number of elements that need to be scanned ................................................. 62
5.1 Screenshot for QBD user interface. ................................................................. 64
5.2 PS Trie Forest. ................................................................................................. 69
5.3 Part of Wikipedia graph with five nodes ......................................................... 73
5.4 Precision against the annotator-nominated relevant phrases. ....................... 77
5.5 Precision after extracting the top- $k$ phrases. ............................................... 78
5.6 Retrieval precision using $k$ phrases ............................................................... 81
5.7 Retrieval precision for QBD-W ..................................................................... 81
6.1 Peckalytics Search UI .................................................................................... 86
6.2 Peckalytics Analytics UI ................................................................................ 88
6.3 The distribution of Twitter users and their list membership ......................... 89
6.4 Sample Twitter user and topic graph ............................................................ 94
6.5 AGGR run time .............................................................................................. 99
6.6 AGGR accuracy ............................................................................................. 100
6.7 AGGR number of topics ............................................................................... 101
6.8 Time taken for TEMPVOL (in ms) vs number of days .................................. 102
6.9 Memory consumption vs number of days ..................................................... 103
6.10 Number of mentions of Lamborghini on Twitter .......................................... 104
6.11 Keywords correlated with HSBC ................................................................. 107
6.12 Number of mentions of “machine learning” on Twitter on day-by-day basis ... 107

7.1 Replicated data indexes in BlogScope. ......................................................... 113
Chapter 1

Introduction

Social media has been proliferating over the last several years. Today over a billion users are sharing content online in numerous ways including blogs, forums, comments, micro-blogs, videos, and images. There are over 180 million active blogs, increasing five folds from 36 million blogs just seven years ago. Twitter has 200 million users, posting 12 billion Tweets every month. The service recorded at average 5,000 Tweets per day in 2007, and now the same number of Tweets are being sent out every second. Facebook graph has 140 billion friend connections. 600 million people worldwide use their mobile devices to access social media services. The amount of information generated on a daily basis is unprecedented and is continuing to grow at a rapid pace [34, 106, 18].

Social media has two unique characteristics:

• **Abundance of data**: Users online post about diverse topics including their personal lives, product reviews, political opinions, technology trends, tourism experiences, sports events, and the entertainment industry. This includes both real-time commentary on current events and news, as well as opinions on subjects that are not specific to current time. For the first time, we have an unprecedented amount of data. Data on every subject, and of all kinds and opinions. Data authored all over the world and all the time. And for the first time, it is not the scarcity of information, but the abundance of it that creates the challenge. There are four main challenges. First, social media data is noisy and full of spam. Second, the analyst doesn’t always know exactly what information he or she seeks but require automated identification of ‘interesting’ and ‘relevant’ knowledge. Third, there is too much data requiring computationally efficient algorithms to analyze. Lastly, all the data is unstructured requiring flexible analysis and scalable visualization.

• **Easy dissemination of information**: Users are connected to their friends, family, and organizations of interest online. For example, the average degree of separation on Twitter is just 4.67 [99]. With the internet, it is easy to create a blog or social profile and voice opinions. Companies and brands frequently have thousands to millions of followers and fans for easy dissemination of company related news. Spreading of information has become cheap and real-time, albeit overwhelming at times. As a result, marketers and advertisers are interested in identification of influencers and online brand advocates. These influencers are selected based on their ability to spread information of a certain topic to the widest possible set of desired audience. But given that there are over a billion users online, it is challenging to study them all and access their reach.

Temporal nature is another notable characteristic of social media, although not unique. Every post in social media has a time of creation. Frequently, the content of a post is influenced by other posts created earlier (e.g., a
blog post in response to another post on the same topic) and the time at which it is posted (e.g., many posts during the US presidential elections are about politics). These three aspects significantly differentiate social media from traditional sources of information. As a result, new set of techniques are needed to work with this new data source.

Social media has utility for both personal and business uses. An estimated 48% of Facebook users, aged 18-34, check the site right after they wake up, and 28% of them do so before even getting out of their bed. Twitter has become one of the primary source of real-time commentary with many high profile events such as the Academy Awards, Olympic games, or news stories like the landing of the US airways flight 1549 in the Hudson river after striking a bird being live tweeted. Social networks have played a prominent role in politics, and have been catalytic for organization of the Arab Spring, Occupy Wall Street, and several other protests. Collecting, monitoring and analyzing information on social networks can provide key insights on ‘public opinion’ on a variety of topics, such as products, political views, or entertainment. Information present in this data can identify events of interest based on the popularity of different topics.

It can also be a source of competitive intelligence information (e.g., Cymfony, a market influence analytics company, monitors blogs [28]). Social media analysis also provides key insights of marketing campaigns. It is projected that spend on social media related marketing and advertising will reach $34 billion by 2016, up from $12 billion in 2012 [51]. As a result, techniques that aid the collection, analysis, mining and efficient querying of social sources are important and this trend is expected to persist, given the growing popularity of social media.

We have social data on every subject and of all kinds, hence we need scalable, online, and flexible methods to uncover insights from this data. In rest of this thesis, we look at different functionality that can be built on top of such high volume social text streams with the goal of taming information overflow. We specifically explore three key predicates:

- **Persistent Chatter Discovery**: exploration of the most discussed topics and trends, specifically those that persist over time. We expect, when there is a lot of discussion on a specific topic or event, a highly correlated set of keywords will appear. For example, we expect the keywords ‘royal’, ‘baby’, ‘kate’, ‘george’, ‘prince’ to have formed a cluster when the royal baby was born on 22 July 2013. We formalize this notion to discover persistent chatter. Furthermore, as ranked lists–of top terms or cluster of terms–are created for select temporal intervals, we introduce new ways of aggregating these lists in presence of hierarchies. This is challenging given the abundance of data.

- **Cross-referencing Media Sources**: connecting posts from one media source to another. The goal is to obtain a holistic coverage on any topic evolving in social media. As an example, given the official news for the birth of the royal baby (from traditional media sources), we wish to find reactions from individuals by querying social media sources. This is challenging given the unstructured textual nature and abundance of data.

- **Contributor Understanding**: studying users creating content online and their ability to spread this information. We propose techniques to associate a user account with a set of topics and expertise. The users can be analyzed collectively to understand their common traits and interests to better understand the dynamics of content creation and reach in social networks. This is motivated by the easy dissemination of information.

To support this functionality, we built the BlogScope system which provides the base data aggregation and management platform. The functionality presented in this thesis was implemented and tested on top of this system. BlogScope has been tracking over 50 million non-spam blogs, indexing over 3.25 billion blog posts in its database. Modified version of the same system was tested on Twitter to successfully crawl over 4000 posts every second in real-time and archiving well above 30 billion Tweets.
1.1 Overview

Each of the following chapter is written to be self contained. They can be read in isolation, and each chapter introduces the specific problem being discussed, references related work, explains the proposed techniques, and concludes with experimental results. Each chapter deals with a significantly different aspect of techniques developed as part of this thesis and minimal notation is common across the techniques. As a result, the notation is not shared across chapters, and each chapter can be read on its own in order of reader’s choice.

The focus of work constituting this thesis lies at the intersection of building large scale real world systems and developing theoretical foundation to support the same. Practical applicability of the algorithms developed is of paramount importance to our work. These practical considerations include the ability to scale, amenability to reliable operation, usability, and ease of implementation in real world systems. Section 7.3 describes in detail the underlying philosophy of simplicity, wherein we explore the spectrum of possible solutions ranging from (1) simple and naive, (2) simple but clever, (3) complex and clever, to (4) complex, and we seek algorithms from the second category. The reader can observe and appreciate this as one the central themes in the following chapters.

We provide an overview of each chapter below. The list of published work covered in the chapter are noted as abbreviated conference name and the year of publication alongside the chapter overview.

The next chapter provides details on functionality offered by BlogScope and its system architecture (WWW’07, VLDB’07, and WebDB’07 [7, 8, 6]). The system is designed to provide interactive response times for all queries while operating on a tens of billions of archived documents with multi-terabyte indexes and processing new posts in real-time. A boolean keyword query and filters on date range, location, age, gender and other attributes, typically takes a few seconds to provide search results along with temporal bursts and summary of correlated keywords.

Chapter 3 discusses techniques for persistent chatter discovery by identification of stable clusters (VLDB’07 [4]). As topics evolve and events take place, sets of keywords become correlated representing the topic. In this chapter, we discuss efficient algorithms to identify such sets of keywords over a time range. Given the vast amounts of data involved, we present algorithms that are fast and take special care to make them efficiently realizable in secondary storage. Experimental results are presented using both real world and synthetic data.

Chapter 4 details the problem of persistent chatter discovery in presence of topic hierarchies and presents efficient algorithms with both probabilistic and approximate stopping conditions (SIGMOD’08 [5]). Specifically, we take a different approach to information discovery from ranked lists, moving beyond the well studied top-$k$ algorithms. We maintain the rank aggregation framework but we elevate terms at a higher level by making use of term hierarchies commonly available. Under such a transformation we show that typical early stopping certificates available for rank aggregation algorithms are no longer applicable. Based on this observation, in this chapter, we present a probabilistic framework and a relaxed deterministic framework. Through a detailed experimental evaluation using synthetic and real datasets we demonstrate the efficiency of our framework and established the fact that relaxing precision requirements results in significant performance gains.

Chapter 5 discusses cross referencing of media sources by querying by document (WSDM’09 [113]). Given the plurality of content there is a pressing need to cross reference information across sources and obtain holistic information about a topic. We present a solution to this problem, allowing the user to submit a text document as a query and identify related documents from another text corpus. To provide such functionality, we present techniques to process text documents on demand and extract key phrases which are used to query BlogScope for retrieving blog posts related to the query document. Further, we show that it is possible to extend these ideas towards the development of additional query types. We validate the utility of our techniques by submitting extensive sets of results to Amazon’s Mechanical Turk (MTurk) for human evaluation. This enables users at large
to act as judges of the quality of our findings.

Chapter 6 focuses on contributor understanding with social author profiles, and describes the Peckalytics system (SIGMOD’13 [20]). The live system utilizes the Twitter Gardenhose streaming API to collect and analyze Tweets. Main features of the system include the ability to identify expert accounts on any topic, and further analysis of these accounts to explore other topics they are experts at, conversations they participate in and types of content they share online. Peckalytics algorithms can scale to operate on real-time Twitter stream with an archive of well above 30 billion documents. Raw data size, counting only a single copy and excluding all redundant copies for parallelization or reliability, run in tens of terrabytes. We make novel use of Twitter Lists to create topic signatures for contributors. We formalize the notion of evolution of topic signatures for contributors as it evolves over time, and develop algorithms for identification of time periods of interest. Experimental results validate the efficiency of our algorithms for both quality and scalability.

Lastly, Chapter 7 concludes this thesis by listing the contributions made in this thesis:

- The BlogScope system was developed to operate on real world data collected from blogs and other social channels. The system scales with data, and maintains interactive query times while operating on database with several billion documents. Core functionality provided by the system is flexible and is able to support specialized predicates developed as part of the thesis.

- Identification of three key predicates to enable online methods of uncovering insights from this data and techniques to implement the same. Experimental results are provided using high volume real world data to establish accuracy and scalability of the proposed techniques.

Techniques and core data aggregation platform developed as part of this thesis is also covered by US patent application [9]. Other researchers have utilized these techniques as building blocks for different projects, including BlicqTimes and Grapevine [2, 75], and research works [86, 94, 74]. The university awarded the 2011 University of Toronto Inventor of the Year in Information and Computer Technology award for this work.

Same techniques have also been applied by the Sysomos [98], a commercial spin-off of the BlogScope project. Sysomos is used by thousands of companies worldwide, primarily in public relations, marketing, advertising, customer support and consumer insights divisions. 70% of the Fortune 50 brands, including Coca Cola, Intel, Microsoft, Adidas, Unilever, Google and Visa, use Sysomos for their global social media analytics needs. With several thousand terrabytes of raw data, Sysomos is also the largest commercial social analytics platform of its kind. At the time of writing, the system is providing interactive analytics on an archive of 200+ billion documents.

The conclusions chapter covers all the contributions, published works, and spin-offs in detail. Finally, the last chapter lists the key learning and cites examples for the same. The thesis concludes with a list of possible future directions for research in this area.
Chapter 2

BlogScope: System Description

Given the exponential growth in social content, we have developed the BlogScope system to provide core search, analytics, and visualization specifically for this new media source. Initial version of the BlogScope was developed specifically for collecting and processing blogs, and was later extended to include other media sources. In the rest of this chapter we will refer to analysis on blogs and the Blogosphere, but all the techniques presented herein are generic enough to apply to other social channels equally well.

Using BlogScope as the core content aggregation and management platform, further specific algorithms and techniques were implemented, as presented in later chapters of this thesis. In this chapter, we focus only on the core functionality of the system as implemented on blogs.

Traditional web search technology can be applied to the Blogosphere to provide keyword search. But information in blogs have two main characteristics not present in traditional web: (1) blog posts have a time of creation adding a temporal dimension, and (2) blog posts may trigger additional posts leading to a discussion in the Blogosphere.

Primary challenge faced in building such as system is the volume of the data. BlogScope has been tracking over 3.25 billion blog posts from 50 million non-spam posts. As new posts are added in real-time the indexes need to be updated reliably and consistently. The system has been used by over 10,000 unique visitors a day on average, requiring high throughput despite operating on a multi-terabyte index. A modified version of the system, Peckalytics, has been tested on a collection of over 30 billion Tweets while still providing interactive query times. We discuss the challenges in detail in Chapter 7.

We now provide an example to motivate temporal analysis capabilities of our system. Consider a search for information related to the actor ‘Phillip Seymour Hoffman’ on the Blogosphere. A traditional search engine would offer a list of all blog posts, containing the search string. Although this is informative, in terms of information discovery, there is more functionality one can offer in the case of blogs. For example, observe a graph displaying the relative popularity of the keywords ‘Philip Seymour Hoffman’ in the Blogosphere as a function of time and mark regions of time that the search string shows unusual popularity. These can be temporal regions that one may wish to focus, refining the search. This particular query ‘Philip Seymour Hoffman’ displayed unexpected popularity in the year 2006 when the actor was nominated for Oscar, when he received the Oscar award and when a subsequent movie that he was acting was released (MI3).

To aid information discovery, details explaining the ‘unusual’ popularity of the keywords ‘Philip Seymour Hoffman’ in the corresponding temporal intervals should be automatically provided. We argue that keywords that are highly correlated (in blog posts) with the search string in a temporal interval of choice are good candidates for
explaining ‘unusual’ popularity. In the first case, the query is closely correlated with the keywords ‘Capote’ (the film he acted and was nominated for an Oscar) and ‘Oscar’. For the second temporal interval with the keywords, ‘Oscar’, ‘Actor’, ‘Capote’ and ‘Crash’ (another movie winning an Oscar) and for the third the correlated keywords were ‘Tom Cruise’, ‘MI3’ (a co-star in the movie MI3). Such keywords provide information on why the query shows relatively ‘unusual’ popularity in the corresponding time interval. Such ‘correlations’ can be discovered repeatedly, possibly triggering additional information discovery. For example one might choose to identify the keywords correlated with both ‘Philip Seymour Hoffman’ and ‘Capote’ in the first temporal interval. Such a functionality would enable a finer exploration of posts in the temporal dimension. Essentially it enables a more focused ‘drill down’ in the temporal dimension.

Information discovery in the Blogosphere is not necessarily query driven. One should be able to monitor posts and automatically suggest ‘interesting’ keywords to explore further. This paradigm is different from the features offered by popular blog search services that solely monitor queries or blog post tags, and rank them based on relative popularity. One can extract a wealth of related information from blogs in order to aid information discovery. The list includes adding a spatial component to queries and correlations, identifying temporal dynamics in the list of keywords correlated to a specific keyword, and mapping correlated keywords to topics.

In this chapter, we present BlogScope. Its features include (a) online, temporal burst detection for keywords, (b) efficient identification of correlated sets of keywords, (c) support for online OLAP style of analysis of the Blogosphere based on correlations and bursts (d) spatial blog post navigation (e) extraction of summaries for effective browsing in the form of hot keywords in the Blogosphere (f) blog post ranking based on authority and (g) support for effective mining of interesting keywords. We have developed efficient algorithms for burst identification, discovering correlated keywords, and mining interesting keywords.

2.1 Related Work

The growing popularity of blogs and the increase in the number of people maintaining blogs resulted in interest in search and analysis engines for the Blogosphere. The engines closely related to BlogScope are Technorati [101], Blogpulse [14], IceRocket [57], and Google Trends [46]. A close examination of the features of such engines in comparison to BlogScope reveals that correlations, bursts, and hot keywords for arbitrary time ranges are features unique to BlogScope.

Several search engines for blogs (e.g., Technorati, IceRocket) display a list of related “tags” along with query search results, for navigation. However, the drawback to this approach is that since tagging requires manual effort by bloggers, most of the content in the Blogosphere is not tagged. Also, the number of tags for a document is usually less than ten, while actual content itself may contain thousands of words. Therefore, tags generally cannot accurately represent the contents of a document (and they may be subjective or prone to spam). Hence, it is more accurate to consider the whole content of the document for analysis. Known methods and systems base their analysis on tags and search volume and not on actual text content. Also, none of the current methods and systems take into account the restrictions on time range and geographical region, if any.

The number of inlinks to a page is a commonly used measure of authority. For example, Google’s page rank algorithm makes use of such information. This measure has proven its effectiveness over time for web documents. However, this simple definition of authority ignores the context (in relationships to a query) and time information and hence is generally inadequate for the Blogosphere, or any other temporally ordered information sources.

TagLines [31] by Yahoo Research is a tool for visualizing interesting tags at flickr.com as they evolve over time. TagLines requires computation of “what is interesting” for arbitrary time periods (e.g., 1st April
2006 to 18th May 2006), just like BlogScope’s hot keywords feature. Such a feature is not available in any of available blog search engines. The list aggregation semantics adopted by TagLines and BlogScope follow different philosophies. TagLines uses scored lists, which can be aggregated by adding the scores of same object across different lists; while BlogScope operates on ranked lists using Footrule based aggregation. Therefore the two proposed precomputation strategies differ significantly.

Data storage and distributed processing framework are important components of the BlogScope system. Since the initial release of BlogScope in 2006, several new software geared specifically towards processing large scale streaming data have been engineered (see [11] and references therein). These primarily include two categories: (1) data stores such as MongoDB and Apache Cassandra [66] and (2) frameworks for distributed data processing such as Muppet [67] and Storm. The new emerging software can provide additional options for building blocks for the data storage and distributed processing framework components of a social media analytics system similar to BlogScope.

2.2 Analyzing The Blogosphere

The analysis paradigm that BlogScope facilitates is segmented in three steps. BlogScope identifies what is ‘interesting’, when it was ‘interesting’ and why it is ‘interesting’. On its front page, BlogScope displays a list of ‘interesting’ keywords to aid information seekers identify what. Based on this list, user can formulate a query to seek for relevant blog posts. On its front page, BlogScope supports a text query interface to identify posts relevant to a query, in case one is seeking for specific information. Once the keywords of interest are identified, a query is formed and relevant blog posts are retrieved. The next question BlogScope aids to answer is when it was interesting. To answer this question, BlogScope plots the popularity of the query keywords in blog posts, as a function of time, and identifies and marks interesting temporal regions as bursts in the keyword popularity. The final step of the analysis is to investigate why it is interesting. Correlated keywords (intuitively defined as keywords closely related to the keyword query at selected temporal interval) are automatically displayed by BlogScope. Such keywords aim to provide explanations or provide insights as to why the keyword experiences a surge in its popularity. Based on these keywords, one can refine the search and drill down in the temporal dimension towards a more focused subset of blog posts. In the rest of this chapter we discuss all components of BlogScope in detail.

2.2.1 Popularity Curve

The popularity curve for a keyword (or set of keywords) displays how often the query terms are mentioned in the Blogosphere as a function to time. Such a curve and its fluctuation can provide insight regarding the keyword popularity evolution over time. Figure 2.1 provides examples of popularity curves for the queries Pixar and Abu Musab al-Zarqawi.

Utilizing popularity curves, one can compare different keywords using BlogScope, the popularity of various keywords. We expect that closely related keywords will have similar popularity curves at least in the temporal interval when the keywords are related. Hence, comparison of such curves provides an alternative approach to the analysis of the temporal relationship between keywords. Figure 2.2 displays the popularity of keywords zidane and soccer. Notice that the keywords exhibit strong similarity in their popularity for a short temporal interval. This interval spans a few days before the 2006 FIFA World Cup final game with a peak the day of the game, given the incidents in the game related to zidane.

Popularity curves can be a useful tool for marketers and public relations executives. They can be used, for
Figure 2.1: Popularity curves for keywords Pixar and Abu Musab al-Zarqawi respectively. The movie Cars by Pixar was released on 9th June 2006. Abu Musab al-Zarqawi, a member of Al-Qaeda in Iraq, was killed in an US air strike on 7th June 2006. Regions marked in red indicate bursts.
example, to measure product penetration by comparing popularity curves of a product along with that of a competitor. Such curves, when coupled with the semantic orientation of the associated blog posts \cite{80, 104} can provide insight for a product popularity in relationship to another. Popularity curves can also be used to access decisions, like marketing strategy changes, by monitoring fluctuations in popularity (e.g., as a result of a marketing campaign).

### 2.2.2 Keyword Bursts

Although blogging activity is uncoordinated, whenever something of interest to a fraction of bloggers takes place (e.g., a natural phenomenon like an earthquake, a new product launch, etc), bloggers write about it. As a result, the popularity of certain keywords increases. This allows BlogScope to identify and mark such interesting events on a popularity curve. We refer to these events as bursts. The notion of burst adopted by BlogScope is related to the notion of unexpected popularity for a keyword within a temporal interval. Bursts play a central role in analysis and blog navigation using BlogScope, as they identify temporal ranges to focus and drill down, refining the search. Figure 2.1 shows two examples of bursts.

Bursts can be categorized in two main types: anticipated and unanticipated. Popularity for anticipated bursts increases steadily, reaches a maximum and then recedes in the same manner. Release of a movie or soccer world cup fall under this category. Unlike anticipated bursts, popularity for unanticipated bursts increases unexpectedly. Hurricane Katrina or the death of Abu Musab al-Zarqawi fall under this category (Figure 2.1).

In addition to aiding navigation in BlogScope, bursts can also be used to produce intelligent alerts for users. Subscribing to specific keyword queries, BlogScope could generate an alert (in the form of email) only when a burst occurs in the popularity of specific keywords. In this way an alert will be raised only when something potentially interesting has occurred. Currently available alert services (e.g., Google Alerts \cite{45}) suffer from two main problems: (1) An alert is raised whenever any new document (e.g., blog post) containing the query is encountered by the crawler. Discovery of a new document may not necessarily imply occurrence of an interesting
event. (2) The number of alerts is large to handle, if the number of documents containing the specified query is large. BlogScope’s alerts service is free from these two problems.

2.2.3 Keyword Correlations

Information in the Blogosphere is highly dynamic in nature. As topics evolve, keywords align together to form stories; and as topics recede, these keyword clusters dissolve. This formation and dissolution of clusters of keywords is captured by BlogScope in the form of correlations.

With every search, a list of keywords in blog posts most closely related to the search query keywords is displayed. Such keywords can be seen as representative tokens for chatter in the Blogosphere, and can be used to obtain insight regarding the posts relevant to a query. Correlations are not static, as they may change according to the temporal interval specified in the query. Users can specify a temporal range for which a list of keywords correlated to query keywords is produced. Provided that users navigate, drilling down to posts related to a burst, such correlations can be used to reason why a burst occurred. Figure 2.3 shows a screenshot of correlations for Philip Seymour Hoffman for two different time periods, 1st-20th March and 1st-20th May 2006. It can be seen that correlations are different for different time intervals, and they reflect the events that occurred then. Choosing one of these keywords (say ‘capote’ in Figure 2.3) a list of keywords correlated to ‘Philip Seymour Hoffman’ and ‘capote’ in the temporal range will be produced, along with the associated popularity curve for the pair. Correlations are also employed by BlogScope to provide an exploratory navigation system. A user can easily navigate from a keyword to related keywords and explore by following correlation links.
2.2.4 Hot Keywords

On its front page BlogScope displays a list of hot keywords for that day in the form of a cloud tag. BlogScope uses a measure of ‘interestingness’ for keywords (see Section 2.3.8) and ranks all keywords for a day according to this measure. Interesting does not necessarily refer to popular. Keywords that exhibit sudden change in their popularities are more interesting. We define our measures of interesting in detail in Section 2.3.8. The highest ranking keywords according to this measure are displayed on the front page for that day with the font size proportional to the measure of interestingness. For example recent highly popular terms in the blogosphere have been, war, iraq, elections, democrats, senate, to name a few (for the week of November 13 2006). In a few occasions, Blogscope tracked popular keywords that corresponded to events that have not made mainstream news media. For example the term math was highly popular on the week of August 7 2006 in the blogosphere as reported by BlogScope. The event corresponded to the news about the Poincare conjecture proof by Grigory Perelman. New York Times had an article on this on August 15 2006. Figure 2.4 shows an example screenshot taken on July 30 2006.

The list of hot keywords is intended to play a guiding role in the analysis process. BlogScope provides a rich interface where the user can specify a temporal range (e.g., 1st March to 31st March, 2006) and a threshold on the “interestingness” to generate a list of hot keywords for that temporal range. This allows analysis of past data. We discuss efficient ranked list aggregation algorithms to support such queries in Chapter 4.

Popularity of various keywords and topics in Blogosphere can be analyzed in detail by aggregating the lists of hot keywords utilizing a hierarchy. For example, popular car sites, contain hierarchies organizing cars by make, model, type, etc. All online shopping sites have hierarchies on the products they offer, such as kind, type, brand, model, features etc. Even an individual analyst can supply his or her own hierarchy. Utilizing such hierarchies would aid discovery of different types of popular events. For example, there might be lots of discussion about several digital camera models, such as the Canon S series of cameras. The terms S700, S600, etc, which are Canon
digital camera models correspond to chatter about Canon digital cameras. However individually such terms might not have enough popularity to make it to the top ranking set of keywords, for rank aggregation queries. Utilizing product hierarchies, enables us to elevate such terms to the term canon digital camera. Now, the total score of that newly introduced term in the presence of the product hierarchy would be higher (equal to the aggregate of the scores of all individual terms mapped to Canon digital camera). We wish to support highly dynamic hierarchies so we chose to impose no restriction on them, their shape or type and we assume that they are supplied on demand (per query). For this purpose, we discuss various probabilistic and deterministic list algorithms in Chapter 4.

2.2.5 Spatio-Temporal Search

Keyword search is an important part of any analysis engine. The search in BlogScope displays search results with snippets and links to full articles (blog posts). A user can choose between a standard and a stemmed index. The standard index conducts searches for exact keywords. For example, when searching with a standard index for the results of the query “consideration”, all articles containing the term consideration will be returned. However, when searching with the stemmed index, all English words are first converted to their roots, and hence search for the query “consideration” will return articles containing either of consider, consideration, considerate or considering.

There are important properties of the Blogosphere that cannot be easily captured by the ranking model in traditional web search. Documents on the web do not have a timestamp associated with them, while blog posts have a definite time of creation. Simple relevance based ranking using \( tf \cdot idf \) will mean ignoring the temporal dimension. Pure temporal recency based ranking will not be very good either. As a first attempt to address the ranking of search results in the Blogosphere, BlogScope employs a combination of both temporal recency and relevance to rank search results.

In addition, BlogScope associates a geographical location with every blog post. With every post that is indexed by BlogScope, a city, state and country is maintained and when possible exact geographical coordinates (in terms of latitude and longitude). As a result of a query, BlogScope has the option to display the blog post results in the selected temporal interval on a map, displaying the distribution of the posts in cities of each country per continent. Users can restrict viewing by selecting countries or cities on the map and drill down to the posts in each geographical region.

BlogScope can easily collect other types of meta data associated with blog posts. For example if instead of blog posts, BlogScope warehouses financial information or news, such textual information will be associated with a source (e.g., Reuters, Thompson financial, Bloomberg news, etc). This information is recorded by BlogScope, and results can be suitably restricted to each source or industry category.

2.2.6 Authoritative Blog Ranking

Two features in BlogScope enhancing the spatio-temporal search are authoritative ranking and burst synopsis. The semantics associated with the burst synopsis set for an initial query \( q \) is that it is the maximal set of keywords associated with \( q \) that exhibits a bursty behavior in the associated popularity curve for the set. Synopsis sets may have an arbitrary size (number of keywords) provided that all included keywords contribute to the burst. Authoritative blogs are blogs read by a large number of readers, and are usually first to report on news. These blogs play an important role in dissemination of opinions in Blogosphere.

Consider the query ‘italy’; blog posts may mention the keyword ‘italy’ in connection to both soccer and political events. All such posts contribute to the burst in the popularity of the keyword ‘italy’. The keywords ‘soccer’
and ‘politics’ are both correlated to keyword ‘italy’ in the associated temporal interval. However expanding the
search and observing the popularity curves of ‘italy, soccer’ and ‘italy, politics’ turns out that only the curve for
‘italy, soccer’ has a burst in the temporal interval of the three summer months of 2006. BlogScope can automat-
ically generate such synopsis keyword sets for a burst. In this case, only the set ‘italy, soccer’ will be identified
and suggested by BlogScope as a synopsis set, associated with the initial keyword query ‘italy’. Notice that the
set ‘italy, politics’ will not be identified as a synopsis set, because ‘italy, politics’ does not have a burst in the
corresponding popularity curve.

Based on such keyword sets, BlogScope automatically ranks blog posts related to the synopsis set based on
authority. Authoritative blogs are the ones that gave rise to the burst on the synopsis keyword set. These are
blogs that are relevant to the synopsis set, temporally close to the occurrence of the burst and most linked in the
Blogosphere.

As an additional example, a search using query ‘cars’ on June 9th 2006 results in the synopsis set {cars, pixar,
disney, movie} which disambiguate the burst resulted from the release of the movie Cars, from general discussion
about automobiles in the Blogosphere. Such set is accompanied with authoritative blog posts that were the first to
report the event and were most linked in the Blogosphere.

2.2.7 Interface

BlogScope adopts a simple and intuitive interface. Popularity curves provide OLAP style drill down and roll
up functionality in the temporal dimension. Outlinks on correlations constitute a network of guided pathways to
assist the user in a journey of Blogosphere exploration. Analysis using BlogScope can be summarized as a four
step process.

1. Select keywords to analyze. BlogScope supports ad hoc keyword queries as well as suggests keywords in
its hot keyword module.

2. Observe the search results (post snippets are displayed) ranked using BlogScope’s ranking function, the
associated popularity curve of the keyword searched and its correlated keywords. Select a spatial region, if
desired, to restrict the search in a specific geographic location.

3. Zoom in or out the popularity curve by selecting regions on it using the mouse. Select the time interval
to analyze based on recommendations by bursts. Select the synopsis keyword set generation feature and
observe blog posts ranked using authoritative ranking.

4. Use correlated keywords to reason about the burst. Outlinks on correlations can also be used to refine the
query or explore further.

2.3 System Architecture

BlogScope tracks over 50 million blogs with around 3.25 billion articles in its database. It is extremely important,
given the analysis the system conducts, for the techniques employed to be computationally efficient in order to
scale at this level. We therefore seek algorithms that are fast and effective. Figure 2.5 presents overall system
architecture of BlogScope. Figure 2.6 describes query execution flow and user navigation. In rest of this chapter
we discuss our techniques in detail.
2.3.1 The BlogScope Crawler

Crawling the Blogosphere is different from web crawling. An RSS feed is available for most blogs, and the crawler can fetch and parse the RSS XML instead of HTML. There is no need to follow outlinks as services like blo.gs [15] and weblogs.com [108] maintain a list of most recently updated blogs. The BlogScope crawler receives a list of blogs updated in the last one hour from weblogs.com [108]. We check this list against the list of spam blogs in our database (see next section for spam removal techniques employed), and schedule the rest to be fetched.

Other social channels provide different APIs depending on the source to collect the data in real-time. Twitter, for example, provides access via several streaming APIs and data partners. Facebook, similarly, provides Graph Search APIs.
2.3.2 Spam Removal

Spam is a big problem in the Blogosphere. Our experience\(^1\) with blogspot.com data shows that half of the blogs are spam. These pages exist to boost page rank of some commercial sites. It is cheap to create many spam entries as hosting is provided for free by blogspot.com. Software is available in the market capable to automatically create thousands of spam blogs in an hour [96].

Spamming techniques are becoming quite intricate which makes the task of spam detection difficult. Language modeling techniques are used to generate sentences that are not just random strings but make some sense. Techniques used by spammers are sophisticated enough to confuse even a human observer at first. Many times, spammers just copy a news article from other sources and insert a few outgoing links to create the spam entry. Main characteristics of a spam blog are, (1) they are created by machine, and (2) they contain outgoing links to commercial sites.

Researchers have studied the problem of web spam in the past. A comprehensive taxonomy of current web spamming techniques is presented by Gyongyi and Garcia-Molina [52]. TrustRank [53] proposes a spam detection framework based on link structure. Spam detection in weblogs is an active research area. Kolari et al. [63] have proposed an SVM based approach to spam removal from the Blogosphere.

Identifying spam is an important but difficult problem. Lesser amount of interlinking in Blogosphere as compared to the traditional web makes it a harder problem. Existence of many personal non-spam blogs with zero or few inlinks make link based techniques ineffective for spam detection. Effectiveness of content based spam detection techniques depends on the fact that spammers are unaware of them (otherwise examples to fool these algorithms can be easily designed).

BlogScope’s spam analyzer builds upon previous work, utilizing a Bayesian classifier [91] in conjunction with many simple (but highly effective) heuristics. For example, spam pages contain a large number of specific characters (e.g., “-” and numerals) and contain certain keywords like free, online and poker both in their urls as

---

\(^1\)We have manually looked at a random sample of few hundred blogs
well as in the urls of outgoing links. Capitalization of the first word of a sentence is often wrong in spam pages. Images are almost never present on spam blogs. These heuristics are based on manual analysis of blogs data.

2.3.3 Searching and Indexing

The crawler stores all its data in a relational database. At a certain time, this data is indexed to generate inverted lists and other statistics. We maintain two types of indices on all posts: standard and stemmed. Standard index maintains inverted lists for all tokens seen in the database while the stemmed index first converts all words to their roots, and maintains lists for all stemmed tokens. We also maintain a list of posts for each day. These indices form the core of our analysis engine. In a separate data structure, for efficiency, we also maintain term frequencies for each day and inverse document frequency over a one year sliding window for all stemmed tokens.

Given the size of the data, the indexes are partitioned by time ranges. For example, we index data for every calendar month on a separate server. Hence, server 1 will have data for January 2012, server 2 for February 2012, server 3 for March, and so on. At the time of the query, we first identify the index servers needed to fulfill the query. As an example, if the query is restricted to time period 13th Feb - 19th Mar 2012, we will need to first query server 2 and 3 individually and then merge the retrieved results to answer the search query. Note that the partition by time range does not have to be by months or any fixed length time period, but it can be very flexible as long as the time is used as the key to partition the data.

2.3.4 Spatial Component

Along with each blog post, while crawling, BlogScope attaches a city, state and country field and when possible geographical coordinates. There are several ways to infer a definite geographical coordinate given a blog post. These include:

- Utilizing metadata about location in the head of the blog. Several html tags and plug-ins exist to associate geographical information in blog posts [42]. BlogScope automatically identifies such tags by parsing them and attaches a geographical set of coordinates to the post.

- Utilizing information related to the address of the blogger from its profile. The profile of a blogger may contain address information. In that case BlogScope extracts this information and maps it to a geographic set of coordinates. Approximate match information offered by tools like Spider [65, 95] enables effective matching of addresses.

- Looking up blog content against a set of standardized zip codes and city names also allows to extract geographic information from blog posts.

With the aid of such coordinates one has the option to identify the posts as a result of a query into a geographical map. The search can be restricted to be specific geographical locations. BlogScope maintains inverted lists for city, state, country for blog posts. When the search is restricted using a spatial restriction, such lists are manipulated to suitably restrict the scope of the search. This enables BlogScope to conduct spatio-temporal navigation for blog posts and correlated keywords.

2.3.5 Popularity and Bursts

We track the Blogosphere popularity of keywords used in a query for a day by tokenizing the query and merging the inverted lists for each of the tokens (keywords) with the list of posts for that day. The popularity curve is
generated by repeating this process for each day. The main cost of this algorithm is fetching the lists from disk; once the lists are read in memory, merging can be conducted very efficiently.

Kleinberg [61, 62] has discussed burst detection in the context of text streams. Their approach is based on modeling the stream using an infinite state automaton. While interesting, this approach is computationally expensive. It requires computing the minimum-cost state sequence, which in turn requires solving a forward dynamic programming algorithm for hidden Markov models. It is therefore not possible to use this approach in our system where bursts need to be computed on the fly. Adapting this technique for on the fly identification of bursts would be prohibitively expensive. Fung et al. [40] have addressed the problem of bursty event detection, and have proposed techniques to identify sets of bursty features from a text stream. Inspired by the work of Fung et al. [40], the following algorithm is employed by BlogScope to detect bursts.

We model the popularity \( x \) of a query as the sum of a base popularity \( \mu \) and a zero mean Gaussian random variable with variance \( \sigma^2 \).

\[
x \sim \mu + N(0, \sigma^2)
\]

We can compute the exact popularity values \( x_1, x_2, \ldots, x_w \) for the last \( w \) days by using our materialized statistics. We then estimate the value of \( \mu \) and \( \sigma \) from this data using the maximum likelihood.

\[
\mu = \frac{1}{w} \sum_{i=1}^{w} x_i \quad \text{and} \quad \sigma^2 = \frac{1}{w} \sum_{i=1}^{w} (x_i - \mu)^2
\]

From the standard normal curve, the probability of the popularity for some day being greater than \( \mu + 2\sigma \) is less than 5%. We consider such cases as outliers and label them as bursts. Therefore, the \( i^{th} \) day will be identified as a burst if the popularity value for the \( i^{th} \) day is greater than \( \mu + 2\sigma \). In our current implementation of BlogScope we use \( w = 90 \) to compute \( \mu \) and \( \sigma \).

### 2.3.6 Keyword Correlations

The notion of correlation of two random variables is a well studied topic in statistics [82]. Quantifying the correlation \( c(a, b) \) between two tokens \( a \) and \( b \) can have many different semantics. One semantics, for example, can be

\[
c(a, b) = \frac{P(a \in D|b \in D)}{P(a \in D)} = \frac{P(b \in D|a \in D)}{P(b \in D)} = \frac{P(a \in D \text{ and } b \in D)}{P(a \in D)P(b \in D)}
\]

where \( P(t \in D) \) denotes the probability of token \( t \) appearing in some document \( D \) in the collection \( D \). In words, correlation between \( a \) and \( b \) is the amplification in probability of finding the token \( a \) in a document given that the document contains the token \( b \). Calculation of correlations using such semantics requires checking each pair of tokens. With tokens in the order of millions, calculating \( c(a, b) \) using the above formula for every possible pair across several temporal granularities would amount to a large computational effort. This is complicated by the fact that such correlations have to be incrementally maintained as new data arrive. Increasing the number of keywords one wishes to maintain correlations for, from two to a higher number, gives rise to a problem of prohibitive complexity.

We describe a fast technique to find correlations which is currently adopted by BlogScope. Consider a query \( q \) and the collection of all documents \( D \). Let \( D_q \subseteq D \) denote the set of documents containing all of query terms.
For a token $t$ we define its score $s(t, q)$ with respect to $q$ as

$$s(t, q) = |\{D|D \in D_q \text{ and } t \in D\}| \times \text{idf}(t) \quad (2.1)$$

where $\text{idf}(t)$ is the inverse document frequency of $t$ in all documents $D$. The first term in Equation 2.1 is the number of documents containing $t$ among those relevant to the query $q$. We multiply this frequency with $\text{idf}(t)$ which represents the inverse of overall popularity of the token in the text corpus. Commonly occurring tokens like “here”, “after”, “when” have high overall popularity and therefore low idf. Hence the proposed scoring function favors tokens which have low overall popularity but high number of occurrences in documents relevant to the query $q$. This represents keywords that are closely related to $q$ as these tokens appear frequently only in the documents containing $q$. The list of top-$k$ tokens having highest score with respect to $q$ forms a representative of $D_q$. We display this list as correlations for query $q$.

This technique requires a single scan over $D_q$. As we scan the documents in $D_q$, we can maintain a count for each token that appears in $D_q$ in a separate hash table. After the scan is complete, we can multiply these count values with precomputed idf values to find the scores, which can then be sorted to get the top-$k$. But even this could be prohibitively time consuming if the set $D_q$ is large. To circumvent this problem we bound the size of set $D_q$ by a number $m$; if there are more than $m$ documents containing query terms, we consider randomly selected $m$ documents from $D_q$. We denote the random sample of size $m$ of $D_q$ by $D_q^m$.

Notice that the proposed technique for finding correlated terms is in the same spirit as the one mentioned above based on amplification in probability. The number of documents containing both $q$ and $t$, $|\{D|D \in D_q \text{ and } t \in D\}|$, follows a binomial distribution, the characteristic probability of which could be approximated by scanning $m$ random documents from $D_q$. If we interpret the idf of a token $t$ as $\frac{|D|}{|D_t|}$, where $D_t$ is the set of documents containing $t$, then the score of $t$ with respect to $q$ is

$$s(t, q) = \frac{|\{D|D \in D_q^m \text{ and } t \in D\}| \cdot |D|}{|D_t|} \times \frac{\hat{P}(q \in D \text{ and } t \in D)}{P(t \in D)P(q \in D)}$$

since $|D|$ and $|D_q|$ are constants for a given $q$. Empirical evaluation however shows that the conventional interpretation of idf, i.e.,

$$\text{idf}(t) = \log \left( \frac{|D|}{|D_t|} \right),$$

is better. This is because interpreting idf as $\frac{|D|}{|D_t|}$ gives too much importance to very rare tokens; and this is further amplified by the fact that we are scanning only $m$ documents.

The proposed technique requires a single scan over $m$ documents among the search results for $q$. BlogScope uses $m = 30$, thus, considering just 30 text articles to find correlated terms for a query. Assuming that we have assessed that keywords $q, t$ above are correlated in a temporal window, repeating this process, using $q$ and $t$ as a query (expanding the query set) would yield keywords correlated with $q$ and $t$ (thus obtain a larger set of correlated keywords).

### 2.3.7 Authoritative Ranking

BlogScope computes the keyword synopsis set employing a greedy expansion technique using the original query keyword(s) as a seed set. We enumerate keywords correlated to the searched query $q$ and identify bursty intervals
in time using the popularity curve of the correlated keyword in combination with \( q \). We select the pair with maximum burstiness and iteratively repeat the same process until additional increase in burstiness is insignificant. For example, given the seed query “cars” the burst on 9th June 2006 (release date of the movie Cars) will be searched in conjunction with all its correlations “mercedes”, “truck” and “pixar”. Since “cars, pixar” gives a burst of higher intensity than both “cars, mercedes” and “cars, truck”, pixar will be selected to expand the set to \{cars, pixar\}. In the second iteration, we consider queries of the form “cars, pixar, disney”, “cars, pixar, nemo” (disney and nemo are both correlated to “cars, pixar”) etc., of which we will select “disney” (it contributes maximum to the burst) to expand our set to \{cars, pixar, disney\}. We continue with these iterations until the intensity of burst stops increasing.

To find authoritative bursts, we search for blogs containing all words in the synopsis keyword set and select those at the beginning of the bursts (earliest in time) having the highest number of incoming links.

2.3.8 Hot Keywords

Interestingness is naturally a subjective measure, as what is interesting varies according to the group of individuals it is intended for. Given the difficulty and the subjective nature of the task, BlogScope adopts a statistical approach to the identification of hot keywords. We employ a mix of scoring functions to identify top keywords for a day. In order to produce a final list we aggregate (using weighted summation) scores from all different scoring functions to find a ranked list of hot keywords.

Let \( x^t \) denote the popularity of some token \( t \) today, and \( x^t_1, x^t_2, \ldots, x^t_w \) be the popularity of the token in the last \( w \) days (except today). Let \( \mu^t \) and \( \sigma^t \) be the mean and standard deviation respectively of these \( w \) numbers. We employ the following two scoring functions:

- **Burstiness** measures the deviation of popularity from the mean value and is defined as \( \frac{x^t - \mu^t}{\sigma^t} \) for a token \( t \). A large deviation (burstiness) of a token implies that its current popularity is much larger than normal. BlogScope, in its current implementation, uses a value \( w = 90 \) in this case. This value is set after conducting several experiments with BlogScope.

- **Surprise** measures the deviation of popularity from the expected value using a regression model. We conduct a regression of popularities for a keyword over the last \( w \) days to compute the expected popularity for today. Let \( r(x^t) \) be this value. Then surprise is computed as \( \frac{|r(x^t) - x^t|}{\mu^t} \). This measure gives preference to tokens demonstrating unanticipated burst, ranking anticipated bursts low. Our implementation uses a value of \( w \) as 15 for this case. The choice of \( w \) in this case is set after experimentation with BlogScope.

Using the burstiness and surprise measures we compute a aggregate ranked list of interesting keywords for each day. To compute the aggregate list we add scores from different scoring functions, but as an alternative, use of ranked list merging techniques as described in the next chapter is also possible [32]. This way, BlogScope materializes a list of hot keywords for each day. BlogScope allows users to query such lists using temporal conditions. For example, one may wish to identify hot keywords in the Blogosphere for a specific week. BlogScope employs algorithms to support such queries; they are detailed later in this thesis.

Next chapter formalizes notion of keywords clusters and studies identification of such clusters. Chapter 4 describes algorithms for aggregation when such queries are accompanied with a hierarchy to elevate the terms to higher level concepts.
Chapter 3

Persistent Chatter Discovery by Identifying Stable Clusters

As topics evolve in the blogosphere, keywords align together and form the heart of various stories. Intuitively we expect that in certain contexts, when there is a lot of discussion on a specific topic or event, a set of keywords will be correlated: the keywords in the set will frequently appear together (pair-wise or in conjunction) forming a cluster. In other words, keywords are correlated if a large number of bloggers use them together in their respective blog posts. Note that such keyword clusters are temporal (associated with specific time periods) and transient. As topics recede, associated keyword clusters dissolve, because their keywords do not appear frequently together anymore. For example, we would expect that the keywords ‘saddam’, ‘hussein’, ‘trial’ formed a cluster when the trial of the former Iraqi president took place (on November 5 2006) as many people blogged about the trial of Saddam Hussein. However the keywords ‘saddam’, ‘hussein’ and ‘dead’ would form a cluster after his execution on December 30 2006. For more examples, consider Figures 3.1 and 3.2. Identifying such clusters for specific time intervals is a challenging problem. The associations between keywords reveals chatter in the blogosphere that may be of significant actionable value for many domains (e.g., marketing, law enforcement). Moreover it can be of value for improving and refining the quality of search results for specific keywords. If a search query for a specific interval falls in a cluster, the rest of the keywords in that cluster are good candidates for query refinement.

In this chapter we formalize and provide solutions for problems related to the temporal association of sets of keywords in the blogosphere. Although we focus on the domain of blogs (since we have a large collection of data via BlogScope), our discussion and techniques are generic enough to apply to any other temporally ordered text source. In particular, we make the following contributions in this chapter:

- We present fast algorithms to identify sets of correlated keywords (keyword clusters) in the blogosphere at any specified temporal interval. BlogScope currently contains more than 13M keywords in its index. Any algorithm aiming to identify keyword associations at this scale needs to be efficient.

- We formalize and present algorithms for the notion of stable keyword clusters. Since associations of sets of keywords is dynamic, stable clusters aim to identify sets of keywords that exhibit associations over several temporal intervals. Such keyword sets would probably point to events of interest, as it is evident that there is significant use of the keywords in the set, in conjunction, for extended periods of time.

- Since temporal information sources evolve continuously we present streaming (online) versions of our algorithms. This enables us to update the result set efficiently as new information arrives without re-computing
Figure 3.1: An example cluster of keywords appearing in the blogosphere on January 8 2007 corresponding to the following event: On January 7 2007, scientists at Wake Forest University led by Dr. Anthony Atala report discovery of a new type of stem cell in amniotic fluid. This may potentially provide an alternative to embryonic stem cells for use in research and therapy.

Figure 3.2: An example cluster of keywords appearing in the blogosphere on January 12 2007 corresponding to the following event: Soccer star David Beckham announces on Jan 11 he is to leave Real Madrid and join Major League Soccer (mls) team LA Galaxy at the end of the season.
everything. Such a requirement on algorithms is essential in order to cope with the temporal nature of our problem domain.

- We present an evaluation of our algorithms demonstrating their practical significance using real data sets and evaluate their scalability for very large data collections and problem settings.

Our core technology extends beyond blogs to social networking sites making heavy use of tagging, such as flickr.com and del.icio.us. Related processing to the one we conduct for keywords in blogs can be conducted on tags as well. This chapter is organized as follows: in Section 3.1 we briefly review related work. Section 3.2 presents our methodology for cluster generation. In Section 3.3 we formally define stable clusters and present our algorithms for identifying them. Section 3.4 presents the results of a quantitative comparison of our algorithms for various parameters of interest. Qualitative results for clusters discovered from real data are also presented in the same section.

### 3.1 Related Work

Graph partitioning has been a topic of active research (see [60] and references therein). A $k$-way graph partitioning is defined as a partitioning of a graph $G$ into $k$ mutually exclusive subsets of vertices of approximately the same size such that the number of edges of $G$ that belong to different subsets is minimized. The problem is hard, and several heuristic approaches have been proposed. In particular, multilevel graph bisection [60] has attracted research attention. Although such heuristic techniques have been tested on fairly large graph sizes (on the order of half a million vertices and few million edges) [60], they have the constraint that the number of partitions has to be specified in advance (as is common with clustering algorithms).

Correlation clustering [3] drops this constraint, and it produces graph cuts by specifying global constraints for the clusters to be produced. More specifically given a graph in which each edge is marked with a ‘+’ or a ‘−’, correlation clustering produces a partitioning of the graph such that the number of ‘+’ edges within each cluster and the number of ‘−’ edges across clusters is maximized. Although approximation algorithms are provided for this problem, the algorithms presented in [3] (as well in subsequent work [44] for a more restricted version of the problem) are very interesting theoretically, but far from practical. Moreover the existing algorithms require the edges to have binary labels, which is not the case in the applications we have in mind.

Flake et al. [38] present an alternative formulation of graph clustering in which they solve the problem using network flows. The drawback of this approach is that it requires the specification of a sensitivity parameter $\alpha$ before executing the algorithm, and the choice of $\alpha$ affects the solutions produced significantly. Moreover the running time of such an algorithm is prohibitively large for the graphs we have in mind, as they require solutions of multiple max-flow problems. Even the fastest algorithms known for max-flow are $O(VE)$, for $V$ vertices and $E$ edges, both of which are in the order of millions in our problem. (In our implementation, the algorithm of Flake et al. required six hours to conduct a graph cut on a graph with a few thousand edges and vertices.) Moreover it is not clear how to set parameters of this algorithm, and no guidelines are proposed in [38].

Various measures have been utilized in the past to assess associations between keywords in a corpus [72]. We employ some of these techniques to infer the strength of association between keywords during our cluster generation process.
Table 3.1: Sizes of resulting keyword graphs (each for a single day) for January 6 and 7 2007 after stemming and removal of stop words.

<table>
<thead>
<tr>
<th>Date</th>
<th>File Size</th>
<th># keywords</th>
<th># edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan 6</td>
<td>3027MB</td>
<td>2889449</td>
<td>138340942</td>
</tr>
<tr>
<td>Jan 7</td>
<td>2968MB</td>
<td>2872363</td>
<td>135869146</td>
</tr>
</tbody>
</table>

3.2 Cluster Generation

Let $D$ denote the set of text documents for the temporal interval of interest. Let $D \in D$ be a document, represented as a bag of words, in this document collection. For each pair of keywords $u, v$, $A_D(u, v)$ is assigned one if both $u$ and $v$ are present in $D$ and zero otherwise. Addition of $A_D(u, v)$ over all documents, $A(u, v) = \sum_{D \in D} A_D(u, v)$, represents the count of documents in $D$ that contain both $u$ and $v$. This way, triplets of the form $(u, v, A(u, v))$ can be computed. Let $V$ be the union of all keywords in these triplets. Each triplet represents an edge $E$ with weight $A(u, v)$ in graph $G$ over vertices $V$. Further, let $A(u)$ denote the number of documents in $D$ containing the keyword $u$. This additional information is required for computing $A(u, v)$, which represents the number of documents containing $u$ but not $v$.

For our specific case, the BlogScope crawler fetches all newly created blog posts at regular time intervals. The document collection $D$ in this case is the set of all blog posts created in a temporal interval (say every hour or every day). The number $A(u, v)$ represents the number of blog posts created in the selected temporal interval containing both $u$ and $v$. The computation of the triplets $(u, v, A(u, v))$ therefore needs to be done efficiently even with large number of posts. We used the following methodology: A single pass is performed over all documents in $D$. For each document $D$, output all pairs of keywords that appear in $D$ after stemming and removal of stop words. Since $A(u)$ also needs to be computed, for each keyword $u \in D$, $(u, u)$ is also included as a keyword pair appearing in $D$. At the end of the pass over $D$ a file with all keyword pairs is generated. The number of times a keyword pair $(u, v)$ appears in this file is exactly the same as $A(u, v)$. This file is sorted lexicography (using external memory merge sort) such that all identical keyword pairs appear together in the output. All the triplets are generated by performing a single pass over the output sorted file. Table 3.1 presents sizes of two of the keyword graphs (each for a single day) after stemming all keywords and removing stop words.

Given graph $G$ we first infer statistically significant associations between pairs of keywords in this graph. Intuitively if one keyword appears in a fraction $n_1$ of the posts and another keyword in a fraction $n_2$ we would expect them both to occur together in a fraction $n_1 n_2$ of posts. If the actual co-occurrence percent deviates significantly from this expected value, the assumption that the two keywords are independent is questionable. This effect can be easily captured by the $\chi^2$ test:

$$
\chi^2 = \left( \frac{(E(uv) - A(uv))^2}{E(uv)} \right) + \left( \frac{(E(\bar{u}v) - A(\bar{u}v))^2}{E(\bar{u}v)} \right) + \left( \frac{(E(\bar{u}\bar{v}) - A(\bar{u}\bar{v}))^2}{E(\bar{u}\bar{v})} \right)
$$

(3.1)

In this formula, $A(uv)$ is the number of times keywords $u, v$ appear in the same post (document). $E(uv)$ is the expected number of posts in which $u$ and $v$ co-occur under the independence assumption. Thus, $E(uv) = \frac{A(u) A(v)}{n}$ where $A(u)$ ($A(v)$) is the total number of times keyword $u$ appears in posts and $n$ is the total number of posts. Similarly, $A(\bar{u})$ is the number of posts not containing keyword $u$. The value $\chi^2$ has a chi-squared distribution. From standard tables, we identify that only 5% of the time does $\chi^2$ exceed 3.84 if the variables are
independent. Therefore, when $\chi^2 > 3.84$ we say that $u$ and $v$ are correlated at the 95% confidence level. This test can act as a filter omitting edges from $G$ not correlated according to the test at the desired level of significance. Note that this test can be computed with a single pass of the edges of $G$.

While this test is sufficient to detect the presence of a correlation, it cannot judge its strength. For example, when $u$ and $v$ are indeed correlated their $\chi^2$ values will increase as the number of data points (number of posts in our case, $n = |D|$) grows. The correlation coefficient $\rho$, is a measure of the strength of correlation. It is defined as follows:

$$
\rho(u, v) = \frac{\sum_i (A_i - \mu_u)(B_i - \mu_v)}{n\sqrt{\sigma_u^2\sigma_v^2}}
$$

(3.2)

where $\mu_u$ is the mean of the number of times keyword $u$ appears in the document collection ($n$ documents in total), that is $\frac{\sum_i A_i}{n}$, $\sigma_u^2$ is the variance of the appearance of $u$ in the posts and $A_i$ is 1 if and only if post $i$ contains $u$. It is evident that $\rho$ is between -1 and 1, and it is zero if $u$ and $v$ are independent. The correlation coefficient is important because it is often the case that we have enough data to find weak but significant correlations. For example once an hour posts might contain two terms together. With enough data over a day, the $\chi^2$ test will (correctly) assess non-independence. The correlation coefficient however will report a weak correlation. For all edges that survive the $\chi^2$ test, we compute the correlation coefficient between the incident vertices. This computation can again be conducted efficiently by re-writing Formula 3.2 as

$$
\rho(u, v) = \frac{nA(u, v) - A(u)A(v)}{\sqrt{(n - A(u))A(u)(n - A(v))A(v)}}
$$

(3.3)

using the fact that $\sum A_i^2 = \sum A_i$.

Given graph $G$ (excluding the edges eliminated by the $\chi^2$ test), assume we have annotated every remaining edge with the value of $\rho$ indicating the strength of the correlation. This graph can be further reduced by eliminating all edges with values of $\rho$ less than a specific threshold. Since our problem is binary (a keyword either appears in the post or not) focusing on edges with $\rho > 0.2$ will further eliminate any non truly correlated vertex pair, making the probability of a false (non correlated pair) being included very small [24]. These correlations are important since the strong ones offer good indicators for query refinement (e.g., for a query keyword we may suggest the strongest correlation as a refinement) and also track the nature of ‘chatter’ around specific keywords.

Let $G'$ be the graph induced by $G$ after pruning edges based on $\chi^2$ and $\rho$. Observe that graph $G'$ contains only edges connecting strongly correlated keyword pairs. We aim to extract keyword clusters of $G'$. Although we can formally cast our problem as an optimization problem for graph clustering [3, 38], adopting any of the known approximation algorithms is impossible as such algorithms are of high polynomial complexity. Running any such algorithm on the problems of interest in this study is prohibitive. Moreover, the access patterns of such approximation algorithms require the entire graphs to be in memory and do not have efficient secondary storage realizations. For this reason, we propose a simple and (as we will demonstrate) effective heuristic algorithm to identify such clusters. Our algorithm is fast, suitable for graphs of the scale encountered in our setting and efficient for graphs that do not fit in memory. We empirically evaluate the quality of the clusters we identify in Section 3.4.

Our algorithm identifies all articulation points in $G'$ and reports all vertices (with their associated edges) in each biconnected component as a cluster. An articulation point in a graph is a vertex such that its removal makes the graph disconnected. A graph with at least two edges is biconnected if it contains no articulation points. A biconnected component of a graph is a maximal biconnected graph. Thus, the set of clusters we report for $G'$ is the set of all biconnected components of $G'$ plus all trees connecting those components. The underlying intuition is
that nodes in a biconnected component survived pruning, due to very strong pair-wise correlations. This problem is a well studied one [25]. We adopt algorithms for its solution and demonstrate via experiments that they are memory efficient. Let \( G_\pi \) be a depth first tree of \( G' \). An edge in \( G' \) is a back edge iff it is not in \( G_\pi \). The root of \( G_\pi \) (the vertex from which we initiated the depth first traversal) is an articulation point of \( G' \) if it has at least two children. A non-root vertex \( u \in G_\pi \) is an articulation point of \( G' \) if and only if \( u \) has a child \( w \) in \( G_\pi \) such that no vertex in the subtree rooted at \( w \) (in \( G_\pi \), denoted subtree\( (w) \)), is connected to a proper ancestor of \( u \) by a back edge. Let \( un[w], w \in G_\pi \) be the order in which \( w \) is visited in the dfs of \( G' \). Such a dfs traversal can be performed efficiently, even if the graph is too large to fit in memory [21, 17]. We define:

\[
\text{low}[w] = \min \begin{cases} 
\text{un}[w] & \text{if } x \text{ is joined to } \text{subtree}(w) \text{ via a back edge where } x \text{ is a proper ancestor of } w \\
\text{un}[x] & \text{for all } x \end{cases}
\]

A non root vertex \( u \in G_\pi \) is an articulation point of \( G' \) if and only if \( u \) has a child \( w \) such that \( \text{low}[w] \geq \text{un}[u] \). Algorithm 1 presents pseudocode for the algorithm to identify all biconnected components of \( G' \). A similar technique can be used to report all articulation points of \( G' \). The algorithm as presented in the pseudo code requires as many accesses to disk as the number of edges in \( G' \). Since in our graphs we expect \(|E| >> |V|\), using the techniques of [21] we can run it in \( O((1 + |V|/M) \text{scan}(E) + |V|) \) I/Os, where \( M \) is the size of available memory. Since the data structure in memory is a stack with well defined access patterns, it can be efficiently paged to secondary storage if its size exceeds available resources. In our experiments, presented in Section 3.4, this was never the case.

**Algorithm 1** Algorithm to Identify Biconnected Components

1. Initialize \( \text{time} = 0 \) and \( \text{un}[u] = 0 \) for all \( u \)
2. \( \text{time} \leftarrow \text{time} + 1 \)
3. \( \text{un}[u] \leftarrow \text{time} \)
4. \( \text{low}[u] \leftarrow \text{time} \)
5. for each vertex \( w \neq u \) such that \((u, w) \in E\) do
6. if \( \text{un}[w] < \text{un}[u] \) then
7. add \((u, w)\) to Stack
8. end if
9. if \( \text{un}[w] = 0 \) then
10. call Art\((w)\)
11. \( \text{low}[u] \leftarrow \min\{\text{low}[u], \text{low}[w]\} \)
12. end if
13. if \( \text{low}[w] \geq \text{un}[u] \) then
14. Pop all edges on top of Stack until (inclusively) edge \((u, w)\), and report as a biconnected component
15. else
16. \( \text{low}[u] \leftarrow \min\{\text{low}[u], \text{un}[w]\} \)
17. end if
18. end for

**Example 3.1** Figure 3.3 shows an example of applying the Algorithm 1 to \( G' \) in (a). The DFS tree, \( G_\pi \) is shown in (b) with the final \( \text{un}(u) \) and \( \text{low}(u) \) values. Back edges \((c, a)\) and \((f, d)\) (shown as dashed edges in \( G_\pi \)) lead to \( \text{low}(u) \) being updated during the backtracking for all parent nodes. Internal nodes \( b \) and \( d \) are articulation points. The biconnected components of \( G' \) are shown in (c).
3.3 Stable Clusters

Let $t_1, \ldots, t_m$ be (without loss of generality) $m$ successive temporal intervals. Let $T_1 \ldots T_m$ be the number of clusters identified for each of the intervals $t_1 \ldots t_m$ using the algorithm in Section 3.2. Let $c_{ij}$ be the clusters identified $1 \leq i \leq m, 1 \leq j \leq T_i$. Analysis of the affinity (e.g., overlap) of the keywords in these clusters across the temporal intervals can provide very valuable information. For example, a cluster of keywords that always appear together across the $m$ temporal intervals probably points to an event that triggered increased use of the keywords in the consecutive temporal intervals by enough people (bloggers) to force a persistent (stable) cluster across the intervals. Similarly, clusters that appear in some of the temporal intervals, or clusters that appear for a few intervals then vanish and appear again, might also be of interest as they point to events that triggered increased use of the keywords for a few intervals.

Let $G_1, \ldots, G_m$ be the sets of clusters produced for each temporal interval $1 \leq i \leq m$. Given two clusters $c_{kj}, c_{k'j'}, k \neq k'$, we can quantify the affinity of the clusters by functions measuring their overlap. For example, $|c_{kj} \cap c_{k'j'}|$ or $\text{Jaccard}(c_{kj}, c_{k'j'})$ are candidate choices. Other choices are possible taking into account the strength of the correlation between the common pairs of keywords. Our framework can easily incorporate any of these choices for quantifying cluster affinity. We consider clusters with affinity values greater than a specific threshold $\theta$ ($\theta = 0.1$) to ensure a minimum level of keyword persistence. Given the clusters $G_i$, we form a graph $G$ by evaluating the affinity between select pairs of $G_i, G_j, i \neq j, i \leq j + g + 1, 1 \leq i, j \leq m$. The choice of pairs to compute, dictates the structure and connectivity of $G$. We refer to the value of $g$ as a gap for a specific construction of $G$. Gaps are useful to account for clusters (chatter) that are persistent for a few intervals, then vanish and appear again (see Figure 3.4 for example). $G$ is a weighted graph with edge weights equal to the affinity of the clusters incident to the edge. For any path in $G$ we define the weight of the path by aggregating the weights of the edges comprising the path. Notice that the types of paths existing in $G$ is a construction choice. Graph $G$ may range from an $m$-partite graph to a fully connected graph, depending on the choice of $g$. $G$ is an undirected graph.

**Problem 3.3.1 (kI-Stable Clusters)** Given a graph $G$ constructed using a specific affinity function, we define the problem of stable clusters as the problem of identifying the $k$ paths of length $l$ of highest weight.

A variant of the above problem is the following:

**Problem 3.3.2 (Normalized Stable Clusters)** Given a graph $G$ constructed using a specific affinity function we
define the problem of normalized stable clusters as the problem of identifying the $k$ paths of length at least $l_{\text{min}}$ of the highest weight normalized by their lengths.

In order to construct the graph $G$ for a set of clusters $G_1, \ldots, G_m$, each computed for an interval $t_1 \ldots t_m$ fixing a gap value $g \geq 0$, we have to compute the affinity between clusters in $G_i, G_j, i \leq j + g + 1, 1 \leq i, j \leq m, 0 \leq g \leq m - 1$. Assuming $T_i$ clusters for each interval $t_i$, the suitable affinity predicate (e.g., intersection size) can be computed between each pair of clusters of the corresponding intervals, assuming the clusters for the pair of intervals fit in memory. If $T_i$ (and the associated cluster descriptions in terms of their keywords) is too large to fit in memory, we can easily adapt technology to quickly compute all pairs of clusters for which the affinity predicate is above some threshold. Notice that each cluster description is a set of keywords. Thus, the problem is easily reduced to that of computing similarity (affinity) between all pairs of strings (clusters) for which the similarity (affinity) is above a threshold. Efficient solutions for conducting such computations for very large data sets are available and can easily be adapted [64].

Given graph $G$, we now present our solutions to the $k l$ stable clusters problem. Note that the top-$k$ paths produced may share common subpaths which, depending on the context, may not be very informative from an information discovery perspective. Variants of the $k l$-stable cluster problem with additional constraints are possible to discard paths with the same prefix or suffix. For simplicity, we focus on the original problem and present three solutions that can later be adapted for more refined variants of the problem. The three associated algorithms are: (a) an algorithm based on breadth first search on $G$, (b) an algorithm based on depth first search, and (c) an adaptation of the well known threshold algorithm [89]. Our focus is on cases for which the number of clusters and their associated descriptions for all temporal intervals are too large to fit in memory and we propose efficient solutions for secondary storage.
3.3.1 The Cluster Graph

Let $G$ denote the cluster graph. Figure 3.5 shows an example cluster graph over 3 temporal intervals. Each interval has 3 nodes (keyword clusters). Edges between two nodes indicate that they have a non-zero affinity. While conceptually the model has undirected edges, we add a source node at the beginning and a sink at the end, and make edges directed. Each edge has a weight in the range $(0, 1]$. Thus the length of an edge over a single gap of length $g$ is considered to be $g + 1$. Edge length is defined as the length of the temporal interval between two participating nodes. For example, in Figure 3.5, the length of edge $c_{11}c_{21}$ and $c_{13}c_{22}$ is one, while that of $c_{11}c_{32}$ is two. The gap size is selected as $g = 1$ in this example, and therefore all edges have length less than or equal to $g + 1 = 2$. The length and weight of edges connecting source or sink with other nodes is assumed to be zero.

Let $G_i = \{c_{i1} \ldots c_{i|T_i|}\}$ be the clusters at interval $t_i$. We refer to a node $c_{ij}$ as a child of another node $c_{i'j'}$ if there is an edge between the two nodes and $i > i'$. In this case, $c_{i'j'}$ is a parent of $c_{ij}$. Let $\text{interval}(c)$ be the index of the temporal interval to the cluster to which $c$ belongs. For example if $c \in G_i$, $\text{interval}(c) = i$.

3.3.2 Breadth First Search

We first present a breadth first search based algorithm for detecting stable clusters. At the end of the algorithm we seek to find the top-$k$ paths with highest weights of length $l$. As the algorithm progresses, we annotate each node in the graph with up to $l$ heaps, each of size less than or equal to $k$. For a node $c_{ij}$, we denote this data structure as $h^x_{ij}$, for $1 \leq x \leq l$, each of which represents top-$k$ (or fewer) highest weighting subpaths of length $x$ ending at $c_{ij}$. Observe that annotating each node of an arbitrary graph with such information is a non-trivial task requiring many random disk I/Os. We take advantage of the special structure of the graph in our case, which is very similar to an $n$-partite graph (except for the gaps). Such graphs have a nice property that a node from $G_i$ cannot have a parent from a temporal interval before $i - g - 1$, where $g$ is the size of the maximum gap allowed. This means that if all nodes from temporal intervals $\{i - g - 1, \ldots, i - 1\}$ can be kept in memory, subpaths ending at all nodes from $G_i$ can be computed without performing any I/O.

For all the nodes belonging to $G_1$, all the associated heaps are initialized to be empty. To compute heaps for a node $c_{ij} \in G_i$, all nodes from the previous $g + 1$ intervals are read in memory along with their $l$ heaps. After

\footnote{Some affinity functions such as intersection do not guarantee weights to be in the range $(0, 1]$. In such cases, the maximum score seen so far can be maintained to normalize all weights to the range $(0, 1]$.}
reading all the nodes from the previous \( g + 1 \) intervals, nodes from \( G_i \) are read one after the other. For each node \( c_{ij} \in G_i \), all its parents are probed (which are already in memory) to update its associated heaps \( h^x_{ij} \). Consider the example cluster graph presented in Figure 3.5 with \( l = 2 \) and \( k = 2 \). Computing heaps for nodes from the second temporal interval, all nodes in \( G_1 \) are read in memory. Each node from the second interval will have only a single heap associated with it, since there are no paths of length two ending there. The heaps for nodes in \( G_2 \) are:

\[
\begin{align*}
  h^1_{21} &= \{c_{11}c_{21}\}; \quad h^1_{22} = \{c_{12}c_{22}, c_{13}c_{22}\}; \quad h^1_{23} = \{c_{12}c_{23}\}
\end{align*}
\]

Computing heaps for nodes from \( G_3 \), all nodes from \( G_1 \) and \( G_2 \) are kept in memory. Since there are three paths of length 2 reaching \( c_{31} \), only the best two are retained. Since the weight of \( c_{12}c_{22}c_{31} \) (which is 0.8) is less than that of \( c_{13}c_{22}c_{31} \) (1.5) and of \( c_{11}c_{21}c_{31} \) (1.2), it is discarded. Although \( c_{11} \) is a parent of \( c_{32} \) (with direct edge between the two), due to the gap, \( c_{11}c_{32} \) is an edge of length two. Thus,

\[
\begin{align*}
  h^1_{31} &= \{c_{21}c_{31}, c_{22}c_{31}\}; \quad h^1_{32} = \{c_{21}c_{32}\}; \\
  h^1_{33} &= \{c_{22}c_{33}, c_{23}c_{33}\}; \quad h^2_{31} = \{c_{11}c_{21}c_{31}, c_{13}c_{22}c_{31}\}; \\
  h^2_{32} &= \{c_{11}c_{21}c_{32}, c_{11}c_{32}\}; \quad h^2_{33} = \{c_{13}c_{22}c_{33}, c_{12}c_{22}c_{33}\}
\end{align*}
\]

Algorithm 2 BFS based algorithm for \( kl \)-clusters

INPUT \( G = \{G_1, \ldots, G_m\}, l, k, g \)

1. Initialize \( H = \emptyset \), heap of size \( k \)
2. for \( i = 2 \) to \( m \) do
3.   Read \( G_i \) in memory, \( i - g - 1 \leq i' \leq i - 1 \)
4.   for \( c_{ij} \in G_i \) do
5.     Initialize \( h^x_{ij} = \emptyset \), heap of size \( k \), \( 1 \leq x \leq l \).
6.     for \( c_{ij} \in \text{parents}(c_{ij}) \) do
7.       \( \text{len} = i - i' \) \{comment: since \( c_{ij}' \in G_{i'} \), this is the length of the edge \( c_{ij}'c_{ij} \)\}
8.       for \( x = 1 \) to \( l - \text{len} \) do
9.         for \( \pi \in h^x_{ij} \) do
10.        \( \pi' = \text{append}(\pi, c_{ij}'c_{ij}) \)
11.       check \( \pi' \) against \( h_{ij}^{x'+\text{len}} \)
12.      check \( \pi' \) against \( H \) \{“check” operation on \( \pi' \) against a fix-sized heap checks for the inclusion of \( \pi' \) in the heap\}
13.     end for
14.   end for
15. end for
16.   save \( c_{ij} \) along with \( h^x_{ij} \) to disk
17. end for
18. return \( H \)

\( G \) is stored in the form of an adjacency matrix so that for any cluster \( c_{ij} \) we can easily retrieve \( \text{parents}(c_{ij}) \) the set of all clusters at intervals \( t_{ij} < t_{i'j'} \in [i - g - 1, i - 1] \) with edges incident to \( c_{ij} \). As an invariant assume that \( h^x_{ij}, i' \in [i - g - 1, i - 1] \) have been computed while building heaps for nodes from \( G_i \). We compute \( \text{allpaths}(c_{ij}, x) \) as the set of all paths of length \( x \) ending at \( c_{ij} \) that can be derived from information maintained in \( \text{parents}(c_{ij}) \). More formally:

\[
\text{allpaths}(c_{ij}, x) = \{ \text{append}(\pi, c_{ij}'c_{ij}) \mid c_{ij}' \in \text{parents}(c_{ij}) \} \quad \text{and} \quad \pi \in h^x_{ij} \]
where \( \text{append}(\pi, c_{i'j}, c_{ij}) \) represents the path obtained by appending the edge \( c_{i'j}c_{ij} \) at the end of subpath \( \pi \). Thus,

\[
h^*_i = \text{top-}k \text{ paths among allpaths}(c_{ij}, x)
\]

Computing \( h^*_i \) using \( \text{parents}(c_{ij}) \) can be conducted in a straightforward way by considering each possible element of \( \text{allpaths}(c_{ij}, x) \). In practice, \( h^*_i \) can be computed directly using a heap without maintaining the intermediate result \( \text{allpaths}(c_{ij}, x) \). For each \( c_{ij} \) in memory we need to maintain at most \( kl \) subpaths of highest weight ending at \( c_{ij} \). These are the best (highest weight) paths of lengths 1, 2, \ldots, \( l \), maintained using heaps, one for each length. This means that \( \text{allpaths}(c_{ij}) \) is updated for each value of \( x \in [1, l] \). Algorithm 2 presents pseudocode.

Maintaining the solution to the \( kl \)-stable clusters problem is conducted by maintaining a heap \( H \) during the execution of the algorithm in which the \( k \) highest weight paths of length exactly \( l \) are identified. When a new interval \( t_{i+1} \) is encountered, \( G_{i+1} \) is brought into memory and \( G_{i-g-1} \) is discarded. This computation is performed for \( 1 \leq i \leq m \). After \( G_m \) is encountered, the solution to the \( kl \)-stable clusters problems is located in \( H \). In the running example of Figure 3.5, whenever a new path of length 2 is discovered, it is checked against the global heap \( H \). Six paths, from \( h^*_31 \), \( h^*_23 \), and \( h^*_33 \), are checked for candidacy in \( H \) in this example. The size of \( H \) is bounded by \( k = 2 \). In the end, the best two paths are identified as \( c_{13}c_{22}c_{31} \) and \( c_{13}c_{22}c_{33} \).

If \( l = m - 1 \), i.e., when finding full paths from \( t_1 \) to \( t_m \), we need not maintain heaps \( h^*_ij \) for each \( 1 \leq x \leq l \). Instead, maintaining one heap per node suffices. For a node \( c_{ij} \in G_i \), only \( h^*_ij \) needs to be computed. This reduces the computation by a factor of \( l \) for this special case.

The algorithm as described requires enough memory to maintain all clusters for \( g + 1 \) intervals in memory. Under this assumption the algorithm requires a single pass over all \( G_i, 1 \leq i \leq m \). The total number of I/Os required is linear in the number of intervals considered and linear in the total number of clusters. Assume that accommodating clusters for \( g + 1 \) intervals in memory requires an amount equal to \( M_{\text{req}} \) but only \( M \) memory units are available (\( M < M_{\text{req}} \)). In order to compute heaps for all clusters in the \( i \)-th interval \( M_{\text{req}}/M \) passes will be required. This situation is very similar to block-nested loops.

**Claim 3.3.1** The BFS based algorithm described outputs the correct set of highest weighting top-\( k \) paths in the cluster graph \( G \).

### 3.3.3 Depth First Search

We present a solution to the \( kl \)-stable clusters problem based on a depth first search (DFS) traversal of the cluster graph \( G \) suitably adapted for secondary storage. DFS can be performed using a stack. We show that the size of the stack will be bounded by \( m \), the number of temporal intervals. The complexity of this algorithm in the worst case is linear to the number of edges in the graph, but practically can be much less due to pruning. Unlike the BFS algorithm presented in the previous subsection, this algorithm requires significantly less memory to operate, but performs much more I/O.

For each node (cluster) \( c_{ij} \), we maintain a list of its children (nodes in \( G_{i'}, i' \in [i+1, i+g+1] \) incident to \( c_{ij} \) as \( \text{children}(c_{ij}) \) which is precomputed during the generation of \( G \). Also we maintain a global top-\( k \) list as a heap (containing the current \( k \) paths of length \( l \) of highest weight) and a stack, both initialized to be empty in the beginning. As the algorithm progresses, we will maintain the following information with each node \( c_{ij} \) (on disk):

- One flag denoting whether the node has already been visited. If the flag is set, we are confident that all descendants of the node have been considered. If not, its descendants may or may not have been traversed.
• If the objective is to find full paths (of length \( m - 1 \)), one number denoting the aggregate weight of the highest weight path from the source to that node. If the objective is to find subpaths of length \( l \), one number for each \( x \), \( \max(1, l + i - m) \leq x \leq \min(l, i - 1) \), denoting the aggregate weight of the highest weighting path of length \( x \) ending at that node. We represent this data structure by \( \maxweight(c_{ij}, x) \) for paths of length \( x \), and use this data structure for pruning.

• If the objective is to find full paths of length \( m - 1 \), a single heap of top-\( k \) best (highest weighting) paths starting at that node is maintained. If the objective is to find subpaths of length \( l \), a heap for each \( \max(1, l + i - m) \leq x \leq \min(l, i - 1) \), containing top-\( k \) best paths of length \( x \) starting at that node is maintained. We denote this data structure by \( \text{bestpaths}(c_{ij}, x) \) for paths of length \( x \). Contrasting this case with the case of the same data structure in the BFS algorithm, we note that paths contained in this case start at \( c_{ij} \) (instead of ending at \( c_{ij} \)). The size of \( \text{bestpaths} \) for any node is bounded by \( k \) when seeking full paths of length \( m - 1 \), and \( k \cdot l \) in the case of subpaths of length \( l \).

The algorithm performs a depth first search on the input cluster graph. Pseudocode for this algorithm is presented in Algorithm 3.

We provide an operational description of the algorithm. Start by pushing the source node along with \( \text{children(source)} \) onto the stack. Now iteratively do the following: Take the top element \( c \) from the stack, remove an element \( c' \) from the list \( \text{children}(c) \). Check if \( c' \) is already visited. If yes, update \( \text{bestpaths}(c) \) using \( \text{bestpaths}(c') \) as described later, and discard \( c' \). If not, mark \( c' \) as visited and push it on the stack. Update \( \maxweight(c', x) \) using \( \maxweight(c, x) \) for each \( x \).

\[
\begin{align*}
\maxweight(c', x) &= \max(\maxweight(c', x), 0), \\
\maxweight(c, x - \text{length}(cc')) + \text{weight}(cc'))
\end{align*}
\]

where \( \text{length}(cc') \) is the length of the edge between \( c \) and \( c' \). The following pruning operation can be conducted (when searching for subpaths of length \( l \)): If for all \( \max(1, l + \text{interval}(c') - m) \leq x \leq \min(l - 1, \text{interval}(c') - 1) \),

\[
\maxweight(c', x) + l - x < \min-k,
\]

where \( \min-k \) is the minimum weight among all paths in \( H \) (the current top-\( k \) ), remove \( c' \) from the stack. Also unmark the visited flag for all the nodes in the stack (including \( c' \)). This is based on the observation that, given the current information about the weight of the path from the source to \( c' \), it is unlikely that any of the paths containing \( cc' \) can be in the top-\( k \). Therefore we postpone considering descendants of \( c' \) until we find (if it exists) a higher weighting path from the source to \( c' \). We unmark the visited flag of all nodes on the stack since the guarantee that all descendants have been traversed no longer holds true for them. Therefore, pruning assumes that all edge weights are between \( (0, 1] \) (which is true for some affinity measures like Jaccard; normalization is required for others e.g., intersect).

If \( c \) is at the top of the stack such that \( \text{children}(c) = \phi \), i.e., all children of \( c \) have been considered (either traversed or discarded by pruning), remove \( c \) from the stack. Let \( c' \) be the next element on the stack. Update \( \text{bestpaths}(c') \) using \( \text{bestpaths}(c) \) (this is actually back tracking an edge in DFS).

To update \( \text{bestpaths}(c) \) using information about one of its children \( c' \); first find all possible paths starting at \( c \) by augmenting the edge \( cc' \) with all paths in \( \text{bestpaths}(c', x) \), and add them to \( \text{bestpaths}(c, x + \text{length}(cc')) \), for all \( x + \text{length}(cc') \leq l \). Now prune \( \text{bestpaths}(c, x + \text{length}(cc')) \) so that it does not contain more than \( k \) paths. When a new path \( \pi \) of length \( l \) is added to \( \text{bestpaths}(c, l) \) for some node \( c \), \( \pi \) is also checked against the
Algorithm 3 DFS based algorithm for $k\ell$-clusters

INPUT $G = \{G_1, \ldots, G_m\}, l, k, g$
1: initialize $H = \phi$, heap of size $k$
2: initialize stack = $\phi$
3: push $(source, children(source))$ to stack
4: while stack is not empty do
5:   $(c, children(c)) = \text{peek from stack}$ \{peek operation returns the top element from the stack without removing it\}
6:   if children($c$) is not empty then
7:     $c' = \text{remove top element from children}(c)$
8:     read from disk information associated with $c'$
9:     if $c'$ is visited then
10:        update $\text{bestpaths}(c, x)$ using info from $c', x \leq l$
11:        for each newly added path $\pi$ in $\text{bestpaths}(c, l)$ do
12:           check $\pi$ against $H$ \{“check” operation on $\pi$ against a fix-sized heap checks for the inclusion of $\pi$ in the heap\}
13:    end for
14:   else
15:      mark $c'$ visited, and push $(c', children(c'))$ on stack
16:      update $\text{maxweight}(c', x)$ using $\text{maxweight}(c, x)$
17:      if CanPrune($c'$) then
18:         unmark visited flag for all nodes in stack
19:         pop $c'$ from stack
20:         save $c'$ and associated information to disk
21:      end if
22:   end if
23: else
24:   pop $c$ from stack and save on disk
25:   $(c', children(c')) = \text{peek from stack}$
26:   update $\text{bestpaths}(c', x)$ using info from $c, x \leq l$
27:   for each newly added path $\pi$ in $\text{bestpaths}(c', l)$ do
28:      check $\pi$ against $H$
29:   end for
30: end if
31: end while
32: output $H$

DEFINE CanPrune($c'$)
1: min-$k = \text{minimum score in } H$
2: for $x = \max(1, l + \text{interval}(c') - m)$ to $\min(l - 1, \text{interval}(c') - 1)$ do
3:   if $\text{maxweights}(c', x) + l - x \geq \text{min}-k$ then
4:      return false
5:   end if
6: end for
7: return true

global top-$k$ heap for inclusion.

The size of the stack is at most $m$ entries during the execution of this algorithm. When the algorithm terminates, the global top-$k$ heap $H$ will contain the required result. Furthermore, each node will be annotated with a list of top-$k$ bestpaths starting at that node.

Addition of each node to the stack requires one random I/O to read the associated data structures from disk. Updating these data structures and marking/unmarking of nodes takes place in main memory. Removal of a node requires an additional random I/O for writing back associated data structures. In the absence of the pruning condition, the number of read operations is bounded by the number of edges, and the number of write operations is bounded by the number of nodes in the graph. With every pruning operation, in the worst case, both these numbers can increase by an amount equal to the size of the stack at that time. But pruning is also expected to discard many
Example 3.2 We show the execution of the algorithm over the cluster graph presented in Figure 3.5 for \(k = 1\) and \(l = 2\). In this example, since we are required to find full paths \((l = 2)\), only one heap and one maxweight structure is associated with each node. Table 3.2 shows the order in which the nodes are considered and actions taken at those steps. Observe that pruning takes place when \(c_{22}\) is first explored. However \(c_{22}\) is explored further when it is reached again via \(c_{13}\). The final result is printed as \(\{c_{13}c_{22}c_{33}\}\). Note that other execution orders are also possible, depending on how the children lists for each node are sorted.

Claim 3.3.2 The DFS based algorithm described outputs the correct set of highest weighting top- \(k\) paths in the cluster graph \(G\).

For effective pruning, it is important that paths of high weights are considered early. For this reason, as a heuristic, while precomputing the list of children for all nodes, we sort them in the descending order of edge weights. Formally if \(c_1, c_2 \in \text{children}(c)\) and \(\text{weight}(c_1, c) > \text{weight}(c_2, c)\), then \(c_1\) precedes \(c_2\) in the list \(\text{children}(c)\). This will ensure that the children connected with edges of high weight are considered first. It must be noted that this heuristic is for efficient execution, and correctness of the algorithm is unaffected by it.

### 3.3.4 Adapting the Threshold Algorithm

The Threshold Algorithm (TA) \([89]\) can also be adapted to find full paths of length \(m - 1\) in \(G\). For each pair of temporal intervals \(t_i\) and \(t_{i'}\), \(|i - i'| \leq g + 1\), one list of edges is maintained. These lists are sorted in descending order of edge weights.

We read edges from sorted lists in a round robin fashion and maintain a global heap \(H\) for intermediate top- \(k\) results. When an edge \(c_{ij}c_{i'j'}\) \((i < i')\) is encountered, we perform random seeks to lookup all paths containing the edge. Let \(d\) be the maximum out-degree in the graph \(G\). Unlike the vanilla-TA, where each attribute belongs to exactly one tuple, in this case there may be multiple paths that contain \(c_{ij}c_{i'j'}\). Perform random seeks in edge lists to find all the paths that start with \(c_{i'j'}\), and all the paths that end at \(c_{ij}\) to construct all paths containing \(c_{ij}c_{i'j'}\). Check all these paths for inclusion in \(H\), and discard them if they fail to qualify. Terminate when the score of the lowest scoring path in the buffer \(H\) falls below that of the virtual tuple. The virtual tuple is the imaginary path consisting of the highest scoring unseen edge from each list.

A path may be discovered more than once in the above algorithm. As an optimization to reduce I/O, two additional hash tables, \(\text{startwts}\) and \(\text{endwts}\), can be maintained. For a node \(c\), \(\text{startwts}(c_{ij})\) (\(\text{endwts}(c_{ij})\)) records the aggregate weight of the highest weighting path starting (ending) at \(c_{ij}\). These hash tables are initialized to be empty at start, and are updated as the algorithm progresses. When all paths starting (ending) at a node \(c_{ij}\) are computed by performing random probes, \(\text{startwts}(c_{ij})\) (\(\text{endwts}(c_{ij})\)) is updated. When an edge \(c_{ij}c_{i'j'}\) is read from the edge list, and if \(\text{startwts}(c_{i'j'})\) and \(\text{endwts}(c_{ij})\) are available in the hash tables, an upper bound on weight of all paths containing \(c_{ij}c_{i'j'}\) can be computed without any I/O. This upper bound can be compared with the score of lowest scoring path in \(H\), and \(c_{ij}c_{i'j'}\) can be discarded without performing any further computation if the former is smaller. This pruning can result in large savings in I/O.

If the maximum out-degree in the graph \(G\) is \(d\), this might lead to as many as \(m^{d-1}\) random seeks in the absence of gaps \((g = 0)\). In the presence of gaps this number can be much higher. Hence this algorithm is not suitable when either of \(m\) or \(d\) is high. We validate this observation in the experimental results section. Further, this algorithm is restricted to discovery of full paths only and thus requires \(l = m - 1\).
<table>
<thead>
<tr>
<th>Node Explored</th>
<th>Action taken and Updates to maxweights and bestpaths</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_{11}$</td>
<td>none</td>
</tr>
<tr>
<td>$c_{21}$</td>
<td>$maxweight(c_{21}, 1) = 0.5$</td>
</tr>
<tr>
<td>$c_{31}$</td>
<td>$maxweight(c_{31}, 2) = 1.2$</td>
</tr>
<tr>
<td>$c_{21}$</td>
<td>$bestpaths(c_{21}, 1) = {c_{21}c_{31}}$</td>
</tr>
<tr>
<td>$c_{32}$</td>
<td>$maxweight(c_{32}, 2) = 0.9$</td>
</tr>
<tr>
<td>$c_{21}$</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>${c_{32}c_{21}}$ failed to qualify for $bestpaths(c_{21}, 1)$</td>
</tr>
<tr>
<td>$c_{11}$</td>
<td>$bestpaths(c_{11}, 2) = {c_{11}c_{21}c_{31}}$</td>
</tr>
<tr>
<td></td>
<td>and $H = {c_{11}c_{21}c_{31}}$</td>
</tr>
<tr>
<td>$c_{32}$</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>$maxweight(c_{32}, 2)$ remains unchanged at 0.9</td>
</tr>
<tr>
<td>$c_{11}$</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>${c_{32}c_{11}}$ failed to qualify for $bestpaths(c_{11}, 1)$</td>
</tr>
<tr>
<td></td>
<td>none</td>
</tr>
<tr>
<td>$c_{source}$</td>
<td>none</td>
</tr>
<tr>
<td>$c_{12}$</td>
<td>$maxweight(c_{22}, 1) = 0.1$</td>
</tr>
<tr>
<td></td>
<td>$c_{22}$ is pruned since min-$k=1.2$</td>
</tr>
<tr>
<td>$c_{12}$</td>
<td>none</td>
</tr>
<tr>
<td>$c_{23}$</td>
<td>$maxweight(c_{23}, 1) = 0.4$</td>
</tr>
<tr>
<td>$c_{33}$</td>
<td>$maxweight(c_{33}, 2) = 0.8$</td>
</tr>
<tr>
<td>$c_{23}$</td>
<td>$bestpaths(c_{23}, 1) = {c_{23}c_{33}}$</td>
</tr>
<tr>
<td>$c_{12}$</td>
<td>$bestpaths(c_{12}, 2) = {c_{12}c_{23}c_{33}}$</td>
</tr>
<tr>
<td>$c_{source}$</td>
<td>none</td>
</tr>
<tr>
<td>$c_{13}$</td>
<td>none</td>
</tr>
<tr>
<td>$c_{22}$</td>
<td>$maxweight(c_{22}, 1) = 0.8$</td>
</tr>
<tr>
<td></td>
<td>$c_{22}$ is not pruned this time</td>
</tr>
<tr>
<td>$c_{31}$</td>
<td>$maxweight(c_{31}, 2) = 1.5$</td>
</tr>
<tr>
<td>$c_{22}$</td>
<td>$bestpaths(c_{22}, 1) = {c_{22}c_{31}}$</td>
</tr>
<tr>
<td>$c_{33}$</td>
<td>$maxweight(c_{33}, 2) = 1.7$</td>
</tr>
<tr>
<td>$c_{22}$</td>
<td>$bestpaths(c_{22}, 1) = {c_{22}c_{33}}$</td>
</tr>
<tr>
<td>$c_{13}$</td>
<td>$bestpaths(c_{13}, 2) = {c_{13}c_{22}c_{33}}$</td>
</tr>
<tr>
<td></td>
<td>and $H = {c_{13}c_{22}c_{33}}$</td>
</tr>
</tbody>
</table>

Table 3.2: Example execution of DFS.
3.3.5 Normalized Stable Clusters

In the previous sections we have presented algorithms for identifying $kl$-Stable Clusters in $\mathcal{G}$. In this section, we present algorithms for identifying normalized stable clusters. Let $\text{length}(\pi)$ define the length of path $\pi$. Let $\text{weight}(\pi)$ define the aggregate weight (sum of edge weights) for path $\pi$. We wish to find top-$k$ paths in $\mathcal{G}$ with the highest normalized weights, $\text{stability}(\pi) = \frac{\text{weight}(\pi)}{\text{length}(\pi)}$. To avoid trivial results, we constrain the paths to be of length at least $l_{\text{min}}$.

In this case we are not required to provide the lengths of paths as input. Pruning paths becomes tricky in the absence of this information. We make the following observation: if a path $\pi$ can be divided in two parts $\pi_{\text{pre}}$ and $\pi_{\text{curr}}$, such that $\pi = \pi_{\text{pre}}\pi_{\text{curr}}$, and stability of $\pi_{\text{pre}}$ is less than that of $\pi_{\text{curr}}$, irrespective of the suffix (unseen part) $\pi_{\text{suff}}$ to follow, one may drop $\pi_{\text{pre}}$ from the path. Formally,

**Theorem 3.3** If $\pi_{\text{pre}}\pi_{\text{curr}}$ is a valid path such that,

$$\text{stability}(\pi_{\text{pre}}) \leq \text{stability}(\pi_{\text{curr}}),$$

then for any possible suffix $\pi_{\text{suff}},$

$$\text{stability}(\pi_{\text{pre}}\pi_{\text{curr}}) \leq \text{stability}(\pi_{\text{pre}}\pi_{\text{curr}}\pi_{\text{suff}})$$

$$\Rightarrow \text{stability}(\pi_{\text{pre}}\pi_{\text{curr}}\pi_{\text{suff}}) \leq \text{stability}(\pi_{\text{curr}}\pi_{\text{suff}}).$$

**Proof** (Sketch) We use the fact that if $a, b, c, d \in \mathbb{R}^+$

$$\frac{a}{b} < \frac{c}{d} \Leftrightarrow \frac{a}{b} < \frac{a+c}{b+d} < \frac{c}{d}$$

Let weights of $\pi_{\text{pre}}, \pi_{\text{curr}},$ and $\pi_{\text{suff}}$ be $w_p, w_c,$ and $w_s,$ and lengths be $n_p, n_c,$ and $n_s$. Given $\frac{w_p}{n_p} < \frac{w_c}{n_c},$ it follows that

$$\frac{w_p + w_c}{n_p + n_c} \leq \frac{w_p + w_c + w_s}{n_p + n_c + n_s} \Rightarrow \frac{w_p + w_c + w_s}{n_p + n_c + n_s} \leq \frac{w_c + w_s}{n_c + n_s}.$$ 

For brevity we omit the complete algorithm and describe only modifications to the algorithm presented in Section 3.3.2. With each node $c_{ij}$, we need to maintain:

- All paths of length less than $l_{\text{min}}$ ending at that node. Let smallpaths($c_{ij}, x$) denote this for all paths of length $x$ ending at $c_{ij}$.

- A list bestpaths($c_{ij}$) of top scoring paths of length $l_{\text{min}}$ or greater ending at that node. This list can be pruned at each node using Theorem 3.3. A path $\pi_{\text{pre}}\pi_{\text{curr}} \in \text{bestpaths}(c_{ij})$ can be pruned to just $\pi_{\text{curr}}$ if $\text{length}(\pi_{\text{curr}}) \geq l_{\text{min}}$ and $\text{stability}(\pi_{\text{pre}}) \leq \text{stability}(\pi_{\text{curr}})$. In words, the prefix can be discarded if its contribution to the stability is less than that of the last $l_{\text{min}}$ edges in the path.

The algorithm in this case proceeds in the same way as in Section 3.3.2. The data structures are updated as follows: to update smallpaths($c$) for node $c$ after discovery of a new edge $c'c$ from $c' \in \text{parents}(c)$,

$$\text{smallpaths}(c, \text{length}(c')) = \text{smallpaths}(c, \text{length}(c')) \cup \{c'c\}$$
and for \( \text{length}(c'e) < x < l_{\text{min}} \),

\[
\text{smallpaths}(c, x) = \text{smallpaths}(c, x) \cup \{\text{append}(\pi, c'e) \mid \pi \in \text{smallpaths}(c', x - \text{length}(c'e))\}.
\]

To update \( \text{bestpaths}(c) \), first all possible candidates are computed as described below

\[
\text{bestpaths}(c) = \text{bestpaths}(c) \\
\cup \{\text{append}(\pi, c'e) \mid \pi \in \text{smallpaths}(c', l_{\text{min}} - \text{length}(c'e))\} \\
\cup \{\text{append}(\pi, c'e) \mid \pi \in \text{bestpaths}(c')\}
\]

After computing all the possible candidates, perform pruning. If \( \pi_1, \pi_2 \in \text{bestpaths}(c) \) and \( \pi_2 \) is a subpath of \( \pi_1 \), then \( \pi_2 \) can be deleted from \( \text{bestpaths}(c) \). Also if \( \pi_{\text{pre}}\pi_{\text{curr}} \in \text{bestpaths}(c) \), \( \text{length}(\pi_{\text{curr}}) \geq l_{\text{min}} \) and \( \text{stability}(\pi_{\text{pre}}) \leq \text{stability}(\pi_{\text{curr}}) \), then delete \( \pi_{\text{pre}}\pi_{\text{curr}} \) and add \( \pi_{\text{curr}} \) to \( \text{bestpaths}(c) \). After updating \( \text{bestpaths} \), check each newly generated path against the global top-k list of paths for inclusion.

The above algorithm can be used with the DFS framework (presented in Section 3.3.3) as well. The basic idea is the same, and pruning uses the result of Theorem 3.3. Details are omitted for brevity.

### 3.3.6 Online Version

New data arrive at every time interval. Hence it is important for the algorithms presented to be amenable to incremental adjustment of the data structures. Notice that the BFS based algorithm of Section 3.3.2 is amenable to such adjustment. Since heaps for each temporal interval are computed separately, when nodes for the next temporal interval \( G_{m+1} \) arrive, heaps for them can be computed without redoing any past computation. If the heaps for all the nodes in \( G \) are maintained on disk, a single pass over them is sufficient to compute the global top-k.

The DFS based algorithm in its original form is not an online streaming algorithm since only the source is known and the sink changes constantly as new data arrives. DFS requires the knowledge of a sink to operate. Observe that the input graph is symmetric. Therefore, \( G \) can be modified by adding the source at the last temporal interval and the sink at the first interval to perform DFS. As the data for new intervals arrive, only the source needs to be shifted (while keeping everything else the same). Therefore, since \( \text{bestpaths} \) for each node in \( G \) is maintained, DFS can be used in an incremental fashion as new data arrives.

Note that when streaming, both BFS and DFS actually perform the same operations at each iteration. The only difference is the bootstrap process.

### 3.4 Experiments

In this section we discuss the results of a detailed experimental evaluation comparing our algorithms in terms of performance, and we present qualitative results. We first present results for our cluster generation procedure and then discuss our stable cluster identification algorithms.
3.4.1 Cluster Generation

In our first experiment we assess the performance of our cluster generation procedure introduced in Section 3.2. We implemented this algorithm, and we report its performance as the pruning threshold (correlation coefficient) increases. Figure 3.6 presents the running time of our entire approach for the data set Jan 6 of Table 3.1. We measure the time required by the entire procedure, namely reading the raw data files, conducting the $\chi^2$ test, pruning based on the correlation coefficient and then running the Art algorithm (Algorithm 1) to find biconnected components. The execution of the Art algorithm is secondary storage based; we only maintain in memory the biconnected component edges in the stack. As $\rho$ increases, time decreases drastically since the number of edges and vertices remaining in the graph decreases due to pruning.

![Figure 3.6: Running time of the Art algorithm.](image)

3.4.2 Stable Clusters

Our algorithms for stable cluster identification were implemented in Java and executed on a Linux machine with a single 2.0 GHz processor. To capture the effect of I/O on performance accurately, the page cache was disabled during the experiments. Enough memory to keep nodes from the last $g + 1$ intervals was available during experimentation with the BFS based algorithms. In order to be able to vary the parameters of interest in our performance study in a consistent fashion (e.g., number of nodes, average node out degree, etc.) we generated synthetic graphs and we report performance on them. We chose the range of our parameters so as to keep response times manageable, while being able to observe performance trends. The data was generated by first creating a set of nodes of size $n$ for each of the $m$ temporal intervals. For pairs of temporal intervals $i$ and $i'$, $i - i' \leq g + 1$ (where $g$ is the gap size), edges were added as follows: for each node $c_{ij}$ from the first temporal interval, its out degree $d_{ij}$ was selected randomly and uniformly between $1$ and $2 \cdot d$, and then $d_{ij}$ nodes were randomly selected from the second temporal interval to construct edges for $c_{ij}$. Edge weights were selected from $(0, 1]$ uniformly.

Table 3.3 presents running times in seconds for identifying top-5 full paths (of length $l = m - 1$) comparing the three algorithms. Each temporal interval had $n = 400$ nodes, gap size was selected as $g = 0$, and average out degree of nodes was $d = 5$. Since the TA based algorithm is exponential in $m$, its running times were significantly higher for $m > 9$ and hence not reported. It can be observed that the BFS based algorithm outperforms DFS by a large margin in terms of running time. But it must be noted that BFS requires significantly larger amounts of memory as compared to the DFS based algorithm. For example, for finding top-3 paths of length 6 on a dataset with $n = 2000$, $m = 9$ and $g = 0$, DFS required less than 2MB RAM as compared to 35MB for BFS.

Since the TA based algorithm is not applicable to identify sub-paths (it requires $l = m - 1$), and due to its high running times on large data sizes when identifying full paths, we focus on the BFS and DFS based algorithms.
Table 3.3: Comparing BFS, DFS and TA based algorithms for different values of $m$.

<table>
<thead>
<tr>
<th>$m$</th>
<th>BFS</th>
<th>DFS</th>
<th>TA</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.65</td>
<td>60.3</td>
<td>0.35</td>
</tr>
<tr>
<td>6</td>
<td>2.09</td>
<td>368.8</td>
<td>11.11</td>
</tr>
<tr>
<td>9</td>
<td>4.49</td>
<td>754.8</td>
<td>133.89</td>
</tr>
<tr>
<td>12</td>
<td>7.95</td>
<td>805.94</td>
<td>&gt; 10 hours</td>
</tr>
<tr>
<td>15</td>
<td>12.49</td>
<td>792.05</td>
<td></td>
</tr>
</tbody>
</table>

In the sequel, we first explore the sensitivity of the BFS based algorithm for different values of the gap size $g$ in Figure 3.7. We next show the sensitivity of the same algorithm for different values of average out degree $d$ in Figure 3.8. In both cases, when either of $g$ or $d$ is increased, the number of edges grows, leading to an increase in the amount of computational effort required. Running times therefore are positively correlated with both $g$ and $d$, as expected.

Figure 3.7: Running times for BFS based algorithm seeking top-5 full paths for different values of $g$ as the number of temporal intervals is increased from 5 to 25. Number of nodes per temporal interval was fixed at $n = 1000$ and average out degree was set to $d = 5$.

Figure 3.8: Running times for BFS based algorithm seeking top-5 full paths for different values of $d$ as the number of temporal intervals is increased from 5 to 25. Number of nodes per temporal interval was fixed at $n = 1000$ and gap size was set to $g = 2$.

Figure 3.9 demonstrates the scalability of the algorithm; we show the performance of the BFS algorithm as...
the number of nodes (clusters) for each temporal interval is increased. Observe that the running times are linear in the number of nodes, establishing scalability. The figure shows running times for \( m = 25 \) and \( m = 50 \).

![Figure 3.9](image_url)

Figure 3.9: Running times for BFS based algorithm seeking top-5 full paths for two different values of \( m \) as the number of nodes in each temporal intervals is increased from 2000 to 14000. Average out degree was set to \( d = 5 \) and gap size was selected as \( g = 1 \).

Figure 3.10 presents performance results for the BFS algorithm when seeking top-5 subpaths of length \( l \). The graphs demonstrate that running times increase as \( l \) increases due to the larger number of heaps maintained with each node. As expected, running times are linear in the number of nodes per temporal interval.

![Figure 3.10](image_url)

Figure 3.10: Running times for BFS based algorithms for different values of \( l \) over \( m = 15 \) temporal intervals, as the number of nodes in each temporal intervals is increased from 500 to 2500. Average out degree was set to \( d = 5 \) and gap size was selected as \( g = 2 \).

Figure 3.11 displays running times of the DFS based algorithm for different values of \( m \) and \( n \). Figure 3.12 shows the sensitivity of the same algorithm for different values of the gap size and average node out degree. As the average out degree or gap size increases, the number of edges increases, directly affecting the running time of the DFS based algorithm. Contrast these results with those of Figure 3.7 and observe that the DFS based algorithm is more sensitive towards \( g \) than the BFS based algorithm. The running times of the DFS based algorithm increases by a factor of more than two as \( g \) is increased from 0 to 2, unlike Figure 3.7, where the effect of an increase in \( g \) is milder. Figure 3.13 shows the performance of the DFS algorithm while seeking subpaths for different values of \( l \). As expected, running times increase with increasing \( l \) and \( n \).

Figure 3.14 displays performance trends for the BFS algorithm seeking normalized stable clusters. Unlike the previous case, where only paths up to length \( l \) had to be maintained, the algorithm seeking normalized stable
Figure 3.11: Running times for DFS based algorithms seeking top-5 full paths for different values of $m$ and $n$. $g = 1$ and $d = 5$ were selected.

Figure 3.12: Running times for DFS based algorithms seeking top-5 full paths for different values of $g$, as the average out degree of nodes is increased. $m = 6$ and $n = 400$ were selected.

Figure 3.13: Running times for DFS based algorithms seeking top-5 sub paths of length $l$ for different values of $l$. $m = 6$, $d = 5$, and $g = 1$ were selected.

clusters needs to maintain paths of all lengths (those which survive pruning). This leads to an increase in running times as $m$ increases. Experimental results validate this intuition. Running times are positively correlated with $l_{min}$ as larger values of $l_{min}$ results in more paths being maintained with each node. We omit graphs where we vary $n$, $g$ and $d$ due to space limitations. Trends are as expected, running times increase gracefully with increase
in \( n, g \) and \( d \).

![Figure 3.14: Running times for BFS based algorithms seeking top-5 normalized stable clusters of length greater or equal to \( l_{\text{min}} \) for different values of \( m \). \( n = 400, d = 3, \) and \( g = 0 \) were selected.](image)

The impact of \( k \), the number of top results required, on the performance of all the algorithms is minimal, and as \( k \) increases running times increase slowly. Experimental results obtained validate that the BFS based algorithm performs better than the DFS based algorithm. The running time of the BFS algorithm increases linearly with an increase in \( n \), while that of DFS increases much more rapidly. This is because the number of edges is proportional to \( n \cdot d \). For our problem setting, the running times of the adaptation of TA based algorithms is exponential in \( m \) and hence not practical for any realistic problem size. The main advantage of DFS is its low memory requirement. DFS should be used as an alternative to BFS in memory constrained environments.

### 3.4.3 Qualitative Results

We have tested our algorithms on large collections of real data obtained from BlogScope. For purposes of exposition, we focus on data obtained for a single week (week on Jan 6 2007) and present results commenting on the output of our algorithms. We set the temporal interval for our construction of graph \( G \) to a day, analyzing seven days. Clusters for a single day were computed using our methodology in Section 3.2 using \( \rho = 0.2 \). Around 1100-1500 connected components (clusters) were produced for each day. Affinity between clusters was computed using the Jaccard coefficient and 42 full paths spanning the complete week were discovered.

Table 3.1 provides data about keyword graph sizes for two days; sizes for the rest of the days were comparable. Figure 3.1 and Figure 3.2 show example clusters we were able to identify after the procedure described in Section 3.2. It is evident that our methodology can indeed capture clusters of keywords with strong pairwise correlations. Taking into account how such correlations are generated (lots of bloggers talking about an event) it is evident that our methodology can identify events that spawn a lot of chatter in the blogosphere. A stable cluster with a path of length 3 and \( g = 2 \) is shown in Figure 3.4. Figure 3.15 presents a stable cluster that persisted for four days without any gaps. An example full length cluster (i.e., that persisted for all seven days) is shown in Figure 3.16.

Our definition of stable clusters computes cluster similarity between clusters from consecutive time periods only, instead of considering similarity between all pairs in a path. This allows us to capture the dynamic nature of stories in the blogosphere, and their evolution with time. For example notice that in Figure 3.15 we are able to identify the shift of discussion from iPhone features to the Apple vs Cisco lawsuit related to iPhone. The nature of stable clusters demonstrated in the figures attests that our methodology can indeed handle topic drifts.

In this chapter, we formally define and provide solutions for problems related to temporal association of sets.
of keywords in the blogosphere (or any other streaming text source for that matter). Our technique consists of two steps, (1) generating the keyword clusters, and (2) identifying stable clusters. For both steps we propose efficient solutions. For the problem of \(kl\)-stable clusters, we propose three solutions, based on breadth first search, depth first search, and one based on an adaptation of the well-known threshold algorithm. Detailed experimental results are provided, demonstrating the efficiency of the proposed algorithms. Qualitative results obtained using real data from BlogScope are reported that attest to the effectiveness of our techniques. The next chapter studies the aggregation of lists of top terms (or cluster of terms) in presence of hierarchies available for mapping terms (or clusters) to higher level topics.
Figure 3.16: Stable clusters with path of full length. The event corresponds to the 2007 battle of Ras Kamboni fought by Islamist militia against the Somali forces and Ethiopian troops. Observe the increase in cluster sizes on Jan 9 after Abdullahi Yusuf arrives in Mogadishu, Somalia on Jan 8 for the first time after being elected as the president. On Jan 8, gunships belonging to the US military had attacked suspected Al-Qaeda operatives in Southern Somalia. Note that the keywords are stemmed.
Chapter 4

Persistent Chatter Discovery in Presence of Topic Hierarchies

In previous two chapters, we studied how BlogScope produces ranked lists – of keywords or clusters of keywords – at select temporal intervals (e.g., per day). As a result, an agglomerative stream of ranked lists is produced, adding a new list every time step, evolving as a function of time. Such lists may be queried specifying temporal restrictions, such as the $k$ most popular blog post terms for the user specified temporal intervals and values of $k$. Identification of the $k$ most popular terms can be conducted using list aggregation techniques described in the previous chapter.

In this chapter we take a different approach to information discovery from such ranked lists that aims to unearth blog-chatter on less expected or anticipated topics. We maintain the same rank aggregation framework but we elevate terms at a higher level by making use of popular term hierarchies commonly available. We wish to support highly dynamic hierarchies so we chose to impose no restriction on them, their shape or type and we assume that they are supplied on demand (per query). Mapping of terms in the ranked list to terms (nodes) in the hierarchy takes place at query time. This is a basic requirement in order to obtain a general solution. We would like to support rank aggregation queries in the presence of such hierarchies. Under the transformation imposed on terms of a ranked list by the use of a hierarchy, rank aggregation models in the literature [35] no longer apply. A fundamental obstacle in applying such algorithms, is that after the transformation the final score of a term is not known. One would have to, in the worst case, scan the entire list in order to correctly determine the score of a term that might be part of the final answer (highest scoring terms). As a result, such transformations in the presence of hierarchies, render early stopping in rank aggregation algorithms impossible.

An example is presented in Figure 4.1, where we have two ranked lists over car companies, camera brands and movie names. A hierarchy is also provided to elevate the keywords to higher level terms. This hierarchy is used to generate two transformed rankings. In both the rankings, Cars is at the top, but after aggregation, Camera has the highest total score. Notice that although popular rank aggregation algorithms [35] provide a certificate to test as an early stopping condition, no such certificate exists in this example. One has to always traverse the lists until the end in order to compute the correct answer.

In order to enable meaningful information discovery, the entire lists have to be scanned and transformed using information from hierarchies\(^1\). Any thresholding based approach to limit the size of the ranked lists is deemed to fail, as it is not clear how to choose a meaningful value for this threshold. This seriously impacts performance as

\(^1\)If mapping for some element is not present in the hierarchy, we copy it as is in the transformed list.
Motivated by the problem of information discovery in the blogosphere, we place the problem of rank aggregation in the presence of hierarchies into perspective and we propose algorithms to efficiently solve it. In particular we make the following contributions:

- We formally define the problem of rank aggregation in the presence of hierarchies ($H-RA$) as an important information discovery primitive in ranked lists of keywords such as those produced by blog search and analysis engines, like BlogScope.
- We present a framework to reason about early stopping during the computation of top-k $H-RA$ in a probabilistic sense.
- Identifying the difficulties in solving the $H-RA$ problem probabilistically, we present algorithms for solving the $H-RA$ problem in a deterministic way with high (user specified) precision ($pH-RA$ problem). This provides a alternative way to facilitate early stopping in the $H-RA$ computations and subsequently obtain improved performance with controlled loss in accuracy.
- Since ranked lists are continuously produced, we present a method to organize them temporally in a streaming fashion enabling precomputation in order to obtain even further performance benefits in solving our problems, exploring space, performance tradeoffs in a controlled way.

### 4.1 Related Work

The problem of merging ranked lists has been addressed by many database researchers in context of meta search [32, 23, 111]. In meta search, when a user submits a query, the meta search engine retrieves ranked list of results from 4-5 different search engines, and sends back a merged list to user. In this case, the lists to
merge are not available to the system until just before merging, and hence precomputation of any intermediate results is not possible. In our case however, the system can leverage on the fact that lists are available in advance and maintaining some auxiliary datastructure is possible.

Top-\(k\) problems are widely studied in the literature. The best known general purpose algorithm for this problem was proposed by Fagin et al. [35], Gunztzer et al. [50], and Nepal et al. [81], and is commonly known as the threshold algorithm or TA. Many variants and enhancements of TA have been proposed [73, 30]. Theobald et al. [70] proposed a probabilistic variant of TA for approximate query processing. This is the only algorithm in the literature addressing probabilistic stopping conditions for the variant of TA called TA-sorted [35]. Bast et al. [10] proposed different schemes to schedule list probes in order to improve the efficiency of TA. However little work has gone in developing algorithms for preprocessing the lists, and preparing them for TA execution. The only work in this area that we know of is by Dubinko et al. [31], which maintains a binary tree on the lists to reduce computation performed at query time. This model however can not be tuned exploring tradeoffs between offline precomputation and online query processing. Guha et al. [49] have proposed a sparse interval set in the context of OLAP to maintain a better histogram approximation of a data distribution.

Manku et al. [71] have proposed algorithms for computing approximate quantiles of large datasets in a single pass. We have discussed an adaptation their approach for computing median ranks fast by maintaining a precomputed summary datastructure. Greenwald et al. [48] improve upon the algorithms by Manku et al. to provide better bounds on memory requirements and accuracy guarantees.

### 4.2 Problem Definition

Let \(X = X_1, X_2, X_3 \ldots\) be a temporally ordered sequence of ranked lists. Each list \(X_i\) consists of ranked terms, according to some specified ranking function \(R\) (e.g., tfidf [92]). Let \(x_{ij}\) denote the \(j\)-th element for list \(X_i\), \(1 \leq j \leq N\), where \(N\) is the size of the lists. Without loss of generality and to simplify our notation we assume that all lists are of the same size. At a specified temporal granularity (e.g., every hour) a new list is included in our sequence. Thus, \(X\) is an agglomerative stream of ranked lists \(X_i\). Let \(s_{ij}\) denote the score according to \(R\) of the \(j\)-th element of the \(i\)-th list.

We consider hierarchies consisting of multiple levels. Each level consists of terms with similar abstraction (for example a level of a product hierarchy may be ‘type of product’, another level may be ‘product brand’, etc). Without loss of generality we use hierarchies to map (or elevate) terms of a list \(X_i\) to the same level of the hierarchy. Thus, a hierarchy \(H\) facilitates a mapping between the set of elements of \(X_i\) and terms at a specified level in the hierarchy. Assume that the domain of terms at the specified level in the hierarchy is \(L\). We treat a hierarchy as a function \(H(x_{ij}) = y\), \(1 \leq j \leq N\), \(y \in L\), where \(|L| \leq N\); from the domain of terms in list \(X_i\) to the domain of terms at a specific level of a hierarchy. The effect of mapping list \(X_i\) using function \(H()\) is to produce a list \(Y_i\) of size equal to the size of \(X_i\). However, since \(|L| \leq N\) there exist multiple elements in \(Y_i\) corresponding to the same term. With each term \(y\) from the hierarchy we associate two numbers: \(M_y\), the maximum multiplicity of term \(y\), is the maximum number of elements from a list \(X_i\) that can be associated to (map to) \(y\). Notice that \(M_y\) is a property of the hierarchy; given any hierarchy we can easily identify \(M_y\) just by counting the children of the node \(y\). We also associate \(m_y\), the actual multiplicity in the hierarchy, across all elements, is denoted by \(M = \max_{y \in L} M_y\).

\(^2\)Without loss of generality assume that all elements of a list \(X_i\) map to terms at a specific hierarchy level. If this is not the case the term(s) not mapped are transfered to the corresponding list \(Y_i\) unmapped.
Let $\phi(g), 1 \leq g \leq m^t_y$ be a function returning the position $j, 1 \leq j \leq N$ in $X_i$ of the element $x_{ij}$ that is mapped for the $g$-th time to $y$. We define the score of term $y$ in list $Y_i$ as

$$s_i(y) = \sum_{g=1}^{m^t_y} s_{i\phi(g)}.$$  

**Problem 4.2.1 ($H - RA$)** Let $Q = [X_i \ldots X_j]$ be a query specified as a temporal restriction on $X$, along with a value $k$ and a mapping $H$. The problem of rank aggregation in the presence of a hierarchy is equivalent to the problem of rank aggregation on lists $Y_i \ldots Y_j$. More specifically is the problem of identifying the $k$ highest ranking terms $y$ in lists $Y_i \ldots Y_j$, obtained by applying $H()$ on the elements of $X_i, \ldots X_j$.

It is evident that the $H - RA$ problem has an obvious solution if one is willing to scan all lists $X_i \ldots X_j$ entirely. We would like to obtain solutions that are able to report the answer faster. Since the score of term $y$ in a list $Y_i$ is not known precisely until the last element in the list is encountered in the worst case, it is evident that early stopping for the $H - RA$ problem is not always possible in a deterministic sense. In Section 4.4 we demonstrate the difficulties associated with obtaining a probabilistic solution to this problem. Consequently, in order to be able to solve this problem faster and still have deterministic guarantees for the answers we are obtaining, we relax the $H - RA$ problem as follows:

**Problem 4.2.2 ($H - RA$ with precision $p$ ($pH - RA$))** Let $Q = [X_i \ldots X_j]$ be a query specified as a temporal restriction on $X$, along with a value $k$, a precision threshold $p, 0 \leq p \leq 1$ and a mapping $H$. The rank aggregation problem problem in the presence of hierarchies with precision $p$ is the problem of identifying the $k$ highest ranking terms $y$ in lists $Y_i \ldots Y_j$, obtained by applying $H()$ on the elements of $X_i, \ldots X_j$, such that at least $p \ast k$ answers correctly belong to the answer of $H - RA$.

Notice that we impose deterministic requirements for the solution of the $pH - RA$ problem. We are seeking solutions that are able to report the answers to $pH - RA$ fast and we describe several performance enhancements to our basic solutions exploring precomputation.

### 4.3 Aggregation in Presence of Hierarchies

We will employ the Fagin’s NRA (No Random Access) version of the threshold algorithm [35] to aggregate $T$ lists, $X_1, \ldots X_T$ using a hierarchy $H$. In the following discussion, for simplicity, we assume the aggregation function is an unweighted summation, though the techniques are generic enough to accept any linear monotone function, in accordance to previous work.

We probe the lists in a round robin manner. After reading an element $x$, we transform it to the appropriate term (on a specified level) in the hierarchy, as $H(x) = y$. If $y$ has never been encountered before, we add it to a buffer that maintains a list of all seen elements along with their worstcase score, and information about how many times they have been seen in each list. Worstcase score $ws(y)$ of $y$ is the sum of scores of all seen instances of $y$. Therefore, after scanning $d$ element from each of the $T$ lists,

$$ws(y) = \sum_{i=1}^{T} \sum_{j=1}^{d} I_{H(x_{ij})=y} \cdot si_j$$

where $I_{H(x_{ij})=y}$ is one if $H(x_{ij}) = y$, and zero otherwise. Also with each element $y$ in the buffer, we maintain
$T$ counters, $c_1(y), \ldots, c_T(y)$, with $c_i(y)$ representing the number of times $y$ is seen in lists $Y_1, \ldots, Y_T$.

$$c_i(y) = \sum_{j=1}^{d} I(H(x_{ij})=y)$$

When we encounter an element that already exists in the buffer, we update its worst-case score and increment the counters appropriately. To consider the elements never seen before, we include a virtual element in the buffer with worst-case score zero, and all counts also equal to zero.

During the execution, the buffer has three types of elements: fully seen, partially seen and completely unseen. For the elements fully seen, their worst-case score is the same as their final score. For the rest of the elements, we associate a quantity, unseen score, which is the sum of scores of unseen instances of the element. Hence, for $y$, unseen score in $X_i$ will be (after reading $d$ elements),

$$us_i(y) = \sum_{j=d+1}^{N} I(H(x_{ij})=y) \cdot s_{ij}.$$

Total unseen score of $y$ therefore will be $us(y) = \sum_{i=1}^{T} us_i(y)$. The problem however is that we don’t know the values of $us(y)$. As a result, we can try to estimate it probabilistically or determine an upper bound for it. We explore both possibilities in the sections that follow.

Periodically we order the elements in the buffer according to their worst-case score, and select the $k$ highest scoring elements as the current top-$k$ (denoted by $currTopK$). The minimum worst-case score among these elements is denoted by min-$k$. The termination condition of the algorithm is determined by comparing the min-$k$ score with the scores of elements not in the current top-$k$. In the next section, we discuss a probabilistic stopping condition utilizing this framework, while in Section 4.5 we propose a relaxed deterministic condition for termination.

### 4.4 Aggregation with Probabilistic Guarantees

A probabilistic stopping condition will enable early termination of the algorithm by relaxing the accuracy requirement at termination. Since the probabilistic algorithm requires knowledge of score distributions to estimate the unseen score, we can approximate the scores either by employing a parameterized distribution (e.g., geometric) or by maintaining precomputed histograms on the scores for each list. We analyze both cases in detail.

#### 4.4.1 Probabilistic TA Assuming Geometric Distribution

The main premise of early stopping in the TA algorithm is that scores decrease as we traverse the list. To capture such decreasing scores, we use the following geometric condition: if $s_{ij}$ denotes the score of $j^{th}$ element in the list $X_i$,

$$s_{i,j+b} \leq r_i \cdot s_{ij} \ \forall j, \ r_i \leq 1,$$

for some fixed constant $b$ in every list $X_i$ (notice that since all lists are available, such a $b$ can always be determined). This means that the score must decrease by a factor of at least $r_i$ every $b$ elements. This implies that the total score of all elements in the list $X_i$ is less than $b(1 + r_i + r_i^2 + \ldots + r_i^{n/b}) = \frac{b(1-r_i^{n/b})}{1-r_i}$, assuming all scores in the range $[0, 1]$. We can precompute the value of $r_i$ for each of the lists and store it as metadata.
For the analysis, we assume that each element \(y\) has a probability \(p_{yi}\) associated with it for every list \(X_i\), and each position in the list \(X_i\) is filled by selecting elements (terms) independently. This means that elements for each position in the list are selected independently, and an element \(y\) has a chance \(p_{yi}\) of being selected at any position in \(X_i\).

\[
\Pr[H(x_{ij}) = y] = p_{yi}, \forall j \leq N \text{ (independent of } j)\]

We now apply the TA algorithm as described in the previous section. The unseen score of \(y\) in a list is the total score of all instances of \(y\) that are yet to be seen. The unseen score can be computed by modeling it as a sum of independent random variables (one for each unseen position). Thus, after reading \(d\) elements from the list \(X_i\), the expected unseen score \(\mu_{yi}\) for \(y\) in \(X_i\) can be computed as

\[
\mu_{yi} = \sum_{j=d+1}^{N} p_{yi} s_{ij} \\
\leq \sum_{j=d+1}^{N} p_{yi} s_{id} r^{[(j-d)-1]/b]} \\
\approx s_{id} p_{yi} b \cdot \frac{1 - r^{(N-d)/b}}{1 - r} \\
\approx s_{id} \cdot \frac{p_{yi} b}{1 - r}
\]

The simplification in the last step assumes that the number of elements \(N\) in the list is much larger than \(b\) (hence \(r^{(N-d)/b}\) is negligible). Similarly the standard deviation of the unseen score \(\sigma_{yi}\) is,

\[
\sigma_{yi} = s_{i,d} \cdot \sqrt{\frac{p_{yi}(1 - p_{yi})b}{1 - r^2}}
\]

Using this information, we can compute the mean and variance of \(us(y)\) as \(\sum_{i=1}^{T} \mu_{yi}\) and \(\sum_{i=1}^{T} \sigma_{yi}^2\). This can be used in conjunction with the Chebyshev [37] inequality—for a random variable \(Z\), \(P(|Z - E[Z]| \geq e) \leq \frac{\sigma_Z^2}{e^2}\)—to estimate the probability that \(us(y) + ws(y) > \min - k\) (with some specified confidence bound). If this bound is not tight enough, we can also use Chernoff [37] bounds as presented below.

The fact that \(y\) appears or not at the \(j\)th position in \(X_i\) is a Bernoulli trial with success probability \(p_{yi}\), moment generating function of which is \(g_{yi}(t) = (1 - p_{yi}) + e^{ts_{ij}} p_{yi}\). The moment generating function of the unseen score of \(y\) in the list \(X_i\) therefore will be,

\[
g_{yi}(t) = \prod_{j=d+1}^{n} g_{ij}^p \\
\leq \prod_{j=d+1}^{n} \left( (1 - p_{yi}) + p_{yi} e^{ts_{id} r^{[(j-d)-1]/b]} \right)
\]

For large values of \(i\), the term in the product becomes insignificant as it approaches one, and hence can be neglected. The moment generating function for \(us(y)\) will be \(g_y(t) = \prod_{i=1}^{T} g_{yi}\). This information can be used with the Chernoff bound for estimating the probability that \(us(y)\) is greater than \(\min - k\) (with a specified confidence
value). Specifically for a random variable $Z$ with moment generating function $g_Z(t)$,

$$
\Pr(Z \geq z) \leq g_Z(t)e^{-tz} \text{ for } t \geq 0,
$$

and

$$
\Pr(Z \leq z) \leq g_Z(t)e^{-tz} \text{ for } t \leq 0.
$$

To complete the TA algorithm described in the previous section, we use the following termination condition. For each element in the buffer (including the virtual element) but not in the current top-$k$, we evaluate the probability that it does not belong to the top-$k$ (using the Chernoff bound with $g_y(t)$, or Chebyshev bound with mean and variance). This probability is,

$$
\Pr[y \text{ belongs to top}-k] = \Pr[us(y) > \min-k - ws(y)]
$$

If the maximum of such probabilities is bounded by some parameter $\epsilon$, we terminate. The probability that the precision of this algorithm is $q/p$ is

$$
\left(\frac{k}{q}\right)(1 - \epsilon)^q e^{-\epsilon q}.
$$

Estimation of $p_{yi}$ is a crucial step. We can either assume that $y$ has the same multiplicity $M_y$ in each list; in this case $p_{yi} = M_y/N$. If we don’t assume this, we can estimate $p_{yi}$ by applying Bayesian Decision Theory [12] since the number of occurrences of $y$ in $X_i$ follows a binomial distribution. After traversing $d$ elements of a list, we have gained some knowledge about this binomial distribution, and hence can guess its characteristic probability. If we assume the prior to be a Beta distribution, $B(\alpha, \beta)$, the posterior will also be Beta distributed, $B(c_i(y) + \alpha, d - c_i(y) + \beta)$, where $d$ is the total number of observations and $c_i(y)$ is the number of successes. With no other information, we can assume the prior to be uniformly distributed (uniform distribution is the same as the Beta distribution with parameters 1 and 1). We can also maintain precomputed information about the distribution of multiplicities for each list (if we have some knowledge about hierarchies) to use as a prior for a more accurate estimation. Therefore, after traversing $d$ elements in a list, and encountering $c_i(y)$ occurrences of $y$,

$$
p_{yi} \sim B(c_i(y) + 1, d - c_i(y) + 1).
$$

For the first case, when we assume we know $p_{yi}$ as $M_y/N$, we can use the Chernoff bound with $g_y(t)$ to estimate the probability in Equation 4.1. This will require a minimization on parameter $t$, to find a tight bound on $us(y)$. If we wish to use the Beta distribution for estimating $p_{yi}$, we can first find a bound on $p_{yi}$ and then use that to compute $g_y(t)$ in order to apply Chernoff bounds. While all this is numerically possible (using tools like Matlab), it involves the solution of complex equations at each estimation interval, which will impose significant computational overheads. Such an approach is interesting from a modeling perspective, however to obtain a practical solution we seek lightweight algorithms. We discuss a solution that utilizes information precomputed on base lists (the $X_i$’s) in the form of histograms in the next section.

### 4.4.2 Probabilistic TA Using Histograms

Probabilistic modeling of the adaptation of the TA algorithm in our setting in order to derive early stopping conditions is an interesting exercise, as discussed in the previous section. A practical realization of such an approach however is not easy, due to the high computational overheads for solving the required equations at realistic scales of the problem. The only work that discusses implementation of a probabilistic top-$k$ approach is by Theobald et. al., [70]. They provide an analysis of their framework with parameterized distributions (like Uniform and Poisson) for modeling purposes, but since real life data usually do not follow such distributions,
they propose the use of histograms for approximating score distributions. Probabilistic bounds can be derived by convoluting these histograms. We discuss an adaptation of such an approach to our problem.

We maintain precomputed histograms over scores for each of the lists. To simplify our presentation assume that \( y \) has multiplicity \( M_y \) in each list. Assume that we have seen an element \( y \), \( c_i(y) \) times, in the list \( X_i \) after \( d \) iterations. Let \( P^d_i \) denote the score distribution of the unseen score part of \( X_i \) (\( P_i \) will change as we progress in the algorithm). As a result the probability distribution of \( \text{us}_i(y) \) will be the convolution of \( P^d_i \) done \( M_y - c_i(y) \) times with itself, i.e.,

\[
\Pr_{\text{us}_i(y)} = \text{CONV}_{j=1}^{M_y-c_i(y)} P^d_i
\]

The probability distribution of \( \text{us}(y) \), which is a sum of \( \text{us}_i(y) \), will therefore be,

\[
\Pr_{\text{us}(y)} = \text{CONV}_{i=1}^{T} \left( \text{CONV}_{j=1}^{M_i-c_i(y)} P^d_i \right)
\]

\( \text{CONV}(P_1, P_2, \ldots) \) here denotes the convolution of \( P_1, P_2, \ldots \) distributions. Using this probability distribution we can evaluate the probability that an element not in the current top-\( k \) can enter the actual top-\( k \). Then we terminate the algorithm when this probability is bounded by \( \epsilon \) for all element in the buffer (including the virtual element). Formally, we stop when,

\[
\max_{y \notin \text{currTopK}} \Pr[\text{ws}(y) + \text{us}(y) > \text{min-}k] < \epsilon
\]

The cost of each convolution required is exponential in the number of lists being aggregated. This is further complicated by the fact that this needs to be done for each element in the buffer, and at each estimation step (since \( P^d_i \) changes as we progress down the lists for each element). Even if we group together the elements with identical counts (number of times seen in each list), there will be \( T^M \) different convolutions, where \( M \) is the maximum multiplicity across all the elements in the hierarchy. It is evident that this technique, being exponential in both \( M \) and \( T \), is not practically viable. For any realistic setting, \( T \) can be expected to be at least in the order of ten or hundred lists and \( M \) equally sizable.

The solution outlined above extends the technique of Theobald et al. [70] in our setting in which we obtain lists as they are transformed by hierarchies on demand. These hierarchies introduce duplicates in the lists, forcing the probabilistic estimation framework to consider each element multiple times for each list. This leads to an increase in the number of histograms convoluted for each step by a factor of \( M \), increasing the time required to convolute by an exponential factor. The number of different possible distributions that the unseen score of an element can follow also increases exponentially from \( 2^T \) to \( 2^{TM} \), since each of the \( T \) counters can now have value between 1 and \( M \). The work by Theobald et al. is focused towards search queries in information retrieval where the number of lists under aggregation is small (2-4), while our application (in BlogScope) requires techniques to be scalable to hundreds of lists.

To confirm our analytical expectation that such approaches are not viable for our setting, we implemented the probabilistic framework discussed above using histograms. Experiments with this approach show that the computational effort required for convolutions is significantly (orders of magnitude) higher than the savings in I/O. Even for small data sizes (10 lists, with multiplicities in range 5 – 10), such a probabilistic approach requires significantly more time than the naive approach of scanning all lists entirely, aggregating term scores in memory and sorting to identify the top-\( k \) elements.
4.5 Deterministic Aggregation with Precision Guarantees

The maximum possible score of every element in the buffer can be computed using the knowledge of the maximum multiplicity of an element in a list as,

$$\text{maxPossibleScore}(y) = ws(y) + \sum_{i=1}^{T} s_{id}(m^i_y - c_i(y)).$$

Since $m^i_y$ may not always be known or available, we can instead use $M_y$ which can be determined by the hierarchy at query time to bound the maximum possible score:

$$\text{maxPossibleScore}(y) \leq ws(y) + \sum_{i=1}^{T} s_{id}(M_y - c_i(y)).$$

(4.4)

One possible termination condition for the algorithm of Section 4.3 is to stop when no element (except those in the current top-$k$ in the buffer (including the virtual element) has its maximum possible score above the min-$k$ score. This stopping condition provides the solution to the $H - RA$ problem, and guarantees that the precision of the top-$k$ elements we output will be 1.0. But such a stopping condition is overly conservative and pessimistic; it essentially assumes that all unseen instances of elements not in the current top-$k$ are right after the current scan position. Contrasting this with the original TA algorithm (operating on lists without duplicates), this problem is more serious here since in our case, the estimate of $\text{maxPossibleScore}(y)$ is amplified by a factor of $M_y$ (notice that $M_y = 1$ in the case of lists without duplicates). Such an overestimate of $\text{maxPossibleScore}$ eliminates the possibility of early stopping. We report experiments in Section 4.7 confirming this expectation.

From an information discovery point of view, obtaining results fast is a primary concern. Normally we do not know what exactly we are looking for, thus we submit multiple queries utilizing diverse hierarchies and seek interesting trends. Fast turnaround time in such a speculative information discovery process is essential. To increase the possibility of early stopping of the algorithm (and thus obtain improved performance) while maintaining a deterministic framework for query answers, we relax our precision requirements supplying a target precision value $\rho$ for the desired result at query time. Our algorithm guarantees that the top-$k$ result set we output will have precision greater than $\rho$. We refer to this problem as $pH - RA$. Setting $\rho$ to 1.0, $pH - RA$ reduces to $H - RA$.

Thus we seek to identify a number of elements $k'$ that have their maximum possible score above the min-$k$ score. If the maximum possible score of the virtual element is less than the min-$k$ score, and $k' < k$, then at least $k - k'$ among the current top-$k$ elements belong to the actual top-$k$. This means that the precision is guaranteed to be at least $1 - k'/k$. Hence, we can terminate when $1 - k'/k \geq \rho$.

As we decrease the value of target precision $\rho$, our algorithm will be able to terminate earlier. Experimental results demonstrate that the observed precision is usually significantly higher than $\rho$. Pseudo code for the complete algorithm $pH - RA$ is presented as Algorithm 4.

Algorithm 4 can also be used as a filter for two step top-$k$ computation. We can set $\rho$ to a low value (e.g., 0.2) in the first step and find top $k/\rho$ elements using the $pH - RA$ algorithm. The output of the first step is guaranteed to contain all correct top-$k$ elements. In the second step, random access can be used to disambiguate the results to identify the actual top-$k$ result set.

As an example, we present in Table 4.1 a snapshot of the buffer after scanning 7 elements from each of the two initial lists displayed in Figure 4.1 using the same hierarchy. In this example, $\text{max}$ scores are calculated
Algorithm 4 TA: Solving $pH - RA$

**INPUT** $k$, $\rho$, $H$ and input lists, $X_1, \ldots, X_T$.

1: Initialize buffer to empty
2: Add one virtual element to the buffer with its worst-case score and all $c_1, \ldots, c_T$ counters set to zero
3: set $falseNegatives = \infty$
4: while $falseNegatives > (1 - \rho)k$ do
5:     for $i = 1$ to $T$ do
6:         read $x =$ next element from $X_i$
7:         read $s =$ score of $x$ from $X_i$
8:         set $y = H(x)$
9:         if $x$ not in buffer then
10:             insert $x$ in the buffer with its worst-case score and all $c_1, \ldots, c_T$ counters set to zero
11:         end if
12:         Increase worst-case score of $x$ in buffer by $s$
13:         Increment the $i^{th}$ counter $c_i$ for $x$ in buffer by 1
14:     end for
15:     if size(buffer) $\geq k$ then
16:         sort the buffer in descending order of worst-case scores
17:         set $\text{min-k} =$ score of $k^{th}$ element in buffer
18:         set $falseNegatives = 0$
19:         for $j = k + 1$ to size(buffer) do
20:             set $x' =$ $j^{th}$ element of buffer
21:             set $\text{max}(x') = \text{ws}(x') + \sum_{i=1}^{T} s(M_{x'} - c_i(x'))$ \{ws($x'$) is the worst-case score of $x'$, $c_i(x')$ is the $i^{th}$ counter for $x'$ in the buffer, and $M_{x'}$ is the multiplicity of $x'$ in $H$\}
22:             if $\text{max}(x') > \text{min-k}$ then
23:                 if $x'$ is the virtual tuple then
24:                     set $falseNegatives = \infty$
25:                 end if
26:             set $falseNegatives = falseNegatives + 1$ \{If max score of virtual tuple is greater than min-k score, number of false negatives can be infinite\}
27:         end if
28:     end for
29: end if
30: end while
31: **OUTPUT** top-$k$ elements in the buffer

<table>
<thead>
<tr>
<th>Element</th>
<th>$\text{ws}$</th>
<th>$c_1$</th>
<th>$c_2$</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Camera</td>
<td>4.82</td>
<td>3</td>
<td>4</td>
<td>5.24</td>
</tr>
<tr>
<td>Cars</td>
<td>3.70</td>
<td>2</td>
<td>2</td>
<td>4.62</td>
</tr>
<tr>
<td>Movies</td>
<td>1.76</td>
<td>2</td>
<td>1</td>
<td>2.26</td>
</tr>
<tr>
<td>Virtual</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>1.84</td>
</tr>
</tbody>
</table>

Table 4.1: Example snapshot of the buffer after completion of 7 iterations during the algorithms’ execution on the lists shown in Figure 4.1. This table presents the element, its worst score, two counters and the maximum possible score.
using Equation 4.4 using $s_{1,7} = 0.42$ and $s_{2,7} = 0.50$ since $d = 7$, and taking in account that $M_{Cars} = 3$, $M_{Camera} = 4$ and $M_{Movies} = 2$. If we were computing top-2, the min-k score would be 3.70, which is greater than the best scores of all other elements (including the virtual tuple), and hence $(Cars, Camera)$ would be the true answer for all values of target precision $\rho$.

4.6 Preprocessing for Aggregation

Since all the lists $X_i$ are available to us in advance, we explore and design a framework for efficient answering of ad hoc rank aggregation queries over subsets of these lists by utilizing a suitable preprocessing strategy. In the absence of user supplied ‘preferences’ towards individual lists, we assume that each list participates in rank aggregation with the same weight. We remark that the preprocessing strategies presented in this section can be extended to incorporate such information if available. Our goal is to obtain even better performance via precomputation. Motivated by the streaming nature of our problem (a new list $X_i$ arrives at every timestep) we design solutions that are amenable to incremental maintenance. Our goal is to support ad hoc range queries on the evolving stream of lists $X_i$. A range query requesting merging all lists between time $[t, t+s]$ requires merging of all lists in the range $[X_t, X_{t+s}]$. Since preprocessing will increase the space requirements of our solution, we seek solutions that offer a tunable trade-off between increased storage requirements and query answering time.

We construct a sparse interval set on our input lists (that is incrementally maintainable in a streaming setting). Maintenance of such a structure guarantees that every range query of size $s$ can be answered by merging less than $2\lceil \log_l s \rceil + 2$ precomputed lists, for some tunable parameter $l$.

Consider $l > 1$ as a parameter and set $r = \log_l n$, where $n$ is the number of lists currently in the system. Assume that the items are indexed $0, 1, \ldots, n-1$; they define the level 0 points.

- Consider the numbers $0, l^j, 2l^j, \ldots$. These define points at level $j$.
- Overall we have $r + 1$ levels with 0 and $n$ as level $r + 1$ points.
- The interval $[0, n-1]$ is in the sparse system $S$. Any pair of level $j$ points between adjacent level $j+1$ points defines an interval in $S$.

Claim 4.6.1 A point is never contained in more than $O(l^2 \log_l n)$ intervals.

Proof: Follows from [49].

The sparse system can be maintained incrementally, since we only need more intervals as lists arrive, never having to delete or remove anything. Each time a new list arrives we increment $n$ to $n+1$, and identify all the intervals that end at the index corresponding to the newly arrived list. Once the intervals are identified, we retrieve and merge base lists corresponding to each of these intervals and save them for later use. To merge the base lists, we read each one in memory (one by one), and compute total score for each element $x$ in them, and save back a merged list with scores sorted in descending order. Claim 4.6.1 ensures that the amount of preprocessing required is bounded.

Claim 4.6.2 Any range $[0, s]$ can be represented as a disjoint union of at most $\lceil \log_l s \rceil + 1$ intervals from the above collection.

Proof: We will first show using induction that, $[1, s]$ can be represented by using at most $j$ intervals where $s \leq U^j$. This is true for the base case $j = 1$. Now consider $U^j < s \leq U^{j+1}$. We can write $[1, s]$ as $[1, \alpha U^j]$ and $[\alpha U^j + 1, s]$.
Chapter 4. Persistent Chatter Discovery in Presence of Topic Hierarchies

Figure 4.2: An example sparse system over 10 initial points with \( l = 3 \). Lines connecting points represent intervals in the system. Three orange intervals show intervals that need to be merged for answering the query \([1, 8]\).

<table>
<thead>
<tr>
<th>List Arrival</th>
<th>Intervals Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>none</td>
</tr>
<tr>
<td>1</td>
<td>none</td>
</tr>
<tr>
<td>2</td>
<td>([1, 2])</td>
</tr>
<tr>
<td>3</td>
<td>([2, 3], [1, 3])</td>
</tr>
<tr>
<td>4</td>
<td>none</td>
</tr>
<tr>
<td>5</td>
<td>([4, 5])</td>
</tr>
<tr>
<td>6</td>
<td>([5, 6], [4, 6], [3, 6])</td>
</tr>
<tr>
<td>7</td>
<td>none</td>
</tr>
<tr>
<td>8</td>
<td>([7, 8])</td>
</tr>
<tr>
<td>9</td>
<td>([8, 9], [7, 9], [6, 9])</td>
</tr>
</tbody>
</table>

Table 4.2: Intervals that are added to the sparse set system of Figure 4.2 as lists arrive in a streaming setting.

where \( \alpha \) is maximal. The later is similar to \([1, s - \alpha l^j]\). Since \( \alpha \) is maximal, \( s - \alpha l^j \leq l^j \), and therefore \([1, s - \alpha l^j]\) can be represented by \( j \) intervals. Also \([1, \alpha l^j]\) is a valid interval in the sparse system. Thus by induction, \([1, s]\) can be represented by using at most \( j \) disjoint intervals.

Since \([1, s]\) can be represented by using at most \( \lceil \log_l s \rceil \) intervals, \([0, s]\) can be represented by \( \lceil \log_l s \rceil + 1 \) intervals.

Claim 4.6.3 Any range \([t_1, t_2]\) can be represented as a disjoint union of at most \( 2\lceil \log_l s \rceil + 2 \) intervals from the above collection, where \( s = t_2 - t_1 + 1 \).

Proof: The query \([t_1, t_2]\) can be divided in two parts \([t_1, \alpha l^j]\) and \([\alpha l^j + 1, t_2]\), but cutting it at a point \( \alpha l^j \), such that \( j \) is maximum. By symmetry, and using the result of previous claim, we can express each of these parts by at most \( \lceil \log_l s \rceil + 1 \) intervals, totaling to \( 2\lceil \log_l s \rceil + 2 \).

We precompute and maintain lists for each of the intervals in the sparse system. At query time, we first find a minimal set of disjoint intervals that cover the query range exactly. Claim 4.6.3 ensures that the number of such intervals will be bounded to be logarithmic in the actual query size. We retrieve the precomputed lists for each of these intervals and apply the \( pH - RA \) algorithm. Note that, by constructing the sparse set, we are not making any approximation, and hence all our correctness guarantees remain valid.

In order to find the minimal set of disjoint intervals, we first identify the largest interval contained in the query and then recurse for the remaining parts. Our experiments show that this step usually takes a few milliseconds for query range size in the order of hundreds, and hence optimizations here are of little significance.

Figure 4.2 shows an example of the sparse system after 9 lists have been added. Table 4.2 shows the intervals which are added to the system after each list arrives. We maintain precomputed aggregated base lists \((X_i)\) for...
when the query \([1, 8]\) arrives, we first determine a minimal disjoint set of intervals required to exactly cover the query. This can be done by recursively identifying the largest interval contained in the query, which results in \([1, 2]\), \([3, 6]\), and \([7, 8]\). We fetch the lists corresponding to these three intervals and apply \(pH - RA\). Utilizing the sparse set system in this case, we had to merge only 3 lists as opposed to 9, hence reducing the query time by a factor of 3.

### 4.6.1 Offline Preprocessing of Hierarchies

If the hierarchies are available to us for offline computation, we can preprocess them and maintain for each element \(y\) its multiplicity \(m_y\) in each list. To save on storage, we may further compress this information by grouping elements with similar multiplicities in the same bucket, and by replacing their multiplicity by its maximum possible value. Such relaxation will not affect the accuracy of the proposed algorithms. More relaxed estimates on multiplicities however may lead to longer running times.

If storage is not a constraint, we can also preprocess each list, by grouping all its elements that map to the same higher level term in the hierarchy. This way, the resulting list will not have any duplicates. For each interval of the sparse system, we can maintain such a list. When a query arrives, we can retrieve the already merged lists without any duplicates and aggregate them using the vanilla TA algorithm [35].

### 4.6.2 Online Processing of Hierarchies

If we don’t have the hierarchy available for preprocessing, we can do a single pass on the hierarchy at query time to find maximum multiplicity \(M_y\) of each element \(y\). This information can be used with the TA algorithm presented in Section 4.5.

For constructing the sparse set system, we can merge the lists (simple aggregation without hierarchy) for each of the interval in the sparse system, and save them for later use. When the query arrives, we can retrieve these already merged lists (maximum \(2^\left\lceil \log_2 s \right\rceil\) for a query) and aggregate them using the \(pH - RA\) described in Section 4.5.

### 4.7 Experiments

We conduct a variety of experiments to evaluate the techniques presented in the previous sections. The experiments were conducted on a 3.0GHz Pentium 4 machine with 3GB RAM, running SUSE Linux. All the algorithms were implemented in Java. For performance comparisons, operating system caches and buffers were cleared before each run (by remounting the underlying filesystem).

In our evaluation we utilize both synthetic and real data. In order to be able to flexibly explore several aspects of our algorithms and vary parameters on demand we utilize synthetic data sets. We demonstrate the practical utility of our techniques with real data sets from BlogScope in Section 4.7.3.

#### 4.7.1 Evaluating \(pH - RA\)

In the experiments presented in this section, we deploy Algorithm 4 to merge lists in order to produce a top-100 result set. We evaluate the termination condition every time 10% of the size of the list has been read. Running times and precision observed were recorded for each experiment. Precision in this case is the fraction of elements in the top-\(k\) the algorithm outputs that also belong to the actual (confidence \(p = 1\) top-\(k\).
Figure 4.3: Observed running times for naive algorithm and for different target precision values (20 to 100) in Algorithm 4 as we vary the list size.

Figure 4.4: Observed running times for naive algorithm and for different target precision values (20 to 100) in Algorithm 4 as we vary the number of lists aggregated.
Observing real data sets extracted from BlogScope, we note that several terms in the base lists $X_i$ appearing in close proximity in $X_i$ (i.e., their relative popularity is close) map to the same hierarchy node. For example, keywords *saddam*, *iraq* and *america* appear very close together in base lists. Same behavior is observed for other keywords such as *sony*, *nintendo*, *xbox*, *PS3*. Thus, to generate the lists, we started with an initial list, and conducted a number of random swaps of consecutive elements (equal to five times the length of the initial list) in order to obtain the next list. We generated 300 lists in this way. In order to generate a hierarchy we start with the initial list, make 50 times the list size number of random swaps (of consecutive elements) and group together consecutive elements to the same higher level term in the hierarchy. This way, a slight correlation between the position of terms and the hierarchy is introduced in the data, i.e., elements that map to the same higher level term are close in positions. Multiplicities are derived from a Normal distribution with $\text{mean} = 6$. The variance of the Normal distribution was set to $\text{mean}/3$ and all multiplicities were restricted between 1 and $2 \cdot \text{mean}$. The scores were derived from an exponential distribution, $\Pr[z < Z] = e^{-\lambda z}$, with the parameter $\lambda = 0.005$.

Figure 4.3 presents a comparison of running times as we merge 100 lists and vary the list size from 0.5 million to 2.5 million. Figure 4.4 presents the same comparison, but this time we vary the number of lists from 100 to 300,
keeping list size fixed at 0.5 million. The observed precision is 100% in all the cases and hence is not reported explicitly in the graph.

We refer to the algorithm that reads the base lists in their entirety and conducts rank aggregation in memory as Naive. It can be observed that as long as the precision parameter (value of $p$) supplied at $pH - RA$ is smaller than 100% the improvements in running time are profound (a factor of 10 improvement). The observed (actual) precision in all cases for Figures 4.3, 4.4 is always 100%. It is only when we set the precision parameter of $pH - RA$ to 100% that the algorithm observes a performance degradation relative to Naive; as explained in the Section 4.5, TA is highly conservative and its overestimation of $maxPossibleScore$ by a factor of $M$ makes early stopping impossible. The overhead in the case of $pH - RA$ comes from additional book keeping costs. From these experiments we conclude that for the case of $pH - RA$ even a small decrease in the precision parameter can bring significant performance advantages and an observed zero loss in accuracy.

The basic premise of early stopping in any TA based approach is that correlations are present in the data. This premise however may not always be true. To complete our evaluation of the proposed algorithm, we conduct the experiments with ‘adverse data’ without any correlation between the hierarchy and the list positions. For this set of experiments, we generate the lists by starting with an initial list, and introducing a number of random swaps (same number as the list size) of consecutive elements to generate the next list. The hierarchy was generated by grouping random elements from the base lists to the same higher level term. This grouping was done so that the multiplicities follow a Normal distribution with specified mean value (varied in the graph), and variance equal to one third the mean value.

The observed precision values for different settings of the precision parameter of $pH - RA$ are shown in Figure 4.5. Observe that the Naive algorithm has precision 100%, and all others are close to this value. Figure 4.6 shows the performance of the algorithms for this experiment. For high values of the target precision parameter specified in $pH - RA$ the observations are similar as before, and the source of $pH - RA$ overheads the same. The performance of $pH - RA$ improves gracefully as the mean multiplicity decreases due to less aggressive overestimation in $maxPossibleScore$. For a low precision parameter specified in $pH - RA$, the algorithm performs well (20-30% savings in running times). Even in this case, the observed precision was much higher than 80%.
4. Persistent Chatter Discovery in Presence of Topic Hierarchies

4.7.2 Sparse Interval Set

When we use the sparse set system for preprocessing, the number of lists that we merge at query time is much less than the number otherwise required. We refer to the ratio between the two as the improvement factor (since running times are proportional to number of lists aggregated). Figure 4.7 shows the average value of the improvement factor for different values of parameter $l$ as we increase the query size. The improvement is in orders of magnitude when the query size is large. Note that, the accuracy is not affected by using the sparse set system, and hence is not reported.

Figure 4.8 shows running times for merging 100 lists each with half a million elements and using a randomly generated hierarchy having mean multiplicity six, with and without the use of the sparse set system. For the case of the sparse set system, the value of $l$ was set to 2. Query time is reduced significantly when preprocessing is used.
Figure 4.10: Merging 90 lists obtained from BlogScope, with uncorrelated hierarchy.

4.7.3 Real Data

To evaluate our proposed techniques in a real world setting, we used BlogScope to generate 90 lists, each list corresponding to a single day, for a 90 day period ending in Nov 15 2006. Each list had 50K keywords ranked according to BlogScope’s keyword popularity ranking methodology on the corresponding day in the blogosphere. Figure 4.9 and Figure 4.10 present performance results for experiments conducted using these data. For Figure 4.9 we used a ‘correlated’ hierarchy, i.e., elements mapping to the same higher level term appear near each other in the lists. Figure 4.10 displays results for the case of ‘uncorrelated’ hierarchies (keywords randomly mappings to hierarchy nodes). The observed precision was 100% in all the cases. It can be observed that the proposed algorithm with relaxed precision requirements performs better than the naive in both the cases.

In our next experiment, we explore the dependency between \( k \) and the user specified absolute value (\( z \)) of correct top-\( k \) results. This relationship provides an insight on the choice of \( k \). For a fixed value of \( z \), we run the TA algorithm (with different values of \( k \)) until the correct top-\( z \) elements are present in the current top-\( k \) (in the buffer of the TA algorithm), and record the number of elements scanned in each list. In other words we stop when top-\( k \) elements in the buffer contain all of actual top-\( z \) elements. Results are reported in Figure 4.11 for \( z = 20 \) and \( z = 40 \). When \( z = k \), the problem is the same as \( H - RA \), as precision requirement is 1.0. It can be observed that relaxing the precision even slightly (i.e., setting \( k = 30 \) when looking for top-20) results in significant performance gains. Increasing the value of \( k \) further exhibits a diminishing returns phenomenon. Empirically, a value of \( k \sim 3z \) is enough to obtain the maximum performance gains at high accuracy.

In this chapter, we propose a novel semantics of scoring ranked lists in presence of hierarchies. Since typical stopping conditions no longer apply in this setting, and relaxing precision guarantees result in performance improvements. The proposed sparse interval set system is an efficient way to reduce query times by preprocessing the lists in advance with guaranteed time/space bounds. Next chapter studies cross referencing of media sources.
Figure 4.11: Number of elements that need to be scanned in each list so that actual top-$z$ results are contained in $currTopK$ in the buffer.
Chapter 5

Cross-referencing Media Sources by Querying by Document

In light of plurality of content, there is a pressing need to cross reference information across sources. Consider for example, a news article from New York Times (either at the nytimes.com site or delivered through an RSS reader) reporting on a breaking news story or on current affairs. It is desirable to cross reference the information reported in the news source with the buzz in blogosphere and other user generated content sources. Commonly the reverse might be true, namely after reading a blog post, to check news sites to obtain additional information.

In this chapter we present a solution to this problem which we refer to as Query by Document (QBD) allowing the user to submit a text document as a query and identify related documents from another text corpus. To provide such functionality, we present techniques to process text documents on demand and extract key phrases which are used to query BlogScope for retrieving blog posts related to the query document. Extracting key phrases from a text document to be used as queries is a challenging problem. We would like such phrases to convey the ‘meaning’ of the document but at the same time distinguishing enough to capture specific events or entities of interest unique to the document. Further, our approach is domain invariant (we use data from blogs and news sources just as a convenience). At a first glance this seems like a difficult cognitive task. However, we emphasize that our main goal is to extract candidate phrases to be used as queries. It is not our intent to summarize the meaning of the document with such phrases, which is an orthogonal problem.

Figure 5.1 shows an screenshot of the user interface for QBD. After the user inputs the text, the system shows a slider bar, which can be adjusted to match the required level of relevance for querying related documents. The slider bar can be set to “Very General” meaning fetch additional documents with somewhat similar content, or “Highly Specific” meaning fetch documents that talk about exactly the same events and topics. Clicking on one of “blogs”, “news”, or “web” retrieves back matching documents from the selected domain.

Further, we show that it is possible to extend these ideas towards the development of additional query types. In particular we utilize a large collection of pages from Wikipedia to extract phrases to enhance or substitute key phrases extracted from the input text document. This version of query by document, referred to as QBD-W, possess highly novel semantics which we detail in Section 5.4. We present algorithm RelevanceRank to select candidate phrases from Wikipedia for this purpose. We validate the utility of our techniques by submitting extensive sets of results to Amazon’s Mechanical Turk (MTurk)[1] for human evaluation. This enables users at large to act as judges of the quality of our findings. We present detailed experimental results validating the applicability of our approach.
To summarize, our main contributions are

- We define the problem of querying a text corpus using another text document. Such functionality can be used for correlating multiple information sources.

- We formalize the problem of extracting relevant phrases from a document for the purpose of constructing queries to search engines (QBD). Two variants for solving this problem are proposed.

- We introduce the notion of using external knowledge sources (Wikipedia in our case) for the purpose of enhancing the set of query phrases. An algorithm for selecting relevant nodes from the Wikipedia graph is presented.

- We evaluate our techniques by employing human judges from Amazon’s MTurk.

- We implement the algorithms presented herein as part of BlogScope, enabling the querying on a document corpus with 3.25 billion posts. This functionality has been actively used by many users of the system.

The rest of the chapter is organized as follows. Section 5.1 reviews related work and provides necessary background. Section 5.2 formally defines the problems of interest. Section 5.3 details our phrase extraction and matching techniques. Algorithms for query enhancement are discussed in Section 5.4. Section 5.5 presents a comprehensive set of experiments validating the efficiency and effectiveness of the proposed techniques.

### 5.1 Related Work

**Relevance Feedback** QBD is a method that enables retrieval of documents related, precisely or more generally to a query document. *Relevance feedback* (RF) [92], is a well-studied technique to improve query relevance which involves multiple interactions between an individual and the keyword search system to iteratively refine the search results. Specifically, in the initial step, one issues a query $Q_0$ which consists of a set of keywords, and the system returns its results $R_0$, comprised of a set of documents. Often, $Q_0$ may not precisely express search
intentions, consequently, only some documents (forming a subset $R^+_0$ of $R_0$) are considered relevant, and others (denoted as $R^-_0 \subset R_0$) are not. RF requires one to identify $R^+_0$ and $R^-_0$, and provide them as feedback. The iterative query refinement process continues until no more feedback is provided. Most solutions are based on Rocchio’s formula, the basic idea of which is to modify the query by increasing (decreasing) the weights assigned to keywords which appear in relevant (irrelevant) documents, respectively; meanwhile, the query is expanded to include all terms present in relevant documents (note that negative term weights are ignored). Several variations and enhancements, most notably by Harman [55] and Ide [58], have been proposed for the Rocchio’s approach.

RF has been utilized to display a “similar pages” link along with search results [13] in a web search scenario. While this functionality bears some resemblance with QBD, there are two fundamental differences between the two problems. First in RF there is an element of interaction, progressively refining the query after examination of search results. In general it has been demonstrated, that RF is more effective in enhancing recall than precision of the results. On the other hand QDB has the ability to identify both highly specific but also generally related information utilizing keywords that are not necessarily present in the query document. Second, RF always has a query as a starting point, which progressively gets refined. In QDB the starting point in an entire document and there is no feedback process for refinement, i.e., the system has additional content at the beginning to utilize in order to specify the retrieval task.

In fact, a direct application of Rocchio’s method to QBD (i.e., by setting $Q_i$ and $R^-_i$ to empty, and $R^+_i$ to contain only the query document) reduces to answering a keyword query comprised of all terms appearing in the query document, which will retrieve back only documents completely containing the original query text. Straightforward modifications of this method, such as taking only the top few terms with highest TF/IDF weights do not yield satisfactory results either: often, the scope of the results are so narrow that they provide little more than the information already present in the query document.

**Phrase Extraction** Another area related to our work is automatic extraction of phrases from documents. Existing solutions can be roughly classified into two categories. The first mainly aims at obtaining high result accuracy. These methods typically apply statistical machine learning techniques, involving expensive computation, and often rely on the availability of a large, high-quality training set. Specifically, several approaches [110, 103] employ a supervised or semi-supervised learning framework using training data to identify suitable key phrases. The main disadvantage is that they are not suitable for general texts as they require training. Recently such techniques have been enhanced with relationships between phrases which is then incorporated in the learning framework [77]. This enhancement however requires additional training. Pantel et. al., [85] consider the term extraction problem using a corpus. They apply several tests for word association [72] utilizing prior information (from the corpus) regarding word co-occurrence counts. This is computationally intensive and requires several pruning thresholds which are difficult to set a priori in a document independent fashion. Tomokiyo et. al., [102] employ language models using a foreground and a background corpus, which after construction they combine in an additive way. This approach however, besides the requirement of multiple corpuses, is difficult to apply for phrases longer than two terms as it requires execution of an a priori style algorithm (to obtain word co-occurrence counts) which is prohibitively expensive.

The second category (e.g., [79], [107]) are practical, efficient solutions proposed in the IR community. Typically, they employ fast, heuristic methods based on various statistics collected from the input document. Syntactic information has been utilized to identify candidate terms [29, 39]. In [29] candidate syntactic constructs were grouped together; each group consisting of sets of words with the same head word. Then within each group term sequences were sorted by size. Maximal phrases within each group were subsequently identified. In [39] this idea is extended further, but this time the frequency of words in the document is considered. The technique identifies
phrases, taking into account the syntactic label of words surrounding a term; ranking however does not utilize any sources or prior statistics.

Our phrase extraction techniques fall in the second category since we seek a practical and computationally tractable approach to implement in the BlogScope system. However, our focus is not to extract a set of phrases to summarize the document, but to construct a query. Moreover, we wish to provide flexibility regarding the relevance of the results. To give an example, consider a query document talking about the wedding of French president Nicolas Sarkozy with Carla Bruni. The Ideal collection of phrases extracted for QBD would be the ordered set \{“France”, “Nicolas Sarkozy”, “Carla Bruni”\}. For this set, use of only the first phrase will bring back all documents talking about France (which is somewhat related to the query document), while use of all three phrases will search for more closely related documents. To our knowledge, none of existing methods for phrase ranking (e.g., [78]) are designed to suit this purpose. Moreover, such phrases are automatically extracted from the document at hand, providing the option to query for highly specific related content. Among a variety of choices, in our experiments we use the Yahoo Phrase Extractor [112] for the purpose of comparison. Besides providing easier repeatability of the experiments, the comparison with a leading commercial system underlines the value of the proposed solutions.

**Query Enhancement** We follow recent efforts to develop search and information discovery tools [97, 27] that greatly benefit from Wikipedia’s authoritative source of common knowledge by exploiting its graph structure, disambiguation pages, hierarchical classification, and concept co-occurrence information. One such system uses Wikipedia for entity extraction from social media conversations [41]. QBD-W, which enhances the initial phrase set with background knowledge extracted from Wikipedia, bears similarities with traditional query expansion [33], whereas the query is automatically expanded. Recent proposals, e.g., [69], have attempted incorporating the information provided by Wikipedia to strengthen query expansion techniques, with considerable success. However, the goals of query expansion and QBD-W are very different. Specifically, the underlying assumption in query expansion is that one is unable to identify the precise keywords to express one’s needs, and thus enhances the query with vague or surrounding concepts. The search engine, therefore, tries to decode the user’s true intentions from this imprecise information, and locate the correct terms to enlarge query. In contrast, QBD-W starts from phrases that best describe the query document, in the sense that they clearly express the one’s knowledge; the goal is to locate new and related concepts to enrich that knowledge. In other words, query expansion aims to improve precision [33], while QBD-W also focuses on boosting recall. RelevanceRank, the proposed solution for QBD-W, is inspired by PageRank [56], TrustRank [54] and spreading activation framework [26]. A more in-depth comparison with these works is presented later in Section 5.

### 5.2 Problem Definition

Let \( B = \{b_1, b_2, \ldots, b_n\} \) be a set of \( n \) blogs. Since a blog is a reverse chronologically ordered stream of text posts written by an individual (or a group of individuals), we model a blog \( b \) as a sequence of \( P_b \) posts. Each post \( p \in P_b \) contains two basic attributes: the textual document content \( p.d \) and the timestamp signifying the creation time of the post \( p.t.s \).

For each blog \( b \), in addition to \( P_b \), BlogScope extracts and maintains information regarding the blog creation time, profile information regarding the blogger including age, profession, gender and geographical location (at the city, state, country level), as well as the aggregate count of in-links to the blog. The in-link count, along with several other properties computed on the posts are used subsequently by ranking algorithms. BlogScope adopts the usual text based search interface to issue keyword or phrase queries with conjunctive or disjunctive semantics.
Moreover queries can be restricted temporally (e.g., only search blogs with posts between Jan 16th and 23rd 2007) with further restrictions on the blogger profile information (e.g., age between 30 and 35 and based in New Jersey, etc). All qualifying posts are searched for matches and those yielding a match are ranked using BlogScope’s ranking mechanisms and subsequently returned as answers.

A QBD query \( q \) consists of a query document \( d \), and optionally, temporal or other metadata restrictions (e.g., age, profession, geographical location) specified by the user. The specific challenge we address is the extraction of a number \( k \) (user specified) of phrases from \( d \) in order to form a query with conjunctive semantics. Ideally we would like them to be the phrases that an average user would extract from \( d \) to retrieve blog posts related to the document.

**Problem 5.2.1 (QBD)** Given a query document \( d \), extract a user specified number \( k \) of phrases to be used as input query with conjunctive semantics to BlogScope. The blog posts retrieved should be rated by an average user as related to the content of the query document.

All phrases extracted by QBD are present in the document. This functionality can be extended by taking into account external information sources. In particular Wikipedia contains a vast collection of information, in pages which exhibit high link connectivity. Consider the graph \( G_w \) extracted from Wikipedia in which each node \( v_i \) corresponds to the title of the \( i \)-th Wikipedia page and is adjacent to a set of nodes corresponding to the titles of all pages that the \( i \)-th page links to. We extracted such a graph, which we maintain up to date, currently consisting of 7M nodes. \( G_w \) encompasses rich amount of information regarding phrases and the way they are related. For example starting with the node for ‘Bill Clinton’ we get links to nodes for the ‘President of the United States’, ‘Governor of Arkansas’, and ‘Hillary Rodham Clinton’. This graph evidently provides the ability to enhance or substitute our collection of phrases extracted by QBD with phrases not present in the query document. Given the numerous outlinks from the ‘Bill Clinton’ page, it is natural to reason regarding the most suitable set of title phrases to choose from Wikipedia. Let \( v_i, v_l \) be two nodes in \( G_w \) corresponding to two phrases in the result of QBD for a document. Intuitively we would like phrases in \( G_w \) corresponding to nodes immediately adjacent to \( v_i \) and \( v_l \) to have higher chances to be selected as candidates for enhancing or substituting the result of QBD. This intuition is captured by an algorithm called RelevanceRank which we propose in Section 5.4.

The choice to enhance or substitute the results of QBD on a document with Wikipedia phrases depends on the semantics of the resulting query. For example consider a document describing an event associated with ‘Bill Clinton’, ‘Al Gore’ and the ‘Kyoto Protocol’ and that these three phrases are the result of QBD on a document. If we add the phrase ‘Global Warming’ extracted from Wikipedia (assuming that this phrase is not present in the result of QBD) we will be retrieving blog posts possibly associating ‘Global Warming’ with the event described in the query document (if any)\(^1\). As an additional example consider a document concerning a new movie released by Pixar animation studios (say Ratatouille); assume that this document does not mention any other animated movies produced by Pixar. Nodes corresponding to other animated movies produced by ‘Pixar’ would be good candidates from Wikipedia since they are pointed by both the node for ‘Pixar’ and the node for ‘Ratatouille’. By substituting (all or some) of the phrases in QBD by phrases extracted from Wikipedia, such as ‘Toy Story’ and ‘Finding Nemo’, we would be able to retrieve posts related to other movies produced by ‘Pixar’. All the above intuitions are formalized in the following problem:

**Problem 5.2.2 (QBD-W)** Given a set of phrases \( C_{qbd} \) extracted by QBD containing \( k \) phrases from \( d \), identify a number of phrases \( k' \) utilizing the result of QBD and the Wikipedia graph \( G_w \). The resulting \( k' \) phrases will

\(^1\)Incidently all three Wikipedia pages for Bill Clinton, Al Gore and the Kyoto Protocol point to the Wikipedia page for Global Warming.
Table 5.1: Example of Noun Phrase Patterns and Instances

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Nintendo</td>
</tr>
<tr>
<td>JN</td>
<td>Global warming</td>
</tr>
<tr>
<td>NN</td>
<td>Apple computer</td>
</tr>
<tr>
<td>JIN</td>
<td>Declarative approximate selection</td>
</tr>
<tr>
<td>JNN</td>
<td>Computer science department</td>
</tr>
<tr>
<td>JCIN</td>
<td>Efficient and effective algorithm</td>
</tr>
<tr>
<td>JNNN</td>
<td>Junior United States Senator</td>
</tr>
<tr>
<td>NNNN</td>
<td>Microsoft Host Integration Server</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>NNNNN</td>
<td>United States President George Bush</td>
</tr>
</tbody>
</table>

be used as input query with conjunctive semantics to BlogScope. The blog posts retrieved should be rated by an average user as related to the content of the query document.

5.3 Phrase Extraction: QBD

In this section we detail the methodology for solving the QBD problem introduced earlier. The basic workflow behind our solutions to QBD is as follows: (1) Identify the set of all candidate key phrases \( C_{all} \) for the query document \( d \). (2) Assess the significance of each candidate phrase \( c \in C_{all} \) assigning a score \( s(c) \) between 0 and 1. (3) Select the top-\( k \) (for a user specified value of \( k \)) phrases as \( C_{qbd} \) as a solution to QBD. We detail generation of candidate key phrases \( C_{qbd} \) in Section 5.3.1 and the mechanisms to score candidate key phrases in Section 5.3.2. Our solution to QBD-W and the algorithm RelevanceRank are presented in Section 5.4.

5.3.1 Extracting Candidate Phrases

We extract candidate phrases \( C_{all} \) from the query document \( d \) with the help of a part-of-speech tagger (POST) [87]. Specifically, for each term \( w \in d \), POST determines its part-of-speech (e.g., noun, verb, or adjective) by applying a pre-trained classifier on \( w \) and its surrounding terms in \( d \). For instance, in sentence “Wii is the most popular gaming console”, term “Wii” is classified as a noun, “popular” as an adjective, and so on. We represent the tagged sentence as “Wii/N is/V the/P most/A popular/J gaming/N console/N”, where “N”, “V”, “P”, “A”, and “J” signify noun, verb, article, adverb, and adjective respectively.

Based on the part-of-speech tags, we consider all noun phrases as candidate phrases, and compute \( C_{all} \) by extracting all such phrases from \( d \). A noun phrase is a sequence of terms in \( d \) whose part-of-speech tags match a noun phrase pattern (NPP). Table 5.1 lists some example NPPs used and their instances (i.e., noun phrases). To facilitate efficient identification of candidate phrases, the pattern set \( PS \) is organized as a forest of tries as illustrated in Figure 5.2. Each path (e.g., \( n_{10} - n_{11} - n_{14} \)) ending with symbol ‘$’ signifies a pattern in \( PS \) (e.g., ‘NN’).

Example 5.1 Let \( d = “Wii is the most popular gaming console” \). First we obtain the POST tags for each word in \( d \), obtaining the tagged document “Wii/N is/V the/P most/A popular/J gaming/N console/N”. The corresponding tag sequence \( d_{POS} \) is thus “NVPAJNN”. We then scan \( d_{POS} \), and match all subsequences of length at most 5 against the \( PS \) trie forest (see Figure 5.2). Continuing the example, the first tag “N” matches node \( n_{10} \); since \( n_{10} \) has ‘$’ (\( n_{13} \), signifying end-of-pattern) as a child, the corresponding term \( c_1 = “Wii” \) in \( d \) is identified as a candidate phrase. Similarly, the subsequence “JNN” and “NN” match the paths \( n_1 - n_2 - n_3 \) and \( n_{10} - n_{11} \) respectively, thus \( c_2 = “popular gaming console” \) and \( c_3 = “gaming console” \).
are extracted as candidate phrases. Note that we do not require a candidate phrase to be maximal, for instance, $c_3$ is a subsequence of $c_2$. This redundancy is eliminated later through scoring and top-$k$ selection of candidate phrases, detailed next.

### 5.3.2 Scoring Candidate Phrases

Once all candidate phrases are identified as $C_{all}$, a scoring function $f$ is applied to each phrase $c \in C_{all}$. The scoring function assigns a score to $c$ based on the properties of $c$, taking into account both the input document, and the background statistics about terms in $c$ from the BlogScope corpus. The candidate phrases are revised in a pruning step to ensure that no redundant phrases are present. We propose two scoring mechanisms, $f_t$ and $f_i$ for this purpose. $f_t$ utilizes the TF/IDF information of terms in $c$ to assign a score, while $f_i$ computes the score based on the mutual information of the terms in phrase $c$. Both ranking mechanisms share the same pruning module to eliminate redundancy in the final result $C_{qbd}$. We provide examples and experimental comparison of these two approaches later in this chapter.

**TF/IDF Based Scoring**

We first describe $f_t$, which is a linear combination of the total TF/IDF score of all terms in $c$ and the degree of coherence of $c$. Coherence quantifies the likelihood these terms have in forming a single concept. Formally, let $|c|$ be the number of terms in $c$; we use $w_1, w_2, \ldots, w_{|c|}$ to denote the actual terms. Let $idf(w_i)$ be the inverse document frequency of $w_i$ as computed over all posts in BlogScope’s corpus. $f_t$ is defined as

$$f_t(c) = \sum_{i=1}^{|c|} tfidf(w_i) + \alpha \cdot \text{coherence}(c) \tag{5.3.1}$$

where $\alpha$ is a tunable parameter.

The first term in $f_t$ aggregates the importance of each term in $c$. A rare term that occurs frequently in $d$ is more important than a common term frequently appearing in $d$ (with low $idf$, e.g., here, when, or hello). This importance is nicely captured by $tfidf$ for the term [93]². We use the total, rather than average $tfidf$ to favor phrases that are relatively long, and usually more descriptive.

²For implementation purposes we use $tfidf(w) = tf(w) \cdot idf(w)$ which is widely used in practice, most notably by the Apache Lucene’s Similarity class.
The second term in $f_t$ captures how coherent the phrase $c$ is. Let $tf(c)$ be the number of times $c$ appears in the document $d$, the coherence of $c$ is defined as

$$coherence(c) = \frac{tf(c) \times (1 + \log tf(c))}{1 + \sum_{i=1}^{|c|} tf(w_i)}$$  \hspace{1cm} (5.3.2)$$

Intuitively, Equation 5.3.2 compares the frequency of $c$ (the numerator) against the average TF of its terms (the denominator). The additional logarithmic term strengthens the numerator, preferring phrases appearing frequently in the input document. For example, consider the text fragment “... at this moment Dow Jones ...”. Since the phrase “moment Dow Jones” matches the pattern “NNN”, it is included in $C_{all}$. However, it is just a coincidence that the three nouns appear adjacent, and “moment Dow Jones” is not a commonly occurring phrase as such. The coherence of this phrase is therefore low (compared to the phrase “Dow Jones”), since the $tf$ of the phrase is divided with the average $tf$ of terms constituting it. This prohibits “moment Dow Jones” to appear high in the overall $f_t$ ranking.

Based on TF/IDF scoring, $f_t$ is good at distinguishing phrases that are characteristic of the input document. In the running example $d = “Wii is the most popular gaming console”$, $f_t$ strongly favors “Wii” over “gaming console” since the former is a much rarer term and thus has a much higher idf score. However, $f_t$ also has the drawback that it is often biased towards rare phrases.

**Mutual Information Based Scoring**

$f_t$ uses mutual information (MI) between the terms of $c$ as a measure of coherence in the phrase $c$ along with idf values from the background corpus. Mutual information is widely used in information theory to measure the dependence of random variables. Specifically, the pointwise mutual information of a pair of outcomes $x$ and $y$ belonging to discrete random variables $X$ and $Y$ is defined as $[22]$

$$PMI(x, y) = \log \left( \frac{prob(x \cap y)}{prob(x)prob(y)} \right)$$  \hspace{1cm} (5.3.3)$$

where $prob(x)$, $prob(y)$, $prob(x \cap y)$ are the probability of $x$, $y$ and the combination of the two respectively. Intuitively, for a phrase $c$ consisting of terms $w_1, w_2, ..., w_{|c|}$, the higher the mutual information among the terms, the higher the chances of the terms appearing frequently together; and thus they are more likely to be combined to form a phrase.

The scoring function $f_t$ takes a linear combination of idf values of terms in $c$, frequency of $c$, and the pointwise mutual information among them. Let $tf(c)$ and $tf(POS_c)$ be the number of times $c$ and its part-of-speech tag sequence $POS_c$ appear in $d$ and $POS_d$ respectively, then

$$f_t(c) = \sum_{i=1}^{|c|} idf(w_i) + \log \frac{tf(c)}{tf(POS_c)} + PMI(c)$$  \hspace{1cm} (5.3.4)$$

The first part in the equation above represents how rare or descriptive each of the terms in $c$ is. The second part denotes how frequent the phrase $c$ is at the corresponding POS tag sequence in the document. The third part captures how likely are the terms to appear together in a phrase.

The $PMI(c)$ for a phrase $c$ is

$$PMI(c) = \log \left( \frac{prob(c)}{\prod_{i=1}^{|c|} prob(w_i)} \right)$$

PMI can be evaluated either at the query document itself or at the background corpus. Computation of these probabilities for the background corpus requires a scan of all documents, which is prohibitively expensive$^3$. In

$^3$If statistics regarding the co-occurrence in the text of $n$ ($n \geq 2$) terms is available one can obtain more precise correlation information for
order to compute PMI using $d$ only, let $\text{prob}(w_i)$ and $\text{prob}(c)$ denote the probability of occurrence of $w_i$ and $c$ respectively at the appropriate part-of-speech tag sequence.

$$\text{prob}(c) = \frac{\text{tf}(c)}{\text{tf}(\text{POS}_c)}, \text{ and } \text{prob}(w_i) = \frac{\text{tf}(w_i)}{\text{tf}(\text{POS}_{w_i})}.$$ 

Substituting these probabilities in Equation 5.3.4,

$$f_1(c) = \sum_{i=1}^{d} \text{idf}(w_i) + \log \frac{\text{tf}(c)}{\text{tf}(\text{POS}_c)} + \log \left( \frac{\text{tf}(c)}{\prod_{i=1}^{d} \text{tf}(\text{POS}_{w_i})} \right)$$

(5.3.5)

The scoring function as defined in Equation 5.3.5 identifies how rare or descriptive each term is and how likely these terms are to form a phrase together. This definition however does not stress adequately the importance of how frequent the phrase is in document $d$; therefore we weight it by $\frac{\text{tf}(c)}{\text{tf}(\text{POS}_c)}$ before computing the final score $f_1$. The scoring function $f_1$ therefore is,

$$f_1(c) = \frac{\text{tf}(c)}{\text{tf}(\text{POS}_c)} \times \left( \sum_{i=1}^{d} \text{idf}(w_i) + \log \frac{\text{tf}(c)}{\text{tf}(\text{POS}_c)} + \log \left( \frac{\text{tf}(c)}{\prod_{i=1}^{d} \text{tf}(\text{POS}_{w_i})} \right) \right)$$

(5.3.6)

The tf values in the above equations are computed by scanning the document $d$ once, while the idf values are maintained precomputed for the corpus.

The scoring function ($f_1$ or $f_2$) evaluates each phrase $c \in C_{all}$ individually. As a result, candidate phrases may contain redundancy. For example, a ranking function may judge that both $c_1 =$ “gaming console” and $c_2 =$ “popular gaming console” as candidate phrases. Since $c_1$ and $c_2$ refer to the same entity, intuitively only one should appear in the final list $C_{qbd}$. We therefore apply a post-processing step after evaluating the ranking function on elements of $C_{all}$. Methodology for computing $C_{qbd}$ is shown in Algorithm 5. Lines 7-14 demonstrate the pruning routine after evaluating the ranking function. Specifically, a phrase $c$ is pruned when there exists another phrase $c' \in C_{qbd}$ such that (i) $c'$ has a higher score than $c$, and (ii) $c'$ is considered redundant in presence of $c$. The function Redundant evaluates whether one of the two phrases $c_1, c_2$ is unnecessary by comparing them literally.

Note that sometimes the shorter phrase may be more relevant, so we should not simply identify longer phrases. For instance, the phrase “drug” may have higher score than a longer phrase “tuberculosis drugs” in a document that talks about drugs in general, and tuberculosis drugs is one of the many different phrases where the term “drug” appears. Also, the candidate set $C_{all}$ may contain phrases with common suffix or prefix, e.g., “drug resistance”, “drug facility” and “drug needs”, in which case we keep only the top few highest scoring phrases to eliminate redundancy. Redundant returns true if and only if either one phrase subsumes the other, or multiple elements in $C_{qbd}$ share common prefix/suffix.

5.4 Using Wikipedia: QBD-W

We have constructed a directed graph $G_w = < V, E >$ by preprocessing a snapshot of Wikipedia, modeling all pages with the vertex set $V$ and the hyperlinks between them with the edge set $E$. Specifically, a phrase sets of terms. However this comes with a high computation, storage and maintenance cost as the number of all such combinations of keywords is very high. In BlogScope we maintain precomputed idf values for single keywords only, and statistics for combinations of keywords is not materialized.


**Algorithm 5 Algorithm for QBD**

**INPUT** document $d$, and required number of phrases $k$

1. Run a POS tagger to obtain the tag sequence $POS_d$ for $d$
2. Initialize $C_{all}$ and $C_{qbd}$ to empty
3. Match $POS_d$ against the PS Trie forest
4. For each subsequent $POS_c \subseteq POS_d$ that matches a NPP, append the corresponding term sequence to $C_{all}$
5. for each $c \in C_{all}$ do
6. Compute the score $s_c$, using either of $f_1$ or $f_2$
7. if NOT exists $c' \in C_{qbd}$ such that $(Redundant(c, c') = true$ and $s_{c'} > s_c$) then
8. Add $c$ to $C_{qbd}$
9. end if
10. for each $c' \in C_{qbd}$ do
11. if $Redundant(c, c')$ and $s_{c'} < s_c$ then
12. Remove $c'$ from $C_{qbd}$
13. end if
14. end for
15. If $|C_{qbd}| > k$, remove the entry with minimum score
16. end for
17. **OUTPUT** $C_{qbd}$

---

$c$ is extracted for each page $p_c$ in Wikipedia as the title of the page. Each such phrase is associated with a vertex in $V$. Hyperlinks between pages in Wikipedia translate to edges in the graph $G_w$. For example, the description page for “Wii” starts with the following sentence: “The Wii is the fifth home video game console released by Nintendo”, which contains hyperlinks (underlined) to the description pages of “video game console” and “Nintendo” respectively. Intuitively, when the Wikipedia page $p_1$ links to another page $p_2$, the underlying phrases $c$ and $c'$ are related. Consider two pages $p_{c_1}$ and $p_{c_2}$ both linking to $p_{c'}$. If the number of links from $p_{c_1}$ to $p_{c'}$ is larger than the number of links from $p_{c_2}$ to $p_{c'}$, we expect $c_1$ to have a stronger relationship with $c'$. This can be easily validated by observing the Wikipedia data.

Formally, the Wikipedia graph $G_w$ is constructed as follows: a vertex $v_c$ is created for each phrase $c$ which is the title of the page $p_c$. A directed edge $e = \langle v_c, v_{c'} \rangle$ is generated if there exists a hyperlink in $p_c$ pointing to $p_{c'}$. A numerical weight $wt_e$ is assigned to the edge $e = \langle v_c, v_{c'} \rangle$ with value equal to the number of hyperlinks from $p_c$ pointing to $p_{c'}$. We refer to the weight of the edge between two vertices in graph $G_w$ as their **affinity**.

**Example 5.2** Figure 5.3 depicts the interconnection between phrases $c_1 = “Wii”, c_2 = “Nintendo”, c_3 = “Sony”, c_4 = “Play Station”, and c_5 = “Tomb Raider”, in the Wikipedia graph. The number beside each edge signifies its weight, e.g., $wt_{c_1, c_2} = 7$ implying that there are 7 links from the description page of “Wii” to that of “Nintendo”. Node $c_2$ is connected to both $c_1$ and $c_5$, signifying that “Nintendo” has affinity with both “Wii” and “Sony”. Edge $c_2 < c_1$, $c_1 >$ has a much higher weight than $c_2 < c_3$, $c_3 >$, signifying that the affinity between “Nintendo” and “Wii” is stronger than that between “Nintendo” and “Sony” (the manufacturer of Play Station 3, a competitor of Wii). Therefore, if “Nintendo” is an important phrase mentioned in the input document $d$, i.e., $c_2 \in C_{qbd}$, it is much more likely that $c_1$ (rather than $c_3$) is closely relevant to $d$, and thus should be included in the enhanced phrase set after QBD-W.

Once $G_w$ is ready and the set $C_{qbd}$ is identified, it can be enhanced using the Wikipedia graph according to the following procedure: (1) Use $C_{qbd}$ to identify a seed set of phrases in the Wikipedia graph $G_w$. (2) Assign an initial score to all nodes in $G_w$. (3) Run the algorithm **RelevanceRank** as described in Algorithm 6 to iteratively assign a relevance score to each node in $G_w$. The **RelevanceRank** algorithm is an iterative procedure in the same spirit as biased PageRank [56] and TrustRank [54]. (4) Select the top-$k'$ highest scoring nodes from $G_w$ (for user specified value of $k'$) as top phrases $C_{wiki}$. 

---

**References**

Algorithm 6 Algorithm to compute RelevanceRank

INPUT Graph $G_w = \langle V, E \rangle$, QBD phrases $C_{qbd}$, $k'$

RelevanceRank
1: Initialize the seed set to empty set
2: for each $c \in C_{qbd}$ do
3: Compute node $v \in V$ with smallest edit distance to $c$
4: If $\text{edit\_distance}(c, v) < \theta$, add $v$ to $S$
5: end for
6: for each $v \in V$ do
7: Assign initial score to $v$ based on Equation 5.4.1
8: end for
9: for $i = 1$ to MaxIterations do
10: Update scores for each $v \in V$ using Equation 5.4.3
11: If convergence, i.e., $RR_i = RR_{i-1}$, break the for loop
12: end for
13: Construct $C_{wiki}$ as the set of top-$k'$ vertices with highest $RR$ scores

The RelevanceRank algorithm starts (Lines 1-5) by computing the seed set $S$ containing the best matches of phrases in $C_{qbd}$. To find best matches, for each phrase $c \in C_{qbd}$, an exact string match over all nodes in $G_w$ is conducted to identify the node matching $c$ exactly. If no such node exists an approximate match is conducted. We deploy edit distance based similarity [68] for our experiments, but other approximate match techniques can also be used [19, 47]. It is possible that a phrase $c \in C_{qbd}$ is not described by any Wikipedia page. A threshold $\theta$ on maximum edit distance is therefore used. The matching phrase $c' \in G_w$ is added to the seed $S$ only if the edit distance between $c'$ and $c$ is below $\theta$.

After generating $S$, RelevanceRank initializes the ranking score $RR_0^i$ of each vertex $v \in V$ (Lines 6-8). Let $c_v$ be the phrase in the seed set corresponding to the vertex $v$. Let $s(c_v)$ be the score assigned to it by one of the two scoring functions ($f_t$ or $f_l$) described in the previous section. $RR_0^i$ is defined by

$$RR_0^i(v) = \begin{cases} 
\frac{s(c_v)}{\sum_{v' \in S} s(c_{v'})} & \text{if } v \in S \\
0 & \text{otherwise}
\end{cases} \quad (5.4.1)$$

This initializes the scores of all vertices not in the seed set to zero. Scores of vertices in the seed set the normalized to lie in $[0, 1]$ such that the sum is 1.

Next RelevanceRank iterates (Lines 9-12) until convergence or reaching a maximum number of iterations MaxIterations. The $i$th iteration computes $RR^i$ based on the results of $RR^{i-1}$ following the spreading activation...
framework [26]. Specifically, the transition matrix \( T \) is defined as

\[
T[v, v'] = \begin{cases} \frac{w_{e}}{\sum_{e'=(v, w)} w_{e'}} & \text{if } \exists e = <v, v'> \in E \\ 0 & \text{otherwise} \end{cases}
\]

(5.4.2)

The entry \( T[v, v'] \) represents the fraction of out-links from the page corresponding to \( v \) in Wikipedia that point to the page associated with \( v' \). Observe that each entry in \( T \) is in range \([0, 1]\) and the sum of all entries in a row is 1. Conceptually \( T \) captures the way a vertex \( v \) passes its affinity to its neighbors, so that when \( v \) is relevant, it is likely that a neighboring phrase \( v' \) with high affinity to \( v \) is also relevant, though to a lesser degree.

**Example 5.3** The transition matrix for vertices in Figure 5.3 is displayed in Table 5.2.

To model the fact that a phrase connected to nodes from \( C_{qbd} \) through many intermediate nodes is only remotely related, the propagation of \( RR \) is dampened as follows: with probability \( \alpha_v \), \( v \) passes its \( RR \) score to its successors, and with probability \( (1 - \alpha_v) \) to one of the seed vertices \( S \). Formally \( RR_i^v \), in the \( i \)th iteration is computed by

\[
RR_i^v = \sum_{e' = <v, v'>} \alpha_{v'} \cdot RR_{i-1}^{v'} \cdot T[v', v] + RR_0^v \sum_{v' \in V} (1 - \alpha_{v'}) RR_{i-1}^{v'}
\]

(5.4.3)

The first term in the equation represents propagation of \( RR \) scores via incoming links to \( v \). The second term accounts for transfer of \( RR \) scores to seed nodes with probability \( 1 - \alpha_v \). Recall that \( RR_0^v \) is zero for phrases not in the seed set, and thus the second term in the equation above is zero for \( v \not\in S \).

The RelevanceRank algorithm can be alternatively explained in terms of the random surfer model. In the Wikipedia graph \( G_w \), first the seed nodes are identified by using the result \( C_{qbd} \) of QBD. Each of these seed nodes is assigned an initial score using a scoring function \( (f_t \text{ or } f_l) \). All other nodes are assigned score zero. The surfer starts from one of the seed nodes. When at node \( v \), the surfer decides to continue forward, selecting a neighboring node \( v' \) with probability \( \alpha_{v'} \cdot T[v, v'] \). With probability \( 1 - \alpha_v \), the surfer picks a node at random from the initial seed set. The probability of selection of the node from the seed set is proportional to the initial \( RR_0 \) scores of the nodes in \( S \). At convergence, \( RR \) score of a node is the same as the probability of finding the random surfer there.

In RelevanceRank, with probability \( 1 - \alpha_v \), the random surfer jumps back to nodes in the seed set only and not to any node in \( G_w \). This is in similar spirit as the topic-sensitive PageRank and TrustRank algorithms [56, 54], which use a global constant value \( \alpha_v = \alpha \) for all \( v \in G_w \) for returning back to one of the seed nodes. Selection of a constant \( \alpha \) is however not suitable for RelevanceRank for the following two reasons:

- The RelevanceRank scoring function must prefer nodes that are close to the initial seed set. In TrustRank, existence of a path between two nodes suffices for propagation of trust (as stationary state probabilities are

<table>
<thead>
<tr>
<th></th>
<th>Wii</th>
<th>Sony</th>
<th>Nintendo</th>
<th>Play Station</th>
<th>Tomb Raider</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wii</td>
<td>0</td>
<td>2/10</td>
<td>7/10</td>
<td>1/10</td>
<td>0</td>
</tr>
<tr>
<td>Sony</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4/4</td>
<td>0</td>
</tr>
<tr>
<td>Nintendo</td>
<td>5/6</td>
<td>1/6</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Play Station</td>
<td>2/11</td>
<td>6/11</td>
<td>1/11</td>
<td>0</td>
<td>2/11</td>
</tr>
<tr>
<td>Tomb Raider</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1/1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 5.2: Transition matrix for graph in Figure 5.3
probability values after the surfer makes infinitely many jumps). The same holds true for PageRank as well, where existence of a path is sufficient for propagation of authority. For the case of RelevanceRank however, the length of the path is an important consideration. Propagation of RR scores over long paths needs to be penalized. Only nodes in the vicinity of seed nodes are relevant to the query document. The value of \( \alpha_v \) therefore must depend on the distance of a node from the seed set.

- \( G_w \) consists of over 7 million nodes. Execution of the iterative algorithm to compute RR scores over the entire graph for every query is not feasible. Unlike TrustRank or PageRank, where one-time offline computation is sufficient, RelevanceRank needs to be evaluated on a per-query basis. Since only nodes close to the seed set are relevant, we set \( \alpha_v \) to zero for vertices \( v \in V \) far from the seed set \( S \). Let \( l_{max} \) be the maximum permissible length of path from a node to \( S \). Define the graph distance \( GD(v) \) of a node \( v \) as its distance from the closest node in the seed set. Formally,

\[
GD(v) = \min_{v' \in S} \text{distance}(v', v)
\]

where \( \text{distance} \) represents the length of the shortest path between two nodes. Thus, if \( GD(v) \geq l_{max} \) for some \( v \in V \), \( \alpha_v \) is assigned value 0. Application of this restriction on \( \alpha_v \) allows us to chop off all nodes from \( G_w \) that are at distance greater than \( l_{max} \) from \( S \), which significantly reduces the size of the graph we need to run the RelevanceRank algorithm on. As the value of \( l_{max} \) increases, the size of sub-graph over which RelevanceRank is to be computed increases, leading to higher running times.

We implemented the version of RelevanceRank with a constant value of \( \alpha \) for the purpose of experimentation. Apart from high computational overhead, we discovered that this implementation always returned irrelevant set of phrases belonging to a densely connected clique far away from starting seed set. For example, query starting from any document related to George Bush returned protein names as result of QBD-W, since some research on proteins was conducted during Mr Bush’s presidency and proteins form a highly dense subgraph in the Wikipedia graph. This is expected, since the stationary probabilities in the random surfer model used for PageRank is very high for nodes in such cliques independent of the starting node. Hence, we experimentally verified that the use of either of TrustRank or topic sensitive PageRank for this problem is not suitable.

For the above mentioned reasons, \( \alpha_v \) for a node \( v \) is defined as a function of its graph distance \( GD(v) \). We would like \( \alpha_v \) to decrease as \( GD(v) \) increases such that \( \alpha_v = 0 \) if \( GD(v) \geq l_{max} \). We define \( \alpha_v \) as

\[
\alpha_v = \max \left( 0, \alpha_{max} - \frac{GD(v)}{l_{max}} \right)
\]

for some constant \( \alpha_{max} \in [0, 1] \).

When the iterative algorithm for computation of RelevanceRank finishes, each node is assigned an RR score. The process is guaranteed to converge to a unique solution, as the algorithm is essentially the same as that of computing stationary state probabilities for an irreducible Markov chain with positive-recurrent states only [37]. These nodes, and thus corresponding phrases, are sorted according to the RR scores, and top-\( k' \) (for a user-defined value of \( k' \)) are selected as the enhanced phrase set \( C_{wiki} \). The new set \( C_{wiki} \) may contain additional phrases that are not present in \( C_{qdb} \). Also, phrases from \( C_{qdb} \) included in \( C_{wiki} \) may have been re-ranked, that is the order of phrases in \( C_{qdb} \) appearing in \( C_{wiki} \) may be different than the corresponding order these phrases have in \( C_{qdb} \). This means, even for \( k' \leq k \), the set \( C_{wiki} \) can be very different from \( C_{qdb} \) depending on the information present in Wikipedia.

**Example 5.4** Consider the graph in Figure 5.3. Assume that the seed set consists of only one node “Nintendo”. Let \( \alpha_{max} = 0.8 \) and \( l_{max} = 2 \). Then, initial score for Nintendo will be 1, \( RR_{Nintendo}^{0} = 1 \); and for Sony, Wii and Play Station, the
initial score will be zero. Also, \( \alpha_{\text{Nintendo}} = 0.8 \), \( \alpha_{\text{Sony}} = 0.3 \), \( \alpha_{\text{Wii}} = 0.3 \), \( \alpha_{\text{Play Station}} = 0 \), and \( \alpha_{\text{Tomb Raider}} = 0 \). Note that, the random surfer can never reach the node “Tomb Raider” in this setting since the surfer must jump back to “Nintendo” when he reaches the node “Play Station”. Hence we can simply remove all nodes, including “Tomb Raider”, with graph distance greater than 2 for calculating RR scores. The transition matrix is presented in Table 5.2. Only the first four rows and columns of the transition matrix are relevant. RelevanceRank scores after few iterations will be as displayed in Table 5.3. At convergence, “Nintendo” has the highest RR score 0.52, with “Wii” at the second position. Scores for “Sony” and “Play Station” are low as expected.

<table>
<thead>
<tr>
<th>iterations</th>
<th>Wii</th>
<th>Sony</th>
<th>Nintendo</th>
<th>Play Station</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.67</td>
<td>0.13</td>
<td>0.20</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0.13</td>
<td>0.06</td>
<td>0.74</td>
<td>0.06</td>
</tr>
<tr>
<td>2</td>
<td>0.49</td>
<td>0.11</td>
<td>0.38</td>
<td>0.02</td>
</tr>
<tr>
<td>3</td>
<td>0.25</td>
<td>0.08</td>
<td>0.62</td>
<td>0.05</td>
</tr>
<tr>
<td>4</td>
<td>0.41</td>
<td>0.10</td>
<td>0.46</td>
<td>0.03</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>infinite</td>
<td>0.35</td>
<td>0.09</td>
<td>0.52</td>
<td>0.03</td>
</tr>
</tbody>
</table>

Table 5.3: RelevanceRank scores after 1-5 iterations and at convergence

**Example 5.5** Consider the news article titled “U.S. Health Insurers Aim to Shape Reform Process” taken from Reuters.

Top 5 phrases in QBD for this article consists of “america’s health care system”, “ahip’s ignani”, “special interests”, “tax credits” and “poorer americans”. While these phrases do relate to the meaning of the document, they do not necessarily constitute the best fit for describing it. The result of running QBD-W with the same value of \( k' = k = 5 \) results in “american health care”, “ahip”, “universal health care”, “united states” and “poore brothers”. Arguably, the latter articulates the theme of the document in a much better way. Enhancement using wikipedia graph has replaced and re-ranked most items from the seed set consisting of 5 initial terms. For example, the phrase “AHIP’s Ignani” that appears thrice in the document, and which refers to the CEO Karan Ignani of America’s Health Insurance Plans, has been replaced with just AHIP. Also, “america’s health care system” is re-written as “american health care” (due to use approximate string matching) which is the title of a page in Wikipedia.

## 5.5 Experiments

We now present the experimental evaluation of our techniques. Section 5.5.1 discusses the data sets utilized for our experiments. Section 5.5.2 describes our use of Amazon Mechanical Turk to evaluate the quality of the phrase extraction techniques. Then, Section 5.5.3 presents our experiments for measuring the retrieval quality under our query-by-document scenario, demonstrating that our techniques significantly outperform existing, strong baselines.

### 5.5.1 Data Sets and Alternative Techniques

**Query Documents:** We extracted and present results for a random sample of 34 news articles from the New York Times, The Economist, Reuters, and Financial Times published during Aug - Sept, 2007. We refer to this set of documents as NYTS. These are the documents that we use as queries to locate related blog postings on BlogScope. We evaluated several collections of data sets and we present NYTS as representative of our results.

\[\text{http://www.reuters.com/article/domesticNews/idUSN2024291720070720}\]
Techniques: We present two algorithms, QDB employing $f_t$ scoring (referred to \textit{QBD-TFIDF} in our experiments) and QDB employing $f_l$ scoring (referred to \textit{QBD-MI}) to extract query phrases from a document. These techniques extract a ranked list of the top-$k$ phrases from a document. In our experiments, we present results for varying values of $k$. Similarly, we experimented with the extraction technique from Section 5.4, which identifies important terms that do not appear in the document, by analyzing the graph of Wikipedia; we refer to this technique as \textit{QBD-W}. As a strong baseline, we use the “Yahoo Term Extraction” (\textit{QBD-YAHOO}) service [112], which takes as input a text document and returns a list of significant words or phrases extracted from the document.

5.5.2 Quality of Phrase Extraction

Our first step is to examine the quality of the phrases obtained by the various techniques. For this, we run two experiments, using human annotators utilizing Amazon’s Mechanical Turk service.

Annotator-Nominated Relevant Phrases: We wish to examine whether phrases identified by human annotators as important, are also identified as important from our techniques. To avoid any bias, we launched a large scale user study using the Amazon Mechanical Turk service [1]. This service offers access to a community of human subjects and tools to distribute small tasks that require human intelligence. In our study, each Mechanical Turk annotator was presented with a document, and had to list up to 10 phrases that are characteristic of the document; in particular we asked the annotators to identify phrases that they would “use as queries to locate this document on Google.” To ensure the quality of this distributed labeling effort, we asked five distinct annotators to extract phrases from each document. Then, for each phrase identified, we measure how many users agreed that a given phrase is relevant, to compute the level of agreement across annotators for each phrase. Since our labelers might use slightly different wording to refer to the same phrase (e.g., phrases “crandon swat team” and “crandon swat” refer to the same entry), we use a string similarity technique [19] in conjunction with our own manual inspection, to group similar phrases together. Annotators did not have access to our techniques and had no knowledge of the output of our algorithms on the same documents for this experiment to avoid any bias.

We compute the precision of each technique, against the pool of annotations produced by Mechanical Turk annotators. We define the precision at agreement level $l$ for a technique $\mathcal{T}$, as $\text{precision} = h/k$, where $h$ is the number of phrases identified by at least $l$ humans \textit{and} by technique $\mathcal{T}$ (i.e., the number of common phrases between humans and $\mathcal{T}$), and $k$ is the total number of phrases extracted by technique $\mathcal{T}$. Figure 5.4 presents the precision of our techniques, for $k = 2$ for different levels of agreement. (The results are similar, qualitatively, for other values of $k$.) We observe that \textit{QBD-MI} outperforms \textit{QBD-TFIDF} and \textit{QBD-YAHOO} across all agreement levels. This indicates that there exists good agreement between the output of our techniques and the expectations of humans.

![Figure 5.4: Precision against the annotator-nominated relevant phrases.](image-url)
Automatically Extracted Relevant Phrases: The previous experiment indicated that the phrases extracted by our techniques agree with the phrases that are independently extracted by humans. Our techniques, however, extracts a larger number of phrases that were not listed by the annotators. In this experiment, we expose the phrases to the annotators asking them the question: are these phrases relevant or non relevant to the contents of the document. When the annotators look at the automatically extracted phrases, they can easily determine whether a particular phrase accurately depicts the contents of the document. So, to examine the accuracy of our techniques, we used each of QBD-MI, QBD-TFIDF, and QBD-YAHOO, and extracted from each document the set of top-k phrases \((k = 20)\) for each technique. This resulted in a pool of a maximum of 60 phrases per document (usually the number was lower, as often the three techniques extracted similar sets of phrases). Each phrase in each pool was labeled by five annotators. At the end of the process, we associated each phrase with an agreement level, counting how many annotators marked the phrase as relevant.

Figure 5.5 presents the precision@k of our techniques, for varying values of \(k\) and for level of agreement 3 and above. We observed that QBD-MI outperforms the other techniques for \(k < 4\), while the techniques tend to perform equally for larger values of \(k\). Restricting our attention to a few phrases only, brings out differences among the techniques considered. For a large number of phrases the techniques appear similar in terms of characterizing the content of the document. Our primary goal however is to utilize such phrases to identify documents (in our case blog posts) related to the query document. We report on such experiments next.

5.5.3 Quality of Document Retrieval

Quality of Querying-by-Document: We deploy our techniques to generate queries from the query document to be used as query phrases in order to retrieve related documents from BlogScope. To measure the quality of the various query-by-document techniques, for each “query-document” in NYTS, we extracted top-k phrases (for varying values of \(k\)) and then submitted these phrases as queries to the BlogScope search engine. We then retrieved the top-5 matching documents for these queries, and generated a pool of retrieved documents for each query-document and each value of \(k\). The query-document and the pool of retrieved documents were submitted to Amazon Mechanical Turk, where we asked 5 annotators to examine individually each retrieved document and decide whether it is relevant to the query-document.

Figure 5.6 illustrates the results. One clear trend is that QBD-MI and QBD-TFIDF systematically outperform QBD-YAHOO for different annotator agreement levels and for different values of \(k\) (the number of phrases used as seed queries to BlogScope). This indicates that QBD-MI and QBD-TFIDF are able to identify documents (blog posts) more relevant to the query document. To understand why QBD-YAHOO performs worse than the other techniques, we manually inspected the queries and the corresponding retrieved documents. The QBD-YAHOO
method tends to generate queries (phrases) that capture the general topical area of the document, but are not specific enough to retrieve documents that will be deemed relevant by the users. In fact, the extracted terms could be used as general-purpose tags for summarizing and classifying the document in a broad classification taxonomy but are relatively inadequate when used as queries due to their generality. Comparing QBD-MI and QBD-TFIDF, we can see that QBD-MI performs better for lower levels of annotator agreement, but this trend reverses when we consider only documents for which 3 and more annotators agree. This result indicates that QBD-MI is better in retrieval environments where users are looking for a diversity of results in the returned matches, while QBD-TFIDF is better suited for environments where the goal is to present results that are commonly accepted as being related to the topic of the query, ignoring potentially less common interpretations of the query-document topics.

We observed that extracting more than $k = 5$ phrases from a query document to utilize as queries to BlogScope was never required for the case of NYTS set. In fact it was not possible to retrieve related blog posts when queries consisted of more than five phrases (the returned result was empty; evidently no article in our NYTS collection appeared verbatim in the Blogosphere).

**Quality of Querying-by-Document-Wikipedia**: Finally, we run a set of experiments using QBD-W, that utilizes the Wikipedia expansion technique, described in Section 5.4, to identify useful query terms for our query-by-document approach. We conducted an experiment using Mechanical Turk similar to that for assessing the quality of QBD asking annotators to characterize the resulting documents (blog posts) as relevant or not to the query document. Figure 5.7 reports obtained precision for different agreement levels as the number of phrases $k'$ used for retrieving the documents is varied. The type of query results produced by this technique tend to be distinct and different than the results returned by QBD-MI, QBD-TFIDF, and QBD-YAHOO. In particular, the fact that QBD-W generates query terms that do not appear in the document makes the returned results more “serendipitous”; the returned documents capture a more implicit notion of relevance, and users tend to prefer the returned results. In fact, the results of QBD-W consistently outperform those of QBD-MI, QBD-TFIDF, and QBD-YAHOO across all values of $k$ and for all different annotator agreement levels. This higher user satisfaction is a direct result of the “novelty” of the resulted documents compared to the initial query-document. The results of QBD-MI, QBD-TFIDF (and in a lesser degree QBD-YAHOO) tend to contain documents that discuss the same topic as the query-document, and tend to repeat the same points of view. In contrast, the results by QBD-W tend to highlight some other, unexpected aspect of the same topic.

![Table 5.4: QBD-W run times for different $l_{max}$](image)

### Performance:
The runtime overheads of our techniques are modest. On a 2.4GHz AMD Opteron processor, both QBD-TFIDF and QBD-MI require under 300 msecs, of which bulk of the computation time is spent in part-of-speech tagging the document. Access to QBD-YAHOO web service takes close to 330 msecs including network latency. Run times for running RelevanceRank algorithm on the wikipedia graph for a typical document as $l_{max}$ is varied are displayed in Table 5.4. Recall that, for the purpose of computing RelevanceRank scores, parts of $G_w$ at distance greater than $l_{max}$ from the seed set can be ignored. As $l_{max}$ increases, the size of subgraph over which RelevanceRank needs to be computed increases drastically (due to heavy interlinking activity in wikipedia), leading to higher running times. Setting $l_{max} = 2$ or $l_{max} = 3$ works well in practice. Experiments reported in the previous subsection used $l_{max} = 3$. 
We have presented techniques to extract candidate phrases from a document in order to utilize them as seed queries to search engines and in particular to BlogScope as search engine for blogs. We have presented an evaluation using Amazon’s MTurk service demonstrating that our retrieval results are of high quality as judged by independent annotators. We believe that the problem of cross referencing documents from different sources is going to become highly significant as the information produced online by services and individuals continues to grow. These features are implemented in BlogScope to enable querying across all the billions of blog posts indexed by the system.

In the next chapter, we focus on the contributors as opposed to just the content and explore ways to study their topical expertise.
Figure 5.6: Retrieval precision@k. The x-axis shows the number of phrases, $k$, used to retrieve documents which were then submitted to MTurk for human evaluation.

Figure 5.7: Retrieval precision@k when using Wikipedia. The x-axis shows the number of phrases, $k$, used to retrieve documents which were then submitted to MTurk for human evaluation.
Chapter 6

Contributor Understanding with Social Author Profiles

Before social media became mainstream, marketers and advertisers resorted to the collection of behavioral online information on individuals to target their messages. Individuals were primarily targeted based on the topical focus of the sites they visited. For example sports sites like ESPN would display sports related advertising or advertising related to the perceived interests of sports fans. The general interests of sports fans would be derived based on third party market research (e.g., males aged 25-35 with interest in sports are also interested in certain types of movies or specific male grooming products).

In the early stages of the social web, bloggers on particular topics with wide following were identified in order to endorse or sponsor specific products. At the same time, bloggers started serving advertisements on their blog real estate. Social media is transforming the way marketers and advertisers spend their budgets. Novel ways to market online are gaining traction both from an academic as well as practical point of view. In particular influencer based targeting in social media has emerged as a very popular way to market in social platforms (such as Twitter and Facebook). Individuals are identified as online experts in particular topics; they are either (1) incentivised to participate in sponsored advertising spreading the messages to their followers or the platforms automatically insert sponsored messages in their activity streams (as in the case of Twitter/Facebook advertising) or (2) targeted with relevant content such that they organically share it with their followers. The goal is to increase brand awareness, by increasing the number of impressions (e.g., how many followers see a particular tweet) and click-throughs to particular campaign (how many click on the link embedded in the tweet) with the ultimate goal to track conversions (how many end up purchasing a product).

Given the large number of users on social networks like Google+, Facebook, Twitter, and Pinterest, these sites have emerged as great marketing vehicle for marketing and branding. In this chapter, we focus on understanding the contributors on these social sites, with the aim of finding new ways of engaging and marketing to consumers. Without loss of generality, we will focus the remainder of our discussion using Twitter as an example of a social platform. The techniques we will be discussing apply equally well to any other social platform as well.

This chapter first describes the Peckalytics system for aiding influencer marketing and advertising on Twitter. We utilize carefully crafted algorithms along with solid state drives (SSDs) to ensure that the system is able to provide results within seconds despite the large dataset size. The system has been running on top of the real-time Twitter data stream utilizing the public Twitter APIs and Gardenhose APIs. The system has been tested to successfully process over 400 million pieces of content daily and operate on an archive of 30 billion Tweets, while
providing interactive response time. We describe the system in detail in the Section 6.1.

Next we propose a technique to associate a user account with a set of topics. Association with topics may be due to a multitude of reasons. For example, one may be an expert in a certain topic and as a result the user may produce content on the topic or engage with content produced by others on the same topic. Another reason may be a non-professional interest or a hobby. Naturally, other options are also possible. As a result, there are different methods one can use to assign a topic signature to a specific user account. Techniques presented later in this chapter are orthogonal to the choice of the specific method. We describe one method utilizing crowd sourcing in Section 6.2, but other options are available as well. Whatever the exact method may be, the end result is to associate each user with a set of topics, possibly with weights to represent their relative importance.

After having determined a collection of topics for each potential user account, the next natural question is to determine what type of content should we be producing for our followers. Clearly we would like the content produced to be about topics our followers are associated with so they find it interesting and engaging. If some of our followers find the content interesting they may choose to share it (e.g., re-tweets) with their followers. If they do so, our content reaches all of their followers and thus enjoys a wider reach (and as a result in the context of marketing, the marketing message is amplified). We would like to make sure that the topic we are producing content about is of interest to the users who themselves have many followers, such that if they choose to share it further, it will reach an even larger number of people.

This points to an technique that aggregates the topic signatures of the followers into a single aggregate signature (AS) that characterizes all of the followers at the same time. This aggregation process should take into consideration the potential reach via sharing (e.g., re-tweeting) from each account when aggregating the content. In this chapter we outline an algorithm called AGGR for this purpose. Given the structure of the social network, (e.g., the Twitter follower graph) a set of accounts that are of interest (e.g., the followers of an twitter account, or the potential followers for brands and companies) along with the topics they are associated with, our algorithm AGGR will produce a single AS for these accounts. Such a signature represents all topics the accounts are associated with weighted by their relative importance; A topic is important if the reach of the accounts associated with this topic is high. This information can be utilized as a guide towards what content should be produced to appeal to the followers.

Next we turn our attention to understand temporal variations of a list of aggregated signatures. Assume that for a specific brand (e.g. ‘coca cola’) we determine all the accounts that have authored Tweets containing the name of the brand. We can do this for different time intervals, such as on a daily basis. For each day we construct the AS using the algorithm AGGR. We would like to develop a principled approach to understand how the AS varies over time.

If we identify significant shift in the topics for a point in time, it could suggest that the potential followers tweeting about this brand have changed. This may point to a significant underlying event or confirm the effectiveness of a marketing campaign. For example, if we observe a sudden appearance of the topic health on the AS for Coca Cola on a specific day, this may be an alarming issue for the brand. Appearance of the topic health in the AS for Coca Cola implies that accounts associated with health are now actively tweeting about the brand Coca Cola which may point to a negative report or a crisis situation warranting an action by the brand. Similarly, assume a marketing campaign with aim of associating the brand with lifestyle or adventure. If many people associated with lifestyle (e.g., contributors in lifestyle magazines) tweet about the brand, this will point to the effectiveness of the campaign. In Section 6.6 we propose an algorithm called TEMPEVOL that given a sequence of topical vectors is able to identify temporal regions of interest to explore further.

Lastly, in Section 6.7, we present both qualitative and performance experiments to validate the efficiency of
presented algorithms using large repository of real data spanning information from tens of millions of users.

6.1 Peckalytics

Advertising on Twitter involves two crucial steps. First being able to identify who are the “experts” on any topic on the platform and second being able to identify sets of users with active “interest” on a particular topic. In the context of Twitter an expert in a particular topic is an account (user) that primarily produces and shares content related to the topic and has a wide following that actively engages with the produced content (sharing, re-tweeting etc). A user has interest in a particular topic if the user follows a number of experts in the topic and engages with the content they produce.

Peckalytics offers a number of important functions that aid marketing and advertising campaigns on Twitter. First it can identify expert accounts on any topic (queries on Peckalytics are topics). Second, it offers analytical functions on the set of expert accounts on a specific topic, such as what other topics they are experts at, what conversations they participate in and what types of content they share online. For example, using Peckalytics, one can learn that experts in ‘social marketing’ on Twitter are also experts in ‘seo’ (search engine optimization), ‘social media’ and ‘pr’ (public relations) among other things. They participate in discussions with hashtags (#smplus, #blogwell, etc) and share content from sites such as practicalcommerce.com and tweetedtimes.com.

The Twitter advertising platform works primarily in one of the following three ways:

- First, the advertiser provides a set of Twitter user handles, and Twitter targets advertisements to the followers of these accounts. The ability of Peckalytics to identify sets of experts at any topic readily aids advertisers to identify the most relevant accounts to provide while instigating a Twitter advertising campaign.

- Second, the advertiser bids on a list of topics on Twitter. Twitter using their own proprietary algorithms identifies which users are interested in the topic and subsequently targets them with tweets “promoted” by the advertisers (inserting them in their tweet stream). Peckalytics can assist in this case as well, as by analyzing related topics for a topic of interest, advertisers can identify possibly cheaper topics to bid on. For example, if the price for ‘social marketing’ is too high, Peckalytics could suggest ‘seo’ as a related topic, which may have a relatively lower bid price. The net effect however in the campaign will be the same, as largely the same audience will be targeted, since the followers of experts in ‘social marketing’ the followers of experts in ‘seo’ are highly related as per Peckalytics.

- Third, the advertisers bid on search keywords (to target searches posed to the Twitter search feature). Information on twitter is temporal by nature and events evolve with time, thus the keywords used in searches evolve over time. When a keyword is used during a search query on Twitter for which an advertisement exists, the platform will display promoted tweets (as advertising) along with the search results. Peckalytics readily assists the keyword bidding process, as for each query identifies the keywords used currently in tweets related to the query, keyword associations prevalent in tweets as well as discussions (hashtags) of interest aiding the advertiser to bid on relevant to the query keywords.

Other functionality supported by Peckalytics, is suggesting specific users that would be highly relevant as followers of a specific Twitter user. These new followers should be highly interested in the topics that the user of interest has expertise at. Taking the concept of expert identification further, Peckalytics can automatically suggest experts in any topic that have interests in some other topic as well; for example it is easy to identify all experts in ‘cloud computing’ with interests in ‘photography’ or experts in ‘food and dining’ with interest in ‘movies’. Such sets of influencers can be targets of novel engagement campaigns that attract attention by combining their area of expertise and their interests.


6.1.1 Technology

There are several components that need to operate together to make Peckalytics a functioning system. All our servers are from Dell, have 4 quad-core processors, 96GB main memory, and 12TB raw disk storage.

The system can processes and add over than 400 million new tweets per day, delivered as a real-time stream. The stream is stored in across a set of servers. Each server materializes the entire tweet, both content and meta data, utilizing a compressed row format into a local MySQL database. We refer to the collection of these MySQL instances as the data store.

A separate process, scans the tweets as they are materialized across the data store, producing a stream of tweets towards the indexer process. The indexer process runs on a node by itself and it is a multi threaded process that materializes a table for each day. Each row in the table is a unique twitter account identifier and a list of all tweet identifiers the account produced that day. This is a fairly heavy read/write scenario as in a day we materialize more than 25 million rows that are read and written on demand as new tweets arrive, for a total of more than 400 million update operations daily. A compressed row format is utilized here as well. Special care is taken to avoid deadlocks, and therefore we run with relaxed transactional semantics to increase throughput across multiple threads reading and writing the table. The tables for the last two weeks are materialized in solid state drives (SSDs) for increased performance. We refer to the collection of tables keeping the association between twitter account identifier and tweet identifiers produced for each day as the index store. The index store can easily retrieve for any day, the identifiers of all tweets produced that day for any set of twitter accounts. The collection of all tweet identifiers can then be provided to the data store to retrieve the actual tweets.

Information about which twitter accounts follow others is constantly crawled from Twitter, materializing the social graph. This graph is stored in a separate MySQL instance; which given a twitter user as a query, returns all twitter users following the queried user.

Along with each tweet, a set of meta data are appended by twitter associated with the twitter user. These meta data contain, aggregate number of followers, self disclosed personal information, location information etc. All these account profile related meta data are stored in the profile store and can be queried on demand.

Next is the process for uncovering the expertise of Twitter accounts and their interests. In order to do so, we associate every user with a topic signature. There are several ways of associating a user with a set of topics, and we present one such way in Section 6.2 which is implemented by the Peckalytics system. We utilize Twitter lists associated with user accounts to infer the topic signatures. This information is then directed to Apache Lucene to populate the index of topics associated with the account. The index supports full Lucene query syntax, including phrase queries and boolean logic. At the same time, the social graph, provides related information about ‘user interests’. Since for each Twitter user, we can also determine all Twitter accounts the user follows, the union of their topic signatures represents the interest of the user.

On top of all this information several algorithms are implemented to extract useful information and conduct analysis. When a query topic is supplied, we extract the set of experts from the Lucene index, we obtain their profile information from the profile store and retrieve all tweets they have conducted in the last weeks utilizing the index store and the data store. We analyze all topics they are experts at by processing their topic signatures, extract all keywords they use, by conducting syntactic parsing of their tweets, we conduct analysis of frequent word associations (words used together frequently in tweets) as well as extract popular hashtags used in their tweets.

We have also implemented a special url dereferencing algorithm utilizing asynchronous IO for maximum performance. Since typically urls in tweets are shortened (using popular url shortening sites like bit.ly or t.co) conducting analysis on the shared domains is challenging as each url has to be un-shortened (possibly multiple
times). Thus special care has to be taken to efficiently un-shorten multiple urls. Utilizing asynchronous IO we can conduct this process for tens of thousands of urls in parallel on a single thread, typically in a second. Contrast this with un-shortening a single url at a time by conducting an http probe, which will typically require at least 100 seconds for the same number of urls.

Specific algorithms help identify expert users on topic A which have interests on topic B. We have also implemented specific algorithms that given a twitter user suggest relevant new followers. New followers are those who are strongly interested in the topics the twitter user has expertise but do not follow it already. This feature is of great importance to brands that wish to attract relevant followers to further spread their messages to the maximum possible number of users in the Twitter ecosystem. Related analysis can be conducted for Twitter accounts that have interests in particular topics, as well as for the followers of any set of twitter users.

### 6.1.2 Usage Experience

Peckalytics is available online at: [www.peckalytics.com](http://www.peckalytics.com). A familiar search box, accepts topic queries with full boolean syntax as shown in Figure 6.1. Upon issuing a query topic, a number of expert accounts are retrieved and displayed along with their profile information (Figure 6.8). Analytics on these accounts are also provided, namely, **TOPICS** representing topics associated with the query topic in the topic signatures of these users, **KEYWORDS** used frequently in the recent tweets of these users, **KEYWORD PAIRS** resulting from a frequency analysis of the keyword associations between keywords in tweets as well as **HASHTAGS** which represent frequent hashtags in the recent tweets of the accounts identified. For each such analysis function a visual word cloud is provided to study the words identified and their associated frequency via suitable font sizing. At the bottom of the page, we display all domains from which content is frequently shared (in the form of url links) via the tweets of these users.

In addition, at the bottom of the list of accounts returned, one can click on ‘Analyze the interests of their followers’ and automatically conduct exactly the same analysis but this time taking into account collective followers of these accounts. This provides an easy way to identify the interests and analyze the ‘audience’ of any expert group on Twitter. Notice that in this case we return exactly the same type of analytics to keep the same user experience. The reader should note that the amount of processing taking place in the back-end this time is orders of magnitude larger than that taking place for the set of expert users. Each of the expert users can have thousands of followers. In addition to the large number of tweets that have to be retrieved from the index and data store, the number of urls that have to be un-shortened could be in the hundreds of thousands. In the cases that the
collective number of followers is very large, the back-end conducts uniform random sampling to make sure that the Peckalytics user interface remains interactive. It is therefore imperative to keep processing manageable in this case.

As is evident in Figure 6.1 Peckalytics also supports a set of more advanced queries. The first type is one that identifies experts in a specific topic A that have interests in a topic B. Notice that topics A and B can be full-fledged boolean queries thus they can be fairly expressive encompassing boolean operators and phrases. The option max reach if selected instructs the algorithm to identify those experts accounts on topic A with interest in topic B, that collectively maximize the follower reach. This effectively means that the set of accounts reported of cardinality $K$ collectively reach the maximum number of unique accounts (thus maximizing user impressions) among all possible subsets of expert accounts of size $K$.

The second option, identifies the set of accounts that have interest on topic A (where A is a full-fledged boolean query). Similarly the option max reach if selected has the same effect on the accounts reported as above.

Finally the last option suggests followers for a specific twitter account, that have interests in a topic A (again by using a full-fledged boolean query). On the back-end all experts on topic A are identified and an analysis of their followers is performed. Among those, the ones with the largest number of followers, that do not currently follow the specific twitter account are identified and returned in the result.

Peckalytics is fully functional and available online and all functionality outlined is available for any visitor to www.peckalytics.com.

In the next section we describe the way we construct topic signatures for Twitter users. The following section uses this information to formalize problems for computing aggregate signatures (AS) and studying evolution of the same over time.

### 6.2 Topic Signatures

Our techniques make use of a topic signature that is derived for every Twitter user. Although there are many possible ways to extract such signature, we detail in this section the specific way we chose for our work. We emphasize however that any other applicable technique can be used without affecting the techniques presented later in this work.

Twitter introduced the concept of a list [105] a few years ago. A Twitter list is a collection of Twitter accounts that can be created by any user. A list contains one or more accounts that the user associates with a particular topic or domain. For example a user might create a list about music that contains all the favorite Twitter accounts that are associated with music. Similarly another user may create a list called fashion containing accounts that the user associates with fashion. A user can create multiple lists. The utility for user creating the list is the ability to filter the content they see on their feed by lists. Thus, it is easy to see tweets only from the music accounts and tweets only from the fashion accounts.

At the same time such user generated actions provide valuable data for extracting useful information regarding Twitter accounts. Our Twitter list crawler, at the time of this writing, has identified 15 million Twitter lists with 12 million unique users. The crawler for lists and the social graph utilizes public APIs provided by Twitter. As each Twitter list declares the total membership, these lists should have a total of 330 million list membership edges. Of these 330 million edges, we have crawled 130 million edges at the time of writing, making our dataset 32.72% complete across all lists. Note that our missing memberships belong users that have been recorded as members of lists only a few times, i.e., Twitter users belonging to less than three lists. Our dataset contains all users who are contained in multiple lists. Commonly list names are descriptive of their content. Thus, if a user names a list...
### Chapter 6. Contributor Understanding with Social Author Profiles

#### Figure 6.2: Peckalytics Analytics UI

<table>
<thead>
<tr>
<th>TOPICS</th>
<th>KEYWORDS</th>
<th>KEYWORD PARTS</th>
<th>#HASHTAGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>new</td>
<td>toronto</td>
<td>bayview collision</td>
<td>#sandy</td>
</tr>
<tr>
<td>media</td>
<td>collision</td>
<td>collision 427</td>
<td>#cto</td>
</tr>
<tr>
<td>canada</td>
<td>city</td>
<td>collision blocking</td>
<td>#tc</td>
</tr>
<tr>
<td>local</td>
<td>time</td>
<td>hwy collision</td>
<td>#hwy</td>
</tr>
<tr>
<td>chl</td>
<td>east</td>
<td>collisionegnton</td>
<td>#edmonton</td>
</tr>
<tr>
<td>polit</td>
<td>people</td>
<td>yonge collision</td>
<td>#gootnewthursday</td>
</tr>
<tr>
<td>canadian</td>
<td>traffic</td>
<td>kipling collision</td>
<td>#fly</td>
</tr>
<tr>
<td>peep</td>
<td>storm</td>
<td>eastbound collision</td>
<td>#opendata</td>
</tr>
<tr>
<td>social</td>
<td>line</td>
<td>brox collision</td>
<td>#mscatlady</td>
</tr>
<tr>
<td>list</td>
<td>hurricane</td>
<td>collision mississauga</td>
<td>#canpol</td>
</tr>
<tr>
<td>tdtr</td>
<td>york</td>
<td>collision streetcar</td>
<td>#toppol</td>
</tr>
<tr>
<td>peopl</td>
<td>street</td>
<td>collision hydro</td>
<td>#startko</td>
</tr>
<tr>
<td>art</td>
<td>power</td>
<td>collision dunas</td>
<td>#100northsya</td>
</tr>
<tr>
<td>twitter</td>
<td>road</td>
<td>collision dup</td>
<td>#bigstrom</td>
</tr>
<tr>
<td>journalist</td>
<td>halloween</td>
<td>collision into</td>
<td>#tw</td>
</tr>
<tr>
<td>word cloud</td>
<td>word cloud</td>
<td>word cloud</td>
<td>word cloud</td>
</tr>
</tbody>
</table>

**CONTENT MOSTLY SHARED FROM THE FOLLOWING DOMAINS**
- torontostandard.com - Toronto Standard
- theglobeandmail.com - Home - The Globe and Mail
- homehonor.com - Home ? a unique horror adventure By Benjamin Rivers
- imgur.com - imgur: the simple image sharer
- torontohydro.com
- microsoft.com - Microsoft Home Page | Devices and Services
- vimeo.com - Vimeo, Your Videos Belong Here
- barackobama.com - Barack Obama
- testtoronto.com - TEDxToronto 2012 Conference | Ideas Worth Spreading
- yfrog.com - yfrog | Share, Converse and Connect
- iloveto.com - I LOVE TO.
Figure 6.3: The distribution of Twitter users and their list membership

music or my favorite musicians it is most likely consisting of accounts who are related to music.

Peckalytics index indexes 240 million Twitter users—those who have tweeted at least once in the last 8 month time period. Of the 240 million Twitter users, about 31.42 million of them belong to at least 1 Twitter list. To
facilitate crawling speed, we first sorted all users in descending order of the number of lists they are member of, and then we crawl up to 2,000 lists per Twitter user.

Figure 6.3 shows the distribution of Twitter users and their list membership. We found that most accounts are listed under less than 10 lists. To be precise, 85.31% of the Twitter users are listed in less than 10 lists. 96.77% of the Twitter users are listed in less than 50 lists, and only 0.10% of all users are listed in more than a thousand lists.

### 6.2.1 Constructing Topic Signatures

We now detail how topic signatures is created for an account. As we process the list names, we tokenize and stem them and remove stop words and other frequently appearing information as well as Twitter idioms that carry no information (e.g., ff, friends etc). Then a dictionary of similar words is used to group together tokens that have the same meaning (utilizing WordNet [36] data). For each Twitter account we assemble a vector of all tokens in the titles of all the lists that they are a member of. Each token is accompanied by a number that expresses the occurrence count, namely, how many times the token was identified in a list title that the account is a member of. We can view the topic signature as a vector, assuming a total ordering on all topics (tokens) and assign a value of zero to the occurrence count for a topic if the account is not associated with that topic at all. Assuming, each token to be a unique dimension in a multi-dimensional space, the occurrence counts are normalized to produce the unit topic signature vector in $L_1$ space. Thus, this vector represents the weight of the account being associated with a topic. We use the $L_1$ space and hence, the length of the expertise vector is normalized to 1 using the manhattan norm. The union of all these vectors will result in a multi-dimensional space with each unique token corresponding to a dimension.

**Example 6.1** Consider a user @john that is member of three Lists \{toronto-dentist, dentists, music-toronto\}. The set of tokens with occurrence counts for this user is \{dentist(2), toronto(2), music(1)\}. After normalization, the unit topic signature vector becomes \(\text{topics}(\text{john}) = \frac{2}{5}\text{dentist} + \frac{1}{5}\text{music} + \frac{2}{5}\text{toronto}\). The vector above is of unit length in $L_1$ space, with non-zero values across three dimensions and zero across all others.

Now, let's consider two more users: @henry belonging to lists \{dentists, squash-london, music\}, and @susan who is a member of lists \{squash, music-london, squash-london\}. After processing all the 3 users, we will have a 5-dimensional topic space as displayed below.

\[
\begin{pmatrix}
\text{dentist} & \text{music} & \text{london} & \text{squash} & \text{toronto} \\
\text{john} & \frac{2}{5} & \frac{1}{5} & 0 & 0 & \frac{2}{5} \\
\text{henry} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 \\
\text{susan} & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 \\
\end{pmatrix}
\]

The above matrix is a compact form notation of individually writing the vectors as,

\(\text{topics}(\text{john}) = \frac{2}{5}\text{dentist} + \frac{1}{5}\text{music} + 0\text{london} + 0\text{squash} + \frac{2}{5}\text{toronto}\)

and so on.

The process of computing topic signatures for each user is linear in the number of users and the length of their topic signatures. This technique of extracting topic signatures is applied on Tweets from 240 million users and 15 million lists. We utilize Apache Lucene for tokenizing, and indexing the Twitter lists. The index allows query with full boolean syntax, and is used to quickly return all users having certain topic association. Therefore, utilizing Peckalytics, it is easy to find all Twitter users associated with the topic of dance music or any boolean query involving topics.
6.3 Related Work

Twitter data have been on the center of research activity in recent years. We consider Twitter lists to assign topics to Twitter users. There exist several approaches for topic extraction specifically on Twitter. Weng et. al., [109] identifies “influential” users utilizing techniques related to PageRank [83]. This work can be easily adapted to identify users influential on a particular topic on Twitter. Similarly Pal and Counts [84] deploy a clustering technique to identify topical authorities on Twitter. An crowd sourcing based approach related to the techniques introduced by Peckalytics is available by Ghosh et. al. [43]. The techniques presented in this paper are independent of the techniques utilized to conduct the topic extraction, and algorithms presented should work without modification.

Better understanding of users on social media sites is of great value for several applications. While we focus primarily on marketing and advertising use cases in this chapter, Pavlidis et al. [88] describe a commercial gift recommendation system also based on social media users.

Techniques to detect changes in time series and sequences have been widely studied. Jagadish et. al. introduced the notion of deviant point/range in a time series, namely ranges such that the associated values are away from the “norm” according to global measures (see [59] and references therein for work in this area). Such techniques are not directly related to those deployed by algorithm TEMPEVOL introduced herein; we are primarily concerned with vectors (aggregate signatures) and their changes in our domain.

There is a vast body of work studying the citation network and evolution of academic publications (see [16] and references therein). The focus of this work is primarily to understand the structure and evolution of science, and to study the patterns of citations and co-authoring over time. Unlike the presented herein, the underlying data is of much smaller size and has different characteristics.

Finally Hidden Markov Models [90] have been widely utilized in various fields. We are utilizing them for algorithm AGGR in this paper.

6.4 Problem Formulation

We represent the set of all users as $U^M$, which has cardinality $M$. Let $u \in U^M$ denote a unique user, with unit topic signature vector $topics(u)$. The vectors are derived from a multi-dimensional space $S^N$ with $N$ dimensions. The matrix representing signature vectors for all users (similar to that in Example 6.1) will then have $M \times N$ entries.

If $s_i$ is a specific dimension in $S^N$, then signature vector may be represented as follows where $w_i$ represents length of the vector across dimension $s_i$, $topics = w_1.s_1 + w_2.s_2 + \ldots + w_N.s_N$

The social graph spanning all users denoting follower relationships is represented by $(U^M, E)$, where $U^M$ is the set of nodes and $E$ is the set of edges. Each user represents a node. Edges are follower relationships, i.e., if a user $u$ follows another user $v$, then the directed edge from $u$ to $v$ will be part of the set of edges $E$. Formally, $u, v \in U^M$, follows$(u, v) = true \iff (u, v) \in E$

Let $q$ represent a keyword query e.g., “hurricane sandy” or “pepsi”; it may be more complex to include boolean operators such as, “elections AND (“barack obama” OR “mitt romney”)” as well. Let $R$ represent the set of tweet results after evaluating the query against the content of all raw tweets on Twitter. If the search has a time restriction $t$ denoting that only results within the time interval $t$ are of interest, we denote the set of results as $R_t$. Each entry $r \in R_t$ is a Tweet such that matches$(q, r) = true$ meaning that the query evaluates to true on tweet $r$ and post_time$(r) \in t$, namely that the tweet was posted within the time interval of interest $t$. Let $A_t$ denote the set of
Given a query message. We propose an efficient algorithm that satisfies these requirements and is computationally efficient given signature aggrsig. We do not wish to consider every member of $A_t$ the users who can spread the content (e.g., by re-tweeting). Hence, when constructing the aggregate signature multiple ways and increases engagement and content sharing.

As time progresses, we expect a natural evolution in the matrix $ASM(q, T)$ over time for different $t$. Consider a long time range $T$ consisting of $D$ smaller time intervals, $T = \{t_1, t_2, \ldots t_D\}$. We compute the aggregate signature for a given query $q$ for each of these intervals as

$ASM(q, T) = \{aggrsig(q, t_1), aggrsig(q, t_2), \ldots aggrsig(q, t_D)\}$

The resulting matrix $ASM(q, T)$ has $N$ rows and $D$ columns. The rows will each correspond to a topic dimension from $S^N$, and columns will each correspond to a time interval from $T$. We refer to this matrix as the aggregate signature matrix ($ASM(q, T)$) over time $T$ for the query $q$.

Problem 6.4.1 Given a query $q$, time interval $t$, and its associated set of authors $A_t$ from result set $R_t$, compute aggregate aggrsig($q, t$) taking into account the potential reach of each author in $A_t$.

We present an algorithm called AGGR to solve this problem. Once we solve Problem 6.4.1, we proceed to investigate how aggrsig($q, t$) evolves over time for different $t$. Consider a long time range $T$ consisting of $D$ smaller time intervals, $T = \{t_1, t_2, \ldots t_D\}$. We compute the aggregate signature for a given query $q$ for each of these intervals as

$ASM(q, T) = \{aggrsig(q, t_1), aggrsig(q, t_2), \ldots aggrsig(q, t_D)\}$

The resulting matrix $ASM(q, T)$ has $N$ rows and $D$ columns. The rows will each correspond to a topic dimension from $S^N$, and columns will each correspond to a time interval from $T$. We refer to this matrix as the aggregate signature matrix ($ASM(q, T)$) over time $T$ for the query $q$.

Consider again the query hurricane sandy; we conduct the search for one day time intervals for a 92 day long period from 1 Oct 2012 to 31 Dec 2012. For each day, we compute the aggregate signature vector based on everyone who is talking about the hurricane on that day (posted a tweet containing the words hurricane and sandy). As time progresses, we expect a natural evolution in the matrix $ASM(q, T)$ for this query. Hurricane Sandy first affected Caribbean and Bermuda on Oct 22nd, and Twitter users actively participating in the discussion where topically associated with these regions \footnote{For example a local news station on Twitter are associated with the topic of news, but also with the specific geographic location. The corresponding Twitter account therefore will belong to several lists containing the location name in the title of the list.}. As days progressed, more American and subsequently global audience started discussing the hurricane. As the hurricane traveled from the southeast of the US (Florida, Virginia, Carolinas), to the mid-atlantic region (Washington DC, Maryland, New Jersey), and finally reaching the New York City, the group of users talking about the hurricane changed. In November, post-hurricane, the discussion shifted...
further to rebuilding efforts and those discussing where associated with politics. Intuitively it is evident that this 92 day time period can be partitioned into a select few time ranges capturing the evolution of this story namely tracing the geographical path of the storm (by observing the topics associated with those taking about it) and then capturing the political discussion centered in re-building efforts. Thus, formally we have,

**Problem 6.4.2** Given an aggregate signature matrix $ASM(q, T) = \{agg sig(q, t_1), \ldots, agg sig(q, t_D)\}$, a pre specified $k < D$, and a function $score$ that measures similarity of aggregate signatures, namely, $score(agg sig(q, t_i) \ldots agg sig(q, t_j))$. Define a disjoint, continuous $k$ partitioning of $[1, 2, \ldots, D]$ as

$$P_k := \{[b_1, e_1], [b_2, e_2], \ldots, [b_k, e_k]\}$$

with

- $b_1 = 1, e_k = D, \forall b_i, e_i \in \mathbb{Z}, e_i \geq b_i$ and $\forall i < j, e_i = b_j - 1$
- by solving for

$$\arg \min_{P_k} \sum_i score(agg sig(q, t_{b_i}), \ldots, agg sig(q, t_{e_i})).$$

We present an algorithm called TEMPEVOL for this problem. A solution to this problem aids immediately to the problem of identifying the ‘right’ number of intervals $k$. Typically, we can iterate over a few values of $k$ and trace the value of the overall function $score$ for each value of $k$. Points at which large discontinuities arise are typically good candidates for $k$.

Solution to this problem is of immediate interest to marketers. Marketers invest a lot of effort to change the way brands are perceived of thought of or associated with. A change in the audience of a brand could be organic over time, or it can be influenced by an event. For example, numerous marketing efforts attempt to reinvent or reposition brands in new target segments and change the way brands are perceived online or offline. An effort to make a brand more fashionable or trendy may be successful if the people taking about the brand online associate themselves with fashion and/or fashion trends. Thus, such changes, if one is able to identify them, may point to the success or failure of marketing efforts online.

In Section 6.6 we discuss alternate solutions to Problem 6.4.2 by presenting different types of objective $score$ functions. In all cases, our solution to Problem 6.4.2 is agnostic with respect to the given function, and remains computationally efficient for processing large amounts of data.

### 6.5 Aggregate Signatures

Our aim is to compute an aggregate signature, using the topic signatures of all users who have mentioned query $q$ on Twitter in the specified time interval $t$. In this section, we outline techniques to model reach of a specific group of users and develop algorithms to construct aggregate signatures taking such models of reach into account, solving Problem 6.4.1.

Having retrieved all tweets $R_t$ with respect to the query $q$ of time interval $t$, we scan through all items in $R_t$ to resolve the set of unique Twitter users from $R_t$ as $A_t$. Then, we get the topic signatures for each user in $A_t$.

One simple strategy to produce aggregate signatures is to sum up the topic signatures retrieved and normalize them to unit length. However, this method fails to capture the relative importance of each user in disseminating a message to their followers with respect to the query $q$. Under this scheme all users are assumed equally important as far as the dissemination potential is concerned which is not the case in real life. For example, the set $A_t$ may contain several users with association in the topic of music but each with very few followers, and few users with association in the topic of travel but each with many followers.
It is desirable to have AGGR compute the relative ranking of \( u \in A_t \). This is because, by looking at the subgraph induced by \( A_t \) on the original Twitter follower graph \((U^M,E)\) only, a user \( u_1 \) may have substantial number of followers in \((U^M,E)\) but have very little number of the followers who also belong to \( A_t \). Since our aim is to find users who can disseminate the message to potentially largest group of relevant users, considering the number of followers in \( A_t \) is more important than the total number of followers across the entire Twitter social graph.

To capture all these intuitions we naturally model this scenario as a Hidden Markov Model, with each user in \( u \in A_t \) as a node in the hidden layer, and each topic in their AS’ as a node in the output layer. For users \( u, v \in A_t \), if user \( u \) follows \( v \), we add a directed edge in the Markov chain from \( u \) to \( v \). Transition from one node to another takes place with equal probability. That is, if there are \( e_u \) edges out of node \( u \), one of the edges is selected for transition with equal probability \( \frac{1}{e_u} \). Since the Markov chain may have disconnected components, with a small pre-specified probability \( \alpha \) a random jump takes place, and with probability \( 1 - \alpha \) one of the outgoing edges is selected.

Traversing the Markov chain, while at node \( u \), having \( e_u \) outgoing edges, the probability of transition is computed as follows. If \( e_u \) is zero, the next node after transition is randomly picked from set \( A_t \). Let \(|A_t| \) be the cardinality of the set \( A_t \). If \( e_u \) is non zero, then the next node will be:

\[
\text{next}(u) = \begin{cases} 
\text{pick random } v \in A_t & \text{with prob } \frac{\alpha}{|A_t| - e_u} \\
\text{pick random } v \mid \text{follows}(u,v) & \text{with prob } \frac{1 - \alpha}{e_u}
\end{cases}
\]

This completes the construction of the Markov chain and we next assign emission probability for the topics at each node. The symbols being emitted from the HMM are dimensions of the topic signature. For example, if the topic signature of a user \( u \) is \( \text{topic}(u) = w_1.s_1 + w_2.s_2 = \frac{1}{2}\text{music} + \frac{1}{2}\text{squash} \), then one of \text{music} or \text{squash} is emitted with equal probability when at the node in the HMM associated with this particular user \( u \). Since the topic signatures are of unit length in \( L_1 \) space, we do not need to do any further normalizations in order to compute symbol emission probabilities. For a topic signature, \( \text{topic}(u) = w_1.s_1 + w_2.s_2 + \ldots + w_N.s_N \). The symbol \( s_i \) will be emitted with probability \( w_i \). Since, \( w_0 + w_1 + \ldots + w_N = 1 \), the probabilities will add up to 1.

Example 6.2 Continuing from Example 6.1, and assuming each of the three users follow each other, we construct the HMM as displayed in Figure 6.4.

The hidden layer is constructed with three nodes, representing the three users. Transition edges are added, each with probability \( \frac{1}{2} \), such that \( Pr(u_1 | u_2) = 0.5 \). As a result, we observe the steady state distribution to be \( Pr(\text{john}) = Pr(\text{henry}) = Pr(\text{susan}) = 1/3 \) for the hidden layer.
Chapter 6. Contributor Understanding with Social Author Profiles

Marginalizing out user probability from \( Pr(\text{topicsignature, user}) \), we can compute the aggregate signature for the entire graph. For example,

\[
Pr(\text{dentist}) = Pr(\text{dentist}|\text{john})Pr(\text{john}) + Pr(\text{dentist}|\text{henry})Pr(\text{henry})
\]

\( \begin{align*}
&= \frac{1}{3} \times \frac{2}{3} + \frac{1}{3} \times \frac{1}{4} = \frac{13}{60}, \\
\end{align*} \)

and,

\[
Pr(\text{toronto}) = Pr(\text{toronto}|\text{john})Pr(\text{john})
\]

\( \begin{align*}
&= \frac{1}{3} \times \frac{2}{3} = \frac{2}{15}.
\end{align*} \)

Similarly, \( Pr(\text{music}) = Pr(\text{london}) = Pr(\text{squash}) = \frac{13}{60} \).

As a final check, \( Pr(\text{dentist}) + Pr(\text{music}) + Pr(\text{london}) + Pr(\text{squash}) + Pr(\text{toronto}) = 1 \)

The resulting AS therefore is

\[
\frac{13}{60} \hat{\text{dentist}} + \frac{13}{60} \hat{\text{music}} + \frac{13}{60} \hat{\text{london}} + \frac{13}{60} \hat{\text{squash}} + \frac{2}{15} \hat{\text{toronto}}
\]

Now we have defined the HMM, with a set of nodes, transition probabilities, and emission probabilities for symbols. The steady state probabilities for this HMM will allow us to compute the aggregate signature across the set of all users \( A_t \).

At steady state, assume, the probability that a symbol \( s_i \) is seen is \( \text{prob}(s_i) \), then the aggregate signature will be, \( \text{aggrsig}(A_t) = \text{prob}(s_1)s_1 + \text{prob}(s_2)s_2 + \ldots + \text{prob}(s_N)s_N \), which is of unit length in \( L_1 \).

Identifying the steady state distribution of a Markov chain is a well studied problem in the literature [90]. An algorithm that iteratively multiplies the current steady state vector with the transition matrix until convergence would suffice for social media graphs such as the Twitter follower graph. To compute the aggregate signature given the steady state distribution from the Twitter follower graph, we can use the definition of conditional probability.

Observe that for topic \( s \):

\[
Pr(s) = \sum_u Pr(s, u) = \sum_u Pr(s|u)Pr(u) \tag{6.5.1}
\]

and since \( Pr(s|u) \) (the topic probability of a user \( u \)) and \( Pr(u) \) (the steady state probability of user \( u \)) are independent and known from the Markov chain and preprocessing, we can proceed to solve the HMM first by the hidden user layers, then the emission (topic) layer.

As we will see later in the experimental section, random sampling can be used to speed up this computation without sacrificing too much quality. This effectively offers a good tradeoff between accuracy and speed.

### 6.6 Temporal Evolution of Aggregate Signatures

In this section, we discuss techniques to automatically identify changes in the aggregate signatures over time for a given query.

In the previous section, we presented algorithms to efficiently compute the aggregate signatures for a given query \( q \) and a specified time interval \( t \), \( \text{aggrsig}(q, t) \). This process can be repeated for several time intervals. If the aggregate topic signature for query “hurricane sandy” is computed for 92 one day periods over the 1st of Oct 2012 to the 31st of Dec 2012, we wish to to determine, algorithmically, the changes in users (authors) writing (tweeting) about the query over time; this can be achieved by determining the changes in aggregate signatures.

For example, in November, the users talking about the hurricane were more political in nature (users tweeting about hurricane sandy were associated with politics) as the discussions had shifted from being Caribbean-focused on 22nd Oct to rebuilding efforts in the USA.

One way to consolidate and summarize the changes in aggregate signatures is to identify continuous date ranges that have similar aggregate signatures. For this particular example, we have divided the time period 1st Oct 2012 to 31st Dec 2012 in 92 shorter 1-day intervals, and hence have essentially constructed a matrix with
92 columns (each column being one AS for the day). We wish to select \( k \) groups of continuous days across the 92 day period for a pre-specified \( k \). Each of these \( k \) date ranges will represent a distinct aspect of the event. For example, if \( k \) was 3, we could expect the resulting date ranges to represent the pre-hurricane period, the period during the hurricane specifically as it passed over New York city, and the period post-hurricane.

Using the notations defined in Section 6.4, given the aggregate signature matrix \( ASM(q, T) \) and specified \( k < D \), we wish to partition \( T \) into \( k \) continuous and disjoint intervals. The aim is to group similar time periods together, and this is formalized by defining a scoring function capturing similarity that we minimize. We present two different scoring functions. Once the scoring function has been chosen, the problem reduces to that of identifying the optimal partitioning.

The first function minimizes the total error represented as the sum of root mean square distance between the average aggregate signature of a collection of signatures and the aggregate signatures in the collection. The second discretizes \( ASM(q, T) \) into an indicator matrix of 0 and 1, and measures similarity as the hamming distance across neighbouring aggregate signatures.

### 6.6.1 Minimizing RMSE

The first measure, given a collection of aggregate signature

\[
ASM = \{aggrsig(q, t1), aggrsig(q, t2), \ldots aggrsig(q, tD)\}
\]

access the distance using the root mean square error:

\[
score = \sum_i \|aggrsig(q, ti) - \left(\frac{\sum_j aggrsig(q, tj)}{D}\right)\|_2
\]

The RMSE \( score \) increases as the distance between aggregate signatures increases, i.e., when the topics across \( \{t1, t2, \ldots, tD\} \) are different, and decreases when the topics are the same. Therefore, with this \( score \) function we wish to single out intervals of time where the aggregate signatures are very similar to each other.

### 6.6.2 Minimizing Hamming Distance

We propose a second error measure that involves the discretization of aggregate signatures. The value in each dimension of \( aggrsig(q, ti) \), is between 0 and 1. The aggregate signature can be discretized by assigning each dimension the value of 0 or 1. There are many ways to discretize the signature; a statistically sound way is to assess the mean of all the values and assign a value as 1 if it is above some standard deviation of the mean and zero otherwise. Denote the discretized \( aggrsig(q, ti) \) as \( \text{aggrsig}(q, ti) \) and similarly, the discretized \( ASM(q, T) \) matrix as \( \text{ASM}(q, T) \). We also write \( t1, t2, \ldots, tD - 1 = T1 \) and \( t2, \ldots, tD = T2 \). With this notation, we rewrite it in the compact form: \( score = \|\text{ASM}(q, T2) - \text{ASM}(q, T1)\|_F \) using the Frobenius norm.

\[
\|A\|_F = \sqrt{\sum_i \sum_j |A_{ij}|^2}
\]

where \( A \) is a matrix.

### 6.6.3 Algorithm

Given a function \( score \) that computes a distance between aggregate signatures of \( ASM(q, T) \), the following recurrence can be used to solve Problem 6.4.2:

Define \( B_{j,k} \) to be the best \( k \) partition score of the first \( j \) columns of \( ASM(q, T) \) using the given \( score \) function

\[
B_{j,k} = \min_{i<j} B_{i-1,k-1} + score(exp(q, ti) \cdots exp(q, tj))
\] (6.6.1)
Algorithm 7 TEMPEVOL: Best K partitioning of $ASM(q, T)$

1: **INPUT** $ASM(q, T)$: the aggregate signature matrix for query $q$.
   $score$: user specified dissimilarity for aggregate signatures
2: **OUTPUT** the best $K$ partition of $ASM(q, T)$ for all $K < D$ where $D$ is the number of columns of $ASM(q, T)$
3: **Procedure** Best K Partition ($ASM(q, T), score$
4:   for all $(i, j)$ such that $i \leq j$
5:     $sco_{i,j} \leftarrow score(ASM(q, Ti) \cdots ASM(q, Tj))$
6:   end for
7:   $bestp \leftarrow null$ (keep tracks of the actual partitioning)
8:   for $i \in 1 \ldots D$ do
9:     $B_{i,1} \leftarrow sco_{1,i}$
10:    $bestp_{i,1} \leftarrow \{1, i\}$
11:  end for
12:  for $k \in 2 \ldots D$ do
13:     for $j \in 2 \ldots D$ do
14:        $i \leftarrow \arg \min_{i \leq j} B_{i-1,k-1} + sco_{i,j}$
15:        $bestp_{j,k} \leftarrow bestp_{i-1,k-1},\{i, j\}$
16:        $B_{j,k} \leftarrow B_{i-1,k-1} + sco_{i,j}$
17:     end for
18:  end for
19:  RETURN $bestp$

Algorithm 7 will compute the best $K$ partition of $ASM(q, T)$ for all $K < D$ using Equation 6.6.1. Notice that it would take $O(D^D)score$ steps to solve the best $K$ partition of $ASM(q, T)$ for all $K$ in a brute force way. Since there are $(D-1\choose K-1)$ ways to produce $k$ disjoint continuous intervals for $[1, 2, \ldots, D]$, we have $\sum_{i=1}^{D} O(D^i) = O(D^D)$.

Looking at the recurrence Equation 6.6.1 we can pre compute $score(ASM(q, Ti) \cdots ASM(q, Tj))$ as it is independent from the recurrence. Line 14 will take $O(D)$ steps. When solving for the best $K$ partition of $ASM(q, T)$ for all $K < D$ using dynamic programming, the runtime is dramatically reduced to $O(D^3)$. The space requirement can also be optimized by noting that in Equation 6.6.1, $B_{j,k}$ depends only on the values from the previous iteration. Therefore, after an iteration is complete, we can discard the optimal interval partitioning and the optimal scoring from the last iteration, bringing the space requirement excluding the precomputed scores, down to $O(D)$.

Algorithm 7 is therefore efficient both in terms of the run time and the storage requirements.

### 6.7 Experiments

We have conducted experiments to evaluate the efficiency of the proposed algorithms $AGGR$ and $TEMPVOL$. We first present quantitative results establishing the suitability of the proposed techniques for very large data sets. Next, we present the qualitative results to validate the effectiveness of the techniques to produce relevant results.

Experiments were conducted using data from August 2012 to January 2013. Our system currently indexes more than 26.02 million Twitter users and their followers from the Twitter graph. All data is crawled using free public APIs as described at http://dev.twitter.com. This equates to more than 17.9 billion graph edges being part of our data set. The experiments are conducted on a Dell PowerEdge 520 server with 96 GB of main memory, and sixteen Intel Xeon processing cores running at 2.53 GHz.
6.7.1 Presentation

Qualitative results for the AGGR algorithm are presented by displaying “top accounts”. We define top accounts as those user nodes with the highest steady state probability in the hidden layer of the HMM as described in Section 6.5.

To construct the input to TEMPEVOL, we run AGGR in 1 day increments in the specified time period. As each invocation of AGGR produces exactly one AS, we construct matrix ASM for input to TEMPEVOL. In Section 6.6 we presented an efficient algorithm for TEMPEVOL which does not require any input parameter and computes the best partitions for all values of $k \in [1, D]$. For our qualitative experiments, we infer the number of partitions to use by comparing the results for increasing values of $k$. For each value of $k$ we assess the error (value of score – which is monotonically non increasing) and choose as the number of partitions (the value of $k$) that causes the maximum drop in the value of score between two successive values of $k$.

Once we obtain the $k$ time intervals partitioning the time domain $D$ of the ASM, we use Equation 6.7.1 to rank and show, the topics for each of the $k$ time intervals. We refer to these topics as “top topics”. Let $s$ be a topic in time interval $I$.

$$\frac{\max_{aggrsig(q,t), t \in I} Pr_{aggrsig(q,t)}(s)}{\min_{aggrsig(q,t), t \in [1,D] \setminus I} Pr_{aggrsig(q,t)}(s)}$$

(6.7.1)

This equation selects the the maximum probability of the topic $s$ in interval $I$, where $I$ is chosen by TEMPEVOL, divided by the minimum probability of $s$ outside $I$. Given that there are thousands of topics it is imperative to rank them in a meaningful way. Equation 6.7.1 demonstrated stable behavior in our experiments. Other choices could apply as well.

For TEMPEVOL, we use the term “overall topics” to refer to the topics obtained when we set $k = 1$. Ranking of the topics in this case is done by their maximal probability inside the single interval. As the interval occupies the entire date range, Equation 6.7.1 does not apply in this case.

Note that this specific presentation methodology is selected only for the purposes of illustrating the results in a concise format. Other presentation methodologies are equally well applicable.

6.7.2 Quantitative Evaluation

In this section we show the run time and memory consumption for both AGGR and TEMPEVOL algorithms. We evaluated our results on several different queries and time ranges, and the quantitative results were similar for each case. Therefore, we only present one set of results for the query “toronto” spanning Aug-Dec 2012. Results for other queries are omitted for brevity. The results we present have been averaged over five repetitions for each experiment.

Given the query $q$, we first retrieve all tweets containing the query terms for the specified time range. We then construct a set $A_t$ of all authors (users) of these tweets. The HMM is constructed, as described in Section 6.5, with each user belonging to $A_t$ as a node. For the query “toronto”, there are 7.3 million Tweets in the specified 5 month period. The author set $A_t$ for this query consists of 6.9 million unique users.

Once the HMM is constructed with 6.9 million users, the AGGR algorithm takes 16 seconds to produce the resulting AS'. We first show that we can further speed up this run time by using random sampling on the set $A_t$. Instead of constructing the HMM with all users in $A_t$, we randomly select only a fraction $f \leq 1.0$.

Figure 6.5 shows the time taken as the fraction $f$ varies. Figure 6.6 shows the (average) cosine similarity of resulting AS as $f$ varies when compared to the one produced using all 6.9 million users (denoted as AS'). Looking at the two figures it is evident that it is not necessary to use all users in $A_t$. Using $f = 0.3$ cuts down the run time
to under a second while still providing over 90% accuracy (measured as cosine similarity). As the fraction $f$ is reduced, the number of topics with non-zero weights also decreases as depicted in Figure 6.7. For $f = 0.3$, the resulting AS has only half the number of topics with non-zero weights compared to AS'; however the topics not present are only those with very small weight in AS'. Therefore, these experiments suggest that random sampling speeds up the AGGR algorithm considerably while producing almost the same results as AS'. Reduction in the number of topics with non-zero weights in AS further helps in reducing the run time and memory usage.

We construct an AS for each day of Jan-Dec 2012 and run the TEMPEVOL algorithm. Figure 6.8 shows the running time of TEMPEVOL as a function of the number of days of tweets considered. Even with a full years worth of tweets, the running time, at less than an hour, is reasonable. We show the overall memory consumption in Figure 6.9. Memory consumption scales linearly with the amount of data, which is a desired behavior.

The algorithm TEMPEVOL is capable of handling realistic data sizes (running on a year on twitter data for less than an hour). As a result it is a practical technique to be deployed in a realistic settings.

### 6.7.3 Qualitative Evaluation

For the qualitative set of our experiments, we issued sample queries and utilized our two algorithms, comparing the results. The queries are chosen to be fairly diverse. The first query is about the city of “Toronto”. Another query pertains to two competing automobile brands (“Lamborghini” versus “Ferrari”); similarly, we run one query to compare and contrast various large commercial banks in Canada (“HSBC”, “RBC”, “CIBC”, “BMO” and “TD”). We also issued queries related to various competing mobile computing brands (like “Apple”, “Samsung”, “Microsoft”, “Nokia”, “HTC”) and also queried for “HSBC”, which is a major international banking institution. Finally we choose a query from the technology domain, namely “machine learning”. All of the qualitative ex-
CHAPTER 6. CONTRIBUTOR UNDERSTANDING WITH SOCIAL AUTHOR PROFILES

Figure 6.6: Cosine similarity to the full expertise vector vs percentage \( f \) users used from search results in \( A_k \).

Experiments were conducted using the score function of Section 6.6.1. When we present the top topics for an AS, we use Formula 6.7.1 as the ranking function for the topics, and only show the top ranked topics. For the queries where output from TEMPEVOL is available, we also provide results for TEMPEVOL with the Hamming distance score function of Section 6.6.2 for qualitative comparison.

With the \( k \) partitions identified, it is desirable to provide a succinct summarization of the most prominent topics from the aggregate signatures for presentation (as the aggregate signature vectors spans hundreds of thousands of dimensions in our experimental setup). We use Equation 6.7.1 to select these topics.

Table 6.1: Top accounts for Toronto

<table>
<thead>
<tr>
<th>Top accounts for query “Toronto” from Aug-Dec 2012</th>
</tr>
</thead>
<tbody>
<tr>
<td>CP24, MarinasDiamonds, stats_canada, wheelingprobs, LindseyStirling, Raptors, MuchMusic, jianghomeshi, iLikeGirlsDaily, ChiefKeef, blogTO, jparencibia9, TorontoStar, TheHockeyProbs, kiss925toronto, threeadaysgrace, globearnmail</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Top accounts for query “Toronto” Oct 29-30, 2012</th>
</tr>
</thead>
<tbody>
<tr>
<td>CP24, psy_oppa, DarrenDreger, TheHockeyNews, KnucklesNilan30, MapleLeafs, Drake, Raptors, globearnmail, TorontoStar, CBCNews, blogTO, gerrydee, theScore, kiss925toronto, stats_canada, 680News, CBCAlerts, TTSystem, WestJet, CTVNews, Flow935, torontoist, BTtoronto, MiguelUnlimited, MacMiller, CBCToronto, YourAnonNews, edsheeran,</td>
</tr>
</tbody>
</table>

Using the ranking of users, based on steady state probabilities in HMM for AGGR, for the query Toronto, one can obtain information regarding the types of social media accounts that are highly associated with Toronto. The overall top accounts for the query Toronto mainly pertain to general fashion, sports, entertainment, and news media. In Table 6.1 we present the corresponding top Twitter accounts for Oct 29-30, 2012; This is the time interval identified by TEMPEVOL on the same input. One can clearly notice a change at a specific temporal
range on the accounts associated with Toronto on that interval, in particular, we see sports, and news related accounts move up in the ranking as compared to the usual top accounts associated with Toronto. We also find that @psy oppa stands out, which is the Twitter handle of international superstar South Korean singer known by the stage name Psy, who was scheduled to visit Toronto on Oct 31, 2012.2

In Table 6.2 we observe the output for TEMPEVOL for query Toronto. Overall topics for Toronto, which is computed by using $k = 1$, confirms that discussions about Toronto mainly pertains to general fashion, sports, entertainment, and news media. However, during the period (Oct 29-30,2012), we identify a temporal change; “weather”, “update”, “nba”, “basketball” were among the most prominent topics associated with Toronto, and this was precisely the period when the hurricane Sandy passed through Toronto, and the Toronto Raptor’s upcoming game against the Pacers took place on the 31st.

In the next experiment we demonstrate how the output of AGGR and TEMPEVOL can provide insights about competing brands. In Table 6.3, we consider overall topics for the queries Lamborghini and Ferrari over the entire data set range (again, by setting $k$ the number of partitions in TEMPEVOL to 1). We determine that the

---

2http://www.huffingtonpost.ca/2012/10/22/gangnam-style-psy-toronto_p_2002663.html
Figure 6.8: Time taken for TEMPVOL (in ms) vs number of days

<table>
<thead>
<tr>
<th>Overall topics for “lamborghini”</th>
</tr>
</thead>
<tbody>
<tr>
<td>funny, people, news, favstar, music, car, auto, twitter, automot, fact, media, fan, entertain, team, love, cool, comedi, game, fun, peep, interest, humor, amigo, stuff, tweep, artist, best, new, sport, famili, tweeter, random, favorit</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Overall topics for “ferrari”</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1, favstar, news, sport, formula, amigo, deport, people, frase, noticia, funny, music, humor, motorsport, motor, car, media, twitter, one, formula1, fan, auto, race, favorito, periodista, love, medio, celeb, best</td>
</tr>
</tbody>
</table>

Table 6.3: Topics from AS for Lamborghini and Ferrari

Lamborghini brand is primarily associated with topics indicative of ‘trendy’ (such as “funny”, “favstar”, “cool”, “fun”, “best”), whereas the Ferrari brand is associated more with high-performance professional racing (“formula [one]”, “sport”, “f1”, “motorsport”). In Table 6.4, we observe the results of the TEMPEVOL algorithm for the query Lamborghini. The algorithm selected the period November 8-9, 2012, which coincided with a lot of Twitter activity pertaining to Lamborghini (Figure 6.10). Much of this activity involved retweets of a tweet by account @UnluckyBrian3, a comedic Twitter account, and observe that “humor”, “funny”, and “comedy” are very prominent. We also see a temporal change for the topic “coco”, which is originating from comedian Conan O’Brien’s team, team coco.

Table 6.4: Topics for the query Lamborghini on Nov 08-09 2012

---

3https://twitter.com/UnluckyBrian/statuses/267112765523628032

| coco, fact, comedi, humor, funny, amigo, one, car, interest, random, favstar, stuff, top, fun, media, favorit, auto, entertain, automot, best, music, love, peep, 2, news, twitter, cool |
Similarly we ran AGGR for various commercial banks in Canada, for the same time duration. Interesting observations can be made: for example, TD, HSBC and CIBC are subject to more politics related discussions than BMO or RBC, possibly due to press releases such as “HSBC to pay 1.9 billion U.S. fine in money-laundering case”\(^4\). BMO and CIBC are more engaged in the real estate topics on Twittersphere (“mortgag[e]” and “mls”, the Canadian multi listing service). And also, by looking at the topics, we single out HSBC as a bank foreign to Canada by looking at its top topics from its AS.

When querying for competing mobile computing device manufacturers (Apple, Samsung, Nokia, Microsoft, HTC), TEMPEVOL identified the intervals October 3-5, 2012 (with prominent topics including Microsoft, Windows, and mobile), November 1-2, 2012 (with prominent topics including jailbreak, Apple, iOS, and iPhone), and December 11-12, 2012 (with prominent topics including “security”, “microsoft”, “appl[e]”, “io[s]”, and “app”) (Table 6.6). These time periods correspond to the release of the Microsoft Surface tablet computer, the release of a jailbreak for Apple iOS, and Microsoft’s failed update of their SkyDrive iOS app which was rejected by Apple, respectively. We also considered results for this query for Twitter data beyond the August-December 2012 time period. One interval of interest identified by TEMPEVOL is January 16-17, 2013 (with prominent topics including finance and stocks). During this period there was a lot of negative activity on the Twittersphere following a Wall Street Journal article\(^5\) about the substantial drop in Apple’s stock price post September 2012.

Table 6.7 presents the result for a query pertaining to the financial institution, HSBC. Using AGGR we can identify HSBC to be associated with “news”, “business”, “media”, and “financ[e]”. TEMPEVOL identifies the period November 21-22, 2012 with prominent topics including “forex”, “asia”, and “china” (Table 6.7). To better understand the output, we looked at tweets from November 21-22, 2012 for HSBC and we used Pearson’s

---

\(^{4}\)http://www.reuters.com/article/2012/12/11/us-hsbc-probe-idUSBRE8BA05M20121211

\(^{5}\)https://twitter.com/WSJ/statuses/291787169624907777
Figure 6.10: Number of mentions of Lamborghini on Twitter on a day-by-day basis, we see an uprising of volume of Lamborghini near the optimal interval selected by TEMPEVOL algorithm

$\chi^2$ test identifying the word pairs mostly associated with the query terms (refer to [4] for specific methodology). During that time the exchange rate between the Chinese yuan and the Australian dollar exceeded a psychologically significant threshold, which was widely tweeted about by interested parties. There was also a lot of chatter about HSBC China flash PMI (purchasing manager’s index) reaching a 13-month high, and lots of tweets about Asian equities trading higher (Figure 6.11). These events are nicely corroborated by TEMPEVOL where the temporal changes for “forex”, “asia”, “finan”, “trader”, “china”, “fx” and “stock” are to be noted.

We also tested AGGR and TEMPEVOL on a more generic topic, with a low volume of activity on the Twitter, namely “machine learning”. In Table 6.8, we observe that machine learning is associated primarily with tech, news, and research, which is expected. TEMPEVOL chooses the interval September 15-17, 2012. In this interval, the most prominent topics include “friendz”, “toronto”, and “women”. Using our index of tweets, we discovered that an event pertaining to women in machine learning took place in Toronto during this time (hash tag #GirlGeeksTo). Furthermore, there is activity related to machine learning under the categories tedplus, pydjang, and pycon, among the tweets for September 15-17, 2012. As machine learning is not usually associated with these topics, TEMPEVOL correctly singled out this interval. Notice that TEMPEVOL need not single out the time period with the most chatter, as we can see in Figure 6.12.

Lastly, we present experimental data for the two different score functions proposed in Section 6.6.1 by running TEMPEVOL with the two score functions separately, on the same query. For this set of experiments, we rank the topics in AS using Formula 6.7.1. The results are presented in Table 6.9 and Table 6.10. Using TEMPEVOL with RMSE appear to yield higher quality results compared to that of Hamming distance. We note that as we change the score function, TEMPEVOL selects very different time span as the interval that optimizes for the most error improvement. While scoring with Hamming distance did yield some further insight for some queries, (eg. “teambreezi” for query Lamborghini as fans for the American entertainer Chris Brown

---

6https://twitter.com/carolinemcggr/status/247843579450511363. Inmar is teaching us about machine learning #GirlGeeksTO pic.twitter.com/wTqYufut

7http://uk.answers.yahoo.com/question/index?qid=20110324133313AAPTEP

<table>
<thead>
<tr>
<th>Bank</th>
<th>Topics</th>
</tr>
</thead>
<tbody>
<tr>
<td>TD Canada Trust</td>
<td>news, job, sport, peopl, bank, media, busi, toronto, financ, local, hockey, financ, canada, twitter, real, music, team, new, fan, market, career, peep, , compani, tweep, polit, social, know, famili, tweeter, other, servic, jobsincanada, boston, tech</td>
</tr>
<tr>
<td>Royal Bank of Canada</td>
<td>news, peopl, rbc, media, busi, sport, financ, real, music, fam, estat, hockey, folk, bank, know, twitter, local, financi, canada, , job, toronto, market, team, canadian, golf, peep, ace, commerci, fan, sporter, tweep, invest</td>
</tr>
<tr>
<td>Canadian Imperial Bank of Commerce</td>
<td>news, job, media, toronto, real, canada, busi, estat, peopl, financ, local, canadian, tech, market, financi, polit, tweep, vancouy, mortgag, career, peep, tweeter, social, health, invest, bank, calgari, fan, ontario, ottawa, sport, run, cbcf, music</td>
</tr>
<tr>
<td>HSBC</td>
<td>news, noticia, amigo, busi, media, financ, peopl, polit, golf, , market, de, sport, medio, twitter, y, rugbi, forex, info, financi, periodista, e, social, music, trade, fan, journalist, world, invest, tweep, stock, tech, econom</td>
</tr>
<tr>
<td>Bank of Montreal</td>
<td>news, peopl, sport, soccer, media, job, team, music, , busi, hockey, toronto, cool, football, twitter, love, funni, fan, local, financ, nba, real, canada, 37entesln, fnr, canadian, peep, tweeter, tweep, fam, market, team37ent, mls</td>
</tr>
</tbody>
</table>

Table 6.5: Overall topics for various Canadian banks

many tweets, pictures for his Lamborghini), for the majority of the queries, \textit{score} with RMSE produced more relevant results when compared to the Hamming distance.

In this chapter we have presented techniques to better understand contributors to the social content. Specifically, we described the Peckalytics system and outlines the algorithms \textit{AGGR AND TEMPEVOL}. Qualitative and quantitative experiments demonstrating the practical utility of proposed algorithms were conducted. The next chapter concludes the thesis by summarizing the contributions and listing possible future research directions.
<table>
<thead>
<tr>
<th>Overall topics</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>news, tech, favstar, media, peopl, appl, funni, technolog, android, noticia, social, busi, tecnologia, market, iphon, info, mobil, music, 2, stuff, y, blog, amigo, interest, web, game, love, tecnologa, fun, 3, geek, humor, blogger, best</td>
<td></td>
</tr>
<tr>
<td><strong>Oct’03 2012 TO Oct’05</strong></td>
<td></td>
</tr>
<tr>
<td>microsoft, health, window, mobil, berita, radio, tecnologia, android, medio, tecnologa, noticia, food, youtub, phone, nokia, market, seo, web, y, digit, compani, app, technolog, geek, tech, design, d, comput, humor</td>
<td></td>
</tr>
<tr>
<td><strong>Nov’01 2012 TO Nov’02 2012</strong></td>
<td></td>
</tr>
<tr>
<td>jailbreak, food, seo, design, io, journalist, mac, financ, develop, microsoft, dev, 6, 7, noticia, web, medio, app, digit, nokia, technolog, tech, appl, iphon, gadget, geek, market, site, comput, mobil, media</td>
<td></td>
</tr>
<tr>
<td><strong>Dec’11 2012 TO Dec’12 2012</strong></td>
<td></td>
</tr>
<tr>
<td>secur, medio, microsoft, nokia, noticia, techi, writer, game, financ, seo, site, tecnologa, digit, scienc, tech, app, tecnologia, compani, io, appl, comput, mobil, mac, android, dev, artist, web, social</td>
<td></td>
</tr>
<tr>
<td><strong>Jan’13 2013 TO Jan’17 2013</strong></td>
<td></td>
</tr>
<tr>
<td>kpop, financ, stock, medio, microsoft, nokia, scienc, noticia, digit, market, game, tecnologia, tecnologa, android, berita, fact, busi, brand, one, compani, writer, social, media, sport, mac, phone, tech, mobil, technolog</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.6: Running TEMPEVOL for mobile handset manufacturers

<table>
<thead>
<tr>
<th>Overall topics for HSBC</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>news, noticia, amigo, busi, media, financ, peopl, polit, golf, favstar, market, sport, medio, twitter, y, rugbi, forex, info, financi, periodista, social, music, trade, fan, journalist, world, invest, tweep, stock, tech, econom,</td>
<td></td>
</tr>
<tr>
<td><strong>Topics for HSBC on Nov’21-22 2012</strong></td>
<td></td>
</tr>
<tr>
<td>forex, restart, finan, empir, avenue, ekonomi, rugbi, haber, manteman, trader, bellado, china, econ, fx, finanza, economia, egypt, updat, trade, stock, asia, eco, twibe, expert, economi, invest, biz, market, financ, tweepl</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.7: TEMPEVOL for HSBC

<table>
<thead>
<tr>
<th>Overall topics for “machine learning”</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>tech, data, news, scienc, peopl, media, develop, social, research, big, technolog, dev, web, busi, analyt, bigdata, learn, program, startup, market, ml, machin, comput, job, geek, python, educ, secur, nlp, 2, math, softwar, design, favstar, scala, cs</td>
<td></td>
</tr>
<tr>
<td><strong>Topics for Sep 15-17 2012 for “machine learning”</strong></td>
<td></td>
</tr>
<tr>
<td>tedplus, pydjango, friendz, frequent, kreativ, sport, write, publish, teaparti, women, money, myfollow, econom, china, livro, campusadmin, digerati, libertarian, pynchon, bud, bookish, emploi, vario, javascript, teman, nativ, toronto, tool, liberti</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.8: TEMPEVOL on machine learning
Figure 6.11: Most correlated pairs of keywords for query HSBC based on the Pearson’s $\chi^2$ est, showing word pairs

Figure 6.12: Number of mentions of “machine learning” on Twitter on day-by-day basis
### Table 6.9: Qualitative comparison of RMSE vs Hamming score function in TEMPEVOL

<table>
<thead>
<tr>
<th>RMSE TEMPEVOL interval with the most improvement in error, and top topics from AS</th>
<th>Hamming TEMPEVOL interval with the most improvement in error, and top topics from AS</th>
</tr>
</thead>
<tbody>
<tr>
<td>cloud computing</td>
<td>Dec’29,2012 TO Dec’30,2012 wahooigan vmug intl techenthusiasts2 negocio publish mkt sm ibm bi myfollow twibe crm microsoft seo y dev ifttt tic virtual entrepreneur new engag job app de thank best retweet compani</td>
</tr>
<tr>
<td>Oct’24,2012 TO Oct’25,2012 cisco sap vmware hp linux ibm salesforc crm empresa secur educ research journalist tool ti analyst event tic de thought work tecnologia infosec tecnologa y enterpris ict app microsoft job</td>
<td>gangnam Oct 02,2012 TO Oct’03,2012 frase comedi joke chist sport funni favstar humor quot direction tweeter fun d los youtub rt believ de favorito y mis noticia radio celeb amigo music one best awesom peopl</td>
</tr>
<tr>
<td>machine learning</td>
<td>Nov’08,2012 TO Nov’09,2012 rv sheffield beheiri transito informa grime malaysia rebeld jornai berita celeb adoro everyon affili egypt geral 2011 guernsey revista work hsbc artist list folk uk network music</td>
</tr>
<tr>
<td>Sep’15,2012 TO Sep’17,2012 pydjang friendz frequent kreativ sport write publish women money econom china campusadmin pynchon bud emploij vario javascript teman toronto tool mason fr access vc seattl nba startupweekend bigdatalist baseball hpc</td>
<td>lamborghini Nov’08,2012 TO Nov’09,2012 coco fact comedi funni amigo quot car interest favstar entertain best music love 2 news Twitter 3 fan peopl</td>
</tr>
<tr>
<td>Sep’07,2012 TO Sep’08,2012 footbal mom secur st spain shirt elit sky universidad contractor geo octob irish everyday nuevo deal informtica sleep linguist bookish author respons andrew geekdom twittero ottawa www enthusiast best mysteri</td>
<td>hsbc Nov’21,2012 TO Nov’22,2012 forex restart empir ekonomi trader china fx finanza economia egypt trade stock asia twibe economi invest biz market financ econom money financi busi bank medio local journalist news world fami</td>
</tr>
<tr>
<td>Sep’08,2012 TO Sep’09,2012 rv sheffield beheiri transito informa grime malaysia rebeld jornai berita celeb adoro everyon affili egypt geral 2011 guernsey revista work hsbc artist list folk uk network music</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 7

Conclusion

This thesis focuses on the goal of uncovering insights from blog posts, Tweets, Facebook wall posts, online message board discussions, and a multitude of other similar sources. Information created in social media has exploded in the past decade, growing by several orders of magnitude. This has resulted in an abundance of data with content being authored on every imaginable topic around the world and all the time. On an average day, counting just the public content, hundreds of millions of posts are created.

The temporal nature of this new media source is one of its most distinctive properties. All posts, as they are created, have a unique timestamp attached to them. While, it is possible for an author to go back and edit an existing post, in reality this is only a rare occurrence. As a result, bulk of the posts are created and never updated or deleted after creation. Hence, one can consider this media source as an append only stream. Specifically, we model it as a high volume social text stream.

7.1 Challenges

We list key challenges along with select examples from the previous chapters to highlight our approach:

- **High Volume** The amount of data produced on a daily basis is unprecedented. Every day, more than half a billion new public posts are created, adding to hundreds of gigabytes of text. At peak hours, 20,000 Tweets are created every second. While researchers have studied the problem of uncovering insights from text over the past few decades, at the time a collection with 20,000 documents in total was considered large – and now we have the same amount of new data being added in a second.

  In Chapter 3, for example, instead of resorting to complex graph partitioning techniques we use biconnected components to generate the clusters of keywords. This makes it possible to scale to keyword graphs with tens of millions of nodes. In the same chapter, the presented algorithms have efficient secondary storage realizations, making them very suitable for high volumes of data. Similarly, the Peckalytics system, from Chapter 6, scales beautifully and makes intelligent use of dynamic programming to speed up the computation.

- **Unstructured** Social media data has no schemas or pre-defined structure. It is mostly free flowing text, and that too usually not properly written. Previous research has primarily focused on analysis of news articles and academic publications which are authored with correct grammar and proper spellings. Social data, on
the contrary, is informal and often lacks linguistic accuracy (given 140 character limit of Twitter, it is not possible to do so).

With unstructured text documents, the task of representing topics as they evolve is not trivial. Chapter 3 represents these topics as a set of keywords that frequently appear together. We stay away from semantic analysis and other more involved natural language processing algorithms to allow for operation on unstructured informal text. Our formulation also allows us to process text in any language. Our definition of the topic is purposefully kept unstructured to match the intricacies of the underlying data.

- **Online and Real-time** We are no longer dealing with a static repository of information. Our data set is continually updated. Hence, the indices and pre-computed data structures need to be updated too. This prohibit us from using algorithms that do not allow incremental updates to the data structure. Both tasks, user queries and addition of new data, happen simultaneously, and the algorithms must allow for this.

While all algorithms presented in this thesis are amenable to real-time addition of new data, the sparse interval set presented in Chapter 4 is a good example of how this can be achieved. The presented methodology is not only simple to maintain as new data arrive but also provides a predictable and tunable trade-off between preprocessing and runtime query execution time.

- **Interactive** We are developing techniques to aid an analyst to uncover insights from a huge information repository. This process is expected to be iterative. The analysis process therefore must be interactive, i.e., results should be displayed within seconds allowing the analyst to adjust the parameters and drill down quickly. The absolute accuracy of the produced result set is less material, but the speed of computation needs to be fast.

For example, Chapter 4 puts speed of computation at the center stage. We demonstrate that trading off the precision a little (with theoretical bounds) results in improvement in running time. For an applications of the nature discussed in this thesis, it is a desirable behavior to have quicker response even if they are not 100% precise.

- **Reliable** We wish to implement the proposed techniques in a real-world system, and have done so for the case in Chapter 2 and 6. As both these systems are used by general public at their own (and not under controlled demo settings), we need to make the components constituting the system reliable and failure proof. All data must be stored in at least two copies to eliminate single point of failure. This means, maintaining consistency across multiple copies of the same data - not only under normal operation but also when failures occur. This adds significant engineering difficulty, and further limit the choice of algorithms that can be used for this data. For example, when creating the inverted index on two different servers simultaneously ensuring the two processes are running consistently is not an easy task. The difficulty is compounded when the volume of data (thousands of Tweets added in a second) is taken in consideration making it difficult to employ any obvious method to compare the two indexes. Second consideration of reliability is operation under failure. When a component fails, the system must automatically direct all the queries to the operational component. When the failure is recovered, the system must provide ways of synchronizing the failed component back to the current state.

The BlogScope system, outlined in Chapter 2, has operated with minimal human intervention for years. The system has continued to collect new data from disparate online sources and serving hundreds of thousands of visitors every month. The system was discontinued from public access eventually but the spin-off Sysomos platform [98] continues to operate albeit at a much larger scale. Over the past six years, the Sysomos
platform has operated with near 100% uptime, serving enterprise customers around the world with analytics based on 100+ billion social conversations [100].

- **Usable** Usability of the system is another important, but often overlooked, consideration. The presented system is used independently by thousands of users on their own, without any supervisions from the authors. The importance of usability can therefore not be overstated, specially given the unstructured high volume nature of the data. Unlike well defined traditional analysis tasks, the analyst in our case is likely looking for the unknown and surprising findings. To support this, the system must allow for flexible querying. The results produced must be visualized in simple, easy to understand, and informative ways.

The query by document functionality described in Chapter 5 provides a good example of this. The analyst can query by not only providing keywords but also by submitting an entire document to the system. Flexibility of the system in offering multiple starting points for analysis process is noteworthy. Additionally, the user interface, as shown in Figure 5.1, is kept simple and usable for this functionality. The same is evident from Figure 6.8 for Peckalytics as well.

To summarize, our main challenge is reliable execution of flexible methods to uncover insights from a high volume data stream in an online and interactive manner.

### 7.2 Contributions

In this work, we explored three key functionalities to aid an analyst in uncovering insights from high volume real-time stream of textual data: (1) Persistent Chatter Discovery (2) Cross-referencing Media Sources and (3) Contributor Understanding.

Our first aim was to create a core data aggregation and management platform. As a result, we first created efficient crawler to collect social media data with a fast spam filtering engine. Next, we added a data storage layer which provided both raw content and processed indexes for quick analysis. Basic searching was extended by core set of analysis functionality, like popularity charts, bursts, and keyword correlations. As the system was used by at average 10,000 visitors every day, and operated on an archive of 3.25 billion posts, we ensured that all components operated reliably and analysis results are produced within seconds. The core BlogScope system is described in detail in Chapter 2. This system forms the basis of functionality described thereafter.

Chapter 3 defines the notion of topics discussed persistently over a period of time. Using co-occurrence of keywords as the basis, the topics are defined loosely as a *clusters* of keywords that frequently appear together. Repeating this process for multiple time intervals, we find clusters that persist over time as stable clusters. Using a large data collection for experiments we validate the efficiency of presented algorithms and quality of generated results.

Aggregation of ranked lists for top-\(k\) computation is well studied in literature. We extend the same problem but with the use of commonly available hierarchies in Chapter 4. This is motivated by the need to find top-\(k\) *topics* by mapping keywords to topics and aggregating the lists of keywords over a time range. We explore both probabilistic and deterministic stopping conditions. We conclude that probabilistic framework, while theoretically interesting, is not practically feasible. Our experiments conclude that relaxing the precision guarantees result in faster execution while still maintaining required precision.

Social data incorporates a multitude of media sources. With the aim of cross referencing these media sources, we define the problem of querying by document in Chapter 5. To achieve this, we extract phrases from the input text document to construct the search query. Using external knowledge source, such as Wikipedia, we further
extend the presented algorithm for enhancing the query phrases. Given the qualitative nature of the problem, we use Amazon’s MTurk to employ human judges to validate our techniques.

Lastly, we turn our attention to understanding contributors. Chapter 6 first describes the Peckalytics system which can scale to analyze over 30 billion Tweets, adding 400 million new every day. We formalize the notion of expertise topic signatures for a contributor and present algorithm for identifying changes in aggregate expertise over time. Experimental results are presented to demonstrate the efficiency of the presented algorithms.

7.3 Learning: The Power of Simplicity

We have built a system that operates reliably on massive amounts of data and has been used by users around the world. Our engineering philosophy can be summarized as: employ the simplest approach that works. While this fact may sound ‘simplistic’ and obvious-once-stated, it indeed has been the biggest learning over for me personally over the past years.

There is a whole spectrum of choices to be made when solving a problem. To allow for scaling to massive levels, while operating without prohibitive system maintenance, we select the choice that is the simplest in the list of solutions that work.

Solutions typically fall in one of the four categories (1) simple and naive, (2) simple but clever, (3) complex and clever, and (4) complex. The second category is the most desired. Below we provide examples to illustrate the concept.

Section 2.2.2 describes bursts, a widely used feature of BlogScope. Consider a solution that defines the burst to have occurred when the number of mentions is over a fixed number (say 100) – this will be overly simplistic and will fall in the first category simple and naive. The algorithms proposed by Kleinberg [61, 62] are much more accurate and theoretically sound, but they fall in third category of complex and clever. To scale to the levels we require, we need something simpler. In Section 2.3.5 we end up defining a burst to have occurred when the number of mentions cross the mean value plus twice the standard deviation over the past 90 day period. This definition, while theoretically sound, is much simpler both to understand and implement, and has also worked well in practice over the last seven years in our system.

In the previous section we talk about reliability as a key challenge given that we need to maintain multiple near-consistent copies of data. Consider the storage of data indexes in the BlogScope system. Assume we keep two copies of the indexed data on two different servers for load balancing and reliability as shown in Figure 7.1. When both the servers are operational, a query can be answered by either of the two indexes in round robin, and if one server is down the other server can keep functioning to answer all queries. If we do nothing more, this is a simple and naive solution, as the two servers will have slightly different versions of the data (when indexing thousands of documents per second, these inconsistencies are bound to happen). As a result, the user will see different results on each page refresh of the BlogScope search page which is not desirable. At the other end of the spectrum is a complex solution, where we develop a new distributed file system with consistency guarantees while offering high performance. This is an overkill. A much simpler solution is to keep the two indexes independent but (a) replace the round robin based load balancing with something smarter and (b) ensure loose consistency. Using the round robin load balancing method to select the server for a query causes the results change on every page refresh for the user. Instead, we hash the userid to select the server, so that all queries from the same user are served by the same index - this will eliminate all data inconsistencies for the user even when they are present across users. This will also assist in caching, as the similar queries (from the same user) will all be served from same sever increasing the cache hit rate. As for point (b), we still need to maintain consistency across the two
data indexes but the requirements are now much more relaxed making the job easier. It must be appreciated that while ‘easier’ here is still quite complicated, it is significantly simpler than the alternative approach of building a fully consistent distributed data index storage layer.

![Figure 7.1: Replicated data indexes in BlogScope.](image)

The algorithm in Section 3.2 operates on a large keyword graph to identify clusters. Graph partitioning has been a topic of active research [60]. A lot of graph partitioning techniques however can not scale to graphs with tens to hundreds of millions of nodes easily, and hence are not applicable in our problem setting. We instead select bi-connected components from the graph as the clusters. This approach is not only scalable, but also produces very relevant results as is evident from our experimentation.

Chapter 4 also explores the spectrum of solutions mentioned above for the problem of list aggregation in presence of hierarchies. We first consider the probabilistic solutions, which while being very interesting, falls in the category four of being complex and impractical. Section 4.5 provides the practical solution which is much simpler but effective. Section 4.6 proposes the sparse interval set to further speed up the query times while preserving the simplicity and providing nice tunable theoretical bounds.

We have provided only a few examples to illustrate the point *keep it simple but clever*, but the reader of this thesis can observe this same theme across the work. It must be clarified that *simple* can indeed be quite complicated on an absolute scale, but we seek the solution that is significantly simpler than its complicated alternatives (and hence not practical in a high volume system with reliability requirements) on a relative scale. We achieve simplicity by better understanding the user needs. For example, an analyst may care more about quickly seeing the “approximate” results and iterating while doing an exploitative research task over waiting for precise results to load. Our experiments demonstrate that all techniques presented herein are not only practical to implement but also produce very relevant results on the qualitative scale.

### 7.4 Impact and Future Directions

This thesis is based on the following published peer reviewed research papers:

- **Chapter 2**: BlogScope System Description covers the poster from WWW’07, the demonstration from VLDB’07, and the workshop paper from WebDB’07 [7, 8, 6].
- **Chapter 3**: Persistent Chatter Discovery by Identifying Stable Clusters covers the paper from VLDB’07 [4].
- **Chapter 4**: Persistent Chatter Discovery in the Presence of Topic Hierarchies covers the paper from SIGMOD’08 [5].
- **Chapter 5**: Cross-referencing Media Sources by Querying by Document covers the paper from WSDM’09 [113].
Chapter 6: Contributor Understanding with Social Author Profiles covers a demonstration from SIGMOD'13 [20].

In addition to the papers listed above, following published works co-authored by us are not covered in this thesis: the workshop paper in ICWSM’09, and the full papers in VLDB’09 and VLDB’10 [86, 94, 74]. Part of the work done in this thesis is also covered by the US patent application no. 12/501,324 [9]. The patent covers the core technology responsible of the BlogScope system and associated algorithms.

Work done in this thesis opens avenues for future work in several directions. This includes both academic and commercial extensions. We first list work that has already been conducted in this regard:

- **Academic** Social media data collected by BlogScope and its core functionality has been used for several other projects, including Grapevine [2] and BlicQtimes [75, 76].

- **Commercial** The BlogScope system has resulted in the creation of the commercial entity Sysomos [98]. Sysomos is a leading provider of social media analytics to companies around the world, including Coca Cola, Intel, Microsoft, Adidas, Unilever, Google and Visa, and powers 70% of the Fortune 50 brands. Our data centers house thousands of terabytes of data, building upon the BlogScope system, and provide interactive web based analytics dashboards for marketing, public relations and consumer insights functions.

Next, we provide possible future research directions, relating each to the three parts of this thesis:

- **Persistent Chatter Discovery** Users wish to see timelines of stories as they evolve. Consider for example, the resignation of the Pope Benedict XVI followed by the papal conclave leading to the election of Pope Francis. In Chapter 3 and 4 we propose techniques to track evolution of such stories as millions of Tweets and posts are authored on the topic with a real-time coverage. Our techniques are primarily based on keywords and keyword clusters. Once these high affinity keyword clusters are identified, the next step is to stitch them together in a readable text format. A desirable output in this scenario will be a well written but automatically generated text, possibly assembled by extracting sentences from the corpus.

An example output could be: *Pope Benedict XVI announced on 11 February 2013, citing health concerns, his decision to step down. On 28 February, with his resignation, he became the first to step down as the leader of the Catholic Church since the medieval times. 115 cardinals from around the world gathered on 12 March to elect the next pope. The conclave elected Jorge Mario Bergoglio from Argentina on 13 March as white smoke was seen coming out of the Sistine Chapel chimney. The new Pope assumed the name Francis.*

Most real-life events and topics can be modeled as significant events and quiet periods. Over a period of time, there are one or more events of major significance relating to the topic and the remaining time period is the quiet period. What constitutes a significant event differs from user to user. Some users may want to see only the most significant of the events, while others may want a more detailed view. Users background knowledge of the subject and the need to know more about the topic both contribute towards this difference in behavior across users. By varying the threshold for quiet period vs significant event, one can vary the level of detail in the produced text. The resulting text as a result can be a single paragraph or run in pages as per user preference.

- **Cross-referencing media sources** Plurality of media sources will continue to grow as social media evolves further. The need to cross reference these sources, as the same topical story evolves in each, will become even more important a problem. In Chapter 5 we formalize the problem of relating a single text document...
with results from another media source. Natural extension of this is to do the same for multiple documents at a time.

Consider again the example of the resignation of Pope Benedict XVI from above. This very same story was discussed by both news journalists and online bloggers. While the events are the same, the point of views across the two group of authors will be widely different. It would be interesting to see a two pane view, with one showing news articles and other with corresponding blog posts, with a timeline to see the comparative evolution of the story.

- **Contributor Understanding** Study of reach and spread of a message in a social graph is of immense use to marketers and advertisers. Researchers in both computer science and sociology have formulated models to study the same, both theoretically and empirically. These models however are usually based on the entire graph and do not take in account the expertise of the user. They only consider the number of outgoing edges from a node as a measure of further spread for the message. Chapter 6 provides a framework to assign expertise and interest to a user in the social graph. It will be interesting to re-formulate the models for studying reach and spread in a social network after differentiating the users based on their expertise topic signature on a per-topic basis.

Since commencement of this thesis work in 2005, social media has evolved significantly. It can be argued that the impact of this new media has been fundamental towards our society and civilization as a whole. We expect a continued evolution, at an even faster pace, which will further disrupt the ways we interact with each other as social beings. We have been fortunate to witness this phenomena - social media - as it grew from its nascent stage to today over the course of this thesis work. We believe this to be just the beginning, and we are excited to watch what the future holds as the research community worldwide continues to study the implications and applications of the same in a greater detail.
Bibliography


[100] Sysomos turns 100,000,000,000. http://blog.sysomos.com/2012/09/19/sysomos-turns-100billion/.


