UNIVERSITY OF CALIFORNIA
RIVERSIDE

Effective Exploration of Web and Social Network Data

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Shiwen Cheng

June 2015

Dissertation Committee:

Dr. Vagelis Hristidis, Chairperson
Dr. Marek Chrobak
Dr. Eamonn Keogh
Dr. Vassilis Tsotras

The Dissertation of Shiwen Cheng is approved:

Committee Chairperson

University of California, Riverside

Acknowledgments

First, I would like to thank my advisor, Professor Vagelis Hristidis. In the past five years, his insightful guidance in research had always pointed me to the new topics and ignited me with novel ideas. He is very responsive whenever I need his input in my research. I appreciate that he put trust on me through my PhD study and gave me enough flexibility on research. I also feel thankful that he gradually offered me opportunities in building and maintaining the data infrastructure for different projects, which I believe the experience is very useful. I would also like to thank other committee members, Professor Marek Chrobak, Professor Eamonn Keogh and Professor Vassilis Tsotras, as well as former committee member Professor Harsha Madhyastha, for their helpful comments on my research.

I thank all my collaborators. Professor Marek Chrobak always brought novel ideas into our projects. His solid knowledge in algorithms and theory greatly helped to move forward our research. I feel lucky to work with Arash Termehchy during my first research project. I appreciate his mentorship which trained me to be a professional researcher. I also thank Mohiuddin Qader and Abhinand Menon for the projects they took the lead. I'm grateful to Anastasios Arvanitis for his valuable input during our collaboration. Especially, Anastasios helped me a lot on my paper writing. I thank Dr. Michael Weiner's valuable contribution in our research project.

I have to thank all my labmates in the Database Lab at UC Riverside. They also contributed to build this collaborative, transparent and enjoyable work environment. Abhijith Kashyap and Eduardo Ruiz instructed me and gave me precious advices on research.

Matthew Wiley is very approachable for discussion and always willing to share his experience and knowledge with me. We also had many fruitful discussions with Moloud Shahbazi, Mohiuddin Abdul Qader and Abhinand Menon during our collaborations.

I would like to thank Zhen Qin for helpful comments and suggestions to improve this thesis. Moloud Shahbazi, despite her tight schedule, spent a large amount of time to read the thesis in detail and offered numerous corrections.

Finally, I would like to thank my family for all the unconditional support. My beautiful wife, Junrong Lei, is always supportive and helpful in every aspect of my life. I'm forever in her debt.

To my wife and parents for all the support.

ABSTRACT OF THE DISSERTATION

Effective Exploration of Web and Social Network Data

by

Shiwen Cheng

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, June 2015
Dr. Vagelis Hristidis, Chairperson

The amount of Internet data is rapidly increasing due to the growth of the Web and the success of Online Social Networks. However, it is challenging to users to effectively explore these dynamic and massive data. Search engines offer a convenient way for users to explore Web and Social data through keyword query interfaces. However it is common that a keyword search query returns many results not relevant to the user's information need in terms of content, time or structure. It is hard for search engines to understand a user's information need purely based on the query keywords, especially when the query is ambiguous in nature. In Social Networks, subscription is another common way to consume data in additional to ad-hoc search, . For example, a user in Facebook and Twitter can subscribe to other users to have their real-time updates shown in her timeline. However, a user could be overloaded by the large number of posts in the subscription due to the high post rate.

In this dissertation we develop solutions to help users effectively explore Web and Social Network data: First, we study the role of the document creation time in Web search queries in relation to the freshness requirement of a query. Second, we estimate the

hardness of keyword queries over structured data. A search engine can use this technique to decide when to employ query suggestion and query reformulation techniques to offer better user experience. Third, we propose context-aware ranking models to improve the search of medical literature by leveraging user query sessions. The user query session can help to understand a user's information need. Finally, we apply novel diversification techniques on Online Social Network data to alleviate the information overload problem and help users to more effectively explore social data through search interfaces or a posts timeline.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Internet data is growing explosively in recent years due to the growth of the Web and thrive of Online Social Networks. Notable search engines (such as Google and Microsoft Bing) allow us to search the Web data with ease. Online Social Networks (e.g., Facebook and Twitter) have become popular where users can publish and consume social data conveniently. It was reported that Twitter users from all over the world post over 500 millions tweets per day to the date of this thesis[1].

Search engines evolved in the past decades to be a mature and standard tool to explore Web and Social data. In early stages of Information Retrieval, which is a core technique of a search engine, researchers focused on indexing and ranking algorithms that are only text based. It is now a broad research area that overlaps with Machine Learning, Data Mining and Natural Language Processing.

[1] https://about.twitter.com/company

In a convenient way, a user can issue a keyword query (normally consisting of only few keywords) to the search engine to retrieve desired information. However, finding desired information or keeping up with the high velocity of general information is still challenging. It is common for a keyword search query to return many results not relevant to a user's information need. One reason is that it is hard for the search engine to estimate the context of the query, such as location or time, purely based on the query keywords, especially for ambiguous queries. This scenario indicates a challenging research direction: understanding the user's information need behind a simple keyword query and making use of meta data of a document and context of a query for better ranking.

Further, in the "big data" era, not only the volume but data presentation also changes.. We can observe that more and more structured databases are accessible on the Web. Many domain specific search engines are built to explore these data. For example, Amazon offers powerful search engine to search products (whose information is in structured format). In these search engines, users still search with keyword queries, because keyword queries advantage in freeing users from knowing the schema of underline data. However, keyword query without schema information specified is ambiguous in nature to search structured data. Databases contain entities, and entities contain attributes that take attribute values. Some of the difficulties of answering a keyword query are as follows: First, users do not specify the desired schema element(s) for each query keyword. For instance, query *Godfather* on the IMDB database does not specify if the user is interested in movies whose *title* is *Godfather* or movies distributed by the *Godfather* company. Thus, a search engine must find the desired attributes associated with each term in the query. Second, the schema

of the output is not specified, i.e., users do not give enough information to single out exactly their desired entities [83]. For example, this query *Godfather* may return movies or actors or producers. It is important for a search engine to recognize such queries and warn the user or employ alternative techniques like query reformulation or query suggestions [88] or query results diversification [32]. This brings us another interesting research direction that is how to accurately and efficiently identify the ambiguous keyword queries over structured databases.

Exploration on Social Networks such as microblogging systems is facing unique challenges. Search engine is a popular and powerful tool to explore social data. Online Social Networks like Twitter, LinkedIn and Facebook provide powerful search engines to explore their data. Subscription is another common way to ingest social data in addition to search. A user can subscribe a set of topics or data producers to receive real-time updates from the social system. If we focus on microblogging systems, timelines (e.g., Twitter timeline) is a common way of consuming subscription data. Typically, the timeline of a user consists of all kinds of updates from her friends (definitions vary in different social networks). For instance, the timeline of a Twitter user is formed by the all tweets posted by the followees of the user with tweets displayed in time order. Given that user's social connection keeps expending, it is now common that user's timeline is updated very frequently. However, in microblogging systems, search and timeline share the same issue that users are overloaded by the high rate of produced microblogging posts. These posts often do not carry any new information and are redundant to other similar posts. This motivates us to find better solutions for users to explore social data.

1.2 Research Problems

In this thesis, I follow two main directions to help users to effectively explore Web and Social Network data. First, I studied methods to better understand a keyword query, which in turn help users to more effectively search Web and Social Network data. Second, I proposed solutions to overcome the issue that users are overloaded by social posts from search or subscriptions.

Estimating the Timelineness of Web Search Queries

Follow this first direction, I specifically studied three problems in this work. In the first problem, I studied the time factor in Web search to better understand the freshness demand behind a keyword query.

Previous works have shown that timeliness is a key aspect for determining the relevance of a web document to a search query [77, 64, 41, 31]. For instance, a user who issues a query for "US elections" in November 2012, is more likely to be interested in web pages related to the 2012 elections. Although there exist several web pages about previous elections that are highly topically relevant to the user's query, it is expected that boosting the more recent documents will improve the retrieval quality. Previous works have proposed time-aware retrieval models with particular focus on *news queries*, where recent web documents related with a real-world event are generally preferable. These queries typically exhibit bursts in the volume of published documents or submitted queries.

Different from previous work, we study a class of queries, named *timely query*, that have no major spikes in either document or query volumes over time, yet they still favor

more recently published documents. For example, query credit card overdraft fees is not a news query and is shown to have steady query and document volume over the time. But from a user's perspective, more recent results are more preferable because the user would not like to see out-of-date bank policies for overdraft fee. Our goal is to accurately estimate the query timeliness and incorporate query timeliness into the ranking model to promote recent results in an appropriate degree.

We describe the details of this work in Chapter 2.

Detecting Difficult Keyword Queries over Databases.

In this problem, I focused on database data (or more generally, structured data). We analyze the properties of ambiguous queries over databases and propose novel methods to detect such ambiguous (difficult) queries.

To the best of our knowledge, there has not been any work on predicting or analyzing the difficulties of queries over databases. Researchers have proposed some methods to detect difficult queries over plain text document collections [115, 131]. However, these techniques are not applicable to our problem since they ignore the structure of the database. In particular, as mentioned earlier, a search engine must assign each query term to a schema element(s) in the database. It must also distinguish the desired result type(s).

As a specific example of how knowledge of the query difficulty may be leveraged, consider Figure 1.1, which shows the ranking results for query *ancient Rome era* by one of our implemented ranking algorithms (details in Section 3.8). Our algorithms determine that this is a hard (ambiguous) query, which guides the system to generate query reformulation

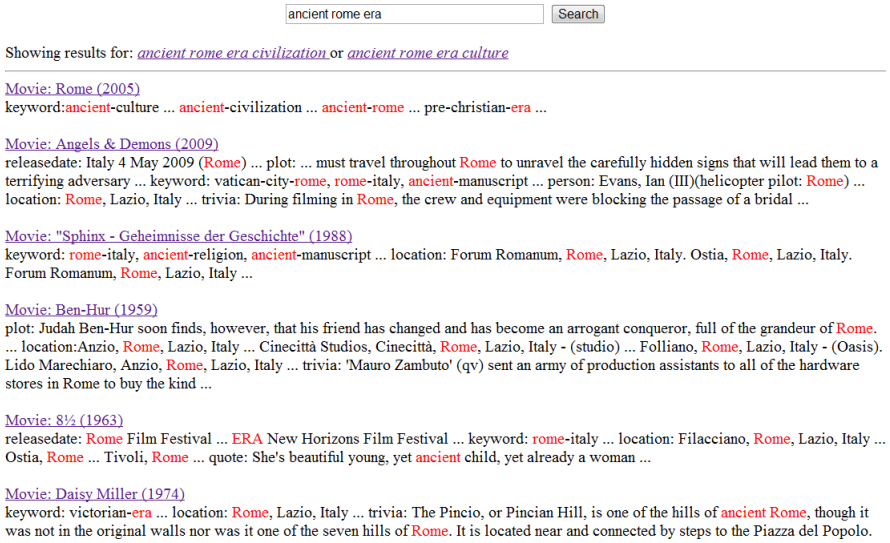


Figure 1.1: Results for query *ancient Rome era* with query suggestions returned.

suggestions. Clarify that the generation of query reformulation technique is beyond our scope in this work.

The study of this problem is presented in Chapter 3.

Leveraging User Query Sessions to Improve Keyword Query Search.

Further, we studied to leverage query session data to better understand the information need behind a query. Specifically, in this problem we focus on searching biomedical data.

It is reported that millions of queries are issued each day on the PubMed system to search medical literature [36]. Due to the importance of searching PubMed, a large number of Web-based applications have been deployed for this [79]. The Text Retrieval Conference (TREC) has included tracks for this domain of data, e.g., the TREC Genomics Tracks in 2006 and 2007 [58, 57]. One interesting fact is that PubMed users reformulate their queries

averagely 4.44 times during a query session [36]. However, to the best of our knowledge, no research has been conducted to utilize query sessions to search biomedical data.

Research in IR community has shown that the past queries in a query session can help understand the user's search topic [124]. The context-aware ranking principle indicates that for two associated (in the same query session) consecutive queries Q_{k-1} and Q_k, the user is likely to prefer the search results related to both queries, and thus such results should be promoted for Q_k [124]. Thus, a users query session can be used to define the *query context* for the current query, and this context can be used to improve the ranking of the returned results.

In this problem, we utilize the context-aware ranking principle in searching medical literature. That is, we incorporate the user query session data (as the query context) to improve the search of medical literature. We aim to build the context-aware search strategies by extending popular ranking algorithms (such as BM25 and Language Model) to incorporate query context in principled ways. We give details for the models in Chapter 4.

Multi-Query Diversification in Microblogging Posts

I worked on two problems to help users more effectively consume social posts and alleviate the data overload issue. In the first one, I applied novel results diversification techniques on microblogging posts. The proposed model can be applied in different applications in a microblogging system, such as microblogging post search and topic subscription. The Multi-Query Diversification in Microblogging Posts (MQDP) problem I studied is abstracted as follows.

Given an input consists of a list of microblogging posts and a set of user queries

(e.g. news topics), where each query matches a subset of posts. Our objective is to compute the smallest subset of posts that *cover* all other posts with respect to a "diversity dimension" that may represent time or, say, sentiment. Roughly, the solution (cover) has the property that each covered post has nearby posts in the cover that are collectively related to all queries relevant to this covered post.

This is distinct from previous single-query diversity problems, as we may have two nearby posts that are related to intersecting but not nested sets of queries, in which case none covers the other. Another key difference is that we do not define diversity in terms of post similarity, since posts are short and thus it is challenging to measure their similarity effectively; instead, we focus on finding representative posts for ordered diversity dimensions like time and sentiment, which are critical in microblogging. For example, for time as the diversity dimension, the selected posts will show how certain news events unfolded over time.

We introduce formal definition of MQDP and its variation and our solutions for them in Chapter 5.

Multi-Dimensional Diversification on Social Post Streams

We assume users subscribe several queries (topics) in the MQDP problem. However, in practice users in a social system are more often subscribing to data producers instead of topics. For example, people can subscribe news agencies' RSS feeds to receive instant news updates. Google Scholar continuously recommends new scientific articles to its users based on a users profile and publication history. A user in Twitter can subscribe to other users posts by following them.

Under this new setting, I studied on the problem of diversifying social post stream: given a social post stream consisting of all the posts from a user's subscribed data producers, our aim is to output in real-time a subset of the stream in which *(i)* all posts are dissimilar to each other and *(ii)* any post in the whole stream will be either included or *covered* by a post in the sub-stream. A post covers another post if the two posts are similar in all three similarity dimensions: (a) content, (b) time and (c) author.

One challenging requirement is that we have to compute the substream in *real-time*, i.e., immediately decide whether a post should be included or not at its arrival. That is, we cannot first view the whole stream and then decide which posts should be included in the substream.

In this problem, we do not have user inputted queries as in MQDP where the set of queries are guiding the content diversity. Instead, we have to measure inter-post content similarity. Given the requirement to make real-time decision at each post's arrival, we cannot afford to use traditional content similarity measures such as cosine similarity. Instead, we turn to hash-based distance measures. The *author* similarity is a subtle dimension that to the best of our knowledge has not been used before for computing diversity in social media. For example, CNN and Fox News, which both have official Twitter accounts, are dissimilar to each other because they generally have different political views. We compute the distance between two authors through their social connections.

There has been much work on diversifying search results [97, 7, 19, 32, 39]. However, none of these works can be applied to our setting where: (i) data is streaming and an instant decision must be made on whether a post should be pushed to the user, and (ii)

a multi-dimensional diversity model is adopted. This stream diversification problem also differs the Twitter stream summarization problem studied in previous work [112, 104, 125]. First, we do not aim to show aggregated tweets to user. Further, we define strict coverage constraints to guarantee that not even one uncovered posts is missed.

We elaborate the problem and our solutions in Chapter 6.

Chapter 2

Estimate Query Timeliness for Web Search Queries

Researchers have recognized the importance of utilizing temporal features for improving the performance of information retrieval systems. Specifically, the timeliness of a web document can be a significant factor for determining whether it is relevant for a search query. Previous works have proposed time-aware retrieval models with particular focus on news queries, where recent web documents related with a real-world event are generally preferable. These queries typically exhibit bursts in the volume of published documents or submitted queries. However, no work has studied the role of time in queries such as "credit card overdraft fees" that have no major spikes in either document or query volumes over time, yet they still favor more recently published documents. In this work, we focus on this class of queries that we refer to as "timely queries". We show that the change in the terms distribution of results of timely queries over time is strongly correlated with the

users' perception of time sensitivity. Based on this observation, we propose a method to estimate the query timeliness requirements and we propose principled ways to incorporate document freshness into the ranking model. Our study shows that our method yields a more accurate estimation of timeliness compared to volume-based approaches. We experimentally compare our ranking strategy with other time-sensitive and non time-sensitive ranking algorithms and we show that it improves the results' retrieval quality for timely queries.

2.1 Introduction

Previous works have shown that timeliness is a key aspect for determining the relevance of a web document for a search query [77, 64, 41, 31]. For instance, a user who issues a query for "US elections" in November 2012, is more likely to be interested for web pages related to the 2012 elections. Although there exist several web pages about previous elections that are highly topically relevant to the user's query, it is expected that boosting the more recent documents will improve the retrieval quality. Under this assumption, several techniques have been proposed to incorporate the time dimension in the ranking model. Previous approaches can be broadly categorized into: *(i)* those that focus on news queries related with real-world events, favoring recent documents [35, 37, 38, 41], and *(ii)* those that target general time-sensitive queries, where the results are preferably published during a specific time range [64, 31].

For both query types, it has been found that the most relevant time range is usually associated with spikes in the number of related user queries, or in the volume of

related documents published during that time frame. Based on this observation, current approaches analyze the time series of related queries or documents, in order to *(i)* classify queries as news queries or not [37] and *(ii)* to identify important time periods [64, 31]. Thereby, they boost the ranking of documents published around that time frame. For instance, a user searching for "Boston Marathon" might be looking for information about the terrorist attack that took place during the marathon on April 15, 2013; then documents published around that date are promoted.

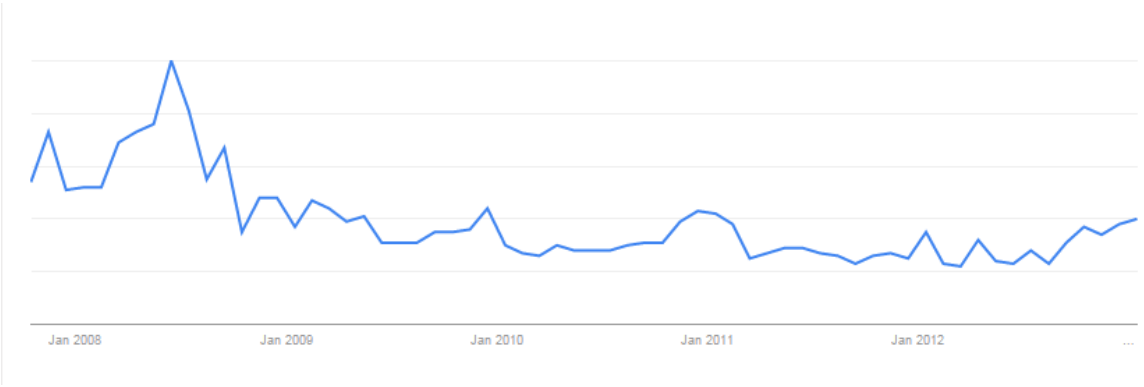


Figure 2.1: Number of search queries for "fashionable haircuts" over time

For a large number of search queries however, recent studies [71, 96] have shown that, while freshness of query results is still very important, their query or document time series are either unclear or misleading. In particular, several queries exhibit no, multiple, random or periodic spikes in query popularity over a time period [71]. For instance consider a query like "credit card overdraft fees". For such a query, the number of published documents and/or search queries remains more or less constant over time, normalized with the total query volume. However, more recent documents are clearly more relevant because, due to policy changes on behalf of banking institutions, information contained in older doc-

uments might no longer be valid. Other queries, exhibit a seasonal or unpredictable rise in popularity. A query such as "tax preparation tips" usually has a burst in popularity around mid April every year; on the other hand a query for "Rihanna new single" follows a more irregular trend pattern.

In general, for this type of queries, all other things being equal, a fresher document is probably much more relevant for a user's search query. However, existing volume-based techniques cannot be applied for such queries, since the number of published documents or issued queries cannot be leveraged as an indication of users' interests. In fact, the number of published documents or search queries might as well be negatively correlated with the users' demands. For instance consider a query such as "long distance phone calls prices" or "fashionable haircuts". Figure 2.1 shows the number of queries in a commercial search engine over years, normalized by the total query traffic during each time period, for the query "fashionable haircuts". As depicted, user searches for this query have dropped during the recent years. At the same time, the keyword "fashionable" indicates that users are interested on new hairstyles and that web pages related to older ones are no longer relevant, which indicates that it is a time-sensitive query. For this query, an approach that focuses only on the query or document volume might instead give higher ranking to older documents.

Moreover, different queries can have very different degree of freshness requirements. For example for a query like "smartphone reviews", users might consider as relevant a document that is up to 6 months old, whereas for a query like "new movies in theaters" the time range of interest is much shorter, typically 1-2 weeks. For the reasons mentioned

above, volume-based approaches will not be able to capture different freshness requirements. Further, in addition to identifying the timely queries, we must also quantify how timely a query is.

In this work we will focus on this important type of queries, which we refer to as *timely queries*. Timely queries have the following properties: *(i)* the interest on the query from document publishers or consumers does not show significant variance over time (normalized by the total query traffic or document volume respectively), and *(ii)* more recently published documents are strongly preferred over older ones. Since, the popularity of such queries can be steady, previous approaches are not effective.

Motivated by the above challenges, in this work we propose a different approach to measure the freshness requirements of a user's query, i.e., the *query timeliness*. In particular, we argue that the users' freshness requirements from search results are strongly correlated with the degree of content change in the relevant documents. In other words, if the terms distribution inside the most relevant documents changes significantly over time, this indicates that older documents become stale shortly, and hence, they should be penalized with lower relevance scores.

In this work we experimentally confirm this correlation. That is, if we identify significant content change in the relevant documents across time, then time becomes a major factor in our proposed ranking. We incorporate the query timeliness in our retrieval model in a principled manner by extending previous works on time-based language models [77, 41].

Note that, since our ranking model always favors more recent documents, our

approach is not meant to handle news queries related with events that took place in the past. For example, for a query like "Supreme Court healthcare act", results around June 28th 2012 are more relevant since this is when the Supreme Court ruled.

Contributions. The contributions of this chapter can be summarized as follows:

- We show that the change in the terms distribution of results of timely queries over time is strongly correlated with the users' perception of time-sensitivity.
- We propose principled ways to incorporate document freshness into the ranking model.
- We experimentally show that our proposed model improves the quality of the results for timely queries.

Outline. Section 2.2 presents related work on temporal ranking of search results. In Section 2.3 we discuss query timeliness and we provide a method to measure timeliness based on the terms distribution. In Section 2.4 we present our time-aware ranking model by incorporating the concept of query timeliness. Section 2.5 contains an experimental evaluation of our methods. In Section 2.6 we discuss how our proposed model can be applied in practice. Finally, in Section 2.7 we draw conclusions and sketch our future work.

2.2 Related Work

Li and Croft [77] were the first that proposed a time-based language model to boost the ranking of more recent results. In particular, they introduced an exponential decay prior to the query likelihood language model, such that more recent documents are assigned a higher probability. Based on the proposed ranking model, experiments show

significant improvements in retrieval quality for TREC queries that are related with recent events.

However, their proposed ranking model treats all recency queries as having the same freshness requirements from the search results. That is, they boost the ranking of recent results uniformly for all queries. Efron and Golovchinsky [41] improve upon this model [77] by proposing a query-specific exponential reranking method. In particular, they calculate a maximum likelihood estimator per query based on the time distribution of the most relevant results, as returned by a non time-aware ranking. A significant limitation with such an approach is that it will boost recent documents, only as long as the document volume increases over time, i.e., it can be applied mainly on news queries and not generally on time-sensitive ones. In contrast, in this work, we provide a ranking model that can be used for different query types, which is not dependent on the distribution of documents over time or the underline (non time-sensitive) ranking. We experimentally compare to both methods [41, 77] in Section 2.5 and we show that our method yields better retrieval quality for timely queries.

Some works [64, 31] focus on more general time-sensitive queries, where the results are preferably published during a specific time range. Jones and Diaz [64] propose building a time series on the number of top ranked documents of the query. According to the detected number of spikes in the time series, they classify queries into three classes: *atemporal*, *temporally ambiguous* and *temporally unambiguous*, which represent queries that exhibit no spike, only one spike and more than one spikes in their document volumes, correspondingly. The class of queries that we study in this work would be classified as atemporal by [64],

since these queries do not exhibit major volume spikes (see Section 2.5.2). Therefore if we followed this approach, the freshness of the results would not be considered as important. Dakka et al. [31] propose alternative methods to learn the most relevant time period for a query, and present solutions to incorporate temporal relevance into several popular ranking algorithms. Both [64, 31] rely on the spikes in the distribution of relevant documents.

It has been empirically shown [77, 41] that applying a time-aware ranking model can generally harm the retrieval quality of non time-sensitive queries. In order to address this problem, Dai et al. [30] introduce a machine learning framework for simultaneously optimize both relevance and freshness of results, by utilizing both temporal and non-temporal document features.

Another approach is to automatically identify whether a search query is time sensitive or not. If a query is not time-sensitive, standard relevance methods can be used instead. Dong et al. [37] use machine learning techniques to classify a query as breaking news query or not. For this purpose, they measure the difference in the query probabilities in various time slots in the past, such as the last day, last week and last month. The probabilities are calculated based on the language model of both the query log and the document collection. If the query is classified as a breaking news query, the freshness of a document becomes important in ranking. Similarly to some of the above methods [64, 31], the underlying intuition is that in a specific time range, the results are much different from a regular search, for instance due to a burst in the number of relevant documents or due to the query being popular in the query stream. Similarly, using features such as the probabilities of the query generated from recent content, Styskin et al. [111] train a linear regression

model to predict a probability for the query's freshness preference and they combine fresh documents with regular ones in order to enhance the temporal diversity of the results.

Assuming a news query, several works deal with how to improve the temporal relevance of returned results. Dong et al. [38, 20] extend former work [37] by enriching the results with documents discovered in the Twitter stream. For this purpose, they extract a set of features from both a regular documents' corpus and a tweets collection, and they learn a ranking model in order to merge recent tweets with regular results. Diaz [35] and König et al. [69] study the utility of showing news results among regular ones by using click-through data.

Elsas et al. [42] study the temporal factors of *navigational* queries, where there is usually a small number of highly relevant documents that are consistently relevant across time. For this type of queries, they experimentally show that there is a strong positive correlation between the relevance of a document and the frequency of the document's content change. However, within the same document, terms that are present across different time ranges are more important in estimating the overall document's relevance. Thereby, they propose the use of a document-specific prior in order to favor more dynamic documents. Our work has a similar motivation, i.e., to leverage the amount of content change in recency ranking. However, we focus on more general informational or transactional time-sensitive queries where more recent (and not necessarily highly dynamic) documents are preferable. Further, we measure the content change in the query rather than in the document level.

2.3 Estimating the Query Timeliness

As we already noted, different queries can have very different timeliness degrees, i.e. requirements on the freshness of the search results. For instance, for a query such as "top graduate schools", a user might find results up to two years old as relevant, whereas for a query such as "Universal Studios coupon", she would be interested only on search results of the last few weeks, since older coupon offers will probably have already expired. Thus, the challenge is how to identify the appropriate timeliness degree for a given search query.

Previous works have used the query volume and the number of published documents as an indicator of the timeliness of a query. The most relevant work to our problem [41] computes a query-specific freshness parameter that is calculated based on the distribution of the publication times of the top k results that would be returned by a non time-sensitive ranking function. However, as we will demonstrate in our experiments in Section 2.5, the proposed approach [41] fails for the class of queries that we study in this work, i.e., those having a relatively steady document volume, such as the one shown in Figure 2.1.

In order to overcome this problem for the class of queries (timely query) we study in this work, we introduce a new method to estimate query timeliness. In particular, we propose to use the degree of change in the content of the most relevant documents of a query, as a measure of the timeliness of the query. Kullback-Leibler (KL) divergence [72] is a popularly used measure to compute the divergence of text documents [85, 60]. In this work, we apply KL divergence to measure the changes of text documents, i.e. we calculate

the difference in term probability distributions of text documents using the KL divergence:

$$KL(P, R) = \sum_{t} P(t) \log \frac{P(t)}{R(t)} \tag{2.1}$$

where P and R are two probability distributions and $P(t)$, $R(t)$ denote the probability of term t in distributions P and R, respectively.

For a query Q, we define the degree of content change between two time slots as the difference in the probability distributions between the sets of documents relevant to Q in these time slots. For simplicity, hereafter we assume discrete time slots, which might denote weeks, months etc. Further, let T_i represent the set of documents that are relevant for query Q, and were published during time slot t_i (e.g., during July 2012). We will also symbolize as $LM(T_i)$ the language model produced by T_i. Then, we define the degree of content change between two time slots t_i, t_j as $KL(LM(T_i), LM(T_j))$.

Assuming n consecutive time slots $t_1, \cdots, t_n$, we define the terms distribution change for a query Q, denoted as $TDC(Q)$ as:

$$TDC(Q) = \frac{1}{n-1} \sum_{i=1}^{n-1} KL(LM(T_i),\ LM(T_{i+1})) \tag{2.2}$$

i.e., we take the average KL-divergence acquired from consecutive pairs of time slots. For calculating $LM(T_i)$ we will apply a unigram language model approach.

Several other measures have been proposed in order to quantify the amount of content change (the opposite of similarity) between documents, such as the cosine similarity, Dice similarity, and Jaccard distance.

Previous work [60] evaluated the effectiveness of these measures for computing the similarity of text documents for document clustering. The results have shown that

all measures deliver similar results with KL divergence, and that differences depend on the particular characteristics of the document collection. Potentially, any of this measures could be used in our model to replace KL divergence.

2.4 Incorporating Freshness into the Ranking Model

In developing our time-aware ranking, we follow the language ranking model [94]. Li and Croft [77] were the first that proposed a ranking that incorporates a temporal dimension into the language model. According to the query likelihood approach, the probability that a document d is relevant to a query Q, $P(d|Q)$, is proportional to *(i)* the probability of deriving Q based on the language model of d, termed as $P(Q|d)$ and *(ii)* an apriori probability of document d that depends on the publication date T_d, termed as $P(d|T_d)$:

$$P(d|Q) \propto P(Q|d) \cdot P(d|T_d) \tag{2.3}$$

In particular, in order to compute $P(d|T_d)$, they assume an exponential decay calculated as:

$$P(d|T_d) = \lambda e^{-\lambda \cdot \Delta t_d} \tag{2.4}$$

where Δt_d is the normalized age of document d, measured as the time distance between T_d and the date of the most recent document in the document collection. Note that Li and Croft [77] use the same freshness parameter λ for a set of queries that is manually identified as time-sensitive, regardless of the different degrees of timeliness requirements each query has. Thus, Efron et al. [41] improve [77] by proposing to use a query-specific λ_Q that is calculated based on the time distribution of relevant documents as returned by a non time-sensitive ranking model.

In this work we use Equations 2.3 and 2.4 as a starting point, and we modify them in order to consider the right amount of freshness for each query using the timeliness requirement estimation method proposed in Section 2.3. Specifically our ranking model assigns scores based on the following function:

$$Score(d, Q) = BM25(d, Q) \cdot \lambda_Q e^{-\lambda_Q \cdot \Delta t_d} \tag{2.5}$$

where $BM25(d, Q)$ denotes the ranking score of document d for a query Q by the popular ranking function Okapi BM25 [93], which is based on the probabilistic model. Based on our ranking model, λ_Q depends on the timeliness requirements of each query as measured by the amount of content change (Section 2.3). Intuitively, we would assign larger values of λ_Q for queries having higher timeliness degrees, as predicted by $TDC(Q)$. Larger values for λ_Q will result in penalizing the scores for older documents, thus favoring the most recent ones. We calculate λ_Q as:

$$\lambda_Q = \alpha \cdot (1 - e^{-TDC(Q)}) \tag{2.6}$$

where $\alpha > 0$ is a parameter of the ranking model and $1-e^{-TDC(Q)}$ is the KL divergence score normalized in (0,1][1]. We will assume a constant value of α for all queries. In Section 2.5.3 we provide more details on how we set α for experiments; in Section 2.6 we explain how α can be set up in practice.

Since the calculation of λ_Q is based on $TDC(Q)$, we will refer to the ranking model in Equation 2.5 as the *Timeliness-Aware Ranking* (TAR). In Section 2.5.3 we experimentally evaluate the retrieval quality of the proposed ranking function with the previously proposed time-aware ranking models [77, 41].

[1]We use a normalized KL divergence score since the original KL divergence is unbounded.

2.5 Experimental Evaluation

In this section, we provide experimental results on the ranking quality of our methods. We first present the datasets that we used for our experiments in Section 2.5.1. Sections 2.5.2 and 2.5.3 contain the experimental evaluation of the proposed timeliness estimation method and timeliness-aware ranking respectively.

2.5.1 Datasets

Query workload. In order to build our query workload we considered the Text REtrieval Conference (TREC) datasets (e.g. TREC Web Tracks [5]). Unfortunately most of the available datasets contain only a few timely queries. Thus, we manually created some additional queries that we considered as timely, following an approach similar to [41]. In particular, we asked 10 graduate students to suggest queries having diverse timeliness requirements. The complete query workload consists of 119 queries (taken from TREC or proposed by students) and is available at [3], in which we specified the queries from TREC.

Documents. For our experimental evaluation we also needed documents published during different time ranges and relevance judgements that take into account both the topical and the temporal relevance of results; however these data where not available in TREC dataset. Hence, we constructed our document collection by submitting each of the queries to a commercial search engine and we conducted a user relevance study as explained in Section 2.5.2.

In order to collect documents published at various time ranges, we specified different start and end dates in our search, such that the returned results contain only documents

that were published during the specified time frame. Note that accurately identifying the publication date of each document is itself a challenging task [8, 37, 18]; moreover often the publication time and the time associated with the content contained in a document might differ. The search engine that we used for our experiments tries to estimate the publication date for each web page by using features such as the date when it was first crawled, or a byline date or an explicitly specified date of a news article or blog post if such information is available. For simplicity, hereafter we will assume that all documents returned by the search engine have been published during the specified time period. Following this method, we retrieved the top 400 results per year for years 2007-2011 and for the first half of 2012, i.e., we obtained for each query 2400 documents in total.

2.5.2 Estimation of the Query Timeliness

In the first set of experiments we compare the performance of our proposed method for estimating the query timeliness with the previous approaches that focused on the document [64, 37, 41, 31, 92] and query volume change [37] as discussed in Sections 2.2 and 2.3.

User Survey on Query Timeliness

In order to calculate the degree of correlation between the document volume and the users' perception of query timeliness, we set up a user survey to collect judgments w.r.t. the timeliness demands for each query on behalf of the users. In addition to the 10 graduate students, our survey's subjects include users recruited through the Amazon Mechanical Turk [1]. In particular, we forwarded all queries to the graduate students, and

60 queries to each Amazon Mechanical Turk worker (110 workers in total). For each query, we asked the users to select among the following five options: *no time preference, up to 2 years old, up to 6 months old, up to 1 month old, up to 1 week old* the one that best describes their preferences in terms of freshness of the search results. In order to increase the quality of the user survey: *(i)* we disqualified low-quality workers from our experimental study as explained in the Appendix, and *(ii)* we filtered out the 5 highest and the 5 lowest outlier timeliness values for each query.

For each query we collected 10 timeliness judgments from students and 20 valid timeliness judgments from Amazon Mechanical Turk workers. Next, for each query we calculated the average Timeliness Requirement in months. For this purpose we mapped each label to a specific number in months that represents the respective timeliness class. Since the maximum age of any document in our collection is 5.5 years, we mapped each label for *no time preference* to 66 months. Similarly we mapped the other timeliness ranges to 24, 6, 1 and 0.25 months respectively. Then we took the average over all judgments, which we will refer to as $TR(Q)$. Figure 2.2 shows some representative queries, along with the respective average timeliness requirements, as specified by the users. The query ids are taken from the complete list of the query workload [3].

As shown, users have very low freshness requirement for queries such as "public speaking tips" and "interview thank you letter". On the other hand, according to our user survey, users find the results published in the last 1-2 years for "passport renewal", "cancel a new car contract", or "low income housing" as relevant. Queries such as "retail sales index" have relevant results published in the last 6 months. Further, results published

#	**Query**	**TR(Q)**	**TDC(Q)**
Q88	public speaking tips	66.000	0.163
Q56	interview thank you letter	61.800	0.138
Q83	passport renewal	21.050	0.180
Q16	cancel a new car contract	17.158	0.193
Q65	low income housing	12.421	0.217
Q89	reality TV stars	11.158	0.267
Q57	keyboard reviews	8.474	0.306
Q90	retail sales index	6.316	0.287
Q95	smartphone reviews	3.500	0.298
Q15	California state parks jobs	2.842	0.335
Q79	newest tablet	1.670	0.363
Q17	celebrity gossips	0.868	0.326
Q75	NBA game schedule	0.838	0.319
Q76	NBA scores	0.408	0.454
Q14	California lottery results	0.288	0.382

Figure 2.2: TR(Q) and TDC(Q) of example queries

during the last month are considered as relevant for queries such as “newest tablet” or “celebrity gossips”. Finally, for other queries such as “NBA scores” or “California lottery results” the relevant documents typically change per week, and users are looking for up-to-date information, as it is confirmed by the user survey.

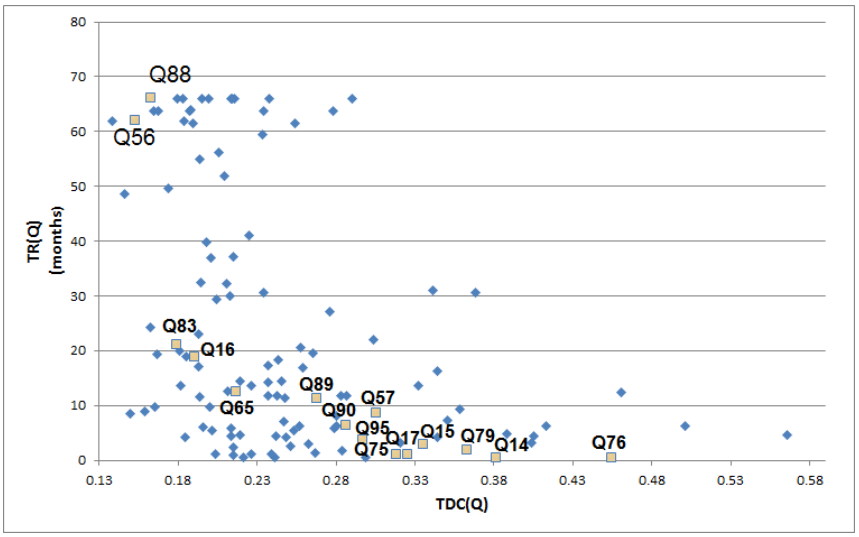


Figure 2.3: TR(Q) vs. TDC(Q) based on timeliness judgments

Timeliness Estimation based on Volume-based Approaches

Studying the Document Volume Change. We first examine to what extent previous approaches [64, 37, 41, 31, 92] can predict the timeliness of timely queries. For this experiment, for each query in our workload, we issued a web search where we also specified an one month time range. For each query, we retrieved the number of documents returned by the search engine for each month range, for each of the last 66 months (5×12 months for years 2007-2011 and 6 months for 2012).

Since the size of the web grows over time, we need to normalize the total number of documents per month, with the size of the web at the time. Since the total size of the (visible) web is unknown, we assume that its size can be approximated by a search query that returns as many relevant documents as possible. In particular, we issued a set of stopwords queries[2] and we calculated the average number of returned documents over all

[2]Each query consists of one of the following stopwords: *a, the, and, to, of*

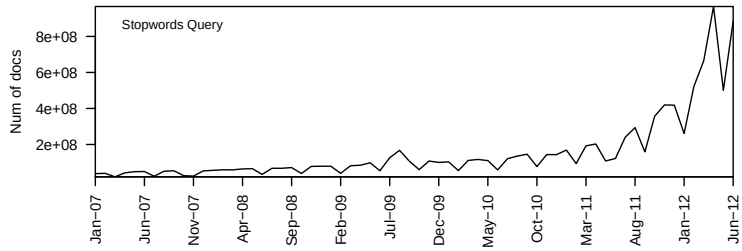


Figure 2.4: Time series of the number of documents returned for stopwords queries

issued queries. Then, we used this number as an approximation of the size of the web for a specified time range. Finally, we normalized the document volumes for each query with the number of results of the stopwords query for this month, as shown in Figure 2.4. Figure 2.5 plots the normalized document volumes for some representative queries.

Anecdotal Examples. The queries "NBA lockout" and "Occupy Wall Street" are news queries that were not included in our query workload. The other two queries "MySQL cluster setup" and "Firefox updates" are from our query workload. By comparing the document volume time series of the news queries with our timely queries, we can observe that news queries have a peak during the specific time when the news event happened. For instance, the time series line for "Occupy Wall Street" suddenly hikes during the end of 2011, whereas the time series for "NBA lockout" exhibits an increase on document volume during the second half of 2011. In contrast, the time series of the document volumes for timely queries might not have any significant spikes. For instance, "MySQL cluster setup" query does not have any spikes and "Firefox updates" has spikes with insignificant variance.

Correlation of Timeliness to Document Volume Change. In order to calculate the correla-

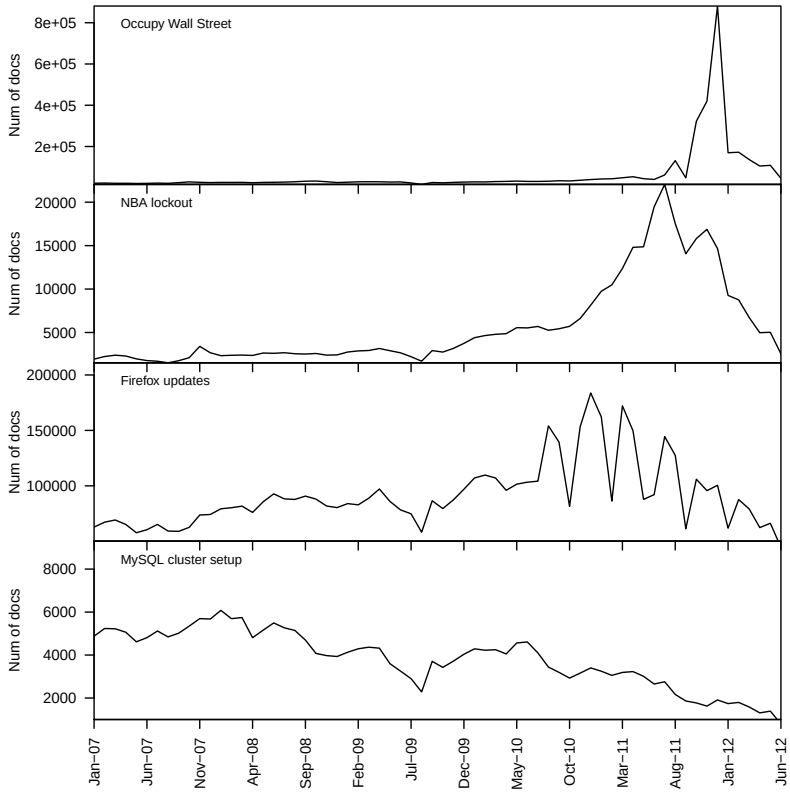


Figure 2.5: Time series of normalized documents number returned for example queries

tion between the document volume change and query timeliness we followed two different approaches based on how we measure the volume change.

First, we used the number of documents published during each of the 66 monthly slots that we considered in our experiments. Then, for each consecutive pair of months we calculated the absolute change in volumes. We also calculated the average number of documents per month, and we used it in order to normalize the differences between months. Finally, we calculated the Pearson correlation coefficient between the normalized document volume changes and TR across all queries in our query workload. The calculated Pearson correlation coefficient is -0.298. Note that the computed correlation value is negative because

larger document volume changes result in more rapid change of the relevant information w.r.t. a query, which means that only the most recent documents should be considered as relevant. In that case the query would have a lower average timeliness value $TR(Q)$.

As a second measure, for each of the examined 119 queries we calculated the coefficient of the variation of its monthly time series, as:

$$CV = \frac{\sigma}{\mu} \tag{2.7}$$

where σ and μ represent the standard deviation and mean of each time series. Similarly, we calculated the Pearson correlation coefficient between CV and TR across all queries and we found a correlation equal to -0.281.

Studying the Query Volume Change. Next, we examined the degree of correlation between the query volume change and the users' perception of timeliness. We built a time series of query volumes, by using the service provided by Google Insights [4]. We issued each query by specifying a date range of 5.5 years. Google Insights could provide monthly query volumes only for 87 out of the 119 queries issued when we collected these data. The results provided by Google Insights are already normalized with the size of the query traffic on Google. Figure 2.6 shows the time series constructed for several representative queries.

Anecdotal Examples. As shown, the query volume time series of "Occupy Wall Street" and "NBA lockout" have quite different behavior compared with the other two queries taken from the query workload. Also note that the hikes in query volumes of "Occupy Wall Street" and "NBA lockout" are consistent with the hikes of their document volumes in Figure 2.5.

Correlation of Timeliness to Query Volume Change. Similarly to how we calculated the timeliness estimation quality for the documents volume time series, we correlate (i) the

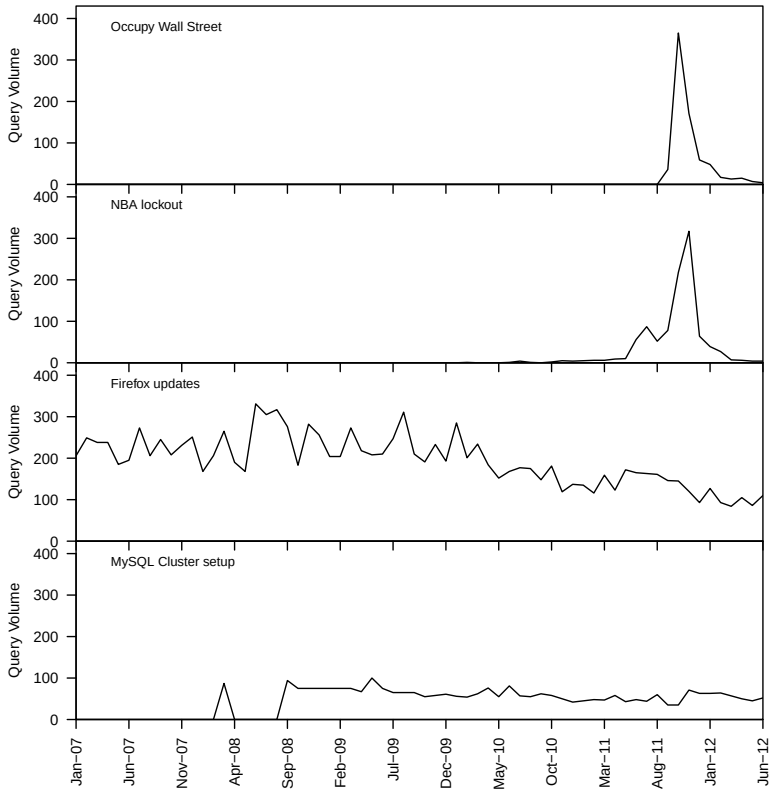


Figure 2.6: Time series of normalized query volume for example queries

normalized query change, and (ii) the coefficient variation of the query volumes time series of each query with the average timeliness requirement TR. The Pearson correlation coefficient calculated over the 87 queries was -0.132 using the normalized query volume change, and -0.130 using the coefficient of the variation.

Discussion. The relatively low correlation of both approaches shows that methods that leverage the document or query volume change are not suitable for timely queries, such as "MySQL cluster setup" and "Firefox updates", but can only be applied on news queries. Note that in addition to monthly, we also tried other range lengths with similar results.

Timeliness Estimation based on Terms Distribution Change

We now evaluate our method for estimating the timeliness of a query based on the terms distribution change in its relevant documents as presented in Section 2.3.

As discussed in Section 2.5.1, we collected 2400 documents per query over a period of 5.5 years. We created 6 document collections, where each collection contains all 400 documents retrieved for the corresponding time slot t_i. In order to build a language model for each document collection we concatenated all 400 documents into a single document T_i. Retrieving the relevant textual content from the HTML body of each document is a challenging and error-prone task [67]. In order to address this problem, mainly for performance reasons, we only considered the titles and snippets of each retrieved document.[3] Finally, when building each language model $LM(T_i)$, we ignored all common stopwords and all terms occurring fewer than 3 times over different time slots (we assumed that they are either typos or irrelevant terms). Then, based on the definition of $TDC(Q)$ in Equation 2.2, we calculated the terms distribution change for a query Q as:

$$TDC(Q) = \frac{1}{5} \sum_{i=2007}^{2011} KL(LM(T_i),\ LM(T_{i+1})) \tag{2.8}$$

Finally, we calculated the Pearson correlation coefficient between $TDC(Q)$ and $TR(Q)$ for all the queries of our workload and we got a correlation score -0.427. This score indicates that there is a strong correlation between the terms distribution change of the documents content and the users' perception of query timeliness. Further, it is much higher compared to the volume-based approaches that we presented in Section 2.5.2.

[3]Using the title and snippet instead of the textual content of a document can improve the efficiency and sometimes also the quality for some applications like search results clustering [127, 128] and query classification [12].

Note that instead of using one year as a unit to define the time slots, we also experimented with different time units. Because of the time range that we specified (5.5 years), there are not sufficient data in order to use a 2-year unit. Thus, we tried to use 1 week, 1 month, and 6 months units to build the language model $LM(T_i)$ for the most recent 10 time slots, and applied a similar method as above to calculate $TDC(Q)$. The Pearson correlation coefficients between $TDC(Q)$ and $TR(Q)$ based on 1 week, 1 month, and 6 months time slots are -0.298, -0.357 and -0.413 respectively, which are all higher than volume-based approaches in Section 2.5.2. As the $TDC(Q)$ score from yearly time slots yields the highest prediction quality, we will use the $TDC(Q)$ results from yearly time slots in the following experiments.

Figure 2.3 plots the $TDC(Q)$ and $TR(Q)$ values for all queries in our workload. Some representative queries (those shown in Figure 2.2) are labeled with different symbols and numbers. For instance, for query Q88: "public speaking tips" and Q56: "interview thank you letter", the relevant documents do not vary largely over time. Thus, the $TDC(Q)$ scores computed for these two queries are relatively small. According to our user survey, users roughly prefer the results published in the last two years for Q83: "passport renewal" and Q16: "cancel a new car contract", and last 1 or 2 months for queries such as Q79: "newest tablet". The $TDC(Q)$ scores linearly decrease according to their $TR(Q)$. For other queries such as Q76: "NBA scores" and Q14: "California lottery results", the relevant documents change very frequently, usually per week for the latter one or even everyday during the season time for the former one. This results in getting higher $TDC(Q)$ scores than other queries. At the same time, users are searching for up-to-date content, as it is

confirmed by the $TR(Q)$ scores. As shown, the users' perception of timeliness for all of the above queries is captured sufficiently using our method.

2.5.3 Improving the Retrieval Performance Using Query Timeliness

In Section 2.4 we proposed a principled way to incorporate timeliness into our ranking algorithm TAR. In this section, we experimentally evaluate the retrieval performance of TAR, compared with other time-aware and non time-aware rankings. We first describe the experimental setup, which is in addition to the setup described in Section 2.5.1. Subsequently, we present our experimental results in Section 2.5.3.

Experimental Setup

Datasets. For our retrieval evaluation experiments we used the 2400 documents that we collected during the timeliness estimation survey. Note that for the performance evaluation experiments instead of using the results' titles and snippets, we built an index on the actual HTML content of each web page.

Effectiveness Metrics. For our retrieval evaluation, we applied two widely used relevance metrics: precision and Discounted Cumulative Gain (DCG) [63]. In particular, we measured the precision and DCG on the top-n results, denoted as $Prec@n$ and $DCG@n$ respectively. Instead of $DCG@n$, we adopted the Normalized Discounted Cumulative Gain (NDCG), which is a normalization of DCG in the range [0, 1] and is calculated as:

$$NDCG@n = \frac{DCG@n}{IDCG@n} \tag{2.9}$$

where IDCG@n is the ideal DCG@n, i.e., the maximum possible DCG value up to the

ranking position n. DCG@n is calculated as [16]:

$$DCG@n = \sum_{i=1}^{n} \frac{2^{rel_i} - 1}{log_2(i+1)} \tag{2.10}$$

where rel_i denotes the binary relevance of the results at ranking position i, i.e., rel_i is equal to 1 if the result at position i is valid and 0 otherwise.

In our experiments we set $n = 5$, i.e., we measured Prec@5 and NDCG@5. In particular we calculated the average Prec@5 and NDCG@5 across the complete query workload [3].

Algorithms. In our evaluation we compared our TAR ranking with the following set of algorithms:

- BM25, the default (non time-aware) ranking provided by Lucene 3.5.0 [2], which uses BM25 ranking [93];
- BM25-T (Time-sorted BM25), which first retrieves the top-n documents based on BM25 and then sorts them by decreasing timestamp[4];
- EXP (Exponential time-based ranking) [77], which uses a constant exponential re-ranking rate for all queries, as defined in Equation 2.4;
- BEX (Bayesian EXponential ranking) [41], which calculates a query-specific exponential re-ranking rate based on the distribution of the top ranked documents obtained from a non time-aware ranking, as explained in Section 2.3.

Li and Croft [77] experimentally show that EXP achieves the best ranking quality by setting $\lambda = 0.01$. Therefore, we used this value for our experiments. For BEX, we used

[4]When we study the effectiveness of top-n results for BM25-T, we retrieve and sort top-n results from BM25.

the recommended parameter settings as described in [41], i.e., we set $k = 500$, $\rho = 100$, and we calculated σ such that $(\rho - 1)/\sigma = 0.015$ (see [41] for more details).

Relevance User Survey. We set up a user study to collect the relevance judgments for our dataset. For this purpose we applied a pooling method that has been popularly used to build test collections in TREC [120, 107]. We randomly mixed the top results from each of the above ranking algorithms and asked a set of workers on the Amazon Mechanical Turk to label the results that are the most relevant to each query. The workers were asked to make their judgments considering both the topical and the temporal relevance of the presented results. The the whole dataset is available at [3].

We retrieved the top-5 results for each ranking algorithm BM25, EXP, BEX and TAR with different values of α[5]. After taking the union of the top-5 rankings of all algorithms (duplicate results from different algorithms will only show once), for each query we got 17 unique document results on average. Note that to compare with other algorithms, we will report the retrieval quality of TAR based on a single value of α. We split our query workload into 6 groups of 20 queries each, such that each worker would have to provide relevance judgments for 20 queries. Thus, each user had to evaluate around 340 (17×20) query-document pairs. A document is considered as relevant to a query if it is labeled by over 50% of the workers that provided judgments for this query. Again, we disqualified some low-quality workers from our study as detailed in the Appendix. We finally assumed as valid only the judgments provided from the 104 most high-quality workers (among the initial 126 workers). Thereby, each query has been evaluated by 17.3 (high-quality) workers

[5]We include various values for α as we will study the setting of α for TAR. In addition, this helps to retrieve results with diverse publication dates to increase the effectiveness of pooling.

on average. In total, for our experimental evaluation, we considered 35248 query-document pairs, out of which 11637 are labeled as relevant.

Setting of α: Additionally, we need to set the parameter α in Equation 2.6 for our TAR ranking. For this purpose we conducted a 5-fold cross-validation to train and test it. In particular, we split our query workload into 5 sets following a lexicographic order. Thereby, 4 out of the 5 test sets consist of 24 queries and one consists of 23 queries. We experimented with the following values for α = 0.01, 0.03, 0.05, 0.07, 0.09, 0.1, 0.3, 0.5, 0.7, 0.9, 1, 3, 5, 7, 9 and 11, which are all included in the above user survey. Each row in Table 2.1 shows the value of α that achieves the best averaged Prec@5 on each training set, and the averaged Prec@5 and NDCG@5 scores based on this α for the respective testing sets. Note that we also conducted experiments using NDCG@5 as our retrieval quality measurement for training; since it produced similar results with Prec@5, we do not show the results of this experiment.

As shown in Table 2.1, the values of α that achieve the best Prec@5 are quite stable on different training sets. Thus, in order to evaluate the overall performance of our TAR method against the baseline algorithms, we calculated the average Prec@5 and NDCG@5 for all testing queries in our query workload under the setting α = 0.3.

Experimental Results

Measuring the ranking differences. First, we experimentally validate that the time-aware rankings yield quite different results compared to the BM25 ranking. For this purpose, we measured the normalized *Spearman footrule* distance [45] between each time-aware

Table 2.1: Cross-validation experiments for α using Prec@5

training set	α	Prec@5 (training set)	Prec@5 (testing set)	NDCG@5 (testing set)
1	0.5	0.383	0.325	0.410
2	0.3	0.371	0.383	0.472
3	0.3	0.375	0.367	0.393
4	0.5	0.371	0.375	0.425
5	0.3	0.371	0.383	0.477

ranked list and the BM25 ranking. Figure 2.7 shows the Spearman footrule distance between BM25 and BM25-T, EXP, BEX and TAR for different values of α, when considering the top-n results for $n = 5$, 10, 15 and 20. Larger values for the Spearman footrule indicate more disagreement between two lists. As shown, the ranking of search results changes largely when applying a time-sensitive ranking algorithm.

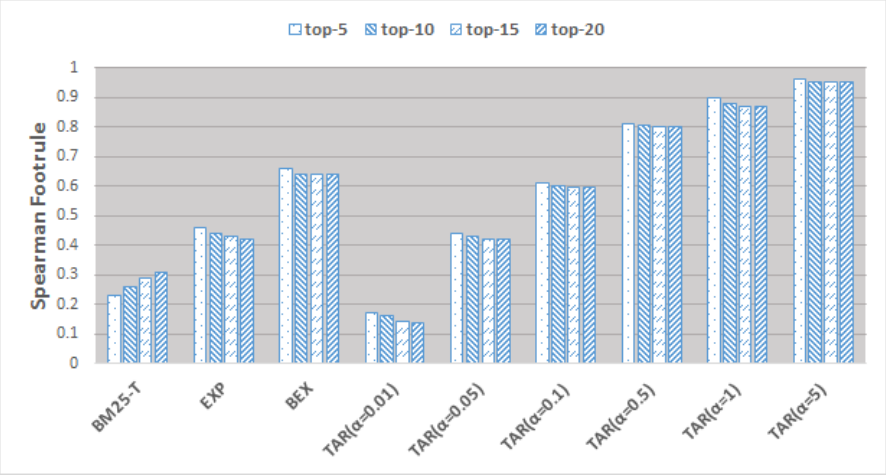


Figure 2.7: Spearman footrule between the BM25 and different time-based rankings for various top-n lists

Retrieval Quality. Table 2.2 compares the retrieval quality of TAR against the competitor methods for the complete query workload. As depicted, TAR achieves superior retrieval quality compared to all four competitor rankings. TAR performs 15% and 11% better than BEX in terms of average Prec@5 and NDCG@5, respectively. Further, BEX has better performance than EXP, which is consistent with the experimental findings in the original paper [41]. BM25-T delivers better NDCG@5 than BM25 because of the property of timely queries: more recently published relevant documents are preferred. Note that, as we described before, top-n results of BM25-T is the reranking of top-n results of BM25 based on time. Thus, for a query the value of Prec@5 will be the same for BM25-T and BM25 but NDCG@5 may be different. Further, the improvements of TAR are statistically significant with p-value < 0.01 using the paired Student's t-test over all competitor methods.

Table 2.2: Retrieval quality for BM25, BM25-T, EXP, BEX and TAR

ranking algorithm	AVG Prec@5	AVG NDCG@5
BM25	0.176	0.202
BM25-T	0.176	0.228
EXP	0.277	0.332
BEX	0.324	0.393
TAR	**0.373**	**0.438**

Sensitivity of Retrieval Quality wrt. Query Timeliness. Next, we studied the retrieval quality of all ranking algorithms for queries with different timeliness requirements.

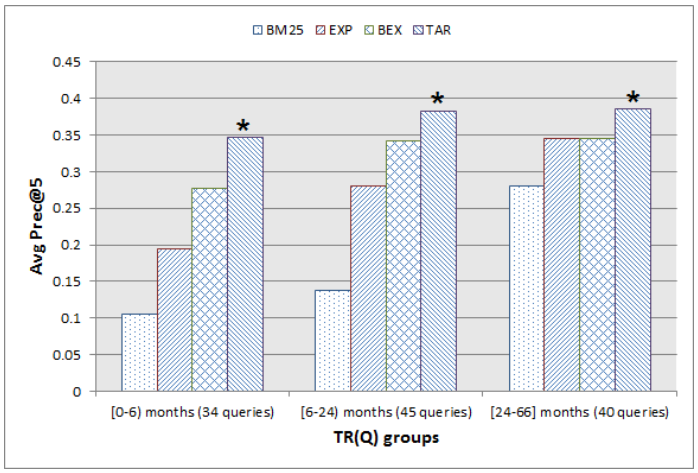


Figure 2.8: Average Prec@5 of examined algorithms on different timeliness groups

Specifically, we split the query workload into three timeliness groups according to the values of $TR(Q)$, as specified by the users in the survey described in Section 2.5.2. The three timeliness groups that we constructed have the following ranges: [0-6 months], [6-24 months] and [24-66 months] and contain 34, 45 and 40 queries respectively.

Figures 2.8 and 2.9 show the average Prec@5 and NDCG@5 results for each timeliness group for the different rankings that we examined. The "*" symbol over the TAR bar denotes that the respective improvements of TAR over all baselines are statistically significant with p-value < 0.05 using the paired Student's t-test.

These two figures show that for all timeliness groups, our proposed ranking achieves better retrieval quality than both BM25 and the other time-aware ranking algorithms. For the case of the [0-6 months] group, in which queries have the highest timeliness requirement, the improvements of TAR over all baselines (over 24% better than BEX on both Prec@5 and NDCG@5) are larger compared to the other two groups. This shows that our proposed

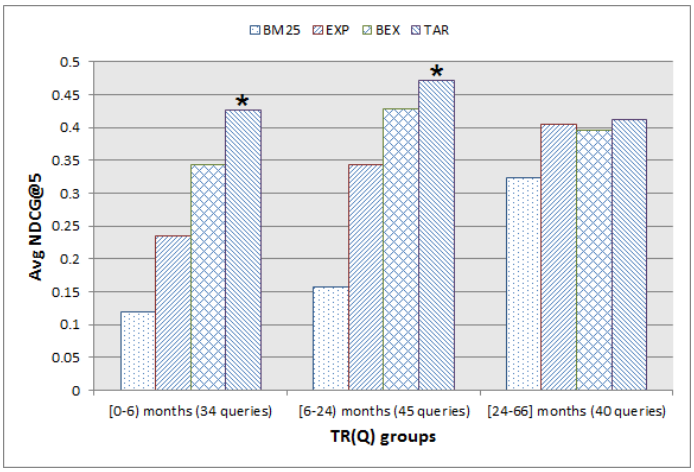


Figure 2.9: Average NDCG@5 of examined algorithms on different timeliness groups

model is especially useful for the queries with very intense timeliness requirements. For the case of the [6-24 months] group, TAR delivers over 10% improvement than BEX which is still the best among the baselines. Finally, for the queries in the [24-66 months] group, where the results freshness is a less important factor than the other two groups, TAR has a slight improvement over the baselines. Further, compared to the previous timeliness groups for the queries of this group we notice that BM25 achieves quite better retrieval quality because the content relevance is becoming the more important factor when the timeliness requirement drops.

Examples. We examine the retrieval quality for the query examples that we studied in Section 2.5.2. The Prec@5 and NDCG@5 scores for the example queries are shown in Tables 2.3 and 2.4 respectively.

Sensitivity of Retrieval Quality wrt. Document Volume Distribution. An interesting observation on our data set is that even if we follow a non time-based ranking

Table 2.3: Prec@5 for a sample of queries using different rankings.

Queries		Algorithms			
		BM25 (BM25-T)	EXP	BEX	TAR
Q88	public speaking tips	0.4	0.4	0.4	0.2
Q56	interview thank you letter	0.4	0.4	0.4	0.4
Q83	passport renewal	0	0	0	0.2
Q16	cancel a new car contract	0.4	0.4	0.6	0.6
Q65	low income housing	0	0	0	0.4
Q89	reality TV stars	0	0.2	0.4	0.4
Q57	keyboard reviews	0.2	0.2	0.6	0.6
Q90	retail sales index	0	0.4	0.4	0.6
Q95	smartphone reviews	0.6	0.6	0.6	0.8
Q15	California state parks jobs	0.2	0.4	0.6	0.6
Q79	newest tablet	0	0.2	0.2	0.4
Q17	celebrity gossips	0	0	0.2	0.4
Q75	NBA game schedule	0.2	0.2	0.2	0.6
Q76	NBA scores	0	0.2	0.2	0.6
Q14	California lottery results	0	0	0	0

(e.g., BM25), the time distribution of the most relevant documents is skewed towards the more recent ones. Specifically, in the top-500 documents of BM25, the average number of documents per query is: 62, 71, 83, 92, 101, 90 for years 2007-2011 and the first half of 2012 respectively. The distributions for some queries are more skewed than the average; we identified 44 out of the 119 queries where the top-500 results contain more that 40% documents that have been published in the last 1.5 year, vs. 60% of documents from 2007-

Table 2.4: NDCG@5 for a sample of queries using different rankings

Queries		Algorithms				
		BM25	BM25-T	EXP	BEX	TAR
Q88	public speaking tips	0.360	0.360	0.383	0.301	0.146
Q56	interview thank you letter	0.586	0.319	0.319	0.319	0.319
Q83	passport renewal	0	0	0	0	0.182
Q16	cancel a new car contract	0.316	0.553	0.485	0.684	0.640
Q65	low income housing	0	0	0	0	0.360
Q89	reality TV stars	0	0	0.182	0.531	0.704
Q57	keyboard reviews	0.246	0.390	0.246	0.710	0.805
Q90	retail sales index	0	0	0.301	0.316	0.655
Q95	smartphone reviews	0.699	0.684	0.699	0.699	0.830
Q15	California state parks jobs	0.246	0.390	0.637	0.805	0.805
Q79	newest tablet	0	0	0.202	0.296	0.704
Q17	celebrity gossips	0	0	0	0.146	0.316
Q75	NBA game schedule	0.170	0.146	0.146	0.214	0.616
Q76	NBA scores	0	0	0.131	0.170	0.655
Q14	California lottery results	0	0	0	0	0

2010. One possible explanation for this is that, since the size of the web grows faster over time, recent documents have a larger number; hence the probability for a recent document to be relevant is higher. Further, some older relevant web pages are no longer accessible or might be penalized with lower scores by commercial search engines. Recall that previous algorithms, especially BEX [41], leverage the time distribution in their rankings and thus

could benefit from this skewed distribution.

We studied the effect of the document distribution on retrieval quality. In particular, we computed the retrieval quality for two sets of queries; 75 queries that exhibit a quite steady time distribution in relevant documents and 44 queries with more skewed distributions towards recent documents. For the former subset, we got an average Prec@5 equal to 0.296 for BEX and 0.352 for TAR, i.e., TAR outperforms BEX (which is the best performing competitor) by 18.9%. For the 44 queries with more skewed distributions, the calculated average Prec@5 for BEX and TAR is 0.373 and 0.409 respectively, which is 9.6% improvement for TAR. The smaller improvement on this subset is expected, since BEX performs better for highly skewed time distributions. For both query sets, the improvement of TAR over BEX is statistically significant with p-value < 0.05.

Summary. All time-sensitive ranking algorithms outperform BM25 with significant improvements on both Prec@5 and NDCG@5 for timely queries. Further, consistent with former research [41], BEX generally exhibits better retrieval quality than EXP.

TAR achieves the best performance among all ranking algorithms. In particular, TAR improves over 10% in terms of both Prec@5 and NDCG@5 over BEX ranking algorithm on our complete query workload. TAR can satisfy queries with different timeliness requirements better than other time-sensitive ranking algorithms as shown in the experiments (Figures 2.8 and 2.9). Further, if we remove the effect of the skewness in the time distribution of the documents, TAR achieves even higher improvement (18.9%) over BEX (which delivers the best ranking quality among the competitor rankings).

The retrieval quality results on queries from different timeliness groups validate

our proposed model, i.e., the timeliness requirements can be predicted accurately based on the degree of content change in relevant documents and it is used in an effective way in our proposed ranking model.

2.6 Discussion

Limitations. In this work, we focus on timely queries that do not exhibit clear or significant variance in query or document popularity over time, but where recent results are preferred. The proposed model is not meant to handle other types of queries such as those targeting specific events. As a direction for future work, we will study a principled way to combine our model with previous works that focus on other types of time-sensitive queries, such as [31] which studies the volume distribution of relevant documents. In other words, we will study how to propose a unified model which considers different signals to estimate the temporal requirements for a broader set of queries.

Because of a lack of public benchmarks that provide both the topical and temporal relevance of results for timely queries, it's hard to conduct experiments on a very large query workload; however we believe that 119 queries is a reasonably large workload.

Practical Issues. In a real-world scenario, instead of conducting a user survey, the timeliness requirements of each query ($TR(Q)$) can be extracted based on clickthrough data, for instance by observing the timestamps of results that are clicked or not-clicked by the users after each web search.

If a new query Q for which the timeliness requirement is unknown is issued to the search engine, we can use the timeliness requirements of similar queries in order to estimate

a TDC value for Q. Similar queries can be found by considering keyword text similarity, or based on a query likelihood approach, etc. Further, in terms of implementation, one alternative approach to estimate $TDC(Q)$ on query time would be to first compute the top-k results in a time-insensitive way for query Q, then compute $TDC(Q)$ using these k results, and then rerank them using Equation 2.5.

With regard to learning an optimal value for α parameter, in a practical use case a search engine can train the model based on user feedback. Different values of α might be suitable for queries that exhibit different timeliness requirements; this is also an interesting direction that we aim to explore as our future work.

In a real-world web search system, our model can be applied as a complement to previous work studying news queries. Previous works have proposed methods to classify queries as news-related or not news-related [37, 30]. If the query is not news-related, our proposed TAR algorithm can be used, otherwise a news ranking approach [35, 37, 38, 30] can be applied.

2.7 Conclusions

In this chapter we studied the freshness factor for a class of queries that we refer to as timely queries. We show that previous works on news queries cannot be applied effectively for predicting the timeliness requirement of queries if the query popularity from document publishers or consumers does not vary significantly over time. We propose a method to estimate query timeliness with high accuracy using the terms distribution change of a query's relevant documents over time. Further, we present a ranking model that

incorporates the timeliness factor in order to improve the results freshness for timely queries, and we experimentally show that our ranking improves upon previous methods over 10% in terms of both precision and NDCG. In our future work we plan to explore methods to automatically learn the TDC scores and to combine our proposed ranking model with other signals in order to support a broader set of queries.

Chapter 3

Efficient Prediction of Difficult Keyword Queries over Databases

Keyword queries on databases provide easy access to data, but often suffer from low ranking quality, i.e., low precision and/or recall, as shown in recent benchmarks. It would be useful to identify queries that are likely to have low ranking quality to improve the user satisfaction. For instance, the system may suggest to the user alternative queries for such hard queries. In this chapter, we analyze the characteristics of hard queries and propose a novel framework to measure the degree of difficulty for a keyword query over a database, considering both the structure and the content of the database and the query results. We evaluate our query difficulty prediction model against two effectiveness benchmarks for popular keyword search ranking methods. Our empirical results show that our model predicts the hard queries with high accuracy. Further, we present a suite of optimizations to minimize the incurred time overhead.

3.1 Introduction

Keyword query interfaces (*KQIs*) for databases have attracted much attention in the last decade due to their flexibility and ease of use in searching and exploring the data [59, 81, 47, 66, 102]. Since any entity in a data set that contains the query keywords is a potential answer, keyword queries typically have many possible answers. KQIs must identify the information needs behind keyword queries and rank the answers so that the desired answers appear at the top of the list [59, 15]. Unless otherwise noted, we refer to *keyword query* as *query* in the remainder of this chapter.

Databases contain entities, and entities contain attributes that take attribute values. Some of the difficulties of answering a query are as follows: First, unlike queries in languages like SQL, users do not normally specify the desired schema element(s) for each query term. For instance, query Q_1: *Godfather* on the IMDB database (*http://www.imdb.com*) does not specify if the user is interested in movies whose *title* is *Godfather* or movies distributed by the *Godfather* company. Thus, a KQI must find the desired attributes associated with each term in the query. Second, the schema of the output is not specified, i.e., users do not give enough information to single out exactly their desired entities [83]. For example, Q_1 may return movies or actors or producers. We present a more complete analysis of the sources of difficulty and ambiguity in Section 3.4.2.

Recently, there have been collaborative efforts to provide standard benchmarks and evaluation platforms for keyword search methods over databases. One effort is the data-centric track of INEX Workshop [118] where KQIs are evaluated over the well-known IMDB data set that contains structured information about movies and people in show business.

Queries were provided by participants of the workshop. Another effort is the series of Semantic Search Challenges (SemSearch) at Semantic Search Workshop [116], where the data set is the Billion Triple Challenge data set at *http://vmlion25.deri.de*. It is extracted from different structured data sources over the Web such as Wikipedia. The queries are taken from Yahoo! keyword query log. Users have provided relevance judgments for both benchmarks.

The Mean Average Precision (MAP) of the best performing method(s) in the last data-centric track in INEX Workshop and Semantic Search Challenge for queries are about 0.36 and 0.2, respectively. The lower MAP values of methods in Semantic Search Challenge are mainly due to the larger size and more heterogeneity of its data set.

These results indicate that even with structured data, finding the desired answers to keyword queries is still a hard task. More interestingly, looking closer to the ranking quality of the best performing methods on both workshops, we notice that they all have been performing very poorly on a subset of queries. For instance, consider the query *ancient Rome era* over the IMDB data set. Users would like to see information about movies that talk about ancient Rome. For this query, the state-of-the-art XML search methods which we implemented return rankings of considerably lower quality than their average ranking quality over all queries. Hence, some queries are more difficult than others. Moreover, no matter which ranking method is used, we cannot deliver a reasonable ranking for these queries. Table 3.1 lists a sample of such hard queries from the two benchmarks. Such a trend has been also observed for keyword queries over text document collections [115]. These queries are usually either under-specified, such as query *carolina* in Table 3.1, or

Table 3.1: Some difficult queries from benchmarks.

INEX	SemSearch
ancient Rome era	Austin Texas
Movies Klaus Kinski actor good rating	Carolina
true story drugs addiction	Earl May
	Lynchburg Virginia
	San Antonio

overspecified, such as query *Movies Klaus Kinski actor good rating* in Table 3.1.

It is important for a KQI to recognize such queries and warn the user or employ alternative techniques like query reformulation or query suggestions [88]. It may also use techniques such as query results diversification [32]. On the other hand, if a KQI would employ these techniques for queries with high-quality results, it may hurt their quality and/or waste computational resources (such as CPU cycle) and the time of users. Hence, it is important that a KQI distinguishes difficult from easy queries and act upon them accordingly (the latter is out of the scope of this work).

In this work, we analyze the characteristics of difficult queries over databases and propose a novel method to detect such queries. We take advantage of the structure of the data to gain insight about the degree of the difficulty of a query given the database. We have implemented some of the most popular and representative algorithms for keyword search on databases and used them to evaluate our techniques on both the INEX and SemSearch benchmarks. The results show that our method predicts the degree of the difficulty of a

query efficiently and effectively.

We make the following contributions:

- We introduce the problem of predicting the degree of the difficulty for queries over databases. We also analyze the reasons that make a query difficult to answer by KQIs (Section 3.4).

- We propose the *Structured Robustness (SR)* score, which measures the difficulty of a query based on the differences between the rankings of the same query over the original and noisy (corrupted) versions of the same database, where the noise spans on both the content and the structure of the result entities (Section 3.5).

- We present an algorithm to compute the SR score, and parameters to tune its performance (Section 3.6).

- We introduce efficient approximate algorithms to estimate the SR score, given that such a measure is only useful when it can be computed with a small time overhead compared to the query execution time (Section 3.7).

- We show the results of extensive experiments using two standard data sets and query workloads: INEX and SemSearch. Our results show that the SR score effectively predicts the ranking quality of representative ranking algorithms, and outperforms non-trivial baselines, introduced in this chapter. Also, the time spent to compute the SR score is negligible compared to the query execution time (Section 3.8).

Section 3.2 discusses related work and Section 3.3 presents basic definitions. Section 3.9 concludes the chapter and presents future directions.

3.2 Related Work

Researchers have proposed methods to predict hard queries over unstructured text documents [115, 131, 56, 29, 105, 132]. We can broadly categorize these methods into two groups: *pre-retrieval* and *post-retrieval* methods.

Pre-retrieval methods [130, 56] predict the difficulty of a query without computing its results. These methods usually use the statistical properties of the terms in the query to measure *specificity*, *ambiguity*, or *term-relatedness* of the query to predict its difficulty [53]. Examples of these statistical characteristics are average inverse document frequency of the query terms or the number of documents that contain at least one query term [56]. These methods generally assume that the more discriminative the query terms are, the easier the query will be. Empirical studies indicate that these methods have limited prediction accuracies [115, 55].

Post-retrieval methods utilize the results of a query to predict its difficulty and generally fall into one of the following categories.

Clarity-score-based: The methods based on the concept of *clarity score* assume that users are interested in a very few topics, so they deem a query easy if its results belong to very few topic(s) and therefore, sufficiently distinguishable from other documents in the collection [115, 56, 29, 55]. Researchers have shown that this approach predicts the difficulty of a query more accurately than pre-retrieval based methods for text documents [115]. Some systems measure the distinguishability of the queries results from the documents in the collection by comparing the probability distribution of terms in the results with the probability distribution of terms in the whole collection. If these probability distributions

are relatively similar, the query results contain information about almost as many topics as the whole collection, thus, the query is considered difficult [115]. Several successors propose methods to improve the efficiency and effectiveness of clarity score [56, 29, 55].

However, one requires domain knowledge about the data sets to extend idea of clarity score for queries over databases. Each topic in a database contains the entities that are about a similar subject. It is generally hard to define a formula that partitions entities into topics as it requires finding an effective similarity function between entities. Such similarity function depends mainly on the domain knowledge and understanding users' preferences [52]. For instance, different attributes may have different impacts on the degree of the similarity between entities. Assume movies A and B in IMDB share some terms in their *genre* attributes, which explain the subjects of the movies. Also, let movies A and C share the same number of terms in their *distributor* attributes, which describe the distribution company of the movies. Given other attributes of A, B, and C do not contain any common term, movies A and B are more likely to be about the same subject and satisfy the same information need than movies A and C. Our empirical results in Section 3.8 confirms this argument and shows that the straightforward extension of clarity score predicts difficulties of queries over databases poorly.

Some systems use a pre-computed set of topics and assign each document to at least one topic in the set in order to compute the clarity score [29]. They compare the probability distribution of topics in the top ranked documents with the probability distribution of topics of the whole collection to predict the degree of the difficulty of the query. One requires domain knowledge about the data sets and its users to create a set of useful topics for the

tuples in the database. We like to find an effective and domain independent approach to predict the difficulties of queries.

Ranking-score-based: The ranking score of a document returned by the retrieval systems for an input query may estimate the similarity of the query and the document. Some recent methods measure the difficulty of a query based on the score distribution of its results [105, 132]. Zhou and Croft argue that the information gained from a desired list of documents should be much more than the information gained from typical documents in the collection for an easy query. They measure the degree of the difficulty of a query by computing the difference between the weighted entropy of the top ranked results' scores and the weighted entropy of other documents' scores in the collection [132]. Shtok et al. argue that the amount of non-query-related information in the top ranked results is negatively correlated with the deviation of their retrieval scores [105]. Using language modeling techniques, they show that the standard deviation of ranking scores of top-k results estimates the quality of the top ranked results effectively. We examine the query difficulty prediction accuracy of this set of methods on databases in Section 3.8, and show that our model outperforms these methods over databases.

Robustness-based: Another group of post-retrieval methods argue that the results of an easy query are relatively stable against the perturbation of queries [126], documents [131] or ranking algorithms [11]. Our proposed query difficulty prediction model falls in this category. More details of some related work will be given in Section 3.4, where we discuss the difference of applying these techniques on text collection and database.

Some methods use machine learning techniques to learn the properties of difficult

queries and predict their hardness [126]. They have similar limitations as the other approaches when applied to structured data. Moreover, their success depends on the amount and quality of their available training data. Sufficient and high quality training data is not normally available for many databases. Some researchers propose frameworks that theoretically explain existing predictors and combine them to achieve higher prediction accuracy [74, 73].

3.3 Data and Query Models

We model a database as a set of entity sets. Each entity set S is a collection of entities E. For instance, *movies* and *people* are two entity sets in IMDB. Figure 3.1 depicts a fragment of a data set where each subtree whose root's label is *movie* represents an entity. Each entity E has a set of attribute values A_i, $1 \leq i \leq |E|$. Each attribute value is a bag of terms. Following current unstructured and (semi-) structure retrieval approaches, we ignore stop words that appear in attribute values, although this is not necessary for our methods. Every attribute value A belongs to an *attribute* T written as $A \in T$. For instance, *Godfather* and *Mafia* are two attribute values in the movie entity shown in the subtree rooted at node 1 in Figure 3.1. Node 2 depicts the attribute of *Godfather*, which is *title*.

The above is an abstract data model. We ignore the physical representation of data in this work. That is, an entity could be stored in an XML file or a set of normalized relational tables. The above model has been widely used in works on *entity search* [102, 47] and *data-centric XML retrieval* [118], and has the advantage that it can be easily mapped to both XML and relational data. Further, if a KQI method relies on the intricacies of the

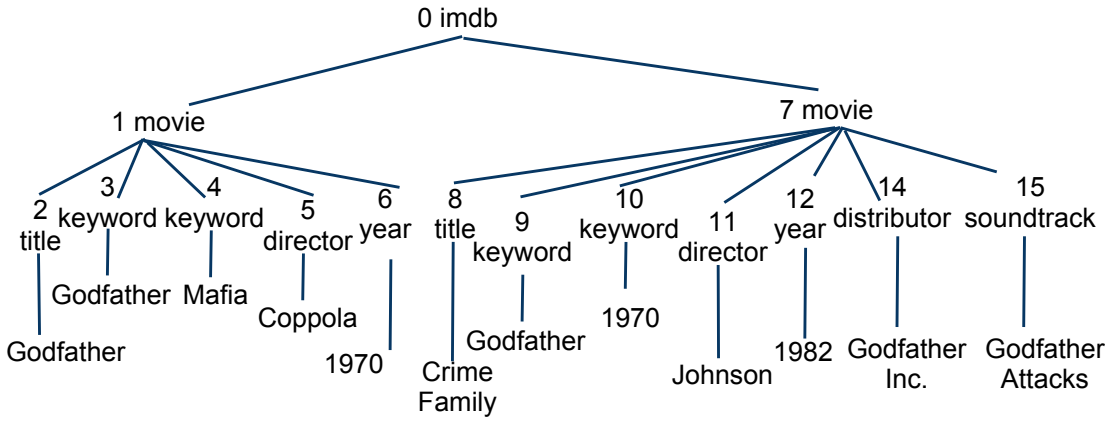


Figure 3.1: IMDB database fragment

database design (e.g. deep syntactic nesting), it will not be robust and will have considerably different degrees of effectiveness over different databases [114]. Hence, since our goal is to develop principled formal models that cover reasonably well all databases and data formats, we do not consider the intricacies of the database design or data format in our models.

A *keyword query* is a set $Q = \{q_1 \cdots q_{|Q|}\}$ of terms, where $|Q|$ is the number of terms in Q. An entity E is an *answer* to Q iff at least one of its attribute values A contains a term q_i in Q, written $q_i \in A$[1]. Given database DB and query Q, retrieval function $g(E, Q, DB)$ returns a real number that reflects the relevance of entity $E \in DB$ to Q. Given database DB and query Q, a keyword search system returns a ranked list of entities in DB called $L(Q, g, DB)$ where entities E are placed in decreasing order of the value of $g(E, Q, DB)$.

[1]Some works on keyword search in databases [59] use conjunctive semantics, where all query keywords must appear in a result.

3.4 Ranking Robustness Principle for Structured Data

In this section we present the *Ranking Robustness Principle*, which argues that there is a (negative) correlation between the difficulty of a query and its ranking robustness in the presence of noise in the data. Section 3.4.1 discusses how this principle has been applied to unstructured text data. Section 3.4.2 presents the factors that make a keyword query on structured data difficult, which explain why we cannot apply the techniques developed for unstructured data. The latter observation is also supported by our experiments in Section 3.8.2 on the *Unstructured Robustness Method* [131], which is a direct adaptation of the Ranking Robustness Principle for unstructured data.

3.4.1 Background: Unstructured Data

Mittendorf has shown that if a text retrieval method effectively ranks the answers to a query in a collection of text documents, it will also perform well for that query over the version of the collection that contains some errors such as repeated terms [86]. In other words, the degree of the difficulty of a query is positively correlated with the robustness of its ranking over the original and the corrupted versions of the collection. We call this observation the *Ranking Robustness Principle*. Zhou and Croft [131] have applied this principle to predict the degree of the difficulty of a query over free text documents. They compute the similarity between the rankings of the query over the original and the artificially corrupted versions of a collection to predict the difficulty of the query over the collection. They deem a query to be more difficult if its rankings over the original and the corrupted versions of the data are less similar. They have empirically shown their claim to be valid.

They have also shown that this approach is generally more effective than using methods based on the similarities of probability distributions, that we reviewed in Section 3.2. This result is especially important for ranking over databases. As we explained in Section 3.2, it is generally hard to define an effective and domain independent categorization function for entities in a database. Hence, we can use Ranking Robustness Principle as a domain independent proxy metric to measure the degree of the difficulties of queries.

3.4.2 Properties of Hard Queries on Databases

As discussed in Section 3.2, it is well established that *the more diverse the candidate answers of a query are, the more difficult the query is* over a collection of the text documents. We extend this idea for queries over databases and propose three sources of difficulty for answering a query over a database as follows:

1. The more entities match the terms in a query, the less specificity of this query and it is harder to answer properly. For example, there are more than one person called *Ford* in the IMDB data set. If a user submits query Q_2: *Ford*, a KQI must resolve the desired *Ford* that satisfy the user's information need. As opposed to Q_2, Q_3: *Spielberg* matches smaller number of people in IMDB, so it is easier for the KQI to return its relevant results.

2. Each attribute describes a different aspect of an entity and defines the context of terms in attribute values of it. If a query matches different attributes in its candidate answers, it will have a more diverse set of potential answers in database, and hence it has higher attribute level ambiguity. For instance, some candidate answers for query

Q_4: *Godfather* in IMDB contain its term in their *title* and some contain its term in their *distributor*. For the sake of this example, we ignore other attributes in IMDB. A KQI must identify the desired matching attribute for *Godfather* to find its relevant answers. As opposed to Q_4, query Q_5: *taxi driver* does not match any instance of attribute *distributor*. Hence, a KQI already knows the desired matching attribute for Q_5 and has an easier task to perform. Assume that Q_4 and Q_5 have almost equal number of answers. Some of the movies whose titles match Q_5 are related, e.g., there are three documentaries in IMDB whose titles match *taxi driver*, which are about making the well-known movie *taxi driver* directed by *Martin Scorsese*. They may partially or fully satisfy the information need behind Q_5. However, the candidate answers whose *title* attribute match Q_4 and the candidate answers whose *distributor* attribute match Q_4 are not generally related.

3. Each entity set contains the information about a different type of entities and defines another level of context (in addition to the context defined by attributes) for terms. Hence, if a query matches entities from more entity sets, it will have higher entity set level ambiguity. For instance, IMDB contains the information about movies in an entity set called *movie* and the information about the people involved in making movies in another entity set called *person*. Consider query Q_6: *divorce* over IMDB data set whose candidate answers come from both entity sets. However, movies about divorce and people who get divorced cannot both satisfy information need of query Q_6. A KQI has a difficult task to do as it has to identify if the information need behind this query is to find people who got divorced or movies about divorce. In contrast

to Q_6, Q_7: *romantic comedy divorce* matches only entities from *movie* entity set. It is less difficult for a KQI to answer Q_7 than Q_6 as Q_7 has only one possible desired entity set. Given Q_6 and Q_7 have almost the same number of candidate answers and matching attributes, it is likely that more candidate answers of Q_6 are relevant to its users' information need than the candidate answers to Q_7.

The aforementioned observations show that we may use the statistical properties of the query terms in the database to compute the diversity of its candidate answers and predict its difficulty, like the pre-retrieval predictors introduced in Section 3.2. One idea is to count the number of possible attributes, entities, and entity sets that contain the query terms to estimate the query specificity and ambiguity and use them to predict the difficulty of the query. The larger this value is the more difficult the query will be. We have shown empirically in Section 3.8.2 that such approach predicts the difficulty of queries quite poorly. This is because the distribution of query terms over attributes and entity sets may also impact the difficulty of the query. For instance, assume database DB_1 contains two entity sets *book* and *movie* and database DB_2 contains entity sets *book* and *article*. Let term *database* appear in both entity sets in DB_1 and DB_2. Assume that there are far fewer movies that contain term *database* compared to books and articles. A KQI can leverage this property and rank books higher than movies when answering query Q_8: *database* over DB_1. However, it will be much harder to decide the desired entity set in DB_2 for Q_8. Hence, a difficulty metric must take in to account the skewness of the distributions of the query term in the database as well. In Section 3.5 we use these ideas to create a concrete noise generation framework that consider attribute values, attributes and entity sets.

3.5 A Framework to Measure Structured Robustness

In Section 3.4 we presented the Ranking Robustness Principle and discussed the specific challenges in applying this principle to structured data. In this section we present concretely how this principle is quantified in structured data. Section 3.5.1 discusses the role of the structure and content of the database in the corruption process, and presents the robustness computation formula given corrupted database instances. Section 3.5.2 provides the details of how we generate corrupted instances of the database. Section 3.5.3 suggests methods to compute the parameters of our model. In Section 3.5.4 we show real examples of how our method corrupts the database and predicts the difficulty of queries.

3.5.1 Structured Robustness

Corruption of structured data. The first challenge in using the Ranking Robustness Principle for databases is to define data corruption for structured data. For that, we model a database DB using a generative probabilistic model based on its building blocks, which are terms, attribute values, attributes, and entity sets. A corrupted version of DB can be seen as a random sample of such a probabilistic model. Given a query Q and a retrieval function g, we rank the candidate answers in DB and its corrupted versions $DB', DB'', \cdots$ to get ranked lists L and $L', L'', \cdots$, respectively. The less similar L is to $L', L'', \cdots$, the more difficult Q will be.

According to the definitions in Section 3.3, we model database DB as a triplet $(\mathcal{S}, \mathcal{T}, \mathcal{A})$, where $\mathcal{S}$, $\mathcal{T}$, and $\mathcal{A}$ denote the sets of entity sets, attributes, and attribute values in DB, respectively. $|\mathcal{A}|$, $|\mathcal{T}|$, $|\mathcal{S}|$ denote the number of attribute values, attributes, and

entity sets in the database, respectively. Let V be the number of distinct terms in database DB. Each attribute value $A_a \in \mathcal{A}$, $1 \leq a \leq |\mathcal{A}|$, can be modeled using a V-dimensional multivariate distribution $X_a = (X_{a,1}, \cdots, X_{a,V})$, where $X_{a,j} \in X_a$ is a random variable that represents the frequency of term w_j in A_a. The probability mass function of X_a is:

$$f_{X_a}(\vec{x}_a) = Pr(X_{a,1} = x_{a,1}, \cdots, X_{a,V} = x_{a,V}) \tag{3.1}$$

where $\vec{x}_a = x_{a,1}, \cdots, x_{a,V}$ and $x_{a,j} \in \vec{x}_a$ are non-negative integers.

Random variable $X_{\mathcal{A}} = (X_1, \cdots, X_{|\mathcal{A}|})$ models attribute value set $\mathcal{A}$, where $X_a \in X_{\mathcal{A}}$ is a vector of size V that denotes the frequencies of terms in A_a. Hence, $X_{\mathcal{A}}$ is a $|\mathcal{A}| \times V$ matrix. The probability mass function for $X_{\mathcal{A}}$ is:

$$f_{X_{\mathcal{A}}}(\vec{x}) = f_{X_{\mathcal{A}}}(\vec{x_1}, \cdots, \vec{x_{|\mathcal{A}|}}) = Pr(X_1 = \vec{x_1}, \cdots, X_{|\mathcal{A}|} = \vec{x_{|\mathcal{A}|}}) \tag{3.2}$$

where $\vec{x_a} \in \vec{x}$ are vectors of size V that contain non-negative integers. The domain of $\vec{x}$ is all $|\mathcal{A}| \times V$ matrices that contain non-negative integers, i.e. $M(|\mathcal{A}| \times V)$.

We can similarly define $X_{\mathcal{T}}$ and $X_{\mathcal{S}}$ that model the set of attributes $\mathcal{T}$ and the set of entity sets $\mathcal{S}$, respectively. The random variable $X_{DB} = (X_{\mathcal{A}}, X_{\mathcal{T}}, X_{\mathcal{S}})$ models corrupted versions of database DB. In this work, we focus only on the noise introduced in the content (values) of the database. In other words, we do not consider other types of noise such as changing the attribute or entity set of an attribute value in the database. Since the membership of attribute values to their attributes and entity sets remains the same across the original and the corrupted versions of the database, we can derive $X_{\mathcal{T}}$ and $X_{\mathcal{S}}$ from $X_{\mathcal{A}}$. Thus, a corrupted version of the database will be a sample from $X_{\mathcal{A}}$; note that the attributes and entity sets play a key role in the computation of $X_{\mathcal{A}}$ as we discuss in Section 3.5.2.

Therefore, we use only $X_{\mathcal{A}}$ to generate the noisy versions of DB, i.e. we assume that $X_{DB} = X_{\mathcal{A}}$. In Section 3.5.2 we present in detail how X_{DB} is computed.

Structured Robustness calculation. We compute the similarity of the answer lists using Spearman rank correlation [49]. It ranges between 1 and -1, where 1, -1, and 0 indicate perfect positive correlation, perfect negative correlation, and almost no correlation, respectively. Equation 3.3 computes the Structured Robustness score (SR score), for query Q over database DB given retrieval function g:

$$\begin{aligned} SR(Q, g, DB, X_{DB}) &= \mathbb{E}\{Sim(L(Q, g, DB), L(Q, g, X_{DB}))\} \\ &= \sum_{\vec{x}} Sim(L(Q, g, DB), L(Q, g, \vec{x})) f_{X_{DB}}(\vec{x}) \end{aligned} \tag{3.3}$$

where $\vec{x} \in M(|\mathcal{A}| \times V)$ and Sim denotes the Spearman rank correlation between the ranked answer lists.

3.5.2 Noise Generation in Databases

In order to compute Equation 3.3, we need to define the noise generation model $f_{X_{DB}}(M)$ for database DB. *We will show that each attribute value is corrupted by a combination of three corruption levels: on the value itself, its attribute and its entity set.* Now the details: Since the ranking methods for queries over structured data do not generally consider the terms in V that do not belong to query Q [59, 66], we consider their frequencies to be the same across the original and noisy versions of DB. Given query Q, let $\vec{x}$ be a vector that contains term frequencies for terms $w \in Q \cap V$. Similarly to [131], we simplify our model by assuming the attribute values in DB and the terms in $Q \cap V$ are independent.

Hence, we have:

$$f_{X_{\mathcal{A}}}(\vec{x}) = \prod_{x_a \in \vec{x}} f_{X_a}(\vec{x_a}). \tag{3.4}$$

and

$$f_{X_a}(\vec{x}_a) = \prod_{x_{a,j} \in \vec{x}_a} f_{X_{a,j}}(x_{a,j}). \tag{3.5}$$

where $x_j \in \vec{x}_i$ depicts the number of times w_j appears in a noisy version of attribute value A_i and $f_{X_{i,j}}(x_j)$ computes the probability of term w_j to appear in A_i x_j times.

The corruption model must reflect the challenges discussed in Section 3.4.2 about search on structured data, where we showed that it is important to capture the statistical properties of the query keywords in the attribute values, attributes and entity sets. We must introduce content noise (recall that we do not corrupt the attributes or entity sets but only the values of attribute values) to the attributes and entity sets, which will propagate down to the attribute values. For instance, if an attribute value of attribute *title* contains keyword *Godfather*, then *Godfather* may appear in any attribute value of attribute *title* in a corrupted database instance. Similarly, if *Godfather* appears in an attribute value of entity set *movie*, then *Godfather* may appear in any attribute value of entity set *movie* in a corrupted instance.

Since the noise introduced in attribute values will propagate up to their attributes and entity sets, one may question the need to introduce additional noise in attribute and entity set levels. The following example illustrates the necessity to generate such noises. Let T_1 be an attribute whose attribute values are A_1 and A_2, where A_1 contains term w_1 and A_2 does not contain w_1. A possible noisy version of T_1 will be a version where A_1 and A_2 both contain w_1. However, the aforementioned noise generation model will not produce

such a version. Similarly, a noisy version of entity set S must introduce or remove terms from its attributes and attribute values. According to our discussion in Section 3.4, we must use a model that generates all possible types of noise in the data.

Hence, we model the noise in a DB as a *mixture* of the noises generated in attribute value, attribute, and entity set levels. Mixture models are typically used to model how the combination of multiple probability distributions generates the data. Let $Y_{t,j}$ be the random variable that represents the frequency of term w_j in attribute T_t. Probability mass function $f_{Y_{t,j}}(y_{t,j})$ computes the probability of w_j to appear $y_{t,j}$ times in T_t. Similarly, $Z_{s,j}$ is the random variable that denotes the frequency of term w_j in entity set S_s and probability mass function $f_{Z_{s,j}}(z_{s,j})$ computes the probability of w_j to appear $z_{s,j}$ times in S_s. Hence, the noise generation models attribute value A_i whose attribute is T_t and entity set is S_s:

$$\hat{f}_{X_{a,j}}(x_{a,j}) = \gamma_A f_{X_{a,j}}(x_{a,j}) + \gamma_T f_{Y_{t,j}}(x_{t,j}) + \gamma_S f_{Z_{s,j}}(x_{s,j}). \tag{3.6}$$

where $0 \leq \gamma_A, \gamma_T, \gamma_S \leq 1$ and $\gamma_A + \gamma_T + \gamma_S = 1$. $f_{X_{a,j}}$, $f_{Y_{t,j}}$, and $f_{Y_{s,j}}$ model the noise in attribute value, attribute, and entity set levels, respectively. Parameters γ_A, γ_T and γ_S have the same values for all terms $w \in Q \cap V$ and are set empirically.

Since each attribute value A_a is a small document, we model $f_{X_{i,j}}$ as a Poisson distribution:

$$f_{X_{a,j}}(x_{a,j}) = \frac{e^{-\lambda_{a,j}} \lambda_{a,j}^{x_{a,j}}}{x_{a,j}!}. \tag{3.7}$$

Similarly, we model each attribute T_t, $1 \leq t \leq |\mathcal{T}|$, as a bag of words and use Poisson distribution to model the noise generation in the attribute level:

$$f_{Y_{t,j}}(x_{t,j}) = \frac{e^{-\lambda_{t,j}} \lambda_{t,j}^{x_{t,j}}}{x_{t,j}!}. \tag{3.8}$$

Using similar assumptions, we model the changes in the frequencies of the terms in entity set S_s, $1 \leq s \leq |\mathcal{S}|$, using Poisson distribution:

$$f_{Z_{s,j}}(x_{s,j}) = \frac{e^{-\lambda_{s,j}} \lambda_{s,j}^{x_{s,j}}}{x_{s,j}!}. \tag{3.9}$$

In order to use the model in Equation 3.6, we have to estimate $\lambda_{A,w}$, $\lambda_{T,w}$, and $\lambda_{S,w}$ for every $w \in Q \cap V$, attribute value A, attribute T and entity set S in DB. We treat the original database as an observed value of the space of all possible noisy versions of the database. Thus, using the maximum likelihood estimation method, we set the value of $\lambda_{A,w}$ to the frequency of w in attribute value A. Assuming that w are distributed uniformly over the attribute values of attribute T, we can set the value of $\lambda_{T,w}$ to the average frequency of w in T. Similarly, we set the value of $\lambda_{S,w}$ as the average frequency of w in S. Using these estimations, we can generate noisy versions of a database according to Equation 3.6.

3.5.3 Smoothing The Noise Generation Model

Equation 3.6 overestimates the frequency of the terms of the original database in the noisy versions of the database. For example, assume a bibliographic database of computer science publications that contains attribute $T_2 =$ *abstract* which constitutes the abstract of a paper. Apparently, many abstracts contain term $w_2 =$ *algorithm*, therefore, this term will appear very frequently with high probability in f_{T_2,w_2} model. On the other hand, a term such as $w_3 =$ *Dirichlet* is very likely to have very low frequency in f_{T_2,w_3} model. Let attribute value A_2 be of attribute *abstract* in the bibliographic DB that contains both w_2 and w_3. Most likely, term *algorithm* will appear more frequently than *Dirichlet* in A_2. Hence, the mean for f_{A_2,w_2} will be also larger than the mean of f_{A_2,w_3}. Thus, a mixture

model of f_{T_2,w_2} and f_{A_2,w_2} will have much larger mean than a mixture model of f_{T_2,w_3} and f_{A_2,w_3}. The same phenomenon occurs if a term is relatively frequent in an entity set. Hence, a mixture model such as Equation 3.6 overestimates the frequency of the terms that are relatively frequent in an attribute or entity set. Researchers have faced a similar issue in language model smoothing for speech recognition [65]. We use a similar approach to resolve this issue. If term w appear in attribute value A, we use only the first term in Equation 3.6 to model the frequency of w in the noisy version of database. Otherwise, we use the second or third terms if w belongs to T and S, respectively. Hence, the noise generation model is:

$$\hat{f}_{X_{a,j}}(x_{a,j}) = \begin{cases} \gamma_A f_{X_{a,j}}(x_{a,j}) & \text{if } w_j \in A_a \\ \gamma_T f_{Y_{t,j}}(x_{t,j}) & \text{if } w_j \notin A_a,\ w_j \in T_t \\ \gamma_S f_{Z_{s,j}}(x_{s,j}) & \text{if } w_j \notin A_a, T_t,\ w_j \in S_s \end{cases} \tag{3.10}$$

where we remove the condition $\gamma_A + \gamma_T + \gamma_S = 1$.

3.5.4 Examples

We illustrate the corruption process and the relationship between the robustness of the ranking of a query and its difficulty using INEX queries $Q9$: *mulan hua animation* and $Q11$: *ancient rome era*, over the IMDB dataset. We set $\gamma_A = 1$, $\gamma_T = 0.9$, $\gamma_S = 0.8$ in Equation 3.10. We use the XML ranking method proposed in [66], called *PRMS*, which we explain in more detail in Section 3.6. Given query Q, PRMS computes the relevance score of entity E based on the weighted linear combination of the relevance scores the attribute values of E.

Example of calculation of $\lambda_{t,j}$ for term t =ancient and attribute T_j =plot in Equation 3.8: In the IMDB dataset, *ancient* occurs in attribute *plot* 2132 times in total, and

total number of attribute values under attribute *plot* is 184,731, $\lambda_{t,j} = 2132/184731$ which is 0.0115. Then, since $\gamma_T = 0.9$, the probability that *ancient* occurs k times in a corrupted *plot* attribute is $\frac{0.9e^{-0.0115}(0.0115)^k}{k!}$.

Q11: Figure 3.2a depicts two of the top results (ranked as 1^{st} and 12^{nd} respectively) for query $Q11$ over IMDB. We omit most attributes (shown as elements in XML lingo in Figure 3.2a) that do not contain any query keywords due to space consideration. Figure 3.2b illustrates a corrupted version of the entities shown in Figure 3.2a. The new keyword instances are underlined. Note that the ordering changed according to PRMS. The reason is that PRMS believes that *title* is an important attribute for *rome* (for attribute weighing in PRMS see Section 3.8.1) and hence having a query keyword (*rome*) there is important. However, after corruption, query word *rome* also appears in the *title* of the other entity, which now ranks higher, because it contains the query words in more attributes.

Word *rome* was added to the *title* attribute of the originally second result through the second level (attribute-based, second branch in Equation 3.10) of corruption, because *rome* appears in the *title* attribute of other entities in the database. If no *title* attribute contained *rome*, then it could have been added through the third level corruption (entity set-based, third branch in Equation 3.10) since it appears in attribute values of other *movie* entities.

The second and third levels corruptions typically have much smaller probability of adding a word than the first level, because they have much smaller λ; specifically λ_T is the average frequency of the term in attribute T. However, in hard queries like $Q11$, the query terms are frequent in the database, and also appear in various entities and attributes, and

<movie id= “1025102”>
<title>rome ...</title>
<keyword>ancient-world</keyword>
<keyword>ancient-art</keyword>
<keyword>ancient-rome</keyword>
<keyword>christian-era</keyword>
</movie>

<movie id=“1149602”>
<title>Gladiator</title>
<keyword>ancient-rome</keyword>
<character>Rome ...</character>
<person>... Rome/UK)</person>
<trivia>”Rome of the imagination... </trivia>
<goof>Rome vs. Carthage ...</goof>
<quote>... enters Rome like a ... Rome ... </quote>
</movie>

(a) Original ranking

<movie id=“1149602”>
<title> Gladiator rome</title>
<keyword>ancient-rome rome</keyword>
<character>Rome ...</character>
<person> ... Rome/UK)</person>
<trivia>of the imagination...</trivia>
<goof>Rome vs. Carthage ...</goof>
<quote>... enters Rome like a ... Rome ...</quote>
</movie>

<movie id= “1025102”>
<title>rome ...</title>
<keyword>ancient-world ancient</keyword>
<keyword>-art</keyword>
<keyword>ancient ancient</keyword>
<keyword>christian-</keyword>
</movie>

(b) Corrupted ranking

Figure 3.2: Original and corrupted results of $Q11$

hence λ_T and λ_S are larger.

In the first *keyword* attribute of the top result in Figure 3.2b, *rome* is added by the first level of corruption, whereas in the *trivia* attribute *rome* is removed by the first level of corruption.

To summarize, $Q11$ is *difficult* because its keywords are spread over a large number of attribute values, attributes and entities in the original database, and also most of the top results have a similar number of occurrences of the keywords. Thus, when the corruption process adds even a small number of query keywords to the attribute values of the entities

<movie id="1492260">
<title>The Legend of Mulan (1998)</title>
<genre>Animation</genre>
<link>Hua Mu Lan (1964)</link>
<link>Hua Mulan cong jun</link>
<link>Mulan (1998)</link>
<link>Mulan (1999)</link>
<link>The Secret of Mulan (1998)</link>
</movie>

<movie id="1180849">
<title>Hua Mulan (2009)</title>
<character>Hua Hu (Mulan's father)</character>
<character>Young Hua Mulan</character>
<character>Hua Mulan</character>
</movie>

(a) Original ranking

<movie id="1492260">
<title>The Legend of Mulan (1998) mulan mulan</title>
<genre> </genre>
<link>Hua Mu Lan (1964)</link>
<link>Hua Mulan cong jun</link>
<link>Mulan (1998) mulan</link>
<link> (1999)</link>
<link>The Secret of Mulan (1998) mulan </link>
</movie>

<movie id="1180849">
<title>Hua (2009) hua</title>
<character>Hua Hu (Mulan's father)</character>
<character>Young Hua Mulan mulan mulan hua</character>
<character>Mulan</character>
</movie>

(b) Corrupted ranking

Figure 3.3: Original and corrupted results of $Q9$

in the original database, it drastically changes the ranking positions of these entities.

Q9: $Q9$ (*mulan hua animation*) is an *easy* query because most its keywords are quite infrequent in the database. Only term *animation* is relatively frequent in the IMDB dataset, but almost all its occurrences are in attribute *genre*. Figures 3.3a and 3.3b present two ordered top answers for $Q9$ over the original and corrupted versions of IMDB, respectively. The two results are originally ranked as 4^{th} and 10^{th}. The attribute values of these two entities contain many query keywords in the original database. Hence, adding and/or

removing some query keyword instances in these results, does not considerably change their relevance score and they preserve their ordering after corruption.

Since keywords *mulan* and *hua* appear in a small number of attribute values and attributes, the value of λ for these terms in the second and the third level of corruption is very small. Similarly, since keyword *animation* only appears in the *genre* attribute, the value of λ for all other attributes (second level corruption) is zero. The value of λ for *animation* in the third level is reasonable, 0.0007 for *movie* entity set, but the noise generated in this level alone is not considerable.

3.6 Efficient Computation of SR Score

A key requirement for this work to be useful in practice is that the computation of the SR score incurs a minimal time overhead compared to the query execution time. In this section we present efficient SR score computation techniques.

3.6.1 Basic Estimation Techniques

Top-K results: Generally, the basic information units in structured data sets, attribute values, are much shorter than text documents. Thus, a structured data set contains a larger number of information units than an unstructured data set of the same size. For instance, each XML document in the INEX data centric collection constitutes hundreds of elements with textual contents. Hence, computing Equation 3.3 for a large DB is so inefficient as to be impractical. Hence, similar to [131], we corrupt only the top-K entity results of the original data set. We re-rank these results and shift them up to be the top-K

answers for the corrupted versions of DB. In addition to the time savings, our empirical results in Section 3.8.2 show that relatively small values for K predict the difficulty of queries better than large values. For instance, we found that $K = 20$ delivers the best performance prediction quality in our datasets. We discuss the impact of different values of K in the query difficulty prediction quality more in Section 3.8.2.

Number of corruption iterations (N): Computing the expectation in Equation 3.3 for all possible values of $\vec{x}$ is very inefficient. Hence, we estimate the expectation using $N > 0$ samples over $M(|\mathcal{A}|\times V)$. That is, we use N corrupted copies of the data. Obviously, smaller N is preferred for the sake of efficiency. However, if we choose very small values for N the corruption model becomes unstable. We further analyze how to choose the value of N in Section 3.8.2.

We can limit the values of K or N in any of the algorithms described below.

3.6.2 Structured Robustness Algorithm

Algorithm 1 shows the Structured Robustness Algorithm (SR Algorithm), which computes the exact SR score based on the top K result entities. Each ranking algorithm uses some statistics about query terms or attributes values over the whole content of DB. Some examples of such statistics are the number of occurrences of a query term in all attributes values of the DB or total number of attribute values in each attribute and entity set. These global statistics are stored in M (metadata) and I (inverted indexes) in the SR Algorithm pseudocode.

SR Algorithm generates the noise in the DB on-the-fly during query processing.

Algorithm 1 *CorruptTopResults*(Q, L, M, I, N)

Input: Query Q, Top-K result list L of Q by ranking function g, Metadata M, Inverted indexes I, Number of corruption iterations N.
Output: SR score for Q.

```
1: SR ← 0; C ← {}; //C caches λ_T, λ_S for keywords in Q
2: FOR i = 1 → N DO
3:    I' ← I; M' ← M; L' ← L; //Corrupted copy of I, M and L
4:    FOR each result R in L DO
5:       FOR each attribute value A in R DO
6:          A' ← A; //Corrupted versions of A
7:          FOR each keywords w in Q DO
8:             Compute # of w in A' by Equation 3.10; //If λ_{T,w}, λ_{S,w} needed but not in C, calculate
               and cache them
9:             IF # of w varies in A' and A THEN
10:               Update A', M' and entry of w in I';
11:         Add A' to R';
12:      Add R' to L';
13:   Rank L' using g, which returns L, based on I', M';
14:   SR += Sim(L, L'); //Sim computes Spearman correlation
15: RETURN SR ← SR/N; //AVG score over N rounds
```

Since it corrupts only the top K entities, which are anyways returned by the ranking module, it does not perform any extra I/O access to the DB, except to lookup some statistics. Moreover, it uses the information which is already computed and stored in inverted indexes and does not require any extra index.

Nevertheless, our empirical results, reported in Section 3.8.2, show that SR Algorithm increases the query processing time considerably. Some of the reasons for SR Algorithm inefficiency are the following: First, Line 5 in SR Algorithm loops every attribute value in each top-K result and tests whether it must be corrupted. As noted before, one entity may have hundreds of attribute values. We must note that the attribute values that do not contain any query term still must be corrupted (Line 8-10 in SR Algorithm) for the second and third levels of corruption defined in Equation 3.10. This is because their attributes or entity sets may contain some query keywords. This will largely increase the

number of attribute values to be corrupted. For instance, for IMDB which has only two entity sets, SR Algorithm corrupts all attribute values in the top-K results for all query keywords. Second, ranking algorithms for DBs are relatively slow. SR Algorithm has to re-rank the top K entities N times which is time consuming.

3.7 Approximation Algorithms

In this section, we propose approximation algorithms to improve the efficiency of SR Algorithm. Our methods are independent of the underlying ranking algorithm.

Query-specific Attribute values Only Approximation (QAO-Approx): QAO-Approx corrupts only the attribute values that match at least one query term. This approximation algorithm leverages the following observations:

Observation 1: The noise in the attribute values that contain query terms dominates the corruption effect.

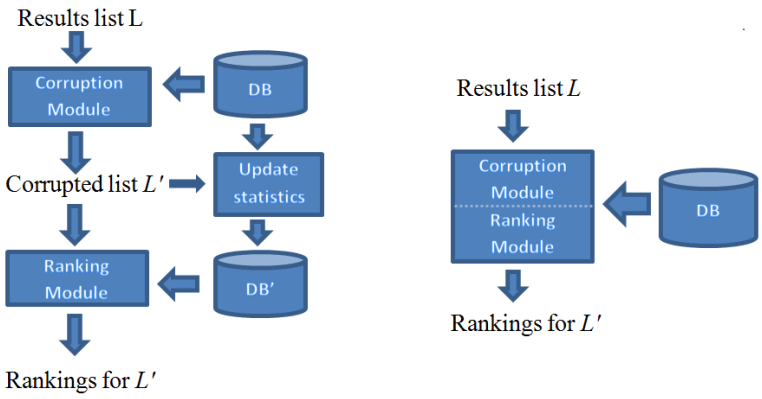
Observation 2: The number of attribute values that contain at least one query term is much smaller than the number of all attribute values in each entity.

Hence, we can significantly decrease the time spent on corruption if we corrupt only the attribute values that contain query terms. We add a check before Line 7 in SR Algorithm to test if A contains any term in Q. Hence, we skip the loop in Line 7. The second and third levels of corruption (on attributes, entity sets, respectively) corrupt a smaller number of attribute values so the time spent on corruption becomes shorter.

Static Global Stats Approximation (SGS-Approx): *SGS-Approx* uses the following observation:

Observation 3: Given that only the top-K result entities are corrupted, the global DB statistics do not change much.

Figure 3.4a shows the execution flow of SR Algorithm. Once we get the ranked list of top K entities for Q, the corruption module produces corrupted entities and updates the global statistics of DB. Then, SR Algorithm passes the corrupted results and updated global statistics to the ranking module to compute the corrupted ranking list.



(a) SR Algorithm (b) SGS-Approx

Figure 3.4: Execution flows of SR Algorithm and SGS-Approx

SR Algorithm spends a large portion of the robustness calculation time on the loop that re-ranks the corrupted results (Line 13 in SR Algorithm), by taking into account the updated global statistics. Since the value of K (e.g., 10 or 20) is much smaller than the number of entities in the DB, the top K entities constitute a very small portion of the DB. Thus, the global statistics largely remain unchanged or change very little. Hence, we use the global statistics of the original version of the DB to re-rank the corrupted entities. If we refrain from updating the global statistics, we can combine the corruption and ranking module together. This way re-ranking is done on-the-fly during corruption. *SGS-Approx*

algorithm is illustrated in Figure 3.4b.

We use the ranking algorithm proposed in [66], called *PRMS*, to better illustrate the details of our approximation algorithm. PRMS employs a language model approach to search over structured data. It computes the language model of each attribute value smoothed by the language model of its attribute. It assigns each attribute a query keyword-specific weight, which specifies its contribution in the ranking score. It computes the keyword-specific weight $\mu_j(q)$ for attribute values whose attributes are T_j and query q as $\mu_j(q) = \frac{P(q|T_j)}{\sum_{T \in DB} P(q|T)}$. The ranking score of entity E for query Q, $P(Q|E)$ is:

$$P(Q|E) = \prod_{q \in Q} P(q|E) = \prod_{q \in Q} \sum_{j=1}^{n} [\mu_j(q)((1-\lambda)P(q|A_j) + \lambda P(q|T_j))] \tag{3.11}$$

where A_j is an attribute value of E, T_j is the attribute of A_j, $0 \leq \lambda \leq 1$ is the smoothing parameter for the language model of A_j, and n is the number of attribute values in E. If we ignore the change of global statistics of DB, then μ_j and $P(q|T_j)$ parts will not change when calculating the score of corrupted version of E, E', for q. Hence, the score of E' will depend only on $P(q|A'_j)$, where A'_j is the corrupted version of A_j. We compute the value of $P(q|A'_j)$ using only the information of A'_j as (# of q in A'_j / # of words in A'_j). SGS-Approx uses the global statistics of the original DB to compute μ_j and $P(q|T_j)$ in order to calculate the value of $P(q|E)$. It re-uses them to compute the score of the corrupted versions of E. However, SR Algorithm has to finish all corruption on all attribute values in top results to update the global statistics and re-rank the corrupted results. Similarly, we can modify other keyword query ranking algorithms over DBs that use query term statistics to score entities.

Combination of QAO-Approx and SGS-Approx: QAO-Approx and SGS-Approx improve the efficiency of robustness calculation by approximating different parts of the corruption and re-ranking process. Hence, we combine these two algorithms to further improve the efficiency of the query difficulty predication.

3.8 Experiments

3.8.1 Experimental Setting

Data sets: Table 3.2 shows the characteristics of two data sets used in our experiments. The INEX data set is from the INEX 2010 Data Centric Track [118] discussed in Section 3.1. The INEX data set contains two entity sets: *movie* and *person*. Each entity in the *movie* entity set represents one movie with attributes like *title*, *keywords*, and *year*. The *person* entity set contains attributes like *name*, *nickname*, and *biography*. The SemSearch data set is a subset of the data set used in Semantic Search 2010 challenge [116]. The original data set contains 116 files with about one billion RDF triplets. Since the size of this data set is extremely large, it takes a very long time to index and run queries over this data set. Hence, we have used a subset of the original data set in our experiments. We first removed duplicate RDF triplets. Then, for each file in SemSearch data set, we calculated the total number of distinct query terms in SemSearch query workload in the file. We selected the 20, out of the 116, files that contain the largest number of query keywords for our experiments. We converted each distinct RDF subject in this data set to an entity whose identifier is the subject identifier. The RDF properties are mapped to attributes in our model. The values of RDF properties that end with substring "#type" indicates

the type of a subject. Hence, we set the entity set of each entity to the concatenation of the values of RDF properties of its RDF subject that end with substring "#type". If the subject of an entity does not have any property that ends with substring "#type", we set its entity set to "UndefinedType". We have added the values of other RDF properties for the subject as attributes of its entity. We stored the information about each entity in a separate XML file. We have removed the relevance judgment information for the subjects that do not reside in these 20 files. The sizes of the two data sets are quite close; however, SemSearch is more heterogeneous than INEX as it contains a larger number of attributes and entity sets. The size of both data sets are about 10GB, which is reasonably large for highly structured data sets, especially given that most empirical studies on keyword query processing over databases have been conducted on much smaller datasets [47, 81, 66]. We should note that structured data sets contain many more finer grained data items than unstructured data sets of the same size. Hence, KQIs over a database must process many more candidate answers than retrieval algorithms over an unstructured set of documents with the same size, and require much more time and resources. Thus, the databases used in evaluating KQIs are generally smaller than typical document collections used in studying document retrieval algorithms.

Query Workloads: Since we use a subset of the dataset from SemSearch, some queries in its query workload may not contain enough candidate answers. We picked the 55 queries from the 92 in the query workload that have at least 50 candidate answers in our dataset. Because the number of entries for each query in the relevance judgment file has also been reduced, we discarded another two queries (Q6 and Q92) without any relevant

Table 3.2: INEX and SemSearch datasets characteristics

	INEX	SemSearch
Size	9.85 GB	9.64 GB
Number of Entities	4,418,081	7,170,445
Number of Entity Sets	2	419,610
Number of Attributes	77	7,869,986
Number of Attribute values	113,603,013	114,056,158

answers in our dataset, according to the relevance judgment file. Hence, our experiments is done using 53 queries (2, 4, 5, 11-12, 14-17, 19-29, 31, 33-34, 37-39, 41-42, 45, 47, 49, 52-54, 56-58, 60, 65, 68, 71, 73-74, 76, 78, 80-83, 88-91) from the SemSearch query workload. 26 query topics are provided with relevance judgments in the INEX 2010 Data Centric Track. Some query topics contain characters "+" and "-" to indicate the conjunctive and exclusive conditions. In our experiments, we do not use these conditions and remove the keywords after character "-". Some searching systems use these operators to improve search quality. Similar to other efforts in predicting query difficulty, we left supporting these operators to the future work. Generally, KQIs over DBs return candidate answers that contain all terms in the query [15, 59, 114]. However, queries in the INEX query workload are relatively long (normally over four distinct keywords). If we retrieve only the entities that contain all query terms, there will not be sufficient number of (in some cases none) candidate answers for many queries in the data. Hence, for every query Q, we use the following procedure to get at least 1,000 candidate answers for each query. First, we retrieve the entities that contain

$|Q|$ distinct terms in query Q. If they are not sufficient, we retrieve the entities that contain at least $|Q| - 1$ distinct query keywords, and so on until we get 1000 candidate answers.

Ranking Algorithms: To evaluate the effectiveness of our model for different ranking algorithms, we have evaluated the query performance prediction model with two representative ranking algorithms: PRMS [66] and IR-Style [59]. Many other algorithms are extensions of these two methods (e.g., [81, 32]).

PRMS: We explained the idea behind PRMS algorithm in Section 3.6.

We adjust parameter λ in PRMS in our experiments to get the best MAP and then use this value of λ for query performance prediction evaluations. Varying λ from 0.1 to 0.9 with 0.1 as the test step, we have found that different values of λ change MAP very slightly on both datasets, and generally smaller λs deliver better MAP. We use $\lambda = 0.1$ on INEX and 0.2 on SemSearch.

IR-Style: We use a variation of the ranking model proposed in [59] for relational data model, referred as IR-Style ranking. Given a query, IR-Style returns a minimal join tree that connects the tuples from different tables in the DB that contain the query terms, called $MTNJT$. However, our datasets are not in relational format and the answers in their relevance judgments files are entities and not $MTNJT$s. Hence, we extend the definition of $MTNJT$ as the minimal subtree that connects the attribute values containing the query keywords in an entity. The root of this subtree is the root of the entity in its XML file. If an entity has multiple $MTNJT$s, we choose the one with the maximum score as explained below. Let M be a $MTNJT$ tree of entity E and $\mathcal{A}_{\mathcal{M}}$ be the attribute values in M. The score of M for query Q is: $\frac{IRScore(M,Q)}{size(M)}$, where $IRScore(M, Q)$ is the score of M for query

Q based on some IR ranking formula. If we use a vector space model ranking formula as in [59] to compute the $IRScore(M, Q)$, we get very low MAP (less than 0.1) for both datasets. Hence, we compute it using a language model ranking formula with Jelink-Mercer smoothing [129] which is shown in equation 3.12. We set the value of smoothing parameter α to 0.2 as it returns the highest MAP for our datasets.

$$IRScore(M, Q) = \prod_{q \in Q} \sum_{A \in \mathcal{A}_{\mathcal{M}}} ((1 - \alpha)P(q|A) + \alpha P(q|T)) \tag{3.12}$$

Configuration: We have performed our experiments on an AMD Phenom II X6 2.8 GHz machine with 8 GB of main memory that runs on 64-bit Windows 7. We use Berkeley DB 5.1.25 to store and index the XML files and implement all algorithms in Java.

3.8.2 Quality results

In this section, we evaluate the effectiveness of the query quality prediction model. We use both Pearson's correlation and Spearman's correlation between the SR score and the average precision of a query to evaluate the prediction quality.

Setting the value of N: Let L and L' be the original and corrupted top-K entities for query Q, respectively. The SR score of Q in each corruption iteration is the Spearman's correlation between L and L'. We corrupt the results N times to get the average SR score for Q. In order to get a stable SR score, the value of N should be sufficiently large, but this increases the computation time of the SR score. We chose the following strategy to find the appropriate value of N: We progressively corrupt L 50 iterations at a time and calculate the average SR score over all iterations. If the last 50 iterations do not change the average SR score over 1%, we terminate. N may vary for different queries in

query workloads. Thus, we set it to the maximum number of iterations over all queries. According to our experiments, the value of N varies very slightly for different value of K. Therefore, we set the value of N to 300 on INEX and 250 on SemSearch for all values of K.

Different Values for K: The number of interesting results for a keyword query is normally small [83]. For example, the average number of relevant results is 9.6 for the SemSearch query workload. In this setting, many low ranked answers may not be relevant and have quite close scores, which makes their relative ranking positions very sensitive to noise. If we use large values for K, the SR score will be dominated by the low ranked non-relevant results and the SR score may deem all queries almost equally difficult. Hence, it is reasonable to use small values of K for query performance prediction. We empirically show that our model prefers smaller K on these two datasets. We conduct our experiments on K=10, 20 and 50. All these values deliver reasonable prediction quality (i.e. the robustness of a query is strongly correlated with its effectiveness). However, on both datasets, K=10 and 20 deliver better prediction quality than K=50, given other parameters are the same. For instance, the value of Pearson's correlation on SemSearch is 0.398 for K=50 but 0.471 for K=10 and 0.556 for K=20. We have achieved the best prediction quality using K=20 for both datasets with various combinations of γ_A, γ_T, and γ_S. We present these experiments in details later in this Section.

Training of γ_A, γ_T, and γ_S: We denote the coefficients combination in Equation 3.10 as (γ_A, γ_T, γ_S) for brevity.

We train (γ_A, γ_T, γ_S) using 5-fold cross validation. We get two settings on each dataset by using Spearman's correlation and Pearson's correlation, respectively, to measure

the prediction quality. After some preliminary experiments, we found that large γ_A is effective. Hence, to reduce the number of possible combinations, we fix γ_A as 1, and vary the other two during the training process to find the highest Pearson's correlation between average precision and SR score. We computed the SR score for γ_T and γ_S from 0 to 3 with step 0.1 for different values of K. We found that the value of (γ_A, γ_T, γ_S) that leads to the best correlation score, is quite stable on different training sets. In the rest of the chapter, we report the results of Pearson's correlation and Spearman's correlation over INEX using the values of (1, 0.9, 0.8) and (1, 0.3, 0.5) for (γ_A, γ_T, γ_S), respectively. We also present the values of Pearson's correlation and Spearman's correlation on SemSearch that are achieved by the setting (γ_A, γ_T, γ_S) to (1, 0.1, 0.6).

Figures 3.5 and 3.6 depict the plot of average precision and SR score for all queries in our query workload on INEX and SemSearch, respectively. In Figure 3.5, we see that $Q9$ is easy (has high average precision) and $Q11$ is relatively hard, as discussed in Section 3.5.4. As shown in Figure 3.6, query $Q78$: *sharp-pc* is easy (has high average precision), because its keywords appear together in few results, which explains its high SR score. On the other hand, $Q19$: *carl lewis* and $Q90$: *university of phoenix* have a very low average precision as their keywords appear in many attributes and entity sets. Figure 3.6 shows that the SR scores of these queries are very small, which confirms our model.

Baseline Prediction Methods: We use Clarity score (CR) [115], Unstructured Robustness Method (URM) [131], Weighted Information Gain (WIG) [132], normalized-query-commitment (NQC) [105], and prevalence of query keywords as baseline query diffi-

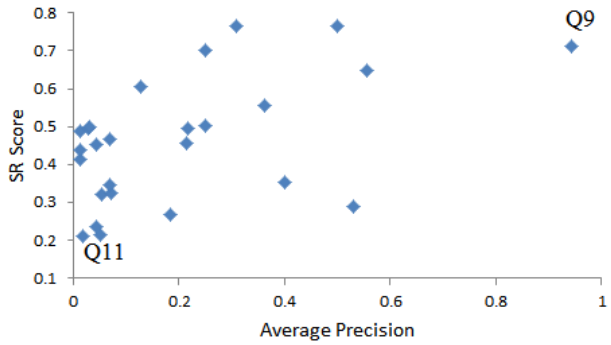


Figure 3.5: Average precision versus SR score for queries on INEX using PRMS, K=20.

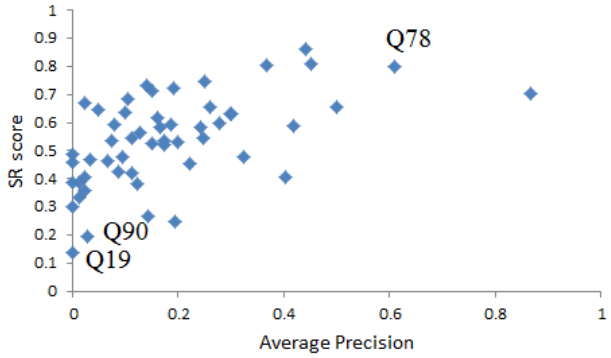


Figure 3.6: Average precision versus SR score for queries on SemSearch using PRMS, K=20.

Table 3.3: Pearson's correlation of average precision against each metric.

K	10									20								
Method	SR	WIG	NQC	URM	CR	iAA	iAES	iAE	iAS	SR	WIG	NQC	URM	CR	iAA	iAES	iAE	iAS
INEX	**0.486**	0.176	0.302	0.093	0.266	0.299	n/a	0.111	0.143	**0.564**	0.187	0.262	0.177	0.257	0.370	n/a	0.255	0.292
SemSearch	**0.471**	0.107	-0.083	0.247	0.111	0.066	0.052	0.040	-0.043	**0.556**	0.109	-0.079	0.311	0.119	0.082	0.068	0.056	-0.046

culty prediction algorithms in databases. CR, URM are two popular post-retrieval query difficulty prediction techniques over text documents. WIG and NQC are also post-retrieval predictors that have been proposed recently and are shown to achieve better query difficulty prediction accuracy than CR and URM [132, 105].

Table 3.4: Spearman's correlation of average precision against each metric.

K	10									20								
Method	SR	WIG	NQC	URM	CR	iAA	iAES	iAE	iAS	SR	WIG	NQC	URM	CR	iAA	iAES	iAE	iAS
INEX	0.303	0.242	0.381	0.196	0.199	**0.409**	n/a	-0.167	0.187	**0.475**	0.218	0.319	0.270	0.202	0.448	n/a	-0.154	0.174
SemSearch	**0.519**	0.270	0.287	-0.012	0.182	0.334	0.282	0.289	0.306	**0.576**	0.253	0.271	0.074	0.179	0.348	0.302	0.310	0.326

To implement CR, URM, WIG, and NQC, we concatenate the XML elements and tags of each entity into a text document and assume all entities (now text documents) belong to one entity set. The values of all μ_j in PRMS ranking formula are set to 1 for every query term. Hence, PRMS becomes a language model retrieval method for text documents [83]. We have separately trained the parameters of these method on each dataset using the whole query workload as the training data to get the optimal settings for these methods.

URM and CR: For URM on both datasets, we corrupted each ranking list 1000 times such that robustness score becomes stable. For CR, we trained three parameters: the number of results (k), the vocabulary size (v) used in computing query language model, and the background language model smoothing factor (λ). We report the results for CR using k=100 and λ=0.7 for INEX and k=500 and λ=0.3 for SemSearch. We have use the whole vocabulary to compute the query language model in both datasets.

WIG and NQC: In order to make a reasonable comparison, we have used the implementations of WIG and NQC from [105]. We trained the number of of top results, k, for both methods . For WIG, we set k=5 on SemSearch and k=10 on INEX. For NQC, we set k=10 on SemSearch and k=150 on INEX. We think smaller k is prefered on SemSearch for both methods because its query workload has smaller average number of relevant results per query.

Prevalence of Query Keywords: As we argued in Section 3.4.2, if the query keywords appear in many entities, attributes, or entity sets, it is harder for a ranking algorithm to locate the desired entities. Given query Q, we compute the average number of attributes ($AA(Q)$), average number of entity sets ($AES(Q)$), and the average number of entities ($AE(Q)$) where each keyword in Q occurs. We consider each of these three values as an individual baseline difficulty prediction metric. We also multiply these three metrics (to avoid normalization issues that summation would have) and create another baseline metric, denoted as $AS(Q)$. Intuitively, if these metrics for query Q have higher values, Q must be harder and have lower average precision. Thus, we use the inverse of these values, denoted as $iAA(Q)$, $iAES(Q)$, $iAE(Q)$, and $iAS(Q)$, respectively.

Comparison to Baseline Methods: Tables 3.3 and 3.4 show Pearson's and Spearman's correlation values between average precision and the metrics for SR, NQC, WIG, URM, CR, iAA(Q), iAES(Q), iAE(Q), and iAS(Q) methods for different values of K over both datasets, respectively. These results are based on all queries in the query workloads without distinguishing between training and testing sets. The n/a value appears in Table 3.3 because all query keywords in our query workloads occur in both entity sets in INEX.

The Pearson's and Spearman's correlation scores for SR Algorithm are significantly higher than those of URM, CR and WIG on both datasets for both cases of $K = 10$ and $K = 20$. SR Algorithm delivers a higher Pearson's and Spearman's correlation than NQC over both data sets for both values cases of $K = 20$, and higher Pearson's correlation over both data sets for the case of $K = 10$. SR algorithm also provides a higher Pearson's correlation

than NQC over SemSearch for $K = 10$. This shows that our prediction model is generally more effective than other methods over databases. Especially, the large improvement over URM confirms that our corruption model better captures the properties of difficult queries on databases. iAA provides a more accurate prediction than all other baseline methods over INEX but slightly worse than NQC in terms of Spearman's correlation for K=10. This indicates that one of the main causes of the difficulties for the queries over the INEX dataset is to find their desired attributes, which confirms our analysis in Section 3.4.2. SR also delivers far better prediction qualities than iAA(Q), iAES(Q), iAE(Q), and iAS(Q) metrics over both data sets. Hence, SR effectively considers all causes of the difficulties for queries over databases.

IR-style ranking algorithm: The best value of MAP for the IR-Style ranking algorithm over INEX is 0.134 for K=20, which is very low. Note that we tried both Equation 3.12 as well as the vector space model originally used in [59]. Thus, we do not study the quality performance prediction for IR-Style ranking algorithm over INEX. On the other hand, the IR-Style ranking algorithm using Equation 3.12 delivers larger MAP value than PRMS on the SemSearch dataset. Hence, we only present results on SemSearch. Table 3.5 shows Pearson's correlation of SR score with the average precision for different values of K, for N=250 and (γ_A, γ_T, γ_S) = (1, 0.1, 0.6). Figure 3.7 plots SR score against the average precision when K=20.

Discussion: Without combining with other predictors, all state-of-the-art predictors on text collections achieve linear/non-linear correlation between average precision and prediction metrics up to 0.65 depending on corpus and query workload [54, 115, 131, 56,

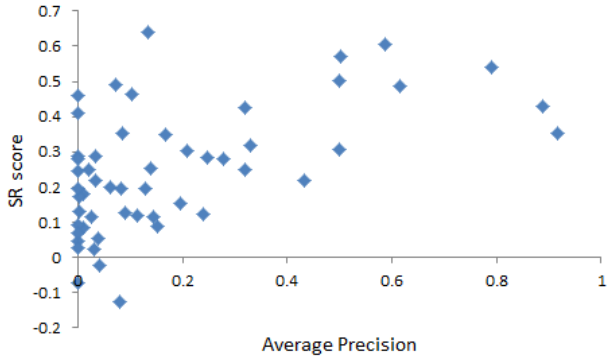


Figure 3.7: Average precision versus SR score using IR-Style over SemSearch with K=20.

Table 3.5: Correlation of average precision and SR score using IR-Style over SemSearch.

K	10	20
Pearson's correlation score	0.565	0.544
Spearman's correlation score	0.502	0.507

29, 105, 132, 55, 74, 73]. As the first work in query difficulty prediction on database, we believe our prediction quality is reasonably high.

Impact of database schema complexity: On one hand, increasing the complexity of the schema (e.g., increasing nesting or number attributes) makes it harder to locate the user-desired results. On the other hand, a richer structure may improve the quality of search if the system is able to locate the right attribute types, e.g., when the keywords only appear in a single attribute type. For the same reasons we believe there is no general rule on the effect of the schema complexity on the effectiveness of SR score.

Using SR Algorithm: Similar to other difficulty metrics, given the result of SR Algorithm for an input query, a KQI may apply a thresholding approach to categorize an input query as "easy" or "hard" [115]. This thresholding approach defines a reasonable

threshold t for a query difficulty metric. If the difficulty metric of the query is below t, it will be considered a "hard" query, and the KQI will apply further treatments like the ones discussed in Section 3.1 to it. One may apply the kernel density estimation technique proposed in [115] to find the value of t for a database. This technique computes the SR score for a large number of syntactic keyword queries that are randomly sampled from the database. It then sets the value of t to the SR score that is estimated to be less than the SR values of 80% of queries. Readers can find more information on the justification and implementation of this approach in [115].

3.8.3 Performance Study

In this section we study the efficiency of our SR score computation algorithms.

SR Algorithm: We report the average computation time of SR score (SR-time) using SR Algorithm and compare it to the average query processing time (Q-time) using PRMS for the queries in our query workloads. These times are presented in Table 3.6 for K=20. SR-time mainly consists of two parts: the time spent on corrupting K results and the time to re-rank the K corrupted results. We have reported SR-time using (corruption time + re-rank time) format. We see that SR Algorithm incurs a considerable time overhead on the query processing. This overhead is higher for queries over the INEX dataset, because there are only two entity sets, (*person* and *movie*), in the INEX dataset, and all query keywords in the query load occur in both entity sets. Hence, according to Equation 3.10, every attribute value in top K entities will be corrupted due to the third level of corruption. Since SemSearch contains far more entity sets and attributes than INEX, this process does

Table 3.6: Average query processing time and average robustness computation for K=20.

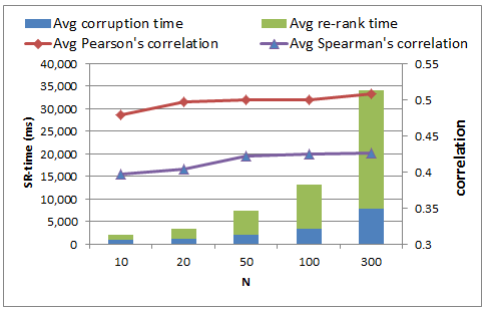
	Avg Q-time (ms)	Avg SR-time (ms)
INEX (N=250)	24,177	(88,271 + 29,585)
SemSearch (N=300)	46,726	(11,121 + 12,110)

not happen for SemSearch.

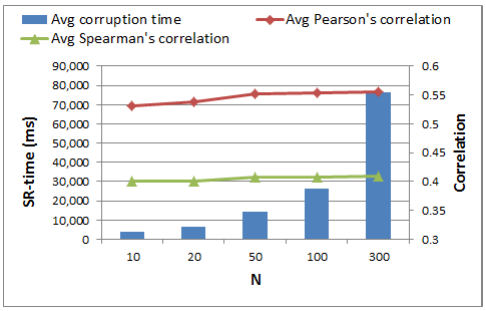
QAO-Approx: Figures 3.8a and 3.9a show the results of using QAO-Approx on INEX and SemSearch, respectively. We measure the prediction effectiveness for smaller values of N using average correlation score. The QAO-Approx algorithm delivers acceptable correlation scores and the corruption times of about 2 seconds for N=10 on INEX and N=20 on SemSearch. Comparing to the results of SR Algorithm for N=250 on SemSearch and N=300 on INEX, the Pearson's correlation score drops, because less noise is added by second and third level corruption. These results show the importance of these two levels of corruption.

SGS-Approx: Figures 3.8b and 3.9b depict the results of applying SGS-Approx on INEX and SemSearch, respectively. Since re-ranking is done on-the-fly during the corruption, SR-time is reported as corruption time only. As shown in Figure 3.8b, the efficiency improvement on the INEX dataset is slightly worse than QAO-Approx, but the quality (correlation score) remains high. SGS-Approx outperforms QAO-Apporx in terms of both efficiency and effectiveness on the SemSearch dataset.

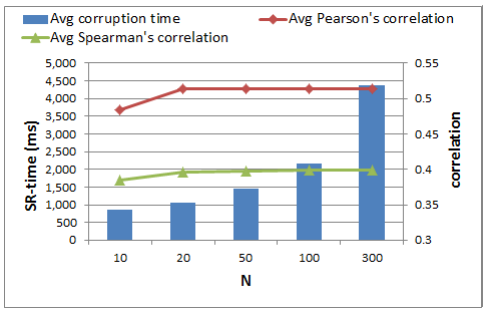
Combination of QAO-Approx and SGS-Approx: As noted in Section 3.6, we can combine QAO-Approx and SGS-Approx algorithms to achieve better performance.



(a) QAO-Approx



(b) SGS-Approx

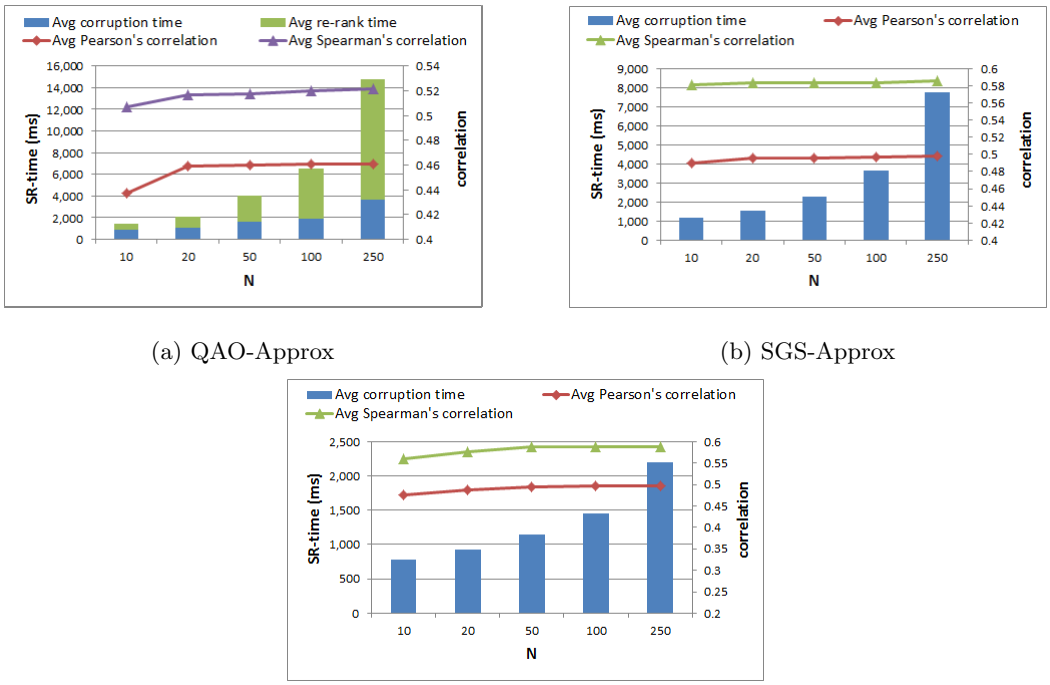


(c) Combination of QAO and SGS

Figure 3.8: Approximations on INEX.

Figures 3.8c and 3.9c present the results of the combined algorithm for INEX and SemSearch databases, respectively. Since we use SGS-Approx, the SR-time consists only of corruption time. Our results show that the combination of two algorithms works more efficiently than either of them with the same value for N.

Summary of the Performance Results: According to our performance study of QAO-Approx, SGS-Approx, and the combined algorithm over both datasets, the combined algorithm delivers the best balance of improvement in efficiency and reduction in effectiveness for both datasets. On both datasets, the combined algorithm achieves high prediction accuracy (the Pearson's correlation score about 0.5) with SR-time around 1 sec-



(a) QAO-Approx

(b) SGS-Approx

(c) Combination of QAO and SGS

Figure 3.9: Approximations on SemSearch.

ond. Using the combined algorithm over INEX when the value of N is set to 20, the the Pearson's and Spearman's correlation scores are 0.513 an 0.396 respectively and the time decreases to about 1 second. For SR Algorithm on INEX, when N decreases to 10, the Pearson's correlation is 0.537, but SR-time is over 9.8 seconds, which is not ideal. If we use the combined algorithm on SemSearch, the Pearson's and Spearman's correlation scores are 0.495 and 0.587 respectively and SR-time is about 1.1 seconds when N=50. However, to achieve a similar running time, SGS-Approx needs to decrease N to 10, with the SR-time of 1.2 seconds, the Pearson's correlation of 0.49 and the Spearman's correlation of 0.581. Thus, the combined algorithm is the best choice to predict the difficulties of queries both efficiently and effectively.

Discussion: The time to compute the SR score only depends on the top-K results, since only the top-K results are corrupted and reranked (see Section 3.6). Increasing the data set size will only increase the query processing time, which is not the focus of this work.

The complexity of data schema could have impact on the efficiency of our model. A simpler schema may not mean shorter SR computation time, since more attribute values need to be corrupted, since more attribute values of the same attribute type of interest exists. The latter is supported by the longer corruption times incurred by INEX, which has simpler schema than SemSearch, as shown in Table 3.6.

3.9 Conclusion and Future Work

We introduced the novel problem of predicting the effectiveness of keyword queries over DBs. We showed that the current prediction methods for queries over unstructured data sources cannot be effectively used to solve this problem. We set forth a principled framework and proposed novel algorithms to measure the degree of the difficulty of a query over a DB, using the ranking robustness principle. Based on our framework, we propose novel algorithms that efficiently predict the effectiveness of a keyword query. Our extensive experiments show that the algorithms predict the difficulty of a query with relatively low errors and negligible time overheads.

An interesting future work is to extend this framework to estimate the query difficulty on other top K ranking problems in DBs such as ranking underspecified SQL statements or keyword queries with exclusion operations or supporting phrases. Another

direction is to experiment with ranking functions that may not fall under the two function classes used in this work. Finally, we will extend our robustness framework for semi-structured queries, where the user specifies both structured conditions and keywords.

Chapter 4

Leveraging User Query Sessions to Improve Searching of Medical Literature

Published reports about searching medical literature do not refer to leveraging the query context, as expressed by previous queries in a session. We aimed to assess novel strategies for context-aware searching, hypothesizing that this would be better than baseline. Building upon methods using term frequency-inverse document frequency, we added extensions such as a function incorporating search results and terms of previous queries, with higher weights for more recent queries. Among 60 medical students generating queries against the TREC 9 benchmark dataset, we assessed recall and mean average precision. For difficult queries, we achieved improvement (27%) in average precision over baseline. Improvements in recall were also seen. Our methods outperformed baseline by 4% to 14%

on average. Furthermore, the effectiveness of context-aware search was greater for longer query sessions, which are typically more challenging. In conclusion, leveraging the previous queries in a session improved overall search quality with this biomedical database.

4.1 Introduction

Millions of queries are issued each day on the PubMed system [36]. Due to the importance of searching PubMed, a large number of Web-based applications have been deployed for this [79]. The Text Retrieval Conference (TREC) has included tracks for this domain of data, e.g., the TREC Genomics Tracks in 2006 and 2007 [58, 57]. An ideal query session would contain only a single query, which would always generate the most relevant results. Much work has proposed methods to improve searching of PubMed [76, 14, 79]: term frequency-inverse document frequency (TF-IDF) ranking, which reflects differences in the frequency of search terms among documents in a corpus of documents, has been shown to outperform binary search, which simply returns all documents that match all search terms [76]. Further, citation data have been used to increase the score of highly-cited publications, similarly to Googles PageRank algorithm6, which has been successful in Web Search, for searching PubMed [14]. More recently, a semantic network was constructed using domain concepts in a document to improve the PubMed data searching [89]. Despite enhancements from these methods, many users must generate more than one query to yield desired results. Improving the methods for searching may increase the chance that relevant documents will be seen, and may decrease the time required to conduct a search.

We propose a new approach to improve searching of biomedical literature, by

identifying the *context of a query*, which is orthogonal to these previous solutions, and hence could complement them. With a *search topic* in mind, users start a *query session* by submitting a query to the search engine (e.g., PubMed) to find relevant documents. While trying to satisfy their information need on the search topic, at each step they either browse more of the returned results (e.g., viewing the next page of results), or they choose to modify (reformulate) their query to convey their search topic more effectively, given the results they have seen so far. To the best of our knowledge, there has not been any study on how to leverage the query session to improve the ranking of results in biomedical literature. That is, the state-of-the-art biomedical search engines handle each query in a query session independently, i.e., a query does not affect the results of any subsequent query.

We believe this is an important direction, as research on other domains in Information Retrieval (IR), like general Web searches by consumers, has shown that the past queries in a query session can help understand the user's search topic [124]; hence, a search engine can compute the ranking not only based on the current query, but also considering the past queries in the same query session. However, to the best of our knowledge, this technique has not been applied to search biomedical data. Motivated by such research, we propose novel methods to perform context-aware search in biomedical data. We postulate that PubMed is an ideal setting for this research, since PubMed users reformulate their queries 4.44 times, on average, in a query session [36].

Objective: In this work, we study the context-aware searching of PubMed data. Our hypothesis is that a user's query session can be used to define the query context for the current query, and this context can be used to improve the ranking of the returned results.

4.2 Methods of Leveraging User Query Sessions

For the purpose of this work, we assume that the user progressively submits related queries, based on the top results of the previous queries. The context-aware ranking principle (CRP) indicates that for two associated (in the same query session) consecutive queries Q_{k-1} and Q_k, the user is likely to prefer the search results related to both queries, and thus such results should be promoted for Q_k [124]. That is, if two results of Q_k have about the same score with respect to Q_k, the user likely prefers the result that is also relevant to Q_{k-1}, since Q_{k-1} is also related to the users search topic. We incorporate this principle in our model.

We propose context-aware search strategies that build upon the two most popular ranking techniques in IR: TF-IDF-based BM25 and Language Models, which are described below. Note that our methods assume that we know when a query session starts and ends, that is, when the user switches to an entirely different search (i.e., different topic). Detecting the start and end of a query session is outside the scope of this work. To handle this, we can use existing query-session detection techniques [51, 80].

4.2.1 Incorporate context-aware search in TF-IDF ranking

We built our ranking strategy on top of the Lucene [2] system, which is a popular, open-source, high-performance text-based search engine. The ranking function of Lucene is based on the well-accepted Okapi BM25 ranking formula [93], which is a state-of-the-art TF-IDF retrieval function and is widely used. Lucene supports weighted queries, i.e., each query term can be assigned a different importance weight.

BaselineBM25: The baseline TF-IDF search method applies the Lucene ranking function and treats each user query independently of the previous queries in the query session. We denote this ranking as BaselineBM25.

M1: We used previous research in IR to represent the query context based on the past queries [103, 110]. We adopted the following ranking function, which incorporates the terms of all past queries in the query session. In particular, the probability of query term w appearing in the current query, given query sequence $Q_1, Q_2, ..., Q_n$ (Q_n is the current query) is:

$$P(w|Q_1, Q_2, ...Q_n) = \frac{1}{n} \sum_{i=1} n \frac{c(w, Q_i)}{|Q_i|} \gamma(1-\gamma)^{(n-1)} \tag{4.1}$$

where, $c(w, Q_i)$ is the number of occurrences of term w in query Q_i, and $|Q_i|$ is the number of terms in Q_i. In this work we do not consider phrases as terms, but only single words, excluding stopwords. γ is the decay factor to reflect that older queries in the query session are less important; we set $\gamma = 0.6$ in our experiments (we found no considerable difference in the results for values of γ around 0.5). We build a context-aware query Q' that includes all terms in the query session $Q_1, Q_2, ..., Q_n$. Each term is assigned an importance weight equal to its probability as computed in Equation 4.1. We refer to this context-aware ranking method as M1. We submit Q' to Lucene, which supports term weighing as mentioned above.

4.2.2 Incorporate context aware-search in Language Model ranking

A popular alternative to TF-IDF ranking is Language Model (LM)-based ranking [108], where the score of a document is the probability that the document will generate the query. In particular, a document R is viewed as a bag of words, and the probability of each word is its frequency in the document over the total number of words in the document. Then, the score $Score(R, Q)$ of a result R for query Q is the probability $Prob(Q|R)$ that a generative model, which at each step picks a term from R according to its frequency-based probability, will generate Q. More details are available in the paper of Song and Croft [108].
BaselineLM: The baseline is to consider each query separately and use the LM ranking formula [108].

Let $L_1, L_2, ..., L_n$ be the ranked results lists for $Q_1, Q_2, ..., Q_n$ (Q_n is the current query) respectively. According to CRP mentioned above, if two ranked lists L_i and L_j from the same query session are similar to each other, the results that appear in both L_i and L_j have a higher chance of being relevant to the users search topic.

Based on the key principle of LM-based ranking [108], for each result R we compute a context-aware score according to the likelihood of generating the query session $Q_1, Q_2, ..., Q_n$ given R. Making the assumption that queries are independent of each other, which is common in IR, the context-aware score of R can be computed using Equation 4.2, which expresses a standard method to combine independent pieces of evidence using a weighted average combination [28].

$$P(Q_1, Q_2, ...Q_n|R) = a_1P(Q_1|R) + a_2P(Q_2|R) + ... + a_nP(Q_n|R) \tag{4.2}$$

where a_i is the weight (importance) of Q_i. We can compute each of the $P(Q_i|R_n)$ terms using the well-studied LM ranking formula [108]. Thus, the key challenge is how to estimate weights a_i, $1 \leq i \leq n$, based on the context-aware ranking principle. We propose to calculate a_i as follows:

$$a_i = \frac{1}{n-i+1} \cdot \frac{\sum_{j=1, j \neq i}^{n} Sim(L_i, L_j)}{n-1} \tag{4.3}$$

Where $Sim(L_i, L_j)$ measures the similarity between two ranked results lists L_i and L_j. The first factor in Equation 4.3 assigns higher weight to more recent queries. Several standard measurements can compute the similarity of ranked results lists [45]. We consider three strategies to compute $Sim(L_i, L_j)$, which correspond to three proposed context-aware ranking strategies:

- **M2:** $Sim(L_i, L_j)$ is the Jaccard similarity, that is, $Sim(L_i, L_j) = \frac{|L_i \cap L_j|}{|L_i \cup L_j|}$.

- **M3:** $Sim(L_i, L_j)$ is one minus the normalized Spearman's Footrule [34] between the ranked results lists. Spearman's Footrule measures the ranking disarray. Two identical lists have Spearman's Footrule equal to 0, and two lists ranked inversely from each other have Spearman's Footrule 1. To compute the Spearman Footrule on two partial lists, where not all elements from the one list appear in the other, we used the formula by Fagin et al [45]. In this variant, two lists have Spearman's Footrule 1 if they have no common items.

- **M4:** We measured the difference between the term distributions of the concatenation of title, abstract, and MeSH terms of the top results of the two lists. For that, we use the Kullback-Leibler distance [72], which is a standard measure for this pur-

pose. Kullback-Leibler distance is non-negative and unbounded. To normalize it, we used $normKL = e^{(-kl)}$ where kl is the original Kullback-Leibler distance, such that $normKL$ is bounded to (0, 1]. When two term distributions are identical, kl is 0, thus $normKL$ is set to 1. If the kl is large, then $normKL$ is close to 0.

In all methods, we use the top-K results from each list in our experiments to measure the lists' similarity, with $K = 100$.

4.3 Related Work

Some ranking models consider the result clicks of the user [124, 103, 110]. Our approach, however, assumes that this information is not available, and only the sequence of user queries is available. For instance, user clicks may be difficult to collect given the query interface, or perhaps the result snippets provide enough information so that the user will not click on results. Further, reports of result clicks have not targeted biomedical data.

Related to how we extent TF-IDF ranking for context search, previous work has been proposed to apply query expansion in biomedical data search. Among them, some work uses domain knowledge to support the expansion [61, 34], using domain knowledge source such as Unified Medical Language System (UMLS) [78]. Relevance feedback has also been used for this purpose [27, 89]. However, this needs the user to explicitly rate the relevance of top ranked results which could be inconvenient for the users. These works are complementary to our approach.

4.4 Experimental Evaluation

4.4.1 Experimental Setting

Data Set. We used a standard benchmark dataset, TREC 9 filtering data [117], in our experiments. TREC 9 filtering contains a set of 348,566 references from MEDLINE consisting of titles and abstracts from 270 medical journals [117]. The original dataset is split into test and training sets. We used the training data set which is from year 1988 to 1991, and contains 293,856 files. Examples of documents are shown in Figure 4.1. The training set contains 63 search topics where, for each search topic, the set of relevant documents is provided. Each search topic comes with a title and description. For example, a search topic is show in Figure 4.1. Given the title and description of the search topic, users enter keyword queries to identify relevant documents in the corpus. Each page contains at most 10 search results. At most 50 unique results will be shown to the user for a search topic.

Given the high cost of recruiting domain experts to evaluate our methods, we limited the number of search topics. We want to avoid "easy" topics, where it is straightforward to formulate a keyword query that will retrieve many relevant results, and also avoid topics with a relatively low number of relevant results. We used the following selection criteria: for each topic, we created three keyword queries, the title, the description and the concatenation of title and description, and submitted these queries to Lucene to get the top-50 results. We selected a search topic if all three of the queries return fewer than 21 relevant results. Further, we required that the total number of relevant results for each selected search topic is over 40. The parameters 21 and 40 were manually selected by progressively

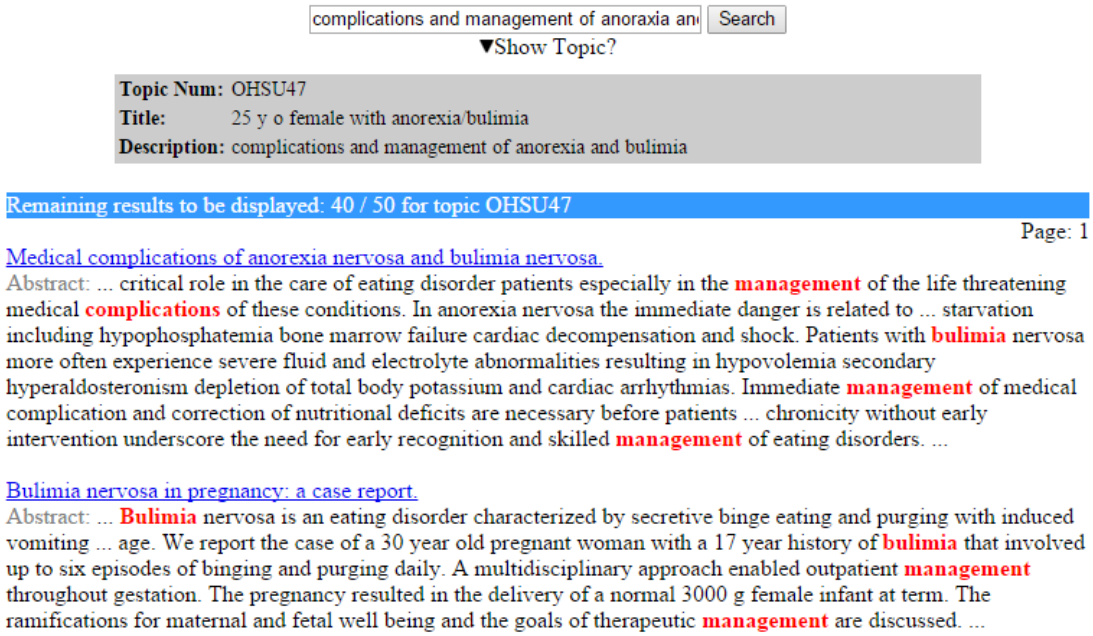


Figure 4.1: User survey interface

changing them until we keep 20-25 search topics. This led to a subset of 22 search topics.

User Study Setting. For each search topic, we display to the user the description of the topic and an empty search textbox, and ask the user to enter a keyword query (see Figure 4.1). We then display the top-10 results. We then ask the user to modify the query if the results do not look relevant enough. This process continues until 5 pages of results are displayed. We do not ask users to mark the relevant results, since we already have this information from the TREC benchmark. We do not display duplicate results across queries of a query session. The key output of the user study is a set of query sessions (sequences of queries) for each search topic.

We recruited medical students from the Indiana University School of Medicine, after receiving approval from the Indiana University Institutional Review Board. We divided

the 22 search topics into 4 groups of 5 or 6 topics per group, such that each group of search topics can be finished in about 20 minutes, which is a reasonable workload. Each user was asked to complete one set of queries, and received a $15 gift card as a reward. Sixty participants (users) completed the survey. The study generated 323 search tasks, i.e. topic-user pairs. That is, each search topic was completed by about 15 users (323 / 22 = 14.7).

4.4.2 Experimental Results

In 86 of the search tasks, the user did not modify her initial query to get the top-30 results (3 pages of 10 results each), and in 58 the user did not modify the initial query to get the top-50 results (5 pages of 10 results each). On average, in the 323 search tasks, 2 queries (i.e., the query session has 2.36 queries on average) are used for top 30 results, and 3 queries (query session has 3.47 on average) are used for top-50 results. Out of the 323 tasks, Figure 4.1 shows the distribution of query session length (the number of queries used in a query session) for top-50 (30) results.

(a) top-30 (b) top-50

Figure 4.2: Distribution of query session lengths to get top-50 results and top-30 results.

Evaluation Measures. We measured the search quality using Precision, Recall, Average Precision, and Normalized Discounted Cumulative Gain (NDCG) [83]. Average Precision and NDCG take into consideration the ranking positions of the returned relevant results. The Average Precision for the top-K results is calculated as follows:

$$AveP@K = \frac{\sum_{i=1}^{K} (p(i) \cdot rel(i))}{number\ of\ relevant\ documents} \tag{4.4}$$

where $p(i)$ is the precision at cut-off i in the ranking list, and $rel(i)$ is 1 if the item at rank i is a relevant document, and 0 otherwise. Mean Average Precision (MAP) is used to measure the average value of Average Precision for a set of queries. NDCG is a normalization of Discounted Cumulative Gain (DCG) in the range [0, 1] and is calculated as:

$$NDCG@K = \frac{DCG@K}{IDCG@K} \tag{4.5}$$

where IDCG@K is the ideal DCG@K, i.e., the maximum possible DCG value up to the ranking position K. DCG@K is calculated as:

$$DCG@K = rel(1) + \sum_{i=2}^{K} \frac{rel(i)}{log_2(i)} \tag{4.6}$$

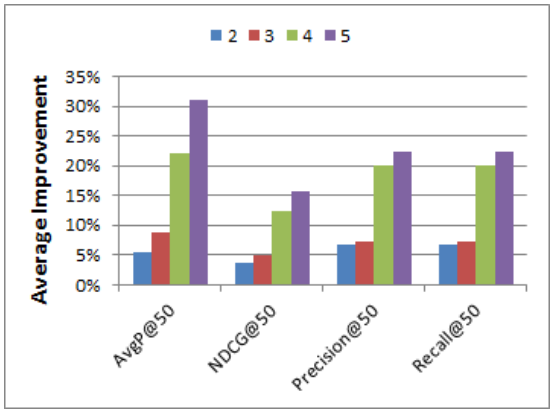
Table 4.1 shows the quality measures for each strategy, averaged over all 22 search topics. Note that our main goal is not to compare BM25 to LM (although this is a side product of our study), but to evaluate the context-aware search strategies for BM25 and LM against the corresponding baseline, that is, compare M1 against BaselineBM25, and M2 - M4 against BaselineLM. In Table 4.1, * indicates the improvement over baseline (shown in previous row) is with p-value < 0.05, and ** indicates the improvement is with p-value < 0.01.

Table 4.1: Search results quality for top-30 and top-50 from different search algorithms.

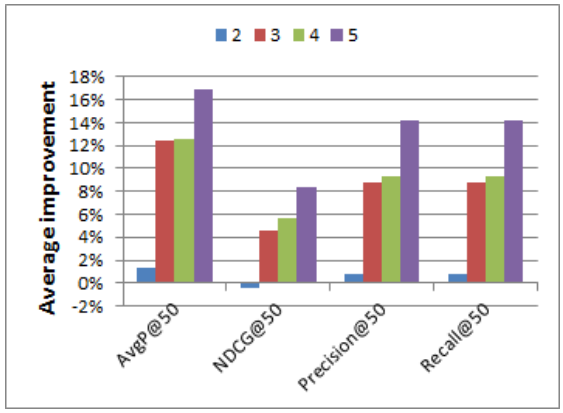
	Top-30				Top-50			
	MAP	NDCG	Precision	Recall	MAP	NDCG	Precision	Recall
BaselineBM25	0.07	0.26	0.26	0.14	0.09	0.26	0.22	0.20
M1	0.08	0.28^{*}	0.29^{*}	0.16^{*}	0.10^{**}	0.27^{**}	0.24^{**}	0.22^{**}
BaselineLM	0.08	0.27	0.28	0.15	0.10	0.26	0.24	0.21
M2, M3, M4	0.08	0.27	0.28	0.15	0.10	0.27^{*}	0.25^{*}	0.22^{*}

In Table 4.1, we observe a 4% to 14% improvement over the BaselineBM25 on different measurements for M1, e.g., M1 has MAP 0.08 on top-30 results while BaselineBM25 has 0.07, that is, the improvement is $\frac{0.08-0.07}{0.07} = 14\%$. We observe up to 5% improvement on different measurements for M1, M2 and M3 in Table 4.1 over BaselineLM. For the rest of this chapter, we will present results for M1 and M2, since M3 and M4 strategies perform very similarly to M2.

Figures 4.3 and 4.4 show the average improvement over their corresponding baselines for M1 and M2 with respect to the number of query modifications for a search topic (distribution of query session lengths is shown in Figure 4.2). Note that here we do not first average for each search topic, but each search task is treated independently, since the same search topic may have tasks with different numbers of modifications (by different users). The improvements of M1 over BaselineBM25 are with p-value < 0.01 on groups where query session length > 2 and M2 over BaselineLM are with p-value < 0.01 on groups where query session length > 3. We also observe that the improvement over the baseline is increasing for longer query sessions, as more context information can be leveraged.

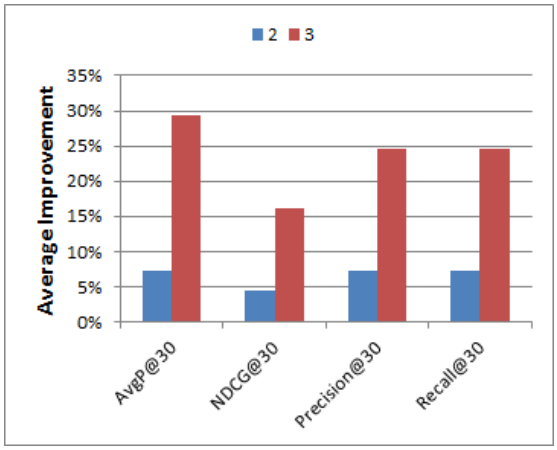


(a) M1 vs. BaselineBM25 on top-50 results

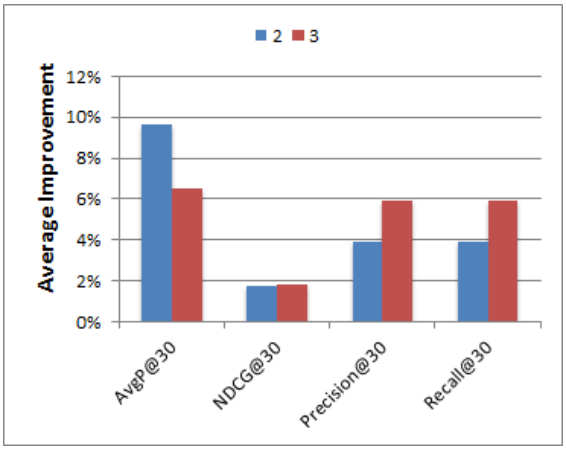


(b) M2 vs. BaselineLM on top-50 results

Figure 4.3: Search quality improvement on top-50 results over baseline for M1 and M2 with respect to the number of query modifications for a search task.



(a) M1 vs. BaselineBM25 on top-30 results



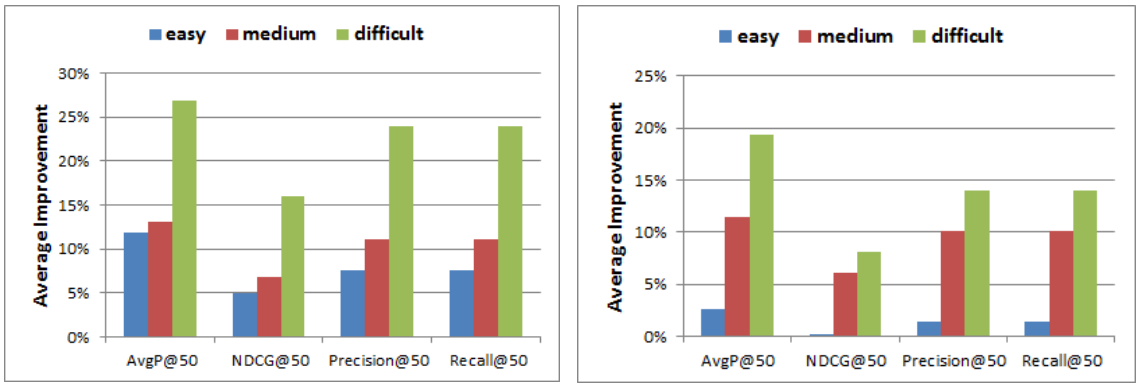
(b) M2 vs. BaselineLM on top-30 results

Figure 4.4: Search quality improvement on top-30 results over baseline for M1 and M2 with respect to the number of query modifications for a search task.

Next, we study how the effectiveness of context-aware ranking varies with the difficulty of the query. We split the 22 queries into three categories: "easy","medium" and "difficult". We use Precision@30 (Precision@50) of baseline as the splitting criterion: the 7 queries with highest Precision@30 (Precision@50) are defined as"easy" queries, the 7 queries

with lowest Precision@30 (Precision@50) are defined as difficult queries, and the remaining 8 queries are "medium". Figures 4.5 and 4.6 show the improvement of the context-aware methods for the three query categories, for top-50 (top-30) results respectively.

In Figures 4.5 and 4.6, we observe that the improvement of context-aware search (M1 and M2) over baselines is generally larger for more difficult queries. Thus, context-aware search is more useful for more difficult queries. An exception is M2 for top-30 results (Figure 4.6b), where we see a larger improvement for medium queries and even a decrease in performance for some easy queries. This shows that the BaselineLM ranking is more effective when we examine only the top-30 results as opposed to examining the top-50 results (Figure 4.5b).

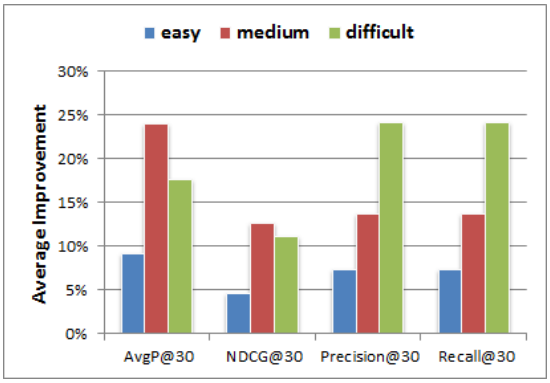


(a) M1 vs. BaselineBM25 on top-50 results

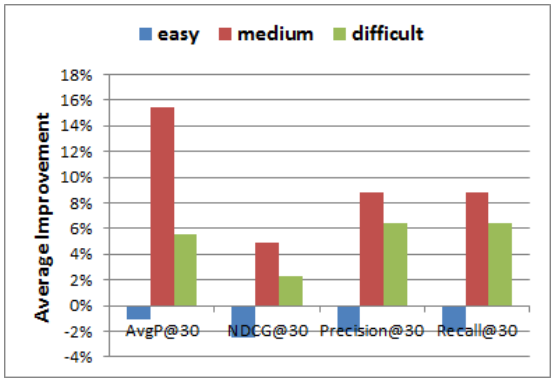
(b) M2 vs. BaselineLM on top-50 results

Figure 4.5: Search quality improvement on top-50 results over baseline for M1 and M2 on the three categories of queries.

Figure 4.7 presents the Recall@50 and Figure 4.8 depicts the AvgP@50 for the 22 search tasks for BaselineBM25, M1, BaselineLM and M2. In Figure 4.7, we observe that for 21 out of the 22 queries M1 performs better than BaselineBM25 and for 18 M2 performs



(a) M1 vs. BaselineBM25 on top-30 results



(b) M2 vs. BaselineLM on top-30 results

Figure 4.6: Search quality improvement on top-30 results over baseline for M1 and M2 on the three categories of queries.

better than BaselineLM. In Figure 4.8, we observe that for 21 out of the 22 queries, M1 outperforms BaselineBM25, and for 17, M2 outperforms BaselineLM.

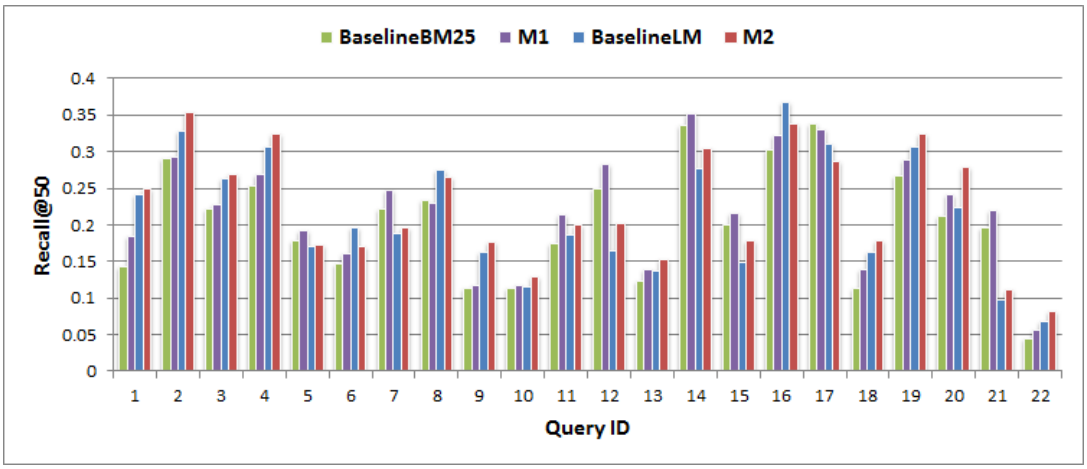


Figure 4.7: Recall@50 for all 22 queries. Recall of each query is averaged over all search tasks for that query.

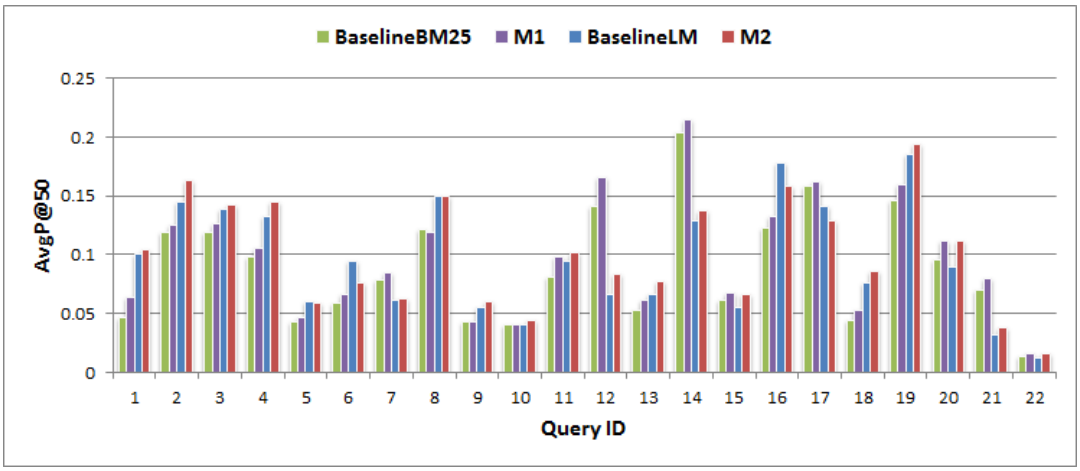


Figure 4.8: AvgP@50 for all 22 queries. Average Precision of each query is averaged over all search tasks for that query.

4.5 Discussion

Motivated by recent research in the IR community, in this work we propose and evaluate the first context-aware ranking strategies in biomedical literature, by leveraging the user query session. Our methods leverage both the search results of the queries in a query session and the terms in the queries, as evidence of a user's search context. We tested multiple strategies to incorporate the context information into state-of-the-art ranking algorithms. The experimental results confirm our hypothesis. We observe that the context-aware ranking strategies improve the quality of PubMed data search by 4% to 14% over the BM25 baseline. Our context-aware search strategies for LM-based search also outperform the baseline.

A key finding is that the improvement over the baseline methods is much higher for more difficult queries, a finding that increases the impact of our results. In particular, for more difficult queries, we achieve 27% improvement in Average Precision over the BM25 ranking baseline. Our context-aware search methods have a smaller improvement for LM-

based ranking, where we observe 8% to 19% improvement for difficult queries for top-50 results. A possible explanation is that for easier topics the users often generate accurate queries to convey the search topic from the beginning, thus the evidence inferred from past queries will not help as much as with difficult topics to remove the disambiguation of the current query. We also found that the effectiveness of context-aware search is greater for longer query sessions, which are typically more challenging, since users modify their query when they do not like the returned results.

A limitation of our experimental method is that we do not ask users to generate separate query sessions for each of the six evaluated ranking methods, but use the same query sessions to evaluate all methods. The reason is that otherwise this would require a six-fold increase in the number of subjects, which are difficult and expensive to recruit. In particular, to record the query sessions we use a TF-IDF ranking for half and an LM ranking for the other half search topics. Another limitation is that the used benchmark TREC 9 filtering dataset is relatively old, but there is no newer dataset that has the user relevance judgments needed in this project.

Our methods can be readily incorporated into existing search systems. Such a system must cache a short history of user queries to apply our methods. To define the query session, a simple but possibly cumbersome (for the user) solution is to let the user define or mark explicitly when a new query session begins. Alternatively, the system could automatically detect the start of a query session [51, 80]. In summary, based on our results, leveraging the previous queries in a query session comes closer to the ideal of one query per session, thereby improving the search quality in a biomedical search engine.

Chapter 5

Multi-Query Diversification in Microblogging Posts

Effectively exploring data generated by microblogging services is challenging due to its high volume and production rate. To address this issue, we propose a solution that helps users effectively consume information from a microblogging stream, by filtering out redundant data. We formalize our approach as a novel optimization problem termed *Multi-Query Diversification Problem (MQDP)*. In MQDP, the input consists of a list of microblogging posts and a set of user queries (e.g. news topics), where each query matches a subset of posts. The objective is to compute the smallest subset of posts that cover all other posts with respect to a "diversity dimension" that may represent time or, say, sentiment. Roughly, the solution (cover) has the property that each covered post has nearby posts in the cover that are collectively related to all queries relevant to this covered post.

5.1 Introduction

User are overloaded by the high rate of produced microblogging posts, which often carry no new information with respect to other similar posts. Our work aims at developing a method for efficiently extracting diversified and representative posts from microblogging data. Here are some examples of applications that motivate our approach: *(i)* A user would like to subscribe to several queries (or topics or hashtags) in order to receive real-time posts relevant to her interests. For example a journalist that is interested in politics might follow a set of topics such as 'White House', 'senate' or 'Barack Obama', which can be represented as hashtags in a microblogging service like Twitter. Or, an investor might subscribe to a monitoring service that provides real time information relevant to terms such as 'GOOG', 'MSFT', or 'NASDAQ'. *(ii)* Alternatively, a user may search a microblogging site by submitting a set of queries instead of individual queries; this has been shown to improve the quality of search on documents [13].

In all these scenarios many microblogging posts will be relevant to the query topics, but the complete data set is likely to include multiple redundant posts with respect to dimensions such as time or sentiment. Sifting through such data would be overwhelming.

There has been work on building efficient indexes and search techniques for real-time search on microblogging data, such as EarlyBird [17], TI [21], and LSII [122]. However, these works do not address the information overload problem. There has been also extensive work on query results diversification [7, 10, 95, 39], where the key idea is to select a small set of posts that are sufficiently dissimilar, according to an appropriate similarity metrics.

For a number of reasons these diversity models are not quite adequate for multi-

query searching or filtering applications as those described above: *(i)* Microblogging posts are too short for text distance functions to be effective – instead, we eliminate near-duplicate posts using existing duplicate detection methods like SimHash [99]. *(ii)* The query set is effectively guiding the content-based diversity, that is, the user expects to see some results for each of the query. *(iii)* Users may want to explore the data according to different diversity dimensions. Two such dimensions, especially relevant in microblogging, are time and sentiment, but other dimensions may also be useful in summarization of microblogging data.

In summary, our problem setting is fundamentally different from previous works on query results diversification in two ways: *(i)* In contrast to previous works that focus on results diversification for a single query, we study diversification with respect to multiple queries. In our setting, the user expresses her information need through a *set* of queries, for instance, by subscribing to a set of topics, like "Obama" or "economy." Since each post could be relevant to several queries, a post can be covered by a post in the results with respect to one query but not with respect to another. This motivates a *multiple-query definition of diversity coverage* where a post is covered only if it is covered with respect to *all* user-specified queries. *(ii)* Our diversity model does not use any inter-post similarity metric; instead, in our approach each post is assigned a value (e.g., timestamp) on the selected *diversity dimension*, and our method produces a subset of posts that covers the whole dimension range. As explained earlier, this model is more appropriate for the applications we are targeting.

To model this novel definition of diversity, we introduce an optimization problem

called the *Multi-Query Diversification Problem (MQDP)*, defined as follows: Given a set of user queries, we aim to identify the minimum representative subset of microblogging posts such that *(i)* posts are diverse to each other (e.g., avoid posts matching the same query with similar sentiment, or publication time), and *(ii)* all posts that are relevant to at least one query are "covered" by a selected post. We define a *diversity threshold* λ on the diversity dimension, such that two posts at distance at most λ apart may cover each other (assuming they are associated to the same queries). E.g., the threshold can be 1 hour if the time dimension is selected.

We study two variations of MQDP. In the offline (static) version of MQDP, for a given dataset of microblogging posts, we seek to identify the minimum number of representative posts that cover every post in this dataset that is related to a set of queries (e.g., represented as hashtags). We show that MQDP is NP-hard, and we propose approximate algorithms to produce the representative results efficiently. In the streaming version of MQDP, we consider the scenario where posts arrive in a streaming fashion. The objective is to extract a small subset of posts that covers all the posts in the data stream, with the selected posts produced with a bounded delay.

A key challenge is that coverage is defined both on the diversity dimension and on the queries matched, that is, two posts relevant to different queries cannot cover each other, even if they have the same value on the diversity dimension. For example, a post that only matches query 'Obama' does not cover a post that only matches query 'economy', even if they have the same timestamp (assuming that time has been selected as the diversity dimension). Further, in the streaming variant, a key challenge is how to minimize the length

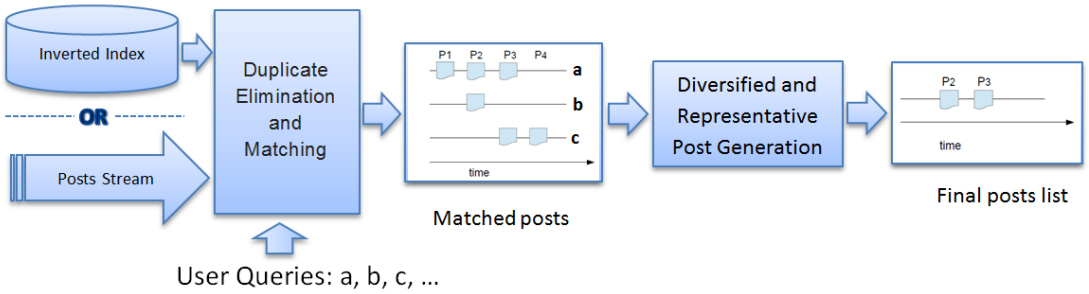


Figure 5.1: System Architecture. In this work, we focus on the Diversified and Representative Post Generation part.

of the returned diversified sub-stream, while at the same time incur a small delay in deciding if a new post should be returned or not. The naive approach would be to wait a long time after a post is published to be able to make a decision given its subsequent posts, but this would be unacceptable, as users expect very short delays when viewing microblogging data. We study these tradeoffs both theoretically and experimentally. Finally, we propose a principled approach to achieve *proportional diversity*, where we want to display to the user more posts from the more popular topics (queries), while at the same time maintaining diversity.

The system architecture is depicted in Figure 5.1, where time is selected as the diversity dimension. There are two options of providing input to the system. The first option, which corresponds to the Multi-Query Diversification problem, is by issuing a search query against an inverted index of microblogging posts. In the second option, corresponding to the Streaming Multi-Query Diversification problem, the matching module works directly on a stream of posts (e.g., Twitter stream).

Our contributions in this chapter can be summarized as follows:

- We introduce and formalize the Multi-Query Diversification problem and its streaming

variant (Section 5.2).

- We show that the Multi-Query Diversification problem is NP-hard (Section 5.3).
- We propose exact and approximation algorithms with provable approximation bounds for the Multi-Query Diversification problem and its streaming variant. We also study the tradeoff between the result size and the acceptable delay in returning a post for the streaming variant (Sections 5.4 and 5.5).
- We show a principled approach to achieve proportional diversity, where the popularity of topics (queries) is reflected in the result. For that, we show how a dynamic post-specific diversity threshold can be defined (Section 5.6).
- We conduct thorough experiments on real Twitter data and show that our proposed approximation algorithms can compute the solution efficiently and effectively (Section 5.7).

Section 5.8 presents related work, and we conclude and present future directions in Section 5.9.

5.2 Problem Formulation

Definitions. Let L be a finite set of *labels* that can represent queries (such as hashtags or news articles) and $LP(a)$ be the list of microblogging posts that are relevant to a label $a \in \mathsf{L}$. Let P be the set of all posts.

We consider a diversity dimension F that defines a total order on the posts. We represent each post $P_i \in \mathsf{P}$ as a pair $(F(P_i), \mathit{label}(P_i))$ where $F(P_i)$ is the value of the post

P_i in dimension F (for example, $F(P_i)$ can be the timestamp, or the sentiment polarity of post P_i) and $label(P_i) \subseteq \mathsf{L}$ is the set of labels that P_i is relevant to.

For ease of presentation and without harming the generality, in the remainder of this work we will assume that F represents the publication time of a post, i.e. $F(P_i) = time(P_i)$. Hence, we represent each post as a $P_i = (t_i, label(P_i))$ where $t_i = time(P_i)$ is the timestamp. If $t_i \leq t_j$, i.e., P_i is older than P_j we represent it as $P_i \prec_{time} P_j$. If both P_i and P_j are relevant to a label a and $|t_i - t_j| \leq \lambda$, then we will write that P_i *λ-covers* $a \in P_j$.

Example 1. *Consider the posts illustrated in Figure 5.2. If we define the threshold $\lambda = \Delta t$ then P_2 λ-covers $a \in P_1$ and $a \in P_3$, P_1 λ-covers $a \in P_2$, P_3 λ-covers $a \in P_2$, P_3 λ-covers $c \in P_4$, and P_4 λ-covers $c \in P_3$.*

All the above coverage examples are with respect to a single label. For our problem definition, we further define the *λ-cover* for a post and a set of posts as follows.

Definition 1. *(Post λ-cover) A post P_i is λ-covered by a set of posts Z if $\forall a \in label(P_i)$: $\exists P_j \in Z$ such that P_j λ-covers $a \in P_i$.*

Definition 2. *(Set λ-cover) Let P be a finite set of posts. $Z \subseteq \mathsf{P}$ is a λ-cover of P if $\forall P_i \in \mathsf{P}$: P_i is λ-covered by Z.*

Based on the above definitions, we formalize our Multi-Query Diversification Problem as follows:

Problem 1. *Multi-Query Diversification Problem (MQDP) Given an instance $\langle \mathsf{P}, \lambda \rangle$: a collection of posts P and a distance threshold λ, compute the minimum cardinality subset of posts $Z \subseteq \mathsf{P}$ that λ-covers P.*

Example 2. *Consider the posts depicted in Figure 5.2. Let* $P = \{P_1, P_2, P_3, P_4\}$. *The time distances between each consecutive pair of posts are all* Δt. *Again, assume* $\lambda = \Delta t$. *If we select* P_2 *and* P_4 *then* $a \in P_1$ *is* λ*-covered by* P_2, $a \in P_3$ *is* λ*-covered by* P_2, *and* $c \in P_3$ *is* λ*-covered by* P_4. *All posts have been* λ*-covered by* P_2 *or* P_4, *hence the set* $\{P_2, P_4\}$ λ*-covers* P.

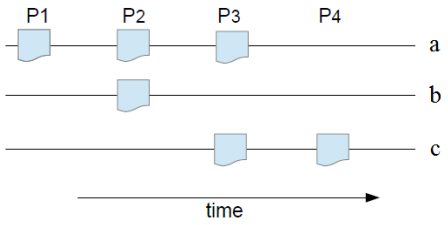


Figure 5.2: Example for coverage relations between posts.

In practice, posts might be arriving constantly. Hence, we also need to progressively report representative posts within a small time window from their publication timestamp.

Problem 2. *Streaming Multi-Query Diversification Problem (StreamMQDP) Given a set of labels (e.g. queries)* L, *a set* P *of incoming posts where each post* P_j *arrives at timestamp* t_j, *and a distance threshold* λ, *progressively report a small cardinality* $Z \subseteq \mathsf{P}$ *that* λ*-covers* P, *with the constraint that each post* $P_i \in Z$ *needs to be reported within time* τ *from time*(P_i).

In general we might want to diversify incoming posts based on a function other than their publication time, e.g. based on their sentiment polarity or distance from a user's location. The above definition of the problem can accommodate this scenario. However

in this setting post coverage will be computed based on the respective distance function defined on sentiment polarity or geolocation.

5.3 NP-Hardness of MQDP

If all posts in an instance of MQDP are issued at exactly the same time, then this instance is in essence an instance of the set cover problem, where the sets are the queries (represented as sets of labels). This immediately implies NP-hardness of MQDP; in fact, it also implies that MQDP cannot be approximated within ratio better than $\ln |L|$ [46] (under appropriate complexity-theoretic assumptions). However, the instances needed for this hardness proof require queries with arbitrary number of labels and such instances would not appear in realistic data sets. In this section we show that MQDP remains NP-hard even for instances with few labels per post.

Lemma 1. *MQDP is NP-hard, even for instances with at most two labels per post.*

Proof. We show that CNF (the satisfiability problem for conjunctive normal form formulas) reduces to MQDP in polynomial time. Let $\alpha = C_1 \wedge ... \wedge C_m$ be a CNF formula with variables $x_1, ..., x_n$, where $C_1, ..., C_m$ are clauses. We transform α into an instance $\langle \mathsf{P}, \lambda \rangle$ of MQDP such that α is satisfiable if and only if P has a λ-cover of cardinality at most $n(2m+3)$.

We now describe this construction. We will take $\lambda = 1$, and the set of labels will be $\mathsf{L} = \{w_i, u_i, \bar{u}_i\}_{i=1,...,n} \cup \{c_j\}_{j=1,...,m}$. We will have posts issued at all integral times $1, ..., 2m+3$. Specifically, for each $i = 1, ..., n$, P will contain the following posts:

(i) $(1, \{u_i, w_i\})$, $(1, \{\bar{u}_i, w_i\})$,

(ii) $(2m+3, \{u_i, w_i\})$, $(2m+3, \{\bar{u}_i, w_i\})$, and

(iii) $(2j, \{u_i\})$, $(2j, \{\bar{u}_i\})$, for all $j = 1, ..., m+1$.

Also, for each $i = 1, ..., n$ and $j = 1, ..., m$, we include posts $(2j+1, U_{ij})$ and $(2j+1, \bar{U}_{ij})$, whose label sets U_{ij} and $\bar{U}_{ij}$ depend on whether clause C_j contains variable x_i or its negation:

(iv) If $x_i \in C_j$ then $U_{ij} = \{u_i, c_j\}$, else $U_{ij} = \{u_i\}$.

(v) If $\bar{x}_i \in C_j$ then $\bar{U}_{ij} = \{\bar{u}_i, c_j\}$, else $\bar{U}_{ij} = \{\bar{u}_i\}$.

There are no other posts in P. (See Figure 5.3 for an example.)

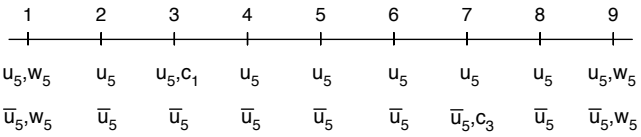


Figure 5.3: An illustration of the construction where $m = 3$, showing the posts for $i = 5$. Only the label sets are shown, to avoid clutter. The example assumes that $x_5 \in C_1$, $\bar{x}_5 \in C_3$, and that these are the only occurrences of x_5 in α.

This reduction clearly runs in polynomial time. So to complete the proof it is sufficient to show the following claim: α is satisfiable if and only if P has a λ-cover of cardinality at most $n(2m+3)$. To prove this claim, we consider both implications separately.

($\Rightarrow$) Suppose that α is satisfiable by some truth assignment $f()$. The corresponding 1-cover Z for P is constructed as follows. If $f(x_i) = 1$, we include the following posts in Z; $(1, \{u_i, w_i\})$, $(2m+3, \{u_i, w_i\})$, $(2j, \{\bar{u}_i\})$ for all $j = 1, ..., m+1$, and $(2j+1, U_{ij})$ for all $j = 1, ..., m$. If $f(x_i) = 0$, we include the following posts in Z: $(1, \{\bar{u}_i, w_i\})$, $(2m+3, \{\bar{u}_i, w_i\})$, $(2j, \{u_i\})$ for all $j = 1, ..., m+1$, and $(2j+1, \bar{U}_{ij})$ for all $j = 1, ..., m$. For each i we thus include $2+m+1+m = 2m+3$ posts, for the total of $n(2m+3)$. All labels w_i, u_i, and $\bar{u}_i$ are

easily seen to be covered. We claim that all labels c_j are covered as well. Consider any label c_j. Since α is satisfied by $f()$, at least one literal in the corresponding clause C_j is true. Suppose that $x_i \in C_j$ satisfies C_j, that is $f(x_i) = 1$. (If C_j is satisfied by $\bar{x}_i$, the argument is similar.) Then, by the definition of Z, post $(2j+1, U_i)$ is in Z and $U_i = \{u_i, c_j\}$. All occurrences of c_j are at time $2j+1$, so c_j is covered by Z, as claimed.

($\Leftarrow$) Now suppose that P has a 1-cover Z of cardinality $n(2m+3)$. Consider a subset Z_i of Z consisting of all posts that contain labels w_i, u_i or $\bar{u}_i$, that is Z_i contains all posts from Z of the following form:

- $(1, \{u_i, w_i\})$, $(1, \{\bar{u}_i, w_i\})$,
- $(2m+3, \{u_i, w_i\})$, $(2m+3, \{\bar{u}_i, w_i\})$,
- $(2j, \{u_i\})$, $(2j, \{\bar{u}_i\})$, for $j = 1, ..., m+1$, and
- $(2j+1, U_{ij})$, $(2j+1, \bar{U}_{ij})$, for $j = 1, ..., m$.

We claim that $|Z_i| \geq 2m+3$. There are $2m+3$ posts with u_i, at times $1, 2, ..., 2m+3$, so to cover them all we need at least $m+1$ posts, and the only way to cover them with $m+1$ posts is by choosing posts $(2j, \{u_i\})$, for $j = 1, ..., m+1$. Similarly, we need at least $m+1$ posts to cover all $\bar{u}_i$'s, and the only way to do that with $m+1$ posts would be to choose posts $(2j, \{\bar{u}_i\})$, for $j = 1, ..., m+1$. Note that these two sets of posts are disjoint and together they have $2m+2$ posts, with all w_i's are still uncovered. Thus Z_i must indeed contain at least $2m+3$ posts.

Since $|Z_i| \geq 2m+3$ for all $i = 1, ..., n$ and our budget for posts is $n(2m+3)$, we must have that in fact $|Z_i| = 2m+3$ for all i. Consider any i. To cover all w_i's, Z_i

must include at least one of $(1, \{u_i, w_i\})$, $(1, \{\bar{u}_i, w_i\})$ and at least one of $(2m+3, \{u_i, w_i\})$, $(2m+3, \{\bar{u}_i, w_i\})$. As covering all u_i's requires $m+1$ posts and covering all $\bar{u}_i$'s required $m+1$ different posts, So Z_i must have $m+1$ posts covering one of u_i, $\bar{u}_i$ and $m+2$ posts covering the other, with these other posts covering also all w_i's. We thus obtain that there are only two choices for Z_i:

$\{(1, \{u_i, w_i\}), (2m+3, \{u_i, w_i\})\}$

$\quad \cup \{(2j+1, U_{ij})\}_{j=1}^{m} \cup \{(2j, \bar{u}_i)\}_{j=1}^{m+1}$ or

$\{(1, \{\bar{u}_i, w_i\}), (2m+3, \{\bar{u}_i, w_i\})\}$

$\quad \cup \{(2j+1, \bar{U}_{ij})\}_{j=1}^{m} \cup \{(2j, u_i)\}_{j=1}^{m+1}$.

With the above in mind, we show that α must be satisfiable. If Z_i is of the first type, we set $f(x_i) = 1$, and if Z_i is of the second type then we set $f(x_i) = 0$. Let C_j be any clause. We need to show that C_j is satisfied. In Z the corresponding label c_j is covered, which means that Z contains some post $(2j+1, U_{ij})$ or $(2j+1, \bar{U}_{ij})$ which contains c_j. By symmetry, we can assume that $c_j \in U_{ij}$. By the form of Z_i, as described above, that means that $x_i \in C_j$. The correspondence between $f()$ and Z implies also that $f(x_i) = 1$. Thus x_i satisfies clause C_j. □

5.4 Algorithms for MQDP

A naive, exhaustive search algorithm to optimally solve MQDP would run in time exponential in $|\mathsf{P}|$, the number of posts. We first show that the computational complexity of this algorithm can be significantly reduced using dynamic programming, to running time that is exponential only in $|\mathsf{L}|$, the number of labels, which in practice is a small integer.

To reduce the running time even further, we present two polynomial-time algorithms that produce approximate solutions. The first one is inspired by solutions to the set cover problem, which has an approximation bound of $ln(|\mathsf{P}||\mathsf{L}|)$. The second one is a novel algorithm based on a traversal of the ordered (by the diversity dimension) list of input posts, which has a tighter approximation bound of s, where s is the maximum number of labels (queries) that a post may be associated with.

5.4.1 Algorithm OPT

We now propose an exact algorithm based on dynamic programming, which we refer to as *OPT*. We start with several definitions.

Definitions. Number the posts $P_1, P_2, ..., P_n$ ordered by their timestamps. Letting $t_j = time(P_j)$ for all j, we then have $t_1 < t_2 < ... < t_n$. (We assume all posts timestamps are different, for simplicity.) To simplify the description of the algorithm, we further assume that we have an additional initial post P_0 that contains all the labels, i.e., $label(P_0) = \mathsf{L}$. Any instance can be modified to have this property, by adding a new post with all labels and $\epsilon > \lambda$ time unit before the first post. This new post will have to belong to any solution, and it increases the optimum solution by exactly one post element.

We need a few other definitions. Let $f(j) = \max\left\{j' \geq j : t_{j'} \leq t_j + \lambda\right\}$ for any $j = 1, ..., n$. Define a (λ, j)-*cover* to be a set of posts $Z \subseteq \left\{P_1, ..., P_{f(j)}\right\}$ that covers all posts $P_1, ..., P_j$. Note that we do not need to include any posts after time $t_{f(j)}$, because they cannot cover any posts $P_1, ..., P_j$.

The *end-pattern* of a (λ, j)-cover Z is defined as a function $\xi : \mathsf{L} \to \{1, 2, ..., f(j)\}$ that to each $a \in \mathsf{L}$ assigns the index $\xi(a)$ of the latest post $P_{\xi(a)}$ in Z that contains a. More

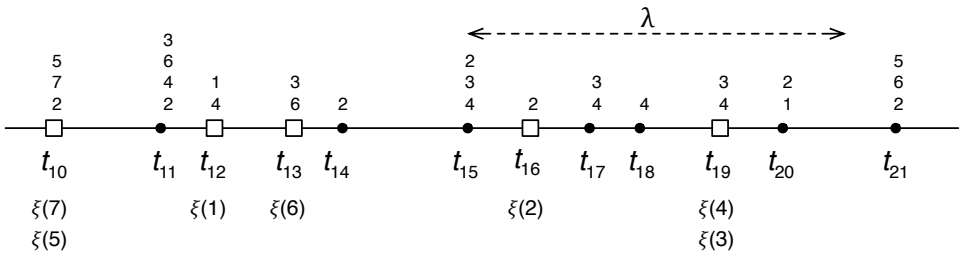


Figure 5.4: An example of an end-pattern. The label set is $\mathsf{L} = \{1, 2, 3, 4, 5, 6, 7\}$. A $(\lambda, 15)$-cover $Z = \{..., P_{10}, P_{12}, P_{13}, P_{16}, P_{19}\}$ is marked with squares. The 15-end-pattern of Z is $\xi(1, 2, 3, 4, 5, 6, 7) = (12, 16, 19, 19, 10, 13, 10)$.

precisely, we have $a \in \mathit{label}(P_{\xi(a)})$, $P_{\xi(a)} \in Z$ and $a \notin \mathit{label}(P_i)$ for each $P_i \in Z$ such that $i > \xi(a)$.

If ξ is the end-pattern of some (λ, j)-cover then we will refer to ξ as a *j-end-pattern*. It is easy to see that **a function $\xi : \mathsf{L} \to \{1, 2, ..., f(j)\}$ is a j-end-pattern if and only if it satisfies the following conditions for each label $a \in \mathsf{L}$**:

(i) For any $b \in \mathsf{L}$, if $\xi(b) > \xi(a)$ then $a \notin \mathit{label}(P_{\xi(b)})$.

(ii) if $t_{\xi(a)} + \lambda < t_i \leq t_j$ then $a \notin \mathit{label}(P_i)$.

We will denote by Ξ_j the set of all j-end-patterns. An example illustrating the definition of end-patterns is shown in Figure 5.4.

The algorithm proceeds from left to right, one post at a time. When processing each P_j, the algorithm will keep track of a set of partial solutions that cover the first j posts $P_1, ..., P_j$. This set of partial solutions is chosen so that at least one of them can be extended to a global optimal solution. On the other hand, we need to make this set small to obtain good running time.

Specifically, for each j and each $\xi \in \Xi_j$ we will compute the cardinality $h_{j,\xi}$ of

the optimal (λ, j)-cover with end-pattern equal ξ. Initially, by our assumption about post P_0, the only 0-end-pattern in Ξ_0 is ξ defined by $\xi(a) = 0$ for all $a \in \mathsf{L}$. For this ξ, we set $h_{0,\xi} = 1$.

Next, suppose that $j \geq 1$ and that we have all values $h_{j-1,\eta}$, for $\eta \in \Xi_{j-1}$, already computed. Then, for each $\xi \in \Xi_j$, $h_{j,\xi}$ is computed according to the following formula:

$$h_{j,\xi} = \min_{\substack{\eta \in \Xi_{j-1} \\ \eta \preceq \xi}} \{h_{j-1,\eta} + \Delta(\eta, \xi)\}. \tag{5.1}$$

We now explain the notations used in this formula:

- $\eta \preceq \xi$ means that η is consistent with ξ, that is, for any $a \in \mathsf{L}$, if $\xi(a) \leq f(j-1)$ then $\xi(a) = \eta(a)$.
- $\Delta(\eta, \xi) = |\{P_{\xi(a)} : \xi(a) > f(j-1)\}|$ is the number of posts in ξ that are not in η.

When the above iteration completes, the algorithm outputs $\min_{\xi \in \Xi_n} h_{n,\xi}$ as the optimum value.

Implementation. We present the algorithm in pseudocode as Algorithm 2, where we return the minimum cardinality instead of the final posts list for simplicity. Algorithm 2 can be easily modified to return the final posts if we maintain all j-end-patterns for each j such that we get the final list of posts by backtracking.

Algorithm 2 starts from the first post P_1. At step j (working on post P_j) we already have Ξ_{j-1}, and we first generate all candidate j-end-patterns, denoted $\hat{\Xi}_j$ (lines 4 - 9). For this, we only need to check the posts in the range $[t_j - \lambda, t_j + \lambda]$ as posts before this range cannot cover P_j nor later posts, so they do not affect the subsequent computation. For each candidate end-pattern $\hat{\xi}$ in $\hat{\Xi}_j$, $\forall a \in \mathit{label}(P_j) : P_{\hat{\xi}(a)}\lambda$-covers $a \in \mathit{label}(P_j)$, and for

a label $a \notin \mathit{label}(P_j)$, $\hat{\xi}(a)$ could be 0 (which means that we do not have to select any post for the labels not in $\mathit{label}(P_j)$ to λ-covers P_j), or any i that $a \in \mathit{label}(P_i)$ with $|t_i - t_j| \leq \lambda$ (lines 7 - 9). This case ($\hat{\xi}(a) = 0$) will be fixed during the computation of the cardinality of a j-end-pattern based on Ξ_{j-1}, where we only considers $\eta \in \Xi_{j-1}$ that $\eta \preceq \xi$ (lines 13 - 14 ensure this) and we update the $\xi(a)$ to $\eta(a)$ where $\xi(a) = 0$ (lines 19 - 20). We put the updated end-pattern into Ξ_j, and save or update its cardinality $h_{j,\xi}$ in H_j (lines 24 - 28) where $H_j(\xi)$ is the optimal cardinality of each end-pattern ξ in Ξ_j.

Correctness. To prove correctness, we first show feasibility, namely that for each j and $\xi \in \Xi_j$ there is a (λ, j)-cover with end-pattern equal to ξ and cardinality $h_{j,\xi}$. This can be shown by simple induction on j. Suppose that the claim holds for $j-1$. Fix any $\xi \in \Xi_j$. For this ξ, let $\eta \in \Xi_{j-1}$ be the end-pattern that realizes the minimum in (5.1). By the inductive assumption, there is a $(\lambda, j-1)$-cover Y with end-pattern η and cardinality $h_{j-1,\eta}$. By adding to Y the posts $P_{\xi(a)}$ for $a \in \mathsf{L}$ such that $\xi(a) > f(j-1)$, we obtain a (λ, j)-cover Z with end-pattern ξ and $h_{j-1,\eta} + \Delta(\eta, \xi) = h_{j,\xi}$ posts. Note that Z is indeed a correct (λ, j)-cover, since all posts $P_1, ..., P_{j-1}$ are covered by Y and P_j is covered by Z, by the definition of ξ.

Next, we argue that the final solution is indeed optimal. It is sufficient to prove that for each j and $\xi \in \Xi_j$ the value of $h_{j,\xi}$ is optimal. We again proceed by induction on j. Assume the claim holds for $j-1$. Fix any $\xi \in \Xi_j$ and define Z^* to be an optimal (λ, j)-cover with end-pattern ξ. Let Y^* be obtained from Z^* by removing the posts (strictly) after $t_{f(j-1)}$ and let η be the end-pattern for Y^*. Then $\eta \preceq \xi$ and Y^* must be an optimal $(\lambda, j-1)$-cover with end-pattern η, since otherwise, if a smaller $(\lambda, j-1)$-cover with end-

Algorithm 2 Algorithm OPT

Input: A list of microblogging posts P sorted by timestamp, each post $P_j \in \mathsf{P}$ with a set of labels $label(P_j)$, a threshold λ.

Output: The minimum cardinality of a λ-cover P.

```
1: Ξ_0 = {(0, ..., 0)}
2: H_0((0, ..., 0)) = 1
3: FOR j = 1 → |P| DO
4:    ppl ← {} //posts per label
5:    FOR a ∈ |L| DO
6:       ppl[a] ← posts in LP(a) in [t_j − λ, t_j + λ]
7:       IF a ∉ label(P_j) THEN
8:          ppl[a].add(0)
9:    Generate j-end-patterns: for each label a get one item from ppl[a], then form an end-pattern
      with these |L| items. If it is valid (see conditions of j-end-pattern), add it to Ξ̂_j.
10:   FOR ξ̂ ∈ Ξ̂_j DO
11:      FOR η ∈ Ξ_{j−1} DO
12:         ξ ← ξ̂
13:         IF ∃a ∈ L : η(a) ≠ ξ(a) ∧ 0 < ξ(a) ≤ f(j − 1) THEN
14:            next //in this case, it is impossible that η ⪯ ξ
15:         Δ ← ∅
16:         FOR a ∈ L DO
17:            IF η(a) ≠ ξ(a) ∧ ξ(a) ≠ 0 ∧ ξ(a) ∉ Δ THEN
18:               Δ.add(ξ(a))
19:            IF ξ(a) ≠ η(a) ∧ ξ(a) == 0 THEN
20:               ξ(a) ← η(a) //handle the 0 items in ξ
21:         IF ξ is not valid THEN
22:            next //see conditions of j-end-pattern
23:         card(ξ) = H_{j−1}(η) + |Δ|
24:         IF ξ ∈ Ξ_j THEN
25:            H_j(ξ) ← min{card(ξ), H_j(ξ)}
26:         else
27:            Ξ_j.add(ξ)
28:            H_j(ξ) ← card(ξ)
29: RETURN min_{ξ∈Ξ_{|P|}} H_{|P|}(ξ) − 1
```

pattern η existed, we could add to it the posts of ξ after $t_{f(j-1)}$ and obtain a (λ, j)-cover with end-pattern ξ with smaller cardinality than Z^*. Thus Y^* has cardinality $h_{j-1,\eta}$, by the inductive assumption. Consequently, the cardinality of Z^* is $h_{j-1,\eta} + \Delta(\eta, \xi)$.

Time Complexity. The number of all end-patterns is $O(|\mathsf{P}|^{|\mathsf{L}|})$, so the loop on j and ξ will iterate $O(|\mathsf{P}|^{|\mathsf{L}|+1})$ times. Computing the minimum in (5.1) takes time $O(|\mathsf{P}|^{|\mathsf{L}|})$ as well, so the time complexity will be $O(|\mathsf{P}|^{2|\mathsf{L}|+1})$.

Space Complexity. The most expensive part in terms of required space is on saving all end-patterns at each position. In order to compute the minimum cardinality, we only need to maintain the patterns for the current position and the previous position, as shown in Algorithm 2. So the space complexity will be $O(|\mathsf{P}|^{|\mathsf{L}|})$. Finally we want to get the minimum set of posts, so we need to keep the end-patterns at every position to be used for backtracking. Thus, the space complexity will be $O(|\mathsf{P}|^{|\mathsf{L}|+1})$.

5.4.2 Algorithm GreedySC

With running time $O(|\mathsf{P}|^{|2\mathsf{L}|+1})$, Algorithm OPT may still be impractical for large data sets. Thus we propose approximation algorithms to solve this problem more efficiently.

The first approach is to transform an MQDP instance $\langle \mathsf{P}, \lambda \rangle$ to a set cover problem instance and then apply the greedy set-cover algorithm. Each element of thus constructed set cover problem instance is a pair of a post $P_i \in \mathsf{P}$ and a label a: $\langle P_i, a \rangle$ where $a \in \mathit{label}(P_i)$. Thus, the universe of the set cover instance is $U = \{\langle P_i, a \rangle\}_{i=1,\ldots,|\mathsf{P}|, a \in \mathit{label}(P_i)}$. We have $|\mathsf{P}|$ sets in the set cover problem (one set for each post). The k-th set S_k contains the set of pairs that are λ-covered by picking P_k, i.e. $S_k = \bigcup_{a \in \mathit{label}(P_k)} \{\langle P_i, a \rangle : a \in \mathit{label}(P_i), |t_k - t_i| \leq \lambda\}$.

Algorithm 3 depicts this approach. For ease of presentation we refer this algorithm as *GreedySC* in the remainder of this chapter. At each iteration, GreedySC selects the set that contains the largest number of yet uncovered elements.

Approximation bound. This algorithm has an approximation ratio of $\ln k$, where k is the maximum set size [46]. In our case, $k \leq |\mathsf{P}||\mathsf{L}|$, so $|S^{GreedySC}| \leq (\ln |\mathsf{P}| + \ln |\mathsf{L}|)|S^{opt}|$. In practice, $|\mathsf{P}|$ is much larger than $|\mathsf{L}|$ and hence the error bound is essentially $\ln |\mathsf{P}|$.

Algorithm 3 Algorithm GreedySC

Input: A list of microblogging posts P sorted by timestamp, a set of labels L, each label $a \in \mathsf{L}$ with a list of relevant posts $LP(a) \subseteq P$ sorted by timestamp, a threshold λ.
Output: A subset of posts $Z \subseteq \mathsf{P}$, such that Z λ-covers P.

$S = \{S_1, S_2, ..., S_{|\mathsf{P}|}\}$, initiate each set $S_i \in S$ as $\emptyset$
$Z = \emptyset$
FOR $a \in \mathsf{L}$ **DO**
  FOR $j = 1 \rightarrow |LP(a)|$ **DO**
    FOR $i = j \rightarrow |LP(a)|$ **DO**
      IF $|time(LP(a)[j]) - time(LP(a)[i])| > \lambda$ **THEN**
        break
      $x \leftarrow$ index of $LP(a)[i]$ in P
      $y \leftarrow$ index of $LP(a)[j]$ in P
      $S_x.add(\langle LP(a)[j], a\rangle)$
      $S_y.add(\langle LP(a)[i], a\rangle)$
while true **DO**
  $i \leftarrow \arg\max_x(|S_x|)$
  IF $|S_i| == 0$ **THEN**
    break
  $Z.add(P_i)$
  FOR $j = 1 \rightarrow |\mathsf{P}|$ **DO**
    $S_j = S_j - S_i$
RETURN Z

5.4.3 Algorithm Scan

We now propose another algorithm with a provable approximation bound and better running time. The algorithm process the relevant posts of each label separately. It *scans* each sorted list $LP(a)$ to find the optimal solution S_a in terms of label a, and it outputs $S^{scan} = \bigcup_{a \in L} S_a$ as its final solution.

For each label a, the scan starts from the first post in $LP(a)$. During the scan, we keep track of the most recent uncovered post P_x. Thus, initially P_x is the first post. We scan forward until finding a post P_y such that $|t_x - t_y| > \lambda$. Then we pick the post P_z that is right before P_y and add it to S_a. Then the posts from P_x to P_z can be all marked as covered (in terms of label a). With the scan continues, we mark the posts within distance of λ to P_z as covered (in terms of label a) and we reset P_x to be the first post that has

Algorithm 4 Algorithm Scan

Input: A set of labels L, each label $a \in \mathsf{L}$ with a list of relevant posts $LP(a)$ sorted by timestamp, a threshold λ.
Output: A subset of posts $Z \subseteq \mathsf{P}$, such that Z λ-covers P.

```
1: Z = ∅
2: FOR a ∈ L DO
3:    IF |P(a)| == 0 THEN
4:       next
5:    j ← 1
6:    left ← LP(a)[j]
7:    picked ← null
8:    while j ≤ |LP(a)| DO
9:       IF |time(LP(a)[j]) − time(left)| ≤ λ THEN
10:         j ← j + 1
11:      else
12:         picked ← LP(a)[j − 1]
13:         Z.add(picked)
14:         while j ≤ |LP(a)| DO
15:            IF |time(LP(a)[j]) − time(picked)| ≤ λ THEN
16:               j ← j + 1
17:            else
18:               left ← LP(a)[j]
19:               break
20:   last ← the last post in LP(a)
21:   IF picked == null ∨ |time(last) − time(picked)| > λ THEN
22:      Z.add(last)
23: RETURN Z
```

distance larger than λ to P_z. The algorithm continues with the above procedure until it reaches the end of the list, then if the last post is not λ-covered by a selected post, then we add it to S_a. The pseudocode is presented in Algorithm 4.

Correctness. For each post P_i in $LP(a)$ there must exist a post in S_a that λ-covers $a \in P_i$ with Algorithm Scan. Since S^{scan} is the union of S_a for all $a \in \mathsf{L}$, then $S^{scan}\lambda$-covers each $P_i \in \mathsf{P}$. Thus S^{scan} is a λ-cover of P.

Approximation bound. Assume that each post is relevant with at most $s \leq |\mathsf{L}|$ labels. Then the approximation bound of Algorithm Scan is s, i.e. $|S^{scan}| \leq s|S^{opt}|$.

Proof. It is routine to prove that S_a is an optimal λ-cover of $LP(a)$. Thus, we have $|LP(a) \cap$

$S^{opt}| \geq |S_a|$. Hence $\sum_{a \in \mathsf{L}} |LP(a) \cap S^{opt}| \geq \sum_{a \in \mathsf{L}} |S_a|$. Since each post relates with at most $s \leq |\mathsf{L}|$ labels, thus $s|S^{opt}| \geq \sum_{a \in \mathsf{L}} |LP(a) \cap S^{opt}| \geq \sum_{a \in \mathsf{L}} |S_a|$. Further it holds that $\sum_{a \in \mathsf{L}} |S_a| \geq \bigcup_{a \in \mathsf{L}} |S_a| = |S^{scan}|$. It follows that $|S^{scan}| \leq s|S^{opt}|$. □

Time Complexity. Algorithm Scan examines each post in $LP(a)$ only once, thus the running time is $\sum_{a \in \mathsf{L}} |LP(a)|$, which is $O(s|\mathsf{P}|)$.

Optimizations of Algorithm Scan. Algorithm Scan processes each label separately, which results in some inefficiency, because the posts selected for one label may also cover posts from other labels. We consider an improvement to address this inefficiency. When selecting a post P_i for a label a, we remove all posts covered by P_i from all subsequent lists $LP(b)$. The effectiveness of this optimization depends on the ordering of the labels processed by Scan. We refer to this variant of Scan as *Scan+*.

5.5 Algorithms for StreamMQDP

For StreamMDQP, the posts/label matching module works directly on the stream of microblogging posts instead of a collection of posts that have been indexed. The algorithm selects a subset stream of the posts to λ-cover the whole stream. For a new relevant post, the algorithm waits a small time τ to make the decision whether output this new post or not. On one hand, to minimize the delay, we want to make a decision as soon as possible on whether a post should be outputted or not. On the other hand, a longer delay increases the probability of finding a smaller cover. Ideally, we should decide if a post should be output immediately, that is, with delay $\tau = 0$. However, as we show, this leads to an increased total number of output posts.

We show how Algorithm GreedySC and Algorithm Scan can be adapted for a streaming setting.

5.5.1 Streaming Scan

We show that when we make an instant decision ($\tau = 0$), we incur an error of up to $2s$, whereas if we have a delay of $\tau \geq \lambda$ then we have the original Algorithm Scan error bound of s.

Delayed ouput of a new post. As we process the stream in chronological order, we keep the list of posts that have been already outputted. New posts can be added to the output if they are not λ-covered by the previously selected ones. In order to decide whether a new post has to be included in the results we can apply Algorithm Scan, which is natural for streaming environments, since it processes the posts in order.

The algorithm, for each label $a \in \mathsf{L}$, keeps track of the following posts: the oldest and the latest uncovered relevant posts, denoted as $P^{ou}(a)$ and $P^{lu}(a)$ respectively, and the latest outputted relevant post $P^{lc}(a)$. And the algorithm waits until time $\min\{time(P^{lu}(a)) + \tau, time(P^{ou}(a)) + \lambda\}$ to output $P^{lu}(a)$, at the same time set $P^{lc}(a)$ to $P^{lu}(a)$ and both $P^{ou}(a)$ and $P^{lu}(a)$ to null. When a new post P_x arrives, for each label $a \in label(P_x)$, if $P^{lc}(a)$ λ-covers $a \in P_x$, the algorithm doesn't update anything for label a. Or else the algorithm sets $P^{lu}(a)$, as well as $P^{ou}(a)$ if it is originally null, to P_x. We denote this algorithm as *StreamScan*.

Similar to the improvement of Algorithm Scan by Scan+, we can apply the same idea to StreamScan as *StreamScan+*.

Approximation bound. This algorithm outputs posts exactly as Algorithm Scan when $\tau \geq \lambda$,

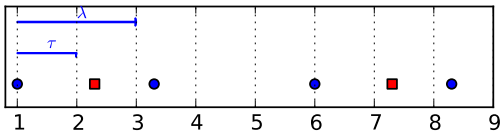


Figure 5.5: An example with approximation ratio 2, for $0 < \tau < \lambda$

thus they have the same approximation error bound, i.e. s.

Instant output of a new post. Instead of waiting up to τ before a post to be outputted, we can directly decide whether to output a post or not right after it arrives. For this, we use a small cache that keeps track of the most recently selected posts for each label $a \in \mathsf{L}$. When a new relevant post P_x arrives, if P_x is not covered, we output P_x and update the cache appropriately.

Approximation bound. We can show that the above algorithm achieves a $2s$ approximation bound if each post is relevant to at most s labels when $0 \leq \tau < \lambda$. In order to prove this, we will firstly consider the case when there is only one label. Assume that the results from Algorithm StreamingScan consists of posts $P_{i_1}, P_{i_2}, ..., P_{i_n}$ with timestamps $t_{i_1} < t_{i_2} < ... < t_{i_n}$. For $1 \leq j < n$, the distance between t_{i_j} and $t_{i_{j+1}}$ is larger than λ (otherwise they will cover each other and the algorithm will not pick both of them). Then $t_{i_{j+2}} - t_{i_j} > 2\lambda$ thus no post can λ-cover both P_{i_j} and $P_{i_{j+2}}$. Hence, an optimal solution must contain no less than $n/2$ posts. Thus the size of solution from this algorithm for label a, $S_a \leq 2S_a^{opt}$, where S_a^{opt} denotes the optimal solution size for the single label a.

We show an example in Figure 5.5 for $0 < \tau < \lambda$, where the optimal solution consists of the posts presented by red squares, and the algorithm will return the posts presented as blue dots. Thus the error factor is 2.

Recall the analysis of approximation bound of Algorithm Scan, the sum of optimal solution size from each single label is less than s times of global optimal solution size, i.e. $\sum_{a \in \mathbb{L}} |S_a^{opt}| \leq sS^{opt}$, where S^{opt} is the global optimal solution. Thus StreamingScan has a $2s$ approximation bound in the case of $0 < \tau < \lambda$.

5.5.2 Streaming Version of GreedySC

Delayed ouput of a new post. If we can tolerate a delay of $\tau > 0$, then the streaming Set-cover-based algorithm works as follows: Assume P' is the oldest post that has not been covered yet, and assume its timestamp is $time(P')$. Then, we wait until time $time(P') + \tau$; let Z be the set of posts with timestamp between $time(P')$ and $time(P') + \tau$. We execute the GreedySC algorithm in Z selecting posts to output, until posts in Z are all covered. We pass the ones that are already covered and set again $time(P')$ to be the oldest uncovered post (the first post in subsequent stream that is not λ-covered by selected posts), and we repeat the same procedure. We denoted this algorithm as *StreamGreedySC*.

We can have a variation of StreamGreedySC, referred as *StreamGreedySC+*: instead of executing GreedySC on Z until all posts are covered, we can stop GreedySC on Z once P' is covered and then update the oldest uncovered post P' (which is possibly in Z).

Instant ouput of a new post. If we want to instantly make a decision on whether to output a post, i.e. $\tau = 0$, then the streaming version of the Set-cover-based algorithm is the same as the one for Algorithm Scan, which has worst case error bound of $2s$, as explained above.

5.6 Proportional Diversity Through Variable λ

In the previous sections, we studied the Multi-Query Diversification problem assuming that the coverage parameter λ is applied uniformly for the whole range. However, this does not address the problem of *representativeness* of the results. For example, if many posts are posted in the morning but few in the afternoon, it may be desirable to return more morning posts in the diversified result. Similarly, if we focus on the sentiment diversity, the selected posts must reflect the distribution of the public's sentiment. For example, news about a decrease in the national unemployment rate would receive more positive posts, and hence we want to display more positive posts.

We propose to consider a different λ for each post, such that λ is larger in sparse areas and smaller in dense areas. The intuition is that if there are many say negative posts, then a negative post should cover another negative post only if they are very close to each other in terms of sentiment score. This will lead to displaying more negative posts, to better represent the complete set of posts.

More specifically, we define a λ value for each pair of post and label that this post matches, i.e., for each post $P_i \in \mathsf{P}$ and label $a \in \mathit{label}(P_i)$ we define $\lambda_a(P_i)$, proportional to the density of posts around P_i that match label a.

However, we don't want the variation of λ to be too drastic, because then rare perspectives would not get represented. For example, if there are 20 negative posts and 2 positive, and we only show 3 to the user, it would make sense to also show one positive one. For this reason, it makes sense to choose λ to be a non-linear function of post density. We propose here a smooth diversity formula, inspired by the work in [123] on single-query

searches, with $\lambda_a(P_i)$ defined by:

$$\lambda_a(P_i) = \lambda_0 e^{1-\frac{density_a(t_i-\lambda_0, t_i+\lambda_0)}{density_0}}, \tag{5.2}$$

where λ_0 is a constant threshold set by a domain expert, $density_a(t_i - \lambda_0, t_i + \lambda_0)$ is the density of posts that match label a in the time range $[t_i - \lambda_0, t_i + \lambda_0]$ (i.e., the number of posts per minute matching label a in this time range) and $density_0$ is the average density of posts in a $2\lambda_0$ time interval across all labels (i.e., the average number of posts per minute relevant to any label $a \in \mathsf{L}$).

By applying Equation 5.2, we achieve proportional diversity in terms of both *(i)* labels, i.e., the output will contain more posts from the labels with larger number of matching posts, and *(ii)* the diversity dimension, e.g., the output will contain more posts from the time intervals with more posts.

The astute reader will notice that in contrast to the fixed λ setting, when λ is post-specific, the post coverage relation becomes directional. That is, it is possible that P_i λ-covers $a \in P_j$ but not P_j λ-covers $a \in P_i$. Nevertheless, all proposed algorithms can be easily adapted to incorporate this property. It does not fundamentally change the implementation of these algorithms except computing λ for each post per label. For Algorithm GreedySC and Algorithm Scan, it is straightforward to apply post and label-specific λs. For Algorithm OPT, one detail is that when processing post P_j we need to generate j-end-patterns and hence we need to find all the posts P_i λ-covers $a \in P_j$, where $|t_i - t_j|$ may be larger than $\lambda_a(P_j)$ as there might be $\lambda_a(P_i) > \lambda_a(P_j)$ s.t. P_i λ-covers $a \in P_j$ but not P_j λ-covers $a \in P_i$. This could potentially reduce the efficiency of OPT.

5.7 Experimental Evaluation

In this section we study the effectiveness and efficiency of the proposed algorithms for MQDP and StreamMQDP. We describe the experimental setting in Section 5.7.1. Sections 5.7.2 and 5.7.3 study the effectiveness and efficiency of the algorithms, respectively.

5.7.1 Experimental Setting

Datasets. We conduct our experiments on Twitter data, that is, each document is a tweet. We use topic modeling to extract a set of topics, which we use as queries (labels), that is, each topic is mapped to a query,

Posts dataset. We used the Twitter Streaming API through which we can collect a random sample of up to 1% of the whole public Twitter stream. We ran the streaming API for 24 hours, on June 12th, 2013, and collected about 4.3 million tweets.

Queries. Recall that a key motivation of our work is to monitor posts related to a user profile, represented as a set of keyword queries. Given the lack of public profile datasets, and the fact that a large ratio of tweets are commenting or referring to news articles [75], we generate a query set by viewing each news topic as a query.

We use a news articles collection to extract topics, instead of using the tweets dataset, because we expect that the topics quality will be higher and also we want to avoid any bias to the algorithms from selecting queries directly based on the documents dataset. In particular, we collected news articles from several popular news websites, such as CNN, BBC, NY Times, LA Times etc., through their RSS feeds during the first half year in 2013 (until Jun 15th). This news collection consists of over 1 million articles. We applied

unsupervised Latent Dirichlet Allocation (LDA) using an open source implementation by Mallet[1] to generate 300 topics on this news collection (the number of topics is an input parameter). Each trained topic is a set of keywords with corresponding weights. We keep the top 40 highest-weight keywords for each topic.

To generate label sets L (user profiles), we assume that each user is interested in a broad topic like politics or sports, and specifies queries inside this broad topic. The 300 extracted topics are grouped into 10 broad topics by three researchers in our lab (if some researcher thought a topic was too ambiguous we discarded the topic, thus we have 215 topics left). Then, to generate a label set L, we first randomly pick a broad topic and then randomly pick $|\mathsf{L}|$ topics within the broad topic. Table 5.1 shows some example topics.

Topic	Keywords
Sports	woods tiger golf masters championship mcilroy garcia pga augusta rory mickelson
	nfl super bowl blog draft ravens ers football baltimore patriots jets quarterback giants eagles
Politics	obama president barack michelle inauguration house administration congress presidential republicans
	election vote poll presidential party president political race candidate campaign electoral coalition

Table 5.1: Example topics with their highest weight keywords.

[1] http://mallet.cs.umass.edu/

$\|L\|$	number of posts
2	136
5	308
20	1180

Table 5.2: Number of matching posts per minute, for a label set for various label set sizes.

We create label sets with different sizes ($|L|$). For each $|L|$, we create 100 label sets. Table 5.2 shows the average number of unique tweets matching at least one label set per minute, where matching is defined as containing at least one keyword of the topic (label).

Implementation and Platform. The tweets inverted index shown in Figure 5.1 was implemented using Apache Lucene[2]. Other real-time indexing systems are also possible, such as EarlyBird [17] or LSII [122], although indexing is out of the scope of this work. Note that an index is only used for the MQDP and not for the streaming problem variants. We have implemented all algorithms in Java, and we conducted our experiment on a Windows 7 machine with Intel i5 3.0GHz CPU and 8 GB RAM.

5.7.2 Effectiveness Study

MQDP. Given that our exact dynamic programming algorithm OPT can only be executed on small problem instances, when evaluating the error of approximation algorithms, we only use a 10 minutes subset of our Twitter dataset, which starts at 12pm on Jun 13, and use small values for λ and $|L|$, such that the number of j-end-patterns in OPT is not excessively large.

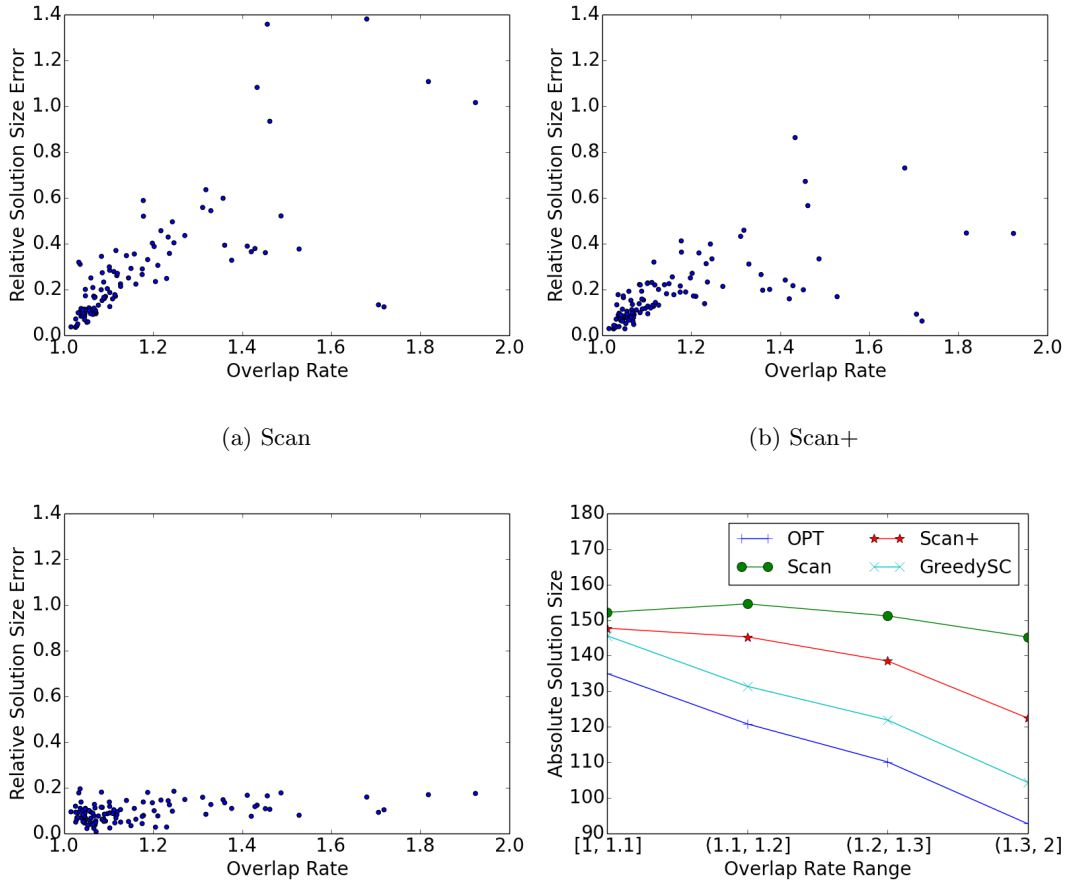
[2]http://lucene.apache.org/

A key factor that may affect the effectiveness of the approximation algorithms is the overlap among tweets with respect to the labels of a label set. That is, if many tweets match multiple labels, then the problem is more challenging and hence the algorithms may have higher error bounds. We define the *post overlap rate* as the average number of labels a post is related to. Figure 5.6 shows the relative solution size error ($|estimated - optimal|/optimal$) of the approximation algorithms for various post overlap rates, for $|\mathsf{L}| = 3$. Each point in Figures 5.6a, 5.6b and 5.6c represents a label set. GreedySC generally has better (smaller) error than Scan and Scan+ except when the overlap rate is very close to 1; recall that Scan and Scan+ are optimal for a single label ($|\mathsf{L}| = 1$), and hence are also optimal for multiple labels if the posts have no overlap (no post is related to multiple labels); this is not the case for GreedySC. Figure 5.6d shows that the solution sizes in all algorithms drops when post overlap rate increases, as they can pick posts that cover posts matching multiple labels.

Figure 5.7 depicts the relative solution size error of the approximation algorithms for various λ values. We see that all approximation algorithms have higher error with larger λ values, because there are more possible choices and hence the problem becomes harder.

Figure 5.8 shows the solution sizes of the approximation algorithms on larger instances for varying number of labels, using the whole 1-day dataset. We see that the solution size of Scan is linear on $|\mathsf{L}|$ since it handles each label separately. GreedySC outperforms the other algorithms, especially as $|\mathsf{L}|$ increases.

StreamMQDP. In the streaming setting, it is tricky to define what the optimal solution is because an algorithm has to make a decision about outputting a post before knowing



(a) Scan

(b) Scan+

(c) GreedySC

(d) Absolute solution size

Figure 5.6: Solution size errors and absolute solution sizes for $|\mathsf{L}| = 3$ and λ=5 seconds on a 10 minute interval, for varying overlap.

what will follow. To avoid this confusion, we consider as optimal the solution of an optimal algorithm that has full knowledge of the future posts. That is, the optimal streaming solution for a time interval is the same as the optimal static solution for the same interval. We again use a 10-minute time interval when the optimal solution is required.

Figure 5.9 presents the relative error for various λ values, given a fixed decision delay τ. We see that the relative errors generally increase as λ increases, since more coverage combinations are possible and hence the problem is harder; this is consistent with the

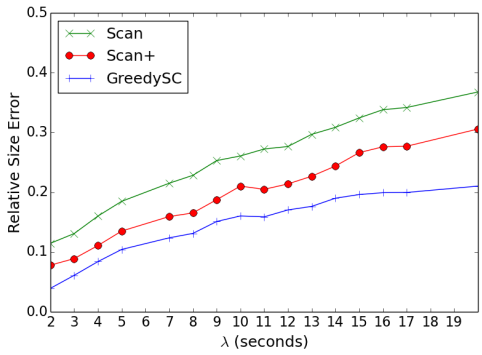
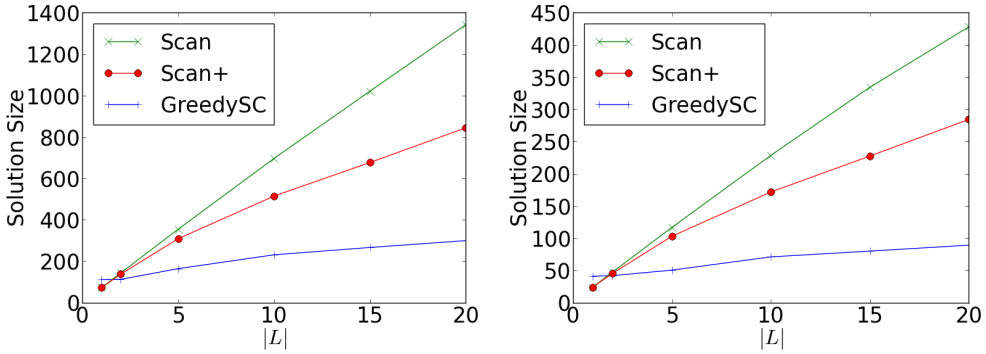


Figure 5.7: Relative solution size error for $|\mathsf{L}| = 2$ for varying λ on a 10 minute interval.



(a) $\lambda = 10$ minutes

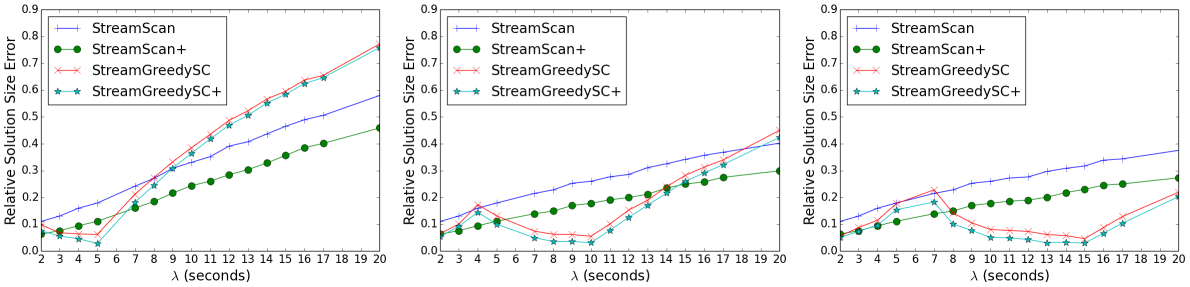
(b) $\lambda = 30$ minutes

Figure 5.8: Solution sizes on 1 day of tweets, for various label set sizes ($|\mathsf{L}|$).

analysis we did for MQDP. We also see that StreamGreedySC+ is consistently slightly better than StreamGreedySC.

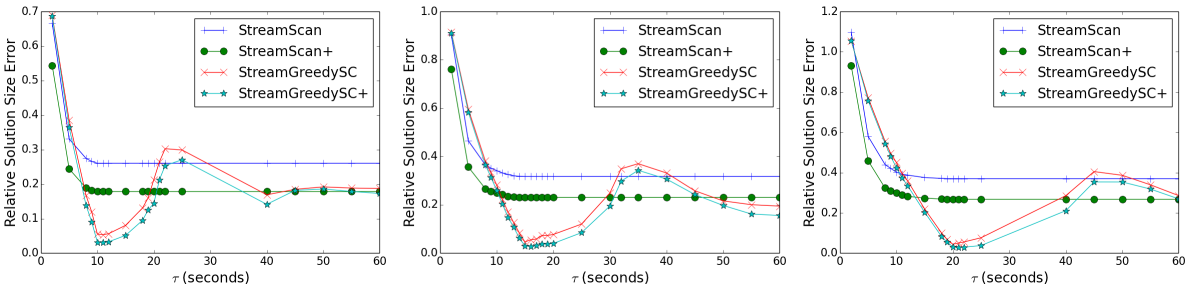
Figure 5.10 shows the relative error for different τ values, given a fixed λ. We see that the Scan-based algorithms have stable error when $\tau > \lambda$ because then the streaming Scan algorithms generate the same solution as their non-streaming counterparts as discussed in Section 5.5.1.

A surprising and interesting observation for the greedy algorithms in both Fig-



(a) $\tau = 5$ seconds (b) $\tau = 10$ seconds (c) $\tau = 15$ seconds

Figure 5.9: Relative solution size errors on 10-minute interval for varying λ when $|\mathsf{L}| = 2$.



(a) $\lambda = 10$ seconds (b) $\lambda = 15$ seconds (c) $\lambda = 20$ seconds

Figure 5.10: Relative solution size errors from approximation algorithms with respect to τ when $|\mathsf{L}| = 2$.

ures 5.9 and 5.10 is that the error has a local peak when τ is slightly larger than 2λ and the smallest error is achieved when $\lambda = \tau$. We can explain the behavior based on the "in-between" posts, that is, short ($<< \lambda$) ranges of uncovered posts that are between already covered ranges, and to cover them, we incur big overlap with what is already covered. Hence: (a) We have a minimum at $\tau = \lambda$ because we are making sure that there are no "in-between" posts. (b) When $\tau >= \lambda$ we have a maximum at τ slightly bigger than 2λ because this maximizes the effect of "in-between" posts. The algorithm has a relatively high probability of using two (or three) posts to cover the posts of a label in this τ interval.

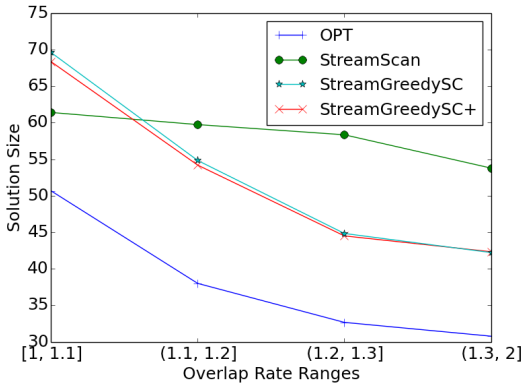


Figure 5.11: Absolute solution size when $|\mathsf{L}| = 2$ for a 10-minute interval for various overlap rate ranges.

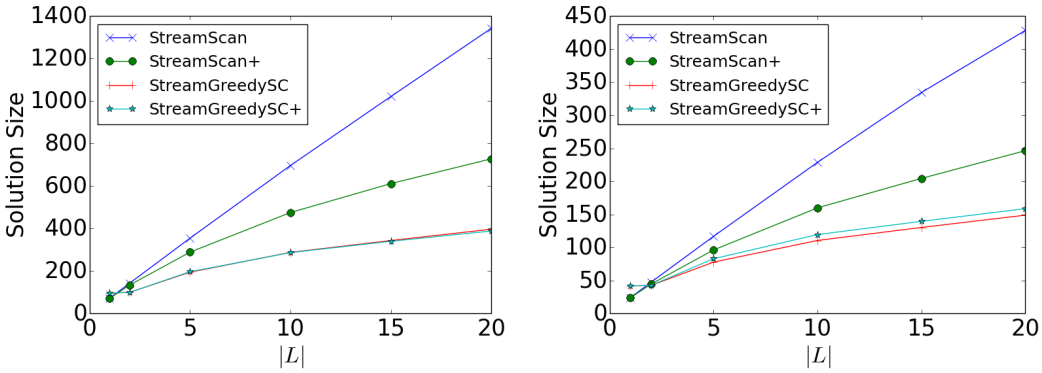
Figure 5.11 shows the effect of the overlap rate on the solution sizes for $\lambda = 10$ seconds and $\tau = 5$ seconds. We see that the approximation algorithms follow the same trend as their static versions, that is, the greedy algorithms are better for higher overlap, whereas the Scan algorithms are better for small overlap (recall that Scan is optimal for overlap = 1).

Similarly to Figure 5.8 for MQDP, Figure 5.12 shows the solution sizes of the approximation algorithms for StreamMQDP on one day of tweets. It shows that StreamGreedySC is better than StreamGreedySC+ on large λ.

5.7.3 Efficiency Study

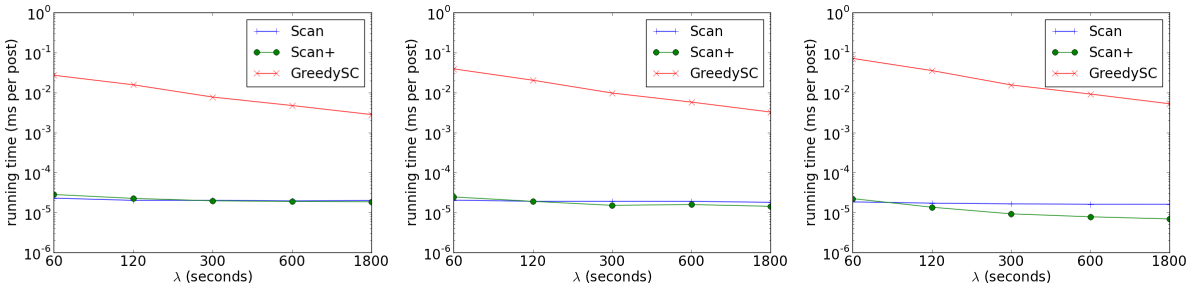
We conduct our experiments on the one-day dataset. We measure the execution time of the algorithms on in-memory data, that is, we do not account for the I/O of loading the inverted indexes into memory.

Since different queries (labels) may return quite different number of relevant posts, we compute the *execution time per post*, which is what is important to understand the



(a) $\lambda = 10$ minutes (b) $\lambda = 30$ minutes

Figure 5.12: Solution sizes of approximation algorithms for 1-day of posts for various $|\mathsf{L}|$, for $\tau = 30$ seconds.

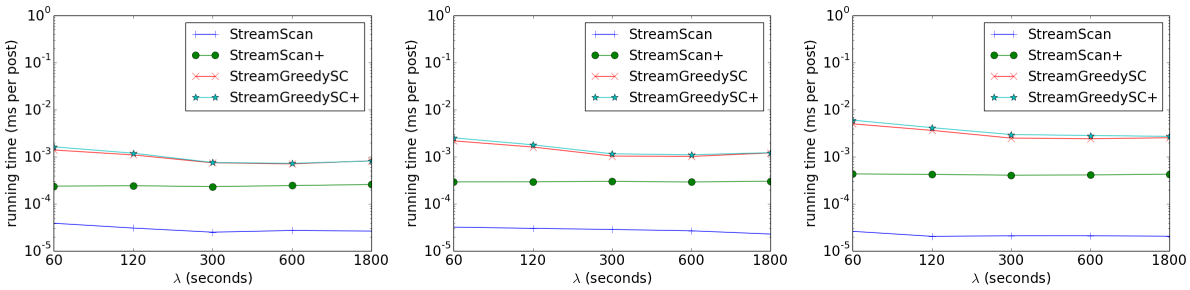


(a) $|\mathsf{L}| = 2$ (b) $|\mathsf{L}| = 5$ (c) $|\mathsf{L}| = 20$

Figure 5.13: Execution time for MQDP on one day of tweets, for varying λ.

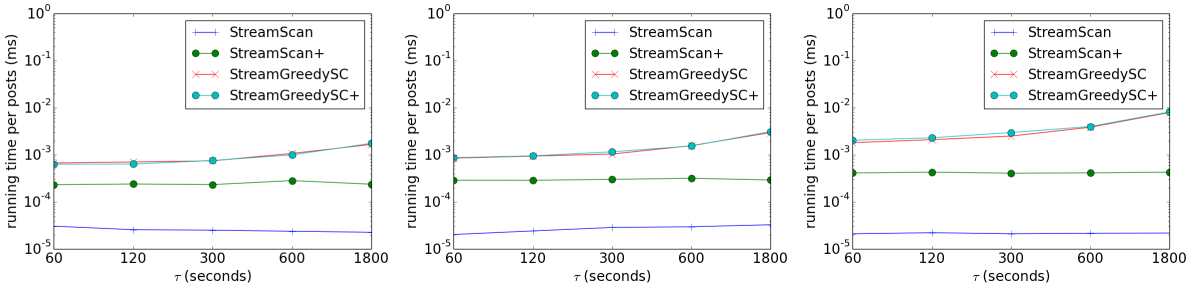
throughput of posts that our algorithms can handle. We measure the execution times of each algorithm for $|\mathsf{L}| = 2$, 5, and 20.

MQDP. Figure 5.13 shows the efficiency results for varying λ on logarithmic axis. We generally see that Scan algorithms are orders of magnitude faster than the greedy ones, since they only make a sequential pass on the data. The running time of Scan and Scan+ is quite stable for different λs, whereas the efficiency of GreedySC increases sharply when λ



(a) $|\mathsf{L}| = 2$ (b) $|\mathsf{L}| = 5$ (c) $|\mathsf{L}| = 20$

Figure 5.14: Execution time for StreamMQDP on one day of tweets, for varying λ with fixed $\tau = 300$ seconds.



(a) $|\mathsf{L}| = 2$ (b) $|\mathsf{L}| = 5$ (c) $|\mathsf{L}| = 20$

Figure 5.15: Execution time for StreamMQDP on one day of tweets, for varying τ with fixed $\lambda = 300$ seconds.

increases, because the solution size is smaller with larger λ and hence GreedySC will have smaller number of rounds to pick posts and update the covered range of left posts.

Another observation is that for larger label set size $|\mathsf{L}|$, the running time of Scan and Scan+ decreases, since the same post may cover more other posts. However, GreedySC becomes slower with larger $|\mathsf{L}|$, since we need to execute more iterations to find the post with maximum cover range and to update other posts' cover ranges.

To better understand this behavior, we provide more details on our implementation of the greedy algorithms. We could maintain a heap (PriorityQueue in Java) in order to

find the set with the maximum size at each step. Every time we select a post (a set in the set cover problem instance), we need to update other posts' cover range. For this, we need to delete the original set from the heap and insert the set back with updated weight (set size). When there are lots of relevant posts in a short time, this leads to big overhead of updating this heap. Thus we don't apply this in our implementation. Instead, we iterate all sets to find the set with maximum size, which is experimentally shown to have better performance on our data set.

StreamMQDP. Figure 5.14 evaluates the execution time of the approximation algorithms by varying λ with a fixed τ value, and Figure 5.15 shows the results by varying τ given a λ. Like Scan and Scan+ on MQDP, the efficiency of StreamScan and StreamScan+ for Stream-MQDP is stable with respect to λ and τ. Given a λ, the efficiency of StreamGreedySC and StreamGreedySC+ decreases slightly with the increases of τ, and given a τ their execution time generally decreases with larger λ, because of smaller number of iterations of execution of set cover algorithm.

5.7.4 Discussion

Our experiments show that the two classes of presented approximate algorithms, Scan and GreedySC (including their variants), offer good relative errors ranging from 0.01 to 0.5 (except the extremely small τs in the streaming setting).

GreedySC has lower error than Scan in most settings; its maximum improvement is about 60% (see difference between GreedySC and Scan+ for $\lambda = 20$ seconds in Figure 5.7) for the static problem setting, whereas in the streaming setting GreedySC's error is unstable, and is often larger than that of Scan (see Figures 5.9 and 5.10).

In addition to the instability, a potentially more serious shortcoming of GreedySC is that it is 1 to 3 orders of magnitude slower than Scan, which makes Scan more desirable for setting of high throughput and especially when the algorithm has to be executed for millions of users (as in Twitter).

Finally, we show that our proposed exact dynamic programming algorithm is feasible for small problem instances, where the number of queries is up to 2-3 and λ is less than a minute.

5.8 Related Work

Vertical and federal search. Previous works in aggregated search argue that a user's intent can be captured in a better way if the web search retrieves and integrates results from different vertical domains. Verticals can include for example microblogging sites (such as Twitter or Google+), user comments, blog feeds or breaking news. Additional information found in those verticals can be used in various ways.

Diaz [35] proposes principled ways to integrate news content into web search results. In [38] the authors rerank the web documents based on a set of social network features, such as the number of tweets that refer to each document, the number of retweets of these posts, etc. Similarly, [106] applies a vector space model technique in order to rerank web results based on their similarity with tweet posts published in a user's social circle. Our approach in this work is different since we do not aim to merge microblogging posts with web search results.

Linking microblogging posts with news articles. With a similar motivation, several

works have focused on the relation between news articles and microblogging posts such as those commenting on, referring to, or directly related through links to a news article or specific event [62, 100, 119, 6, 48, 70]. These works can be considered complementary to our approach. That is, instead of treating a news article as a query to retrieve related microblogging posts, we can apply these works in order to build the relation between articles and posts. However, efficiency could be a potential issue in such a scenario.

Diversity. Query results diversification is a well-studied problem in the field of information retrieval [7, 97] as well as in data management [95, 39]. Because of the ambiguity of queries, results diversification is very helpful to satisfy the search intent of different users. In this work, we take a different definition of diversification from these works, which is based on multiple queries. Further, similarly to [39], we put more focus on results coverage. Santos et al. [101] work on diversification on explicit sub-queries of the original query. They maximize the semantic coverage with respect to different aspects of the original query.

Giannopoulos et al. [48] address the problem of diversifying user comments found on news articles such that the selected comments cover different aspects of the article. Compared to [48], our work focuses on identifying microblogging posts that refer to multiple news articles, whereas [48] provides a solution only for one article. Further, instead of maximizing diversity, we use a coverage-based optimization goal. Finally, we mainly focus on the efficiency of the proposed methods, whereas [48] do not consider this aspect.

Publish/subscribe. Different variations of *Publish/Subscribe* systems have been proposed such as Topic-Based, Content-Based, and Type-Based [44]. However, to the best of our knowledge, there has not been considerable work on building publish/subscribe systems on

microblogging services. The authors in [33] suggest a publish/subscribe infrastructure for microblogging services to better support crowdsourced sensing and collaboration applications.

5.9 Conclusions and Future Work

In this chapter we introduced and formalized the Multi-Query Diversification Problem (MQDP). We proved MQDP is NP-hard and we proposed several algorithms for solving it: an exact dynamic programming algorithm, two efficient approximation algorithms, as well as some algorithms for the streaming variant. We confirmed the effectiveness of our approach through extensive experiments on real Twitter data set.

In the future we will study how our solutions can be adapted for more diversity dimensions. In particular, we would like to extend them to the spatiotemporal space, where the selected posts need to cover both the time and geospatial dimension. Incorporating geographical information is likely to become more important as, increasingly, more posts are geotagged.

Further, we will study how content-based intra-result diversity methods [7, 10, 95, 39] can be effectively applied to short posts [84] – for instance through context expansion – and then how this content-based diversity can be represented in our multi-query diversity framework.

Chapter 6

Multi-Dimensional Diversification on Social Post Streams

Web 2.0 users conveniently consume content through subscribing to content generators such as Twitter users or news agencies. However, given the number of subscriptions and the rate of the subscription streams, users suffer from the information overload problem. To address this issue, we propose a novel and flexible diversification paradigm to prune redundant posts from a collection of streams. A key novelty of our diversification model is that it holistically incorporates three important dimensions of social posts, namely content, time and author. We show how different applications, such as microblogging, news or bibliographic services, require different settings for these three dimensions. Further, each dimension poses unique performance challenges towards scaling the diversification model for many users and many high-throughput streams. We show that hash-based content distance measures and graph-based author distance measures are both effective and efficient for so-

cial posts. We propose scalable real-time stream processing algorithms leveraging efficient indexes that input a social post stream and output a diversified version of the stream, diversified across all three dimensions. Next, we show how these techniques can be extended to serve multiple users by appropriately reusing indexing and computation where possible. Through extensive experiments on real Twitter data, we show that our diversification model is effective and our solutions are scalable. We show that different algorithms perform best for different application settings.

6.1 Introduction

Tremendous amounts of online social data are generated every day. For instance, Twitter has reported over 280 million monthly active users in its microblogging service and 500 million Tweets posted per day[1]. One common way to consume social data is through implicit or explicit subscription. For example, almost all news agencies offer RSS feeds for people to subscribe. Google Scholar continuously recommends new publications to its users based on a user's profile and publication history. In a microblogging system like Twitter, one can subscribe to other users' posts by following them.

All posts matching a user's subscriptions are typically displayed in a convenient central place, such as the user's timeline in Twitter or Facebook. These timelines are updated in real time. A key challenge is that a user could be easily overwhelmed by the number of posts in the timeline, especially if the user is subscribed to many post producers. Further, a user's timeline often contains lots of posts that carry no new information with

[1]https://about.twitter.com/company

respect to other similar posts. This data overload issue also happens in other applications with smaller data throughput such as news and research papers. For instance, it has been shown that a primary care physician should read hundreds of medical publications per day to keep up with the medical literature [9].

To alleviate the data overload problem, in this work we propose a novel way to efficiently and effectively diversify social post streams by pruning redundant posts. By social post streams we mean a broad class of content generated by services where each post, in addition to its textual content, has a unique author and a unique timestamp, and where authors are associated through various social relationships. For instance, in Google Scholar authors are connected by relations such as co-authorship or overlapping research interests. In microblogging sites users are connected by follower/followee relations.

Given a stream consisting of all the posts from a user's subscriptions, our goal is to output in real-time a subset of the stream in which *(i)* all posts are dissimilar to each other and *(ii)* any post in the whole stream will be either included or *covered* by a post in the sub-stream. A post covers another post if the two posts are similar in all three similarity dimensions: (a) content, (b) time and (c) author.

Two posts have similar *content* if their text components are similar. Intuitively, all other dimensions being equal, users want to avoid seeing two posts with very similar content. Similarly, the *timestamp* distance of two posts is important in social post diversification. Two posts that have similar content but are far away in terms of post time, may both be of interest to the user. Note that time is widely used for diversifying search results in microblogging systems [68, 91, 22].

The *author* similarity is a more subtle dimension that to the best of our knowledge has not been used before for computing diversity in social media. For example, CNN and Fox News, which both have official Twitter accounts, are dissimilar to each other because they generally have different political views. We compute the distance between two authors through their social connections. In particular, we compare the sets of friends (or followers in the case of Twitter) of the two authors, which has been shown to be a good author similarity measure in social networks [113, 50].

Challenges: To summarize, in our model two posts are redundant with respect to each other if they are similar in all of the three dimensions. It is challenging to apply the proposed diversification model in a large scale social service with high posts throughput. First, we must efficiently compare the content of a new post to the content of all previous posts (within a time window). For this, we apply Hash-based techniques to measure the content similarity between social posts. Hash-based techniques have been applied before to Web documents [82], but not to social posts, which are generally shorter and may heavily rely on abbreviations or URLs.

Second, handling the author dimension is challenging. A naive approach is to check if the author of each new post is similar to the author of each existing post (within a time window). However, we show that depending on the setting (similarity thresholds across the three dimensions), a different indexing data structure is more efficient to achieve real-time posts processing.

Third, the three diversity dimensions offer an opportunity to use the results of the one dimension to prune the work needed for the other dimension. For instance, if I know

that posts P_1 and P_2 have high content similarity, then I don't need to check if their authors or time are similar.

Fourth, if we move from one user to many users, where each user has a collection of subscriptions, the challenge is how to reuse the computation performed for diversifying one user's stream to diversify streams of other users. We show that we can reuse computation across users only if their shared subscriptions meet a strict condition.

Previous work on diversity: There has been much work on diversifying results for documents [97, 7, 19], social posts [68, 91, 22] and database records [32, 39]. However, none of these works can be applied to our setting where: (i) data is streaming and an instant decision must be made on whether a post should be pushed to the user, and (ii) a multi-dimensional diversity model is adopted. In contrast, most previous works focus on the search setting, where a user submits a query and the set of results must be diversified based on content, including work on social posts [68, 91].

The problem studied in this work is also fundamentally different from previous work on stream summarization [112, 104, 98, 125], because: *(i)* we do not aim to generate an aggregation of documents, but instead select a subset of posts, and *(ii)* we define strict coverage constraints to guarantee that not even one uncovered posts is missed.

Contributions: In this work, we make following contributions:

- We propose a new paradigm to define diversity on social posts, by incorporating three important dimensions – content, time and author – and we define corresponding optimization problems (Section 6.2).

- We study how content similarity can be efficiently applied to social posts, which are

generally short and contain abbreviations (Section 6.3).

- We propose efficient data structures and algorithms to solve the social posts stream diversification problem (Section 6.4).

- We show how the single-user algorithm can be extended to handle many users, by reusing computation across users (Section 6.5).

- We perform a comprehensive experimental evaluation, where we focus on microblogging data, which poses the most serious scalability challenges. We show how different algorithms perform better for different diversity needs (Section 6.6).

Section 6.7 reviews related work. We conclude in Section 6.8.

6.2 Framework and Problem Definition

Let P represent a stream (ordered set) of social posts. Each post P_i in P has an author $author(P_i)$, textual content $text(P_i)$ and a timestamp $time(P_i)$ (also referred as t_i). We define the distance measures across the three diversity dimensions as follows.

- **Content Distance.** We represent the content distance between two posts P_i and P_j as $dist_c(P_i, P_j)$. Cosine similarity is a possible way to define the distance, but for efficiency purposes we employ the hash-based simhash measure as explained in Section 6.3, where we show that simhash is effective for social posts.

- **Time Distance.** The time distance between two posts P_i and P_j is denoted as $dist_t(P_i, P_j) = |t_i - t_j|$.

- **Author Distance.** We denote the author distance between P_i and P_j as $dist_a(P_i, P_j)$. For social data, we define the similarity between two authors as the cosine similarity between their friends' vectors, which has been successfully used in previous work to measure the user similarity in Twitter [113, 50]. The author distance is $(1 - similarity)$. For other domains other distance measures may be more appropriate.

Next, we define the coverage semantics between posts.

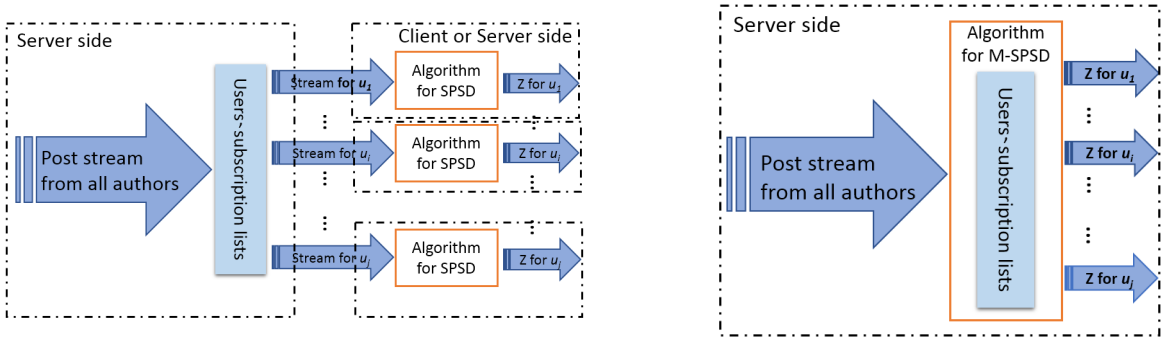
Definition 3. *(Post Coverage) Given a content diversity threshold λ_c, a time diversity threshold λ_t and an author diversity threshold λ_a, two social posts P_i and P_j cover each other if:*

- $dist_c(P_i, P_j) \leq \lambda_c$ *and*
- $dist_t(P_i, P_j) \leq \lambda_t$ *and*
- $dist_a(P_i, P_j) \leq \lambda_a$.

Note that the coverage semantics between two posts is symmetric. The three thresholds may vary according to the characteristics of a social system as we discuss below. The primary focus of this work is to study the efficient processing of a posts stream and not to set these threshold values.

We next define the Social Post Stream Diversification (SPSD) problem.

Problem 3. *Social Post Stream Diversification (SPSD) Given a social post stream P, and diversity thresholds λ_c, λ_t and λ_a, compute a sub-stream of posts $Z \subseteq \mathsf{P}$ that covers P, that is, $\forall P_i \in \mathsf{P}\ \exists P_j \in Z$, such that P_j covers P_i.*



(a) SPSD

(b) M-SPSD

Figure 6.1: Settings of SPDP and M-SPDP.

Note we have to compute Z in *real-time*, i.e., immediately decide whether a post P_i should be included in Z at its arrival. That is, we cannot first view the whole stream and then decide which posts should be included in the substream.

In SPSD, there is a single user who consumes the stream and many authors who generate the posts of the stream (a user may also be an author and vice versa). That is, a solution to SPSD should be deployed for each user, for example, as part of the Twitter app of a user. On the other hand, a social network service would rather have a central diversification engine that diversifies the posts for each of its users, so that no client side post processing is required. We refer to this version of SPSD as Multiple-Users SPSD (M-SPSD). Another difference between SPSD and M-SPSD is that in SPSD we can easily support user customized diversity thresholds. Figure 6.1 shows how SPSD and M-SPSD differ in terms of the setting and deployment.

Problem 4. *Multiple-Users Social Post Stream Diversification (M-SPSD) Given a social post stream P, diversity thresholds λ_c, λ_t and λ_a, and a set of users where each user is subscribed to a subset of the authors, compute a diversified sub-stream for each user.*

6.3 Content Distance Estimation for Microblogging Posts

Among the three diversity dimensions, the content distance is the most expensive to compute, because it must be computed for each new post. This is especially true given our real-time decision semantics described above. In contrast, the author similarity between each pair of authors may be precomputed (e.g., once every week), as it changes slowly over time. For that reason, we cannot afford to use traditional content similarity measures such as cosine similarity. Instead, we turn to hash-based distance measures. In this section we present the details of the employed content distance technique along with an analysis of its effectiveness for microblogging data.

We define the content distance between two posts P_i and P_j as the Hamming distance of their SimHash [99] fingerprints. Previous work has applied SimHash on web documents [82] and showed that it is efficient and effective. We represent the SimHash of $text(P_i)$ as S_i, which is a 64-bit fingerprint. The Hamming distance of two SimHash fingerprints is the number of different bits between them. According to the experimental analysis in [109], the cosine distance between two texts positively correlates to the Hamming distance of their corresponding SimHash fingerprints.

Distribution of SimHash distances in Twitter

First, we study the distribution of SimHash distances on Twitter data. We collected a dataset of 200 thousand tweets from the Twitter Streaming API, which returns a stream of randomly selected substream of Twitter ([87] showed that the stream is not exactly random but this is not too important for our problem). The distribution of the Hamming distances for these tweets is depicted in Figure 6.2, which shows a perfect normal

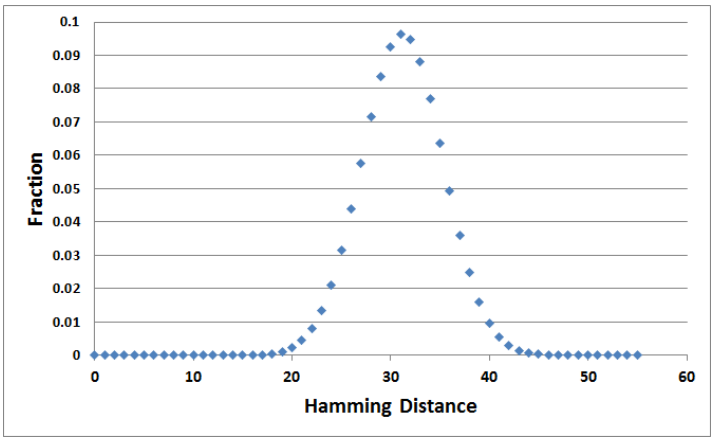


Figure 6.2: Hamming distance distribution

distribution with mean value 32, as expected, and with most of the distances between 24 to 40.

User Study

To further evaluate the effectiveness of SimHash for social posts, we conducted a user study to learn the relationship between the SimHash distance between two posts and the perceived dissimilarity between the posts. A second goal of the study is to learn what is a good SimHash distance threshold (e.g., a threshold of 3 bits was chosen to define redundant Web pages [82]) and if any preprocessing of the tweet text (e.g., expand shortened URLs) may improve the effectiveness of SimHash.

Setup and Methods: In particular, we collected a dataset of 2000 pairs of tweets randomly selected from the 200,000 tweets returned by the Twitter Streaming API, with SimHash distances between 3 and 22 – 100 tweets from each distance value. We chose 3 to 22 because this is the range where we expect to find posts that are very similar (redundant with respect to each other). This range choice is supported by our results below. We

recruited 12 undergraduate and graduate students.

We evenly divided these 2000 pairs into 4 groups and distributed them to the 12 students for labeling. The author and timestamp of the posts are hidden. Some examples of these pairs are shown in Table 6.1. Each group of tweets is labeled by 3 students. The students were asked to mark whether the two tweets in a pair are redundant with respect to each other.

Table 6.1: Example tweet pairs and their Hamming distances

Tweet pair	*Hamming distance*
Over 300 people missing after South Korean ferry sinks. (Reuters) Story: http://t.co/9w2JrurhKm Over 300 people missing after South Korean ferry sinks. (Reuters) Story: http://t.co/E1vKp9JJfe	3
"In order to succeed, your desire for success should be greater than your fear of failure" Bill Cosby In order to succeed, your desire for success should be greater than your fear of failure. #quote #success - Bill Cosby	8
Alibaba's growth accelerates, U.S. IPO filing expected next week http://t.co/mUcmLJ4cpc #Technology #Reuters Alibaba's growth accelerates, U.S. IPO filing expected next week: SAN FRANCISCO (Reuters) - Alibaba Group Hold... http://t.co/aLAV8w4gWF	13

To help the users more accurately label the similarity between two posts, we showed the expanded URL (instead of the shortened one shown in Table 6.1). We used a majority vote, that is, if at least 2 out of the 3 students labelled a pair as redundant, we labelled the pair as near-duplicates.

Results: Out of the 2000 pairs, the users marked 949 pairs as redundant. Figure 6.3 shows the precision and recall achieved by various SimHash distance values. For each

Hamming distance h, the precision is defined as the fraction of pairs with Hamming distance no more than h that are true near-duplicates. Recall is the fraction of the total number of near-duplicate pairs that are detected with Hamming distance at most h. This graph shows that SimHash distance is an effective measure to identify similar posts.

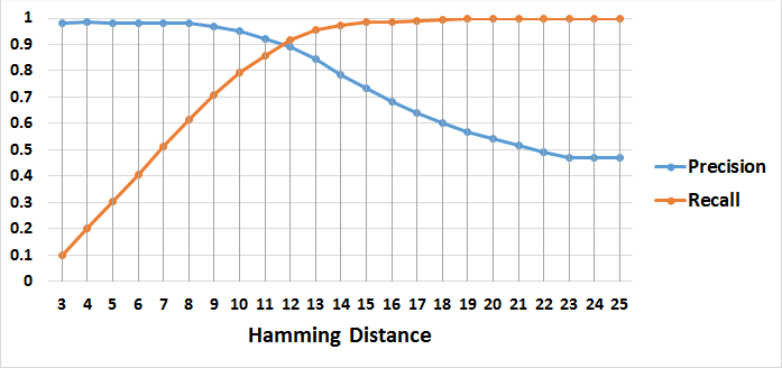


Figure 6.3: Precision and Recall for Hamming distance. SimHash fingerprints are generated from raw texts of tweets

Next, we study if various text preprocessing methods may improve the precision or recall of SimHash distance measure for microblogs. We first normalize the text by (a) changing all text to lowercase, (b) removing extra white spaces between words, and (c) removing non-alphanumeric characters (such as $*, -, +, /$, etc.). Figure 6.4 plots the precision and recall after we apply the normalization. We see that this graph achieves higher precision and recall values than the original analysis in Figure 6.3. We also see that the the two lines cross for $distance = 18$, which achieves precision $= 0.96$ and recall $= 0.95$.Hence, we use $\lambda_c = 18$ as the default content distance threshold in the experiments in Section 6.6.

We also tried other methods of text preprocessing such as expanding shortened URLs (URLs in tweets are shortened by Twitter), varying the weights of user mentions

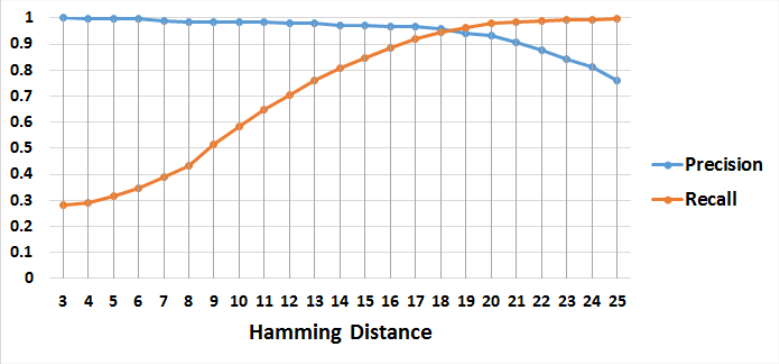


Figure 6.4: Precision and Recall for Hamming distance. SimHash fingerprints are generated from normalized texts of tweets

and hashtags (by creating artificial copies), and expanding abbreviations. However, these methods had no significant impact to the precision and recall.

For completeness, we compared the effectiveness of SimHash to that of cosine similarity (which is much slower as discussed above) in terms of detecting posts with near-duplicate content (redundant). We tried different cosine threshold values and found that the precision and recall lines across at cosine similarity 0.7, where all posts with cosine similarity above 0.7 are marked as redundant. This achieves precision and recall of 0.96 and 0.95 respectively, which is the same as what we achieved using SimHash above. This means that, for detecting near-duplicate in our dataset, SimHash achieves effectiveness similar to cosine similarity. Hence, given the time performance advantage of SimHash, it is the best choice for our problem.

The high threshold value of $\lambda_c = 18$ for SimHash precludes the use of the efficient SimHash index proposed in [82] which relies on building several copies of the SimHash values table for several permutations of the bits, since the number of these copies is exponential

in λ_c (which was only 3 in [82]). Hence, as we discuss in Section 6.4, other indexing and searching techniques are required.

6.4 Algorithms for SPSD

In this section, we describe our algorithmic solutions for the SPSD problem. As explained earlier in Section 6.3, due to the high Hamming distance threshold we are unable to use existing SimHash indexing techniques, and we must rely on comparing the SimHash value of each new post with those of all the previous ones, leading inevitably to linear time complexity per post in the worst case. We reduce the number of these comparisons by leveraging the other two dimensions, time and author. We first discuss how we handle time diversity, which is simpler, and then we present various approaches for handling author diversity.

Handling Time Diversity. According to the diversity model, at the arrival of a post P_i it can only be covered by the previous posts within a λ_t time distance. Thus, it is sufficient to store only the posts from previous λ_t time in memory for checking the coverage of a new post. One possible implementation is that we could store the posts in a circular array. We track two post indices for the oldest post within a λ_t distance to current time (a) and the most recent post (b). At the arrival of each post P_i, we compare it to the posts from most recent post to the oldest (i.e., from index b to a). If we encouter a post P_j with $t_i - t_j > \lambda_t$, we update a to be index of the post right after P_j. And we insert a non-redundant post to the array with index $(b+1)$ and update $b = b + 1$.

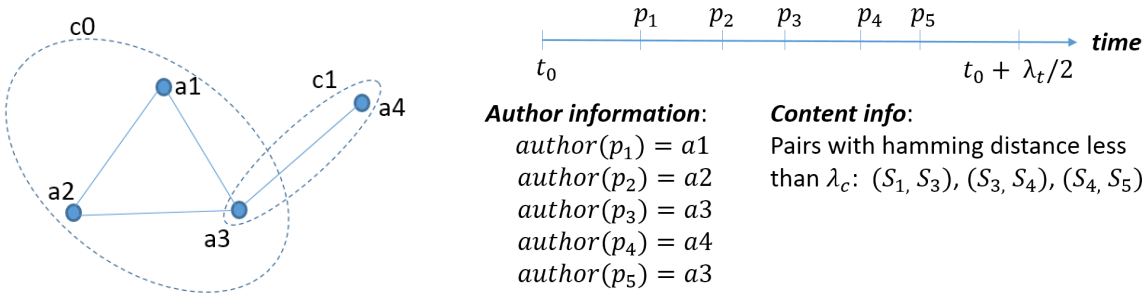
Now that we have discussed how to handle time diversity, we focus on the author

diversity among the posts in the last λ_t time units. The author similarity relations between all authors form an *author similarity graph* G, which as we discussed above may be periodically precomputed. There is an edge between two authors in G if their distance is below the threshold λ_a. For each user u_i who subscribes to a set A of authors, we define G_i as the subgraph of G that contains all the A authors and the edges among them. In this section, we assume there is only one user (hence, one G_i) and in Section 6.5 we assume multiple users (and G_i's).

6.4.1 UniBin

Our first method to solve SPSD, which we refer as *UniBin*, works as follows: At the arrival of each post P_i in P, we sequentially (from the most recent post to the older ones) compare P_i to each post in the past λ_t time range in the diversified sub-stream Z. For each post P_j, we check whether P_j meets both: (1) Hamming distance between S_i and S_j (SimHash fingerprints of P_i and P_j, respectively) $\leq \lambda_c$, and (2) $dist_a(P_i, P_j) \leq \lambda_a$, which can be achieved by checking whether $author(P_i)$ and $author(P_j)$ are the same or neighbors in G. If no post from the past λ_t time range meets the above two conditions (i.e., P_i is not covered by Z), then we add P_i to Z. Otherwise we do not include P_i in Z.

We denote this method as **UniBin** indicating that the posts from all authors are stored in a single *post bin* (e.g., a circular array as described earlier). We illustrate UniBin with an example. In Figure 6.5a, each node represents an author. Two authors are connected by an edge if they are similar to each other (i.e., the author distance $\leq \lambda_a$). Figure 6.5b shows the posts from these authors with post distance information in terms of



(a) Author similarity graph G_i

(b) Social Posts

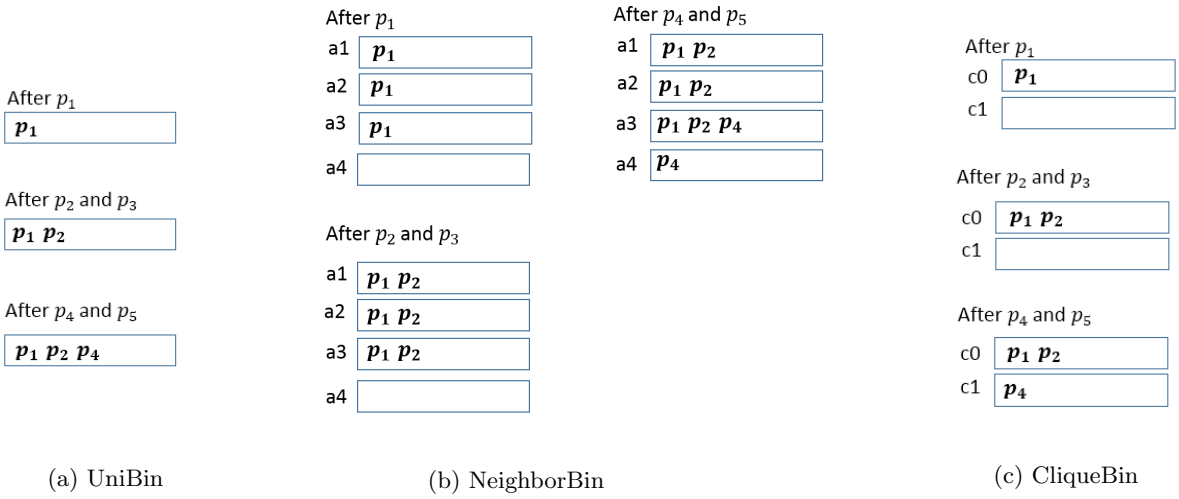
Figure 6.5: Example of author similarity graph and posts

all three diversity dimensions.

We show the update of a post bin for UniBin in Figure 6.6a. When P_1 arrives, there is no posts in the bin yet. Thus P_1 is not covered hence is added to the bin. P_2 is also added as it is not covered by P_1 (the Hamming distance between S_1 and S_2, $dist_c(P_1, P_2)$, is larger than the threshold λ_c). For P_3, the algorithm first compares it to P_2 which does not cover P_3 (because $dist_c(P_2, P_3) > \lambda_c$). However, it is covered by P_1 because in all three diversity dimensions they are within the distance thresholds (or above similarity threshold). Thus, P_3 is not added. So forth, P_4 is not covered by either P_1 and P_2 and is included in the bin. However, we note that P_4 and P_3 cover each other. Finally, P_5 is covered by P_4.

6.4.2 NeighborBin

UniBin has to compare a new post (both its author and content SimHash) to all posts in the last λ_t time units. This aggregated time may be considerable given the high frequency of posts, even if the author similarity graph G_i and the post bin are maintained in memory.



(a) UniBin (b) NeighborBin (c) CliqueBin

Figure 6.6: Running example for the three algorithms for SPSD.

To improve this, we partition the posts by their authors such that for a new post P_i we only check its coverage by comparing with the posts from $author(P_i)$ or from $author(P_i)$'s similar authors. Specifically, we create a post bin for each author.and when a new post P_i comes, the algorithm sequentially checks posts in the bin identified by $author(P_i)$ but not other posts. However, we must note that posts from the authors that are neighbors of $author(P_i)$ in G_i can potentially cover P_i. Hence, the post bin of an author also includes the posts of similar authors (neighbors in G_i). Thus, we add P_i to all bins of $author(P_i)$'s neighbors in addition to the bin of $author(P_i)$, if P_i is detected as a non-redundant post. We denote this method as **NeighborBin**.

Figure 6.6b depicts the execution of NeighborBin for the data shown in Figure 6.5. P_1 is added not only to the bin of its author a1, but also to the bins of a2 and a3, because they are neighbors of a1, as shown in Figure 6.5a. To check the coverage of P_2, only the post bin of a2 is accessed where P_1 does not cover P_2. After that, P_2 is also added to the

post bins of a1, a2 and a3. NeighborBin checks the coverage of P_3 by iterating posts in the bin of a3 where P_1 covers P_3. When P_4 comes, a4's post bin is blank and thus P_4 is added to the post bins of a3 and a4 without incurring any post comparisons. Finally, P_5 is detected as redundant by checking the bin of $a3$ ($author(P_5) = \text{a3}$) where P_4 covers P_5.

6.4.3 CliqueBin

In NeighborBin, we index the posts by author aiming to reduce the pairwise post comparisons. But the tradeoff is memory consumption: we have multiple copies of a post in different authors' post bins.

To reduce the overhead on memory consumption incurred by NeighborBin, we identify groups (cliques) of authors that are similar to each other and assign a single bin to them, such that a post generated by any of these authors is only stored in that bin. Specifically we find a *clique edge cover* of G_i, that is a collection of cliques whose union contains all edges of G_i. We maintain a post bin per clique (e.g., a map from clique ID to a list of posts). Only the posts from authors in a same clique as $author(P_i)$ can possibly cover post P_i. Thus, at the arrival of post P_i, we check whether it is covered by sequentially comparing it to the posts from only the cliques that contain $author(P_i)$. Thus a post P_i in Z is stored once for every clique that contains $author(P_i)$ – instead of once for each neighbor of $author(P_i)$ in NeighborBin. Note that this approach guarantees that the coverage requirement for posts is satisfied: when a new post P_i authored by a_j appears, and P_i is not similar to earlier posts of a_j or its neighbors then P_i will be added to the cliques involving a_j, because a_j's edges are covered by the cliques.

Considering the space consumption, our objective should be to minimize the sum

of the sizes of cliques, i.e., the average number of cliques per author is minimized and thus number of copies per post is reduced. This is an NP-hard problem, and hence we have decided to use a simple greedy heuristic. It starts by picking an edge in G_i to form an initial clique. Then it extends the clique by adding nodes that are neighbors to all the nodes in the clique. When there is no such node, the clique is saved and the algorithm picks another edge not yet included in any found cliques and repeats the above process. We stop when all edges are covered.

Upon a new post P_i, we use a hashmap (Author2Cliques) to get all the cliques that contains $author(P_i)$, and then we check the posts in the corresponding bins. Recall that NeighborBin and UniBin load the author similarity graph G_i in memory. We can make the same assumption that Author2Cliques is loaded in memory for applying CliqueBin. Similar to the computation of author similarity graph, we assume the clique partition of G_i and the Author2Cliques mapping are computed offline. We denote this algorithm as **CliqueBin**.

The update of a post bin by CliqueBin is depicted in Figure 6.6c. Cliques C0 and C1 together cover all the edges in the graph. We can see that P_1 is only stored once in C0's bin (because a1 is in C0) instead of saving 3 copies in NeighborBin as Figure 6.6b. The same applies to P_2. Since a3 is in both C0 and C1, during the processing of P_3 CliqueBin may check both bins of C0 and C1. P_4 will only be compared with the bin of C1 because a4 belongs to only C1. Again, CliqueBin checks the coverage of P_5 by iterating both bins of C0 and C1. This example illustrates how CliqueBin can reduce space requirements compared to NeighborBin.

We note that in some cases CliqueBin may have to do a larger number of pairwise

post comparisons than NeighborBin. Suppose that after P_5 in the above example author a3 posts P_6 and then author a4 posts P_7. If P_6 and P_7 are not redundant to any other posts, then P_6 should be added to all four post bins in NeighborBin, and to both post bins in CliqueBin. For P_7, NeighborBin only accesses the bin of a4 and thus only needs to do two comparisons (with P_4 and P_6). In contrast, CliqueBin has to do 5 comparisons: with P_1, P_2, P_4 and twice with P_6 (once in post bin of each clique). We study this experimentally in Section 6.6.

6.4.4 Performance Analysis

In this section we show an estimate of the time and space complexity of our algorithms, attempting to capture their performance on realistic data, rather than the worst-case performance. Rigorous derivation of such estimates is challenging, because the behavior of these algorithms heavily depends on the specifics of the data sets, including the topology of the social network. Instead, we provide informal derivations based on several reasonable assumptions about the data set and the graph's topology.

Suppose there are m subscribed authors, and the total number of posts from these m authors in a λ_t time range is n. We assume a ratio of r (≤ 1) posts left after diversification, that is, $r \cdot n$ non-redundant posts per λ_t time. We also assume that the each author generates the same number of $\frac{n}{m}$ posts with $\frac{r \cdot n}{m}$ left after diversification. Further we assume in the author similarity graph, each author has d neighbors and is in c ($\leq d$) cliques. We denote s as the average number of authors in a clique.

Note that cliques may have overlaps. If we define q as the number of edges in G over the total number of edges in c cliques from G, we have $\frac{m \cdot c}{s} = \frac{m \cdot d}{s \cdot (s-1) \cdot q}$, where both

sides compute the number of distinct cliques. Thus we can expect $c \cdot (s-1) \cdot q = d$ with $0 \leq q \leq 1$.

Recall that UniBin puts posts from all authors in Z into a single post bin. Thus, the total bin size is $r \cdot n$ in UniBin. Each new post is sequentially compared to each post in the bin and thus the number of post comparisons per new post is $r \cdot n$. Each non-redundant post incurs one insertion into the bin.

NeighborBin maintains a set of per-author bins with each bin storing posts from an author and her similar authors. Roughly, each per-author bin stores $\frac{d+1}{m} \cdot r \cdot n$ posts. Thus the total number of post copies stored in memory is $(d+1) \cdot r \cdot n$. At the arrival of a new post P_i, the number of post comparisons made by NeighborBin is $\frac{d+1}{m} \cdot r \cdot n$ (compare P_i to all posts in $author(P_i)$'s post bin). Each non-redundant post incurs a total of $(d+1)$ insertions into the bins.

In CliqueBin, for each non-redundant post P_i we store its c copies: one copy in the bin of each clique containing $author(P_i)$. Thus, the total size of the clique bins is $c \cdot r \cdot n$. CliqueBin compares each new post P_i to posts in the bins of c cliques that contain $author(P_i)$, which leads to a total of $\frac{s \cdot c}{m} \cdot r \cdot n$ comparisons. Each non-redundant post incurs a total of c insertions into the bins.

Table 6.2 summarizes the performance analysis. We can see that all these results contain the same component $r \cdot n$. Obviously, all three diversity thresholds effects the ratio of non-redundant post r. The value of n is affected by several factors, such as the frequency of the post stream P and the setting of time diversity threshold λ_t.

An important factor that affects the performance of the algorithms, especially

Table 6.2: Performance estimation of the algorithms for SPSD

	UniBin	NeighborBin	CliqueBin
RAM	$r \cdot n$	$(d+1) \cdot r \cdot n$	$c \cdot r \cdot n$
Comparisons in λ_t	$r \cdot n^2$	$\frac{d+1}{m} \cdot r \cdot n^2$	$\frac{s \cdot c}{m} \cdot r \cdot n^2$
Insertions in λ_t	$r \cdot n$	$(d+1) \cdot r \cdot n$	$c \cdot r \cdot n$

NeighborBin and CliqueBin, is the topology of the author similarity graph G. In the above estimates, we use parameters d, c, s and m to capture the topology properties. We note that the values of the ratios of d, c, s to m are functions of the author diversity threshold λ_a. Given a set of subscribed authors (i.e., with m fixed), the larger λ_a the denser G is (in terms of the number of edges). Thus, the number of neighbors per author (d) increases with λ_a, which means the performance of NeighborBin will drop if all other settings remain unchanged. We also argue that c and $c \cdot s$ increase with the graph's density, and hence we expect CliqueBin to perform better for smaller λ_as. We confirm this through experiments on real data set in Section 6.6, where we will summarize the use cases for each algorithm based on this theoretical analysis combined with our experimental results.

6.4.5 Summary

We summarize the characteristics of the three algorithms in Table 6.3. In terms of data structure, UniBin and NeighborBin need the author similarity graph, while CliqueBin needs the mapping of each author to the set of cliques containing the author. As we mentioned, we assume that all these data structures are maintained in memory.

Table 6.3: Differences between the three algorithms for SPSD

		UniBin	NeighborBin	CliqueBin
Data Structures		(1) Author similarity graph (2) A single post bin storing posts from all authors.	(1) Author similarity graph (2) A post bin per author storing posts from the author and her neighbors.	(1) Author clique mapping (2) A post bin per clique storing posts from all the authors in the clique.
Properties	RAM	Low	High	Moderate
	Comparisons	High	Low	Moderate
	Insertions	Low	High	Moderate

We can see that UniBin requires the least RAM. NeighborBin reduces the post comparisons compared to UniBin, but has high RAM consumption because it maintains multiple copies of a post. CliqueBin outperforms NieghborBin in terms of RAM consumption, by reducing the number of copies per post (and thus insertions per post), but it incurs more post comparisons. Since CliqueBin still maintains multiple copies of a post, it requires more insertions and higher RAM consumption than UniBin. Also, since CliqueBin does not compare posts from non-similar authors, we expect the number of comparisons in CliqueBin to be lower than in UniBin.

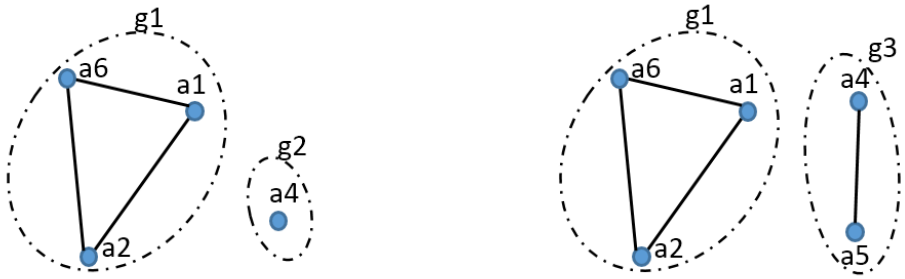
6.5 Algorithms for Multiple-Users SPSD (M-SPSD)

In this section, we extend our ideas to solving M-SPSD. When we move from applying the diversity model for one user to multiple users, the crucial question is whether

it is possible to reuse the computation performed for diversifying one user's stream to diversify the other users' streams.

A simple way to solve M-SPSD is to process the post stream for each user individually. That is, we can apply the algorithm for SPSD on each user's post stream separately. We denote the corresponding algorithms for M-SPSD as **M_UniBin**, **M_NeighborBin** and **M_CliqueBin** respectively, to distinguish them from the algorithms for SPSD. In this section, we present variations of these algorithms to optimize the diversification process by reusing computations for multiple users who share subscriptions.

If two users do not share any common subscriptions, then their post streams are disjoint and thus the computation of diversifying one's stream cannot be reused for diversifying the other users' post streams. Hence we only consider the cases for optimization when users share the same subset of subscriptions.



(a) G_1 (b) G_2

Figure 6.7: Author similarity graphs of two users u_1 and u_2.

However, we notice several limitations to reusing the diversification computation across multiple users, even if they share some subscriptions. We use examples to illustrate

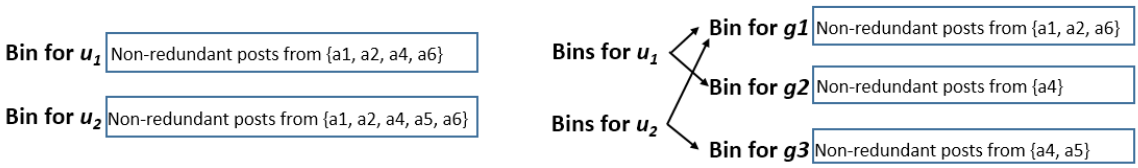
this. Figure 6.7 shows two users, u_1 and u_2, sharing a set of subscriptions {a1, a2, a4, a6}.

We notice that after diversification u_1 may see a different subset of the posts from a4 as u_2. u_2 subscribes to a5 which is a similar author to a4. Thus, it is possible that some posts from a4 are shown to u_1 but not to u_2 if they are covered by a5's posts.

However, the same diversified set of posts from {a1, a2, a6} will be shown to u_1 and u_2. The three authors form a *connected component* (denoted as g1 in Figure 6.7) in both G_1 and G_2. That is, in both G_1 and G_2 there are no other authors similar to any author in {a1, a2, a6}. Hence, posts from other subscribed authors can not cover the posts from {a1, a2, a6}. Thus, the diversification processes on the posts from {a1, a2, a6} are exactly the same for u_1 and u_2. This means that we can reuse the data structures and computation across u_1 and u_2 for diversifying the post stream from {a1, a2, a6}.

Based on these observations, we can optimize the diversification process for multiple users if they subscribe to a same set of authors that form a connected component. We can then consider a post stream (a subset of P) of each connected component separately, apply the diversification algorithm on it, and then merge the diversified post streams together.

For this, we first process the author similarity graph G_i of each user u_i to compute all connected components of all G_is. (Since different G_is may overlap, some nodes may appear in several components.) For each distinct connected component g_i, we run one of the proposed algorithms for SPSD on the post stream by the authors in g_i. User u_i's post stream consists of the union of the diversified post streams from all connected components in G_i.



(a) M_UniBin (b) S_UniBin

Figure 6.8: Example of M_UniBin and S_UniBin.

For example, as shown in Figure 6.8b, we can apply the UniBin algorithm for three distinct connected components (g1, g2 and g3), that is, we maintain a single post bin for each of the three components. Then the posts shown to u_1 is the union of the two diversified post streams from g1 and g2. We refer this algorithm as *S_UniBin*. For comparison, we show the example for M_UniBin in Figure 6.8a. M_UniBin maintains a post bin for each user seperately. To extend NeighborBin, we maintain a per-author post bin for each author in a distinct connected component g_i. To extend CliqueBin, we do the clique partition for each g_i, then maintain a per-clique post bin as described earlier.

The three algorithms with the above optimization are denoted as **S_UniBin**, **S_NeighborBin** and **S_CliqueBin** respectively.

6.6 Experimental Evaluation

6.6.1 Data Set and Experimental Settings

We conducted our experiments on Twitter data. The authors in [121] published a Twitter social graph dataset consisting of more than 660,000 Twitter authors (accounts). Computing the author similarity graph for the whole data set would be prohibitive, as it

requires comparing all pairs of authors. Instead, we used a subgraph of 20,150 authors obtained by randomly picking an initial author, and adding authors that are reachable through Breadth First Search on the follower-followee graph.

We computed all pairwise author similarity for these 20,150 Twitter authors. The author similarity distribution is depicted in Figure 6.9, where the x-axis shows the author similarity value and y-axis shows the fraction of author pairs with similarity values larger than the value indicated by x-axis. It shows that 2.3% author pairs are with similarity ≥ 0.2 and 0.6% pairs are with similarity ≥ 0.3.

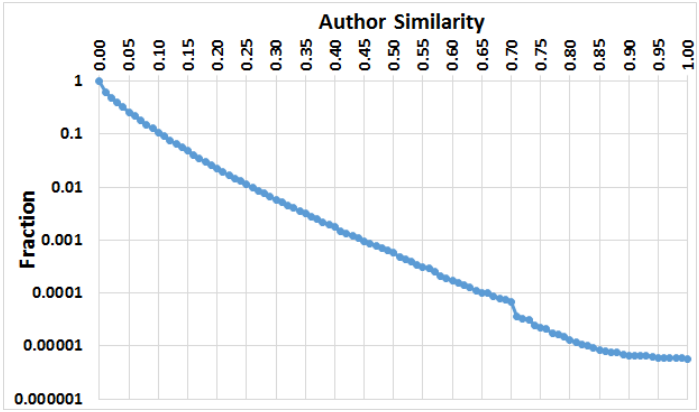


Figure 6.9: Author similarity distribution in our data set

Further, we crawled the tweets of these twitter authors using Twitter REST API[2] for one day. The tweets data set contains 233,311 tweets, which means these Twitter authors post slightly over 10 tweets per author per day. After we removed some short tweets that have less than two words or only contain meaningless tokens, there are 213,175 tweets left.

We implemented all algorithms in Java. We ran our experiments on machines with Quad Core Intel(R) Xeon(R) E3-1230 v2@3.30GHz CPU and 16GB RAM.

[2]https://dev.twitter.com/overview/documentation

6.6.2 Performance of the algorithms for SPSD

In this section, we evaluate the performance of the three algorithms for SPSD. We assume that a user follows all the Twitter authors in our dataset, and we run the algorithms on the user's post stream which consists of 213,175 posts in one day.

First, we study the effect of the three diversity dimensions: time, content and author. Figure 6.10 shows the number of tweets left after diversification under different settings by removing diversity dimensions and varying diversity thresholds. Incorporating all three diversity dimensions with reasonable diversity thresholds, the diversification model prunes about 10% redundant posts. We notice that incorporating only some of these dimensions will largely change the size of diversified stream. It means that all three dimensions play an important role in diversifying tweet data.

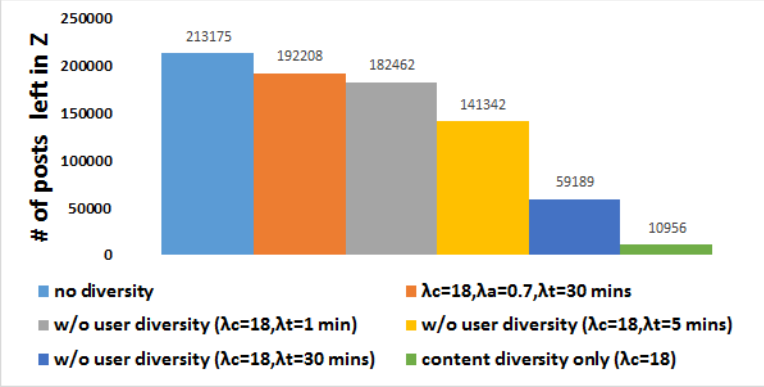


Figure 6.10: Number of tweets left after applying diversification in our data set

Performance of the algorithms under different diversity settings

The analysis in Section 6.4.4 indicates that the performance of the three algorithms for SPSD is effected by several factors such as the diversity thresholds and the post stream

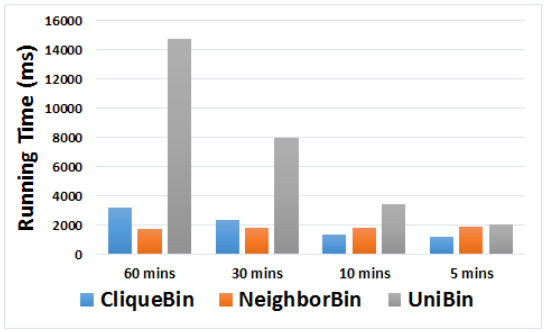
throughput. These diversity settings could change the relative performance between the three algorithms. In this section, we study the performance of each algorithm under different settings and we experimentally show that each algorithm outperforms the other two in certain settings. Supported by former analysis and experimental results, we will summarize use cases for each algorithm.

Varying time diversity threshold λ_t. In Figure 6.11, we present the performance of UniBin, NeighborBin and CliqueBin under different time diversity thresholds (λ_t). In this experiment, we set $\lambda_c = 18$ (according to the results in Figure 6.4) and $\lambda_a = 0.7$ (i.e., we consider two authors are similar if the cosine similarity between their followee vector is ≥ 0.3 and thus distance is ≤ 0.7). The running time shows the execution time for an algorithm to ingest the 213,175 posts.

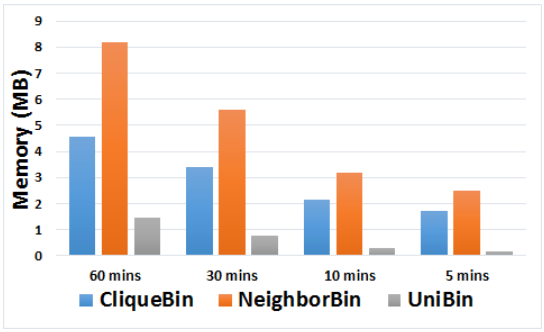
In Figure 6.11a we can see that the running time of all three algorithms decreases with smaller λ_ts. The reason is that with a smaller λ_t, the algorithms perform fewer pairwise post comparisons (depicted in Figure 6.11c). NeighborBin and CliqueBin outperforms UniBin in terms of running time. We also notice that CliqueBin is more efficient than NeighborBin when λ_t is small (e.g., ≤ 10 minutes). This gives us evidence for the summarization of use cases in Table 6.4 for NeighborBin and CliqueBin.

Smaller λ_t also reduces the RAM consumption because the algorithms store shorter history of Z in post bins. As expected, NeighborBin requires more memory than UniBin and CliqueBin.

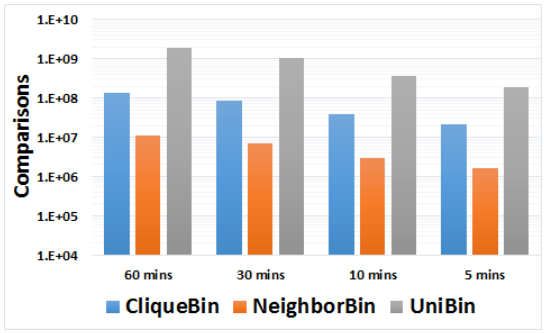
Varying content diversity threshold λ_c. We also study the performance of the three algorithms by varying λ_c. For this, we set $\lambda_t = 30$ *mins* and $\lambda_a = 0.7$ and we vary the λ_c



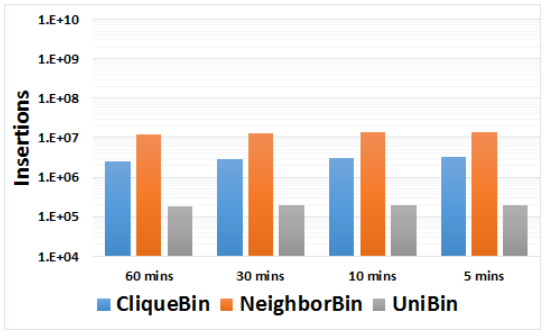
(a) Running time



(b) RAM



(c) Post comparisons

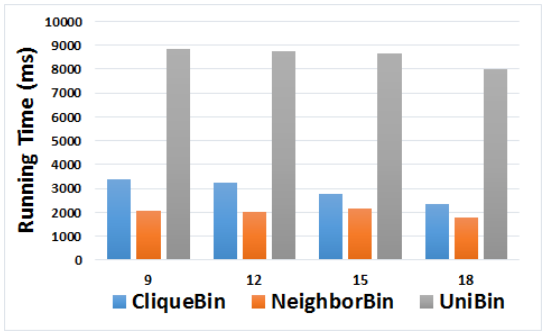


(d) Post insertions

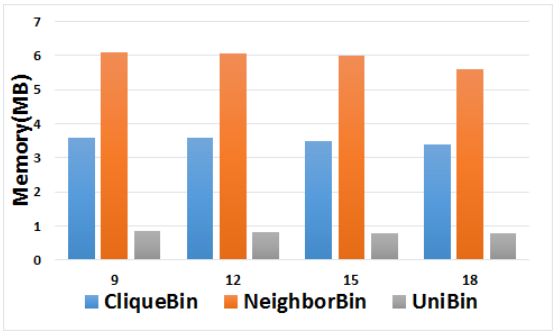
Figure 6.11: Performance of the algorithms under different time diversity thresholds.

from 9 to 18. Figure 6.12 depicts the results. It shows that, for all the three algorithms, the change of content diversity threshold only slightly affects the performance. The reason is that SimHash can effectively detect tweets with near-duplicate content for $\lambda_c \geq 9$ as we can see in Figure 6.4. With λ_c changing from 9 to 18, the precision is already stable. The recall is lower with smaller λ_c, which means more posts will be detected as non-redundant. But this increase in number of non-redundant posts is slight, and thus the increase in the number of comparisons and insertions does not affect the overall efficiency significantly.

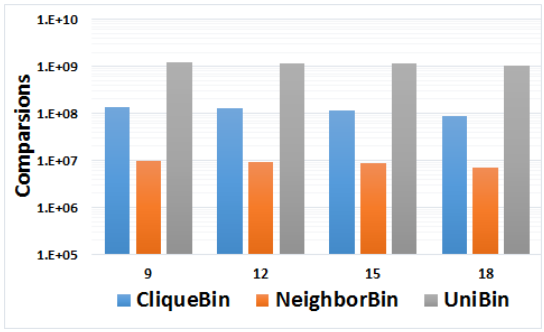
Varying author diversity threshold λ_a. Further, we study the performance by varying



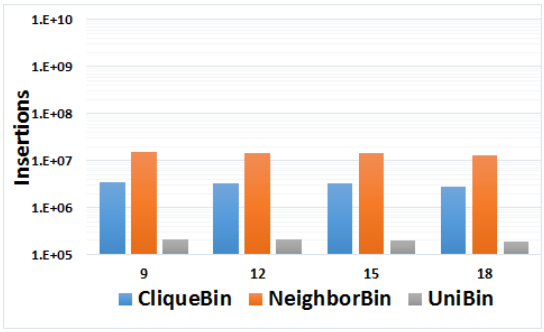
(a) Running time



(b) RAM



(c) Post comparisons

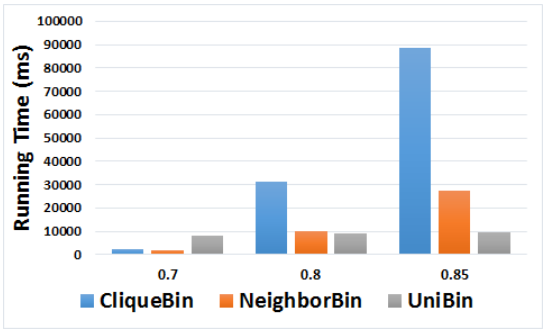


(d) Post insertions

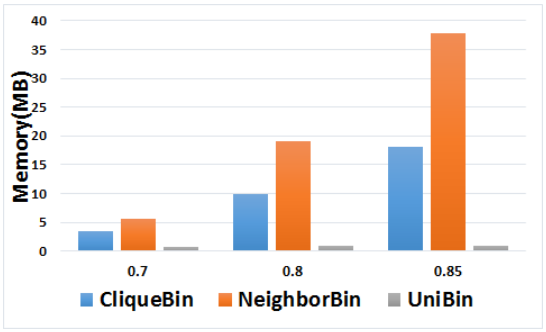
Figure 6.12: Performance of the algorithms under different content diversity thresholds.

λ_a. The results are presented in Figure 6.13 where we set $\lambda_t = 30$ *mins* and $\lambda_c = 18$.

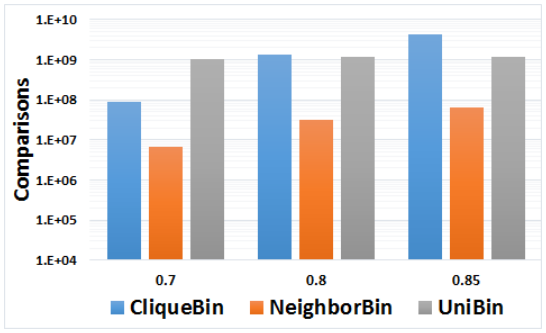
We observe that the author diversity threshold λ_a significantly affects the overall performance of NeighborBin and CliqueBin but not UniBin. When λ_a increases, the author similarity graph gets denser and thus the number of neighbors per author and the number of cliques per author both increase. For instance, when $\lambda_a = 0.7$ the number of neighbors per author (d) is 113.7, the number of cliques per author (c) is 29 and the average size of a clique (s) is 20 in our data set. They change to 437.3, 106 and 38 correspondingly with $\lambda_a = 0.8$. Hence, the number of copies per post in NeighborBin and CliqueBin increases. This explains that in Figure 6.13 the memory consumption by NeighborBin and CliqueBin



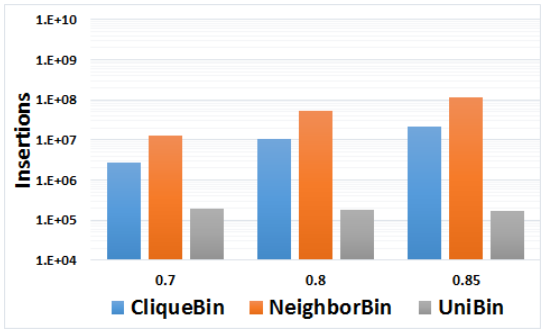
(a) Running time



(b) RAM



(c) Post comparisons



(d) Post insertions

Figure 6.13: Performance of the algorithms under different author diversity thresholds.

increases sharply with larger λ_as. However, the number of non-duplicate posts does not vary much with different λ_as in our data set; thus the performance of UniBin is stable.

We note that when λ_a is large the performance of NeighborBin and CliqueBin (in terms of both memory consumption and running time) is significantly worse than UniBin. Hence, we expect UniBin is the best choice among these three algorithms in use cases where λ_a is large, as we summarize in Table 6.4.

Varying post stream throughputs. We also study the performance of the algorithms under different post stream throughputs. We test this in two ways: (i) varying subscriptions' post rate, and (ii) varying the number of subscriptions. For both, we keep $\lambda_t = 30$ *mins*,

$\lambda_a = 0.7$ and $\lambda_c = 18$.

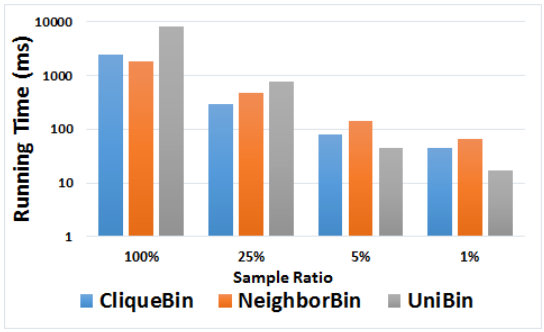
Varying post generation rate. For this, we randomly sample the posts from the 21,050 authors and solve SPSD on the sampled post stream. We conduct experiments for the sample ratio 25%, 5% and 1% and present the results in Figure 6.14. The results show that when the throughput is low (the same ratio is low) UniBin outperforms the other two algorithms. We can also see that CliqueBin performs better than NeighborBin with a moderate or small post generation rate.

Varying the number of subscribed authors. The results shown above are for the case of one user subscribing (following) all Twitter authors in our dataset. In this experiment, we randomly sample Twitter authors in our dataset with different sample sizes. We assume that a user subscribes to all authors in one sample and we run the algorithms on the user's post stream. The results in Figure 6.15 show that UniBin slightly outperforms the other two when the number of subscriptions is small.

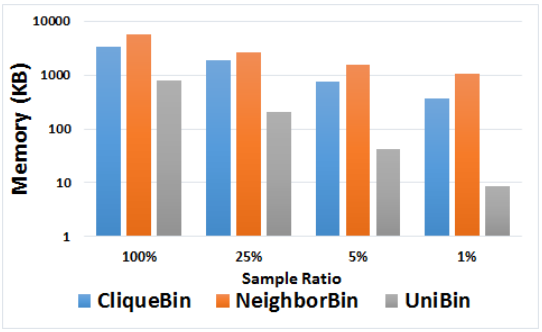
To summarize, UniBin delivers better performance than NeighborBin and CliqueBin when the stream throughput is low. This is consistent with our analysis in Section 6.4.4 – see also Table 6.4.

Discussion

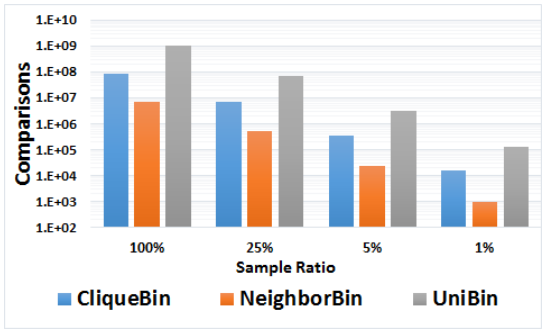
Through extensive experiments, we observe that each algorithm outperforms the other two in certain cases. In Table 6.4 we summarize the best choice of algorithm in different use cases based on our analysis and experimental study.



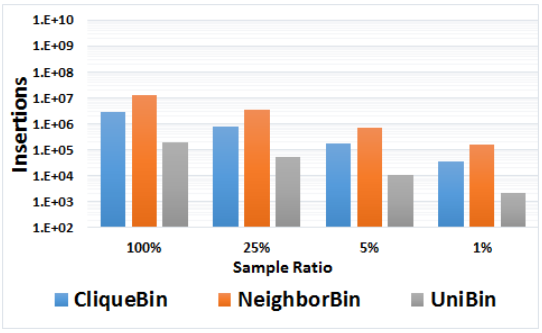
(a) Running time



(b) RAM



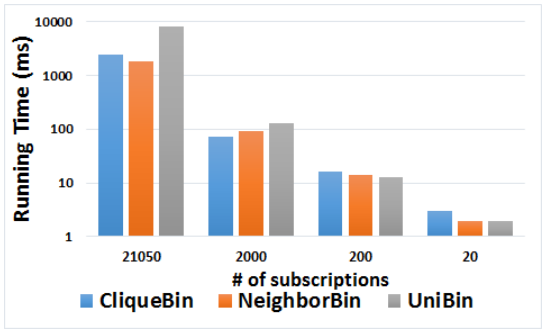
(c) Post comparisons



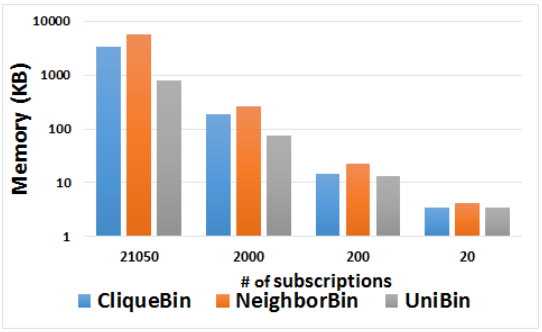
(d) Post insertions

Figure 6.14: Performance of the three algorithms under different post rates.

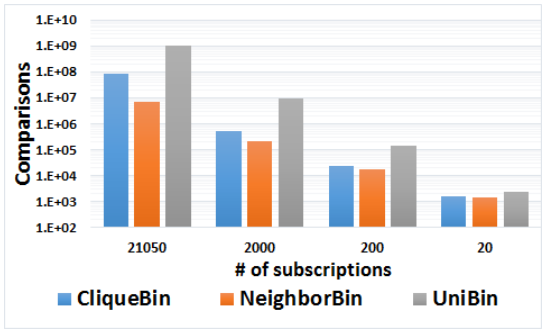
UniBin is the most memory efficient among the three algorithms. Thus in applications with limited RAM UniBin should be considered. Further, when the stream throughput is low (we tested it with small number of subscriptions and low post generation rate), UniBin performs better than the other two. According to the analysis in Table 6.2, we expect that the number of comparisons increases super-linearly with n (the number of posts in a λ_t time range), however the number of insertions increases sub-linearly with n. With a lower stream throughput (smaller n) the overhead of insertions in NeighborBin and CliqueBin is a large contribution to the total running time. When n is small enough, the overhead on insertions becomes larger than the saving on comparisons for NeighborBin and CliqueBin compared



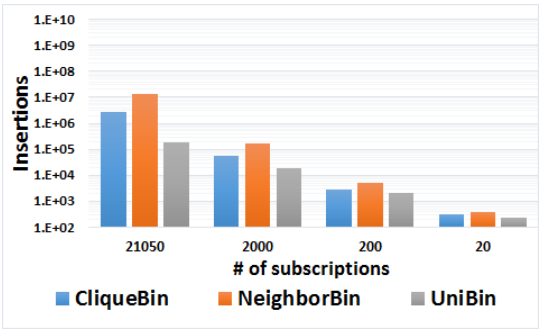
(a) Running time



(b) RAM



(c) Post comparisons



(d) Post insertions

Figure 6.15: Performance of the three algorithms varying the number of subscribed authors.

with UniBin. The similar reasoning can be applied to explain why UniBin is the best choice when λ_t is very small. To clarify, in Figure 6.11 we did not include the results by setting $\lambda_t = 1\ min$ where UniBin performs best among the three algorithms. We argued that with a larger λ_a both d (number of neighbors per author) and c (number of cliques per author) increase and thus NeighborBin and CliqueBin both have higher number of comparisons and insertions. Thus we can see UniBin is preferable when λ_a is set large. One example use case for UniBin is News RSS Feed reader, where the author similarity graph is dense. Generally, news agents form clusters (e.g., by their political views) such that in each cluster the news agents are similar to each other from a user's perspective. Another use case could be Google

Table 6.4: Use cases of the three algorithms for SPSD

Conditions	*Algorithm choice*	*Example use case*
Very small λ_t OR low stream throughput OR large λ_a (dense G) OR RAM is a critical limitation	UniBin	News RSS Feed, Google Scholar
Large λ_t AND small λ_a (sparse G) AND high stream throughput	NeighborBin	Twitch
Moderate λ_t AND small λ_a (sparse G) AND high stream throughput	CliqueBin	Twitter

Scholar where the post (scientific publication) throughput is low.

In other cases, CliqueBin or NeighborBin will be the better choice. They both perform well in cases with a high or moderate stream throughput, which is very common for online social networks. The tie breaker between them is the time diversity threshold λ_t, as we analyzed λ_t determines the tradeoffs between costs of comparisons and insertions. CliqueBin is a better choice if λ_t is set moderately. For example, in Twitter information is time sensitive and thus people may be interested in reading posts with related content but with time distance larger than, say, minutes. For applications where the value of λ_t could be in hours or even days, NeighborBin can be applied. For example, Twitch[3] is a platform on which people can watch and share video game shows. Users may not be interested in watching the video record of the same match that posted at different time. Even in Twitter some users may prefer to customize the λ_t to a larger value, in order to reduce the post volume if they follow a large number of authors.

[3]http://www.twitch.tv/

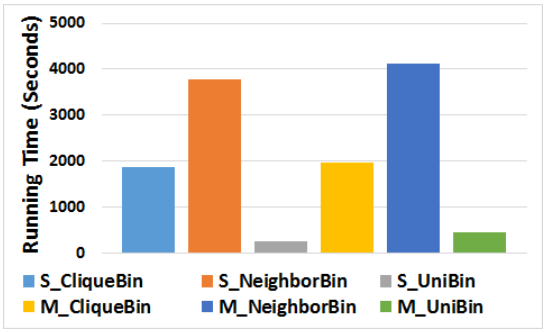
6.6.3 Performance of the algorithms for M-SPSD

We consider now the scenario where each Twitter author is also a user. Each user subscribes to (follows) a set of authors which we can get from the original follower-followee social graph. Then we run the algorithms solving M-SPSD for these 21,050 users in our data set. For the experiments in this section, we set $\lambda_t = 30\ mins$, $\lambda_a = 0.7$ and $\lambda_c = 18$.

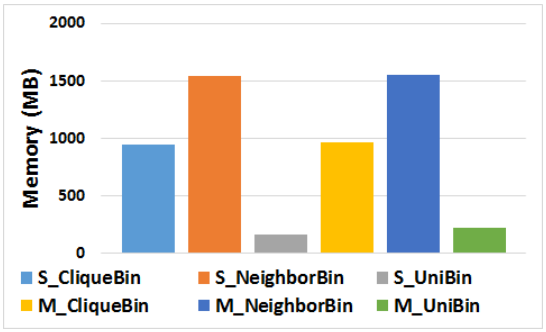
The average number of subscriptions in our sampled user data set is 443.6 and the median is 187. Since we only crawled the posts and computed the author similarity graph for the set of 21,050 authors, we ignored the subscriptions that are not in this set. Then the average number of subscriptions per user drops to 130 and the median is 20. We should note that this reduces the probability of different users sharing common subscriptions.

Figure 6.16 presents the performance of the algorithms. It shows that the proposed optimization (reusing computation and data structure across multiple users described in Section 6.5) improves time efficiency as well as memory consumption. Specifically, S_UniBin uses 43% less running time and 27% less memory than M_UniBin. In the S_UniBin method, posts are stored separately by connected components. This reduces the number of comparisons significantly over M_UniBin. We also observe tthat S_NeighborBin reduces the running time of M_NeighborBin by 8% while S_CliqueBin improves M_NeighborBin by 4% in running time.

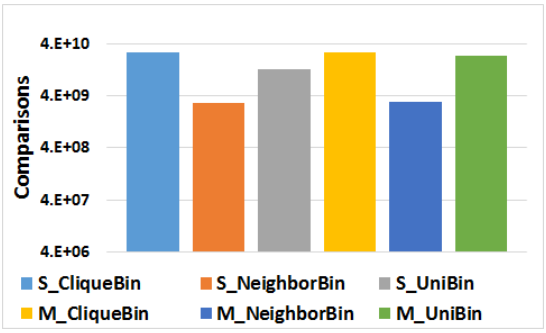
S_UniBin achieves superior performance. We also notice that S_NeighborBin requires fewer post comparisons than S_UniBin but many more insertions. We think that S_UniBin outperforms S_NeighborBin and S_CliqueBin also because its post access pattern is sequential while in the other two are not (each post bin is a map).



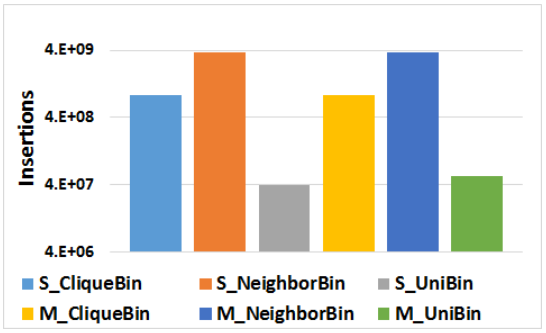
(a) Running Time



(b) RAM



(c) Post comparisons



(d) Post insertions

Figure 6.16: Performance of the algorithms for M-SPSD.

6.7 Related Work

Time Aware Diversity. The authors of [40] solve the problem of maintaining the k most diverse results in a sliding window over a stream. MaxMin semantics is used. They maintain a data structure called the cover tree and show how to incrementally add new and remove expired results from this tree. The cover tree cannot be used for our diversity semantics because it cannot handle simultaneous similarity in three dimensions: time, content and author.

Diversification on Microblogging Posts. The work of [22] studies the problem

of diversifying posts in microblogging systems. In their problem setting, users subscribe several queries (topics). However, in practice users are more often subscribing to authors, which is the setting of the problem we studied in this work. In [22] they apply strict coverage semantics similar to ours, but limited only to time and content diversity. Unlike in our model, in [22] the content diversity is guided by the inputted queries where no inter-post content similarity is considered. They also studied the stream variation of their problem in which they allow a lag upon a new post to decide whether it should be outputted. Our diversity model is required to make the decision immediately at the arrival of a post.

Document Stream Summarization. The authors of [112] work on the problem of summarizing a Twitter stream. They model the summarization problem as a facility location problem. Give a budget of k, they aim to select k tweets that maximize the similarity to the whole tweets set. They incorporate the time factor to measure the document similarity of two posts. But unlike in our problem, instead of using a hard (boolean) threshold, they consider an exponential decay to the content similarity based on their timestamp difference. In the work of [90], the authors apply clustering techniques for Twitter stream summarization. Tweets are clustered according to content similarity. For each cluster, they build a word graph or phrase graph and pick frequent sentences ("paths" in the graph) to construct a summary. The sentences in the summary may not be in any original tweet. The authors of [104] propose a one-pass online clustering algorithm to cluster tweets, and then they generate online summaries by selecting k tweets (one from each cluster) that have high LexRank [43] score. In [98], the authors apply topic modeling for personalized time-aware tweet summarization. However, all these work do not consider author similarity to measure

the similarity between tweets.

Detecting Duplicate Tweets. In [113], the authors propose to use machine learning methods to detect near-duplicates in tweets. For this, they construct a rich set of syntactic, semantic and contextual features. They aim to distinguish different levels of near-duplicates, e.g. exact copy, strong near-duplicate, or weak near-duplicate.

6.8 Conclusion

In this work, we studied the novel problem of diversifying social post streams by incorporating diversity in three dimensions: content, time and author. We illustrated the challenges of solving the problem and proposed various algorithms to efficiently handle these challenges. We show the tradeoffs between our proposed algorithms and argue the use cases for them. We also studied the problem of applying the proposed diversification model for multiple users in a social system. Extensive experiments prove the effectiveness of our model and efficiency of proposed algorithms.

Chapter 7

Conclusions

In this thesis, we studied several problems to help users effectively explore Web and Social Network data.

In Chapter 2, we showed that the change in the terms distribution of results of timely queries over time is strongly correlated with the users' perception of time-sensitivity. We proposed principled ways to incorporate document freshness into the ranking model [23].

In Chapter 3, we proposed a method to estimate the difficulty of a query over structured data. The proposed method is based on the differences between the rankings of the same query over the original and noisy (corrupted) versions of the same database, where the noise spans on both the content and the structure of the result entities. To improve the efficiency of the prediction model, we introduced efficient approximate algorithms with controlled quality trade-offs [25, 26].

In Chapter 4, we proposed context-aware query ranking models by leveraging the user query session [24]. Specifically, we proposed context-aware search strategies that

build upon two popular ranking algorithms: BM25 and Language Model. Our results on biomedical data show that the context-aware query ranking can significantly improve the search results.

In Chapter 5, we introduced and formalized the MQDP problem and its streaming variant. We showed that MQDP is NP-hard and proposed exact and approximation algorithms with provable approximation bounds for MQDP and its stream variant. We also showed a principled approach to achieve proportional diversity, where the popularity of topics (queries) is reflected in the result. For that, we showed how a dynamic post-specific diversity threshold can be defined [22].

In Chapter 6, we formalized the SPSD problem. To solve SPSD in a system with high post throughput, we studied how content similarity can be efficiently applied to social posts. Further, we proposed efficient data structures and algorithms to solve SPSD and showed how different algorithms perform better for different diversity needs. We extended the proposed algorithms to handle many users in a social system, by reusing diversification computation across users.

Bibliography

[1] Amazon Mechanical Turk. http://www.mturk.com/.

[2] Apache Lucene. http://lucene.apache.org/core/.

[3] Data set for time queries. http://dblab.cs.ucr.edu/projects/TimelyQueries.

[4] Google Insights. http://www.google.com/insights/search/.

[5] TREC Web Track Datasets. http://trec.nist.gov/data/webmain.html.

[6] F. Abel, Q. Gao, G.-J. Houben, and K. Tao. Semantic enrichment of twitter posts for user profile construction on the social web. In *ESWC*, pages 375–389, 2011.

[7] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong. Diversifying search results. In *WSDM*, pages 5–14, 2009.

[8] O. Alonso, M. Gertz, and R. A. Baeza-Yates. Clustering and exploring search results using timeline constructions. In *CIKM*, pages 97–106, 2009.

[9] B. S. Alper, J. A. Hand, S. G. Elliott, S. Kinkade, M. J. Hauan, D. K. Onion, and B. M. Sklar. How much effort is needed to keep up with the literature relevant for primary care? *Journal of the Medical Library association*, 92(4):429, 2004.

[10] A. Angel and N. Koudas. Efficient diversity-aware search. In *SIGMOD*, pages 781–792, 2011.

[11] J. A. Aslam and V. Pavlu. Query hardness estimation using jensen-shannon divergence among multiple scoring functions. In *ECIR*, 2007.

[12] S. M. Beitzel, E. C. Jensen, A. Chowdhury, and O. Frieder. Varying approaches to topical web query classification. In *SIGIR*, pages 783–784, 2007.

[13] N. J. Belkin, P. B. Kantor, E. A. Fox, and J. A. Shaw. Combining the evidence of multiple query representations for information retrieval. *Inf. Process. Manage.*, 31(3):431–448, 1995.

[14] E. V. Bernstam, J. R. Herskovic, Y. Aphinyanaphongs, C. F. Aliferis, M. G. Sriram, and W. R. Hersh. Using citation data to improve retrieval from medline. *Journal of the American Medical Informatics Association*, 13(1):96–105, 2006.

[15] G. Bhalotoa, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using BANKS. In *ICDE*, 2002.

[16] C. J. C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. N. Hullender. Learning to rank using gradient descent. In *ICML*, pages 89–96, 2005.

[17] M. Busch, K. Gade, B. Larson, P. Lok, S. Luckenbill, and J. Lin. Earlybird: Real-time search at twitter. In *ICDE*, pages 1360–1369, 2012.

[18] T. Campos and R. Nuno. Using top-k retrieved web snippets to date temporal implicit queries based on web content analysis. In *SIGIR*, pages 1325–1326, 2011.

[19] G. Capannini, F. M. Nardini, R. Perego, and F. Silvestri. Efficient diversification of web search results. *Proceedings of the VLDB Endowment*, 4(7):451–459, 2011.

[20] Y. Chang, A. Dong, P. Kolari, R. Zhang, Y. Inagaki, F. Diaz, H. Zha, and Y. Liu. Improving recency ranking using twitter data. *ACM Trans. Intell. Syst. Technol.*, 4:4:1–4:24, 2013.

[21] C. Chen, F. Li, B. C. Ooi, and S. Wu. TI: an efficient indexing mechanism for real-time search on tweets. In *SIGMOD*, pages 649–660, 2011.

[22] S. Cheng, A. Arvanitis, M. Chrobak, and V. Hristidis. Multi-query diversification in microblogging posts. In *EDBT*, pages 133–144, 2014.

[23] S. Cheng, A. Arvanitis, and V. Hristidis. How fresh do you want your search results? In *CIKM*, pages 1271–1280, 2013.

[24] S. Cheng, V. Hristidis, and M. Weiner. Leveraging user query sessions to improve searching of medical literature. In *AMIA Annual Symposium Proceedings*, volume 2013, page 214. American Medical Informatics Association, 2013.

[25] S. Cheng, A. Termehchy, and V. Hristidis. Predicting the Effectiveness of Keyword Queries on Databases. In *ACM CIKM*, 2012.

[26] S. Cheng, A. Termehchy, and V. Hristidis. Efficient prediction of difficult keyword queries over databases. *TKDE*, 26(6):1507–1520, 2014.

[27] A. Christiansen and D. Lee. Relevance feedback query refinement for pdf medical journal articles. In *Computer-Based Medical Systems, 2006. CBMS 2006. 19th IEEE International Symposium on*, pages 57–62. IEEE, 2006.

[28] R. T. Clemen and R. L. Winkler. Unanimity and compromise among probability forecasters. *Management Science*, 36(7):767–779, 1990.

[29] K. Collins-Thompson and P. N. Bennett. Predicting query performance via classification. In *ECIR*, 2010.

[30] N. Dai, M. Shokouhi, and B. D. Davison. Learning to rank for freshness and relevance. In *SIGIR*, pages 95–104, 2011.

[31] W. Dakka, L. Gravano, and P. G. Ipeirotis. Answering general time-sensitive queries. *TKDE*, 24(2):220–235, 2012.

[32] E. Demidova, P. Fankhauser, X. Zhou, and W. Nejdl. DivQ: Diversification for Keyword Search over Structured Databases. In *SIGIR*, 2010.

[33] M. Demirbas, M. A. Bayir, C. G. Akcora, Y. S. Yilmaz, and H. Ferhatosmanoglu. Crowd-sourced sensing and collaboration using twitter. In *WOWMOM*, pages 1–9, 2010.

[34] P. Diaconis and R. L. Graham. Spearman's footrule as a measure of disarray. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 262–268, 1977.

[35] F. Diaz. Integration of news content into web results. In *WSDM*, pages 182–191, 2009.

[36] R. I. Dogan, G. C. Murray, A. Névéol, and Z. Lu. Understanding pubmed® user search behavior through log analysis. *Database*, 2009:bap018, 2009.

[37] A. Dong, Y. Chang, Z. Zheng, G. Mishne, J. Bai, R. Zhang, K. Buchner, C. Liao, and F. Diaz. Towards recency ranking in web search. In *WSDM*, pages 11–20, 2010.

[38] A. Dong, R. Zhang, P. Kolari, J. Bai, F. Diaz, Y. Chang, Z. Zheng, and H. Zha. Time is of the essence: improving recency ranking using Twitter data. In *WWW*, pages 331–340, 2010.

[39] M. Drosou and E. Pitoura. Disc diversity: result diversification based on dissimilarity and coverage. *PVLDB*, 6(1):13–24, 2012.

[40] M. Drosou and E. Pitoura. Dynamic diversification of continuous data. In *EDBT*, pages 216–227, 2012.

[41] M. Efron and G. Golovchinsky. Estimation methods for ranking recent information. In *SIGIR*, pages 495–504, 2011.

[42] J. L. Elsas and S. T. Dumais. Leveraging temporal dynamics of document content in relevance ranking. In *WSDM*, pages 1–10, 2010.

[43] G. Erkan and D. R. Radev. Lexrank: Graph-based lexical centrality as salience in text summarization. *J. Artif. Int. Res.*, 22(1):457–479, 2004.

[44] P. T. Eugster, P. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, 2003.

[45] R. Fagin, R. Kumar, and D. Sivakumar. Comparing top-k lists. *SIAM J. Discrete Math.*, 17(1):134–160, 2003.

[46] U. Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

[47] V. Ganti, Y. He, and D. Xin. Keyword++: A framework to improve keyword search over entity databases. *PVLDB*, 3:711–722, 2010.

[48] G. Giannopoulos, I. Weber, A. Jaimes, and T. K. Sellis. Diversifying user comments on news articles. In *WISE*, pages 100–113, 2012.

[49] J. Gibbons and S.Chakraborty. *Nonparametric Statistical Inference*. Marcel Dekker, New York, 1992.

[50] A. Goel, A. Sharma, D. Wang, and Z. Yin. Discovering similar users on twitter. In *11th Workshop on Mining and Learning with Graphs*, 2013.

[51] M. Hagen, B. Stein, and T. Rüb. Query session detection as a cascade. In *CIKM*, pages 147–152, 2011.

[52] J. Han, M. Kamber, and J. Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2011.

[53] C. Hauff, L. Azzopardi, and D. Hiemstra. The combination and evaluation of query performance prediction methods. In *Advances in Information Retrieval*, 2009.

[54] C. Hauff, L. Azzopardi, D. Hiemstra, and F. Jong. Query performance prediction: Evaluation contrasted with effectiveness. In *Advances in Information Retrieval*, 2010.

[55] C. Hauff, V. Murdock, and R. Baeza-Yates. Improved query difficulty prediction for the web. In *CIKM*, 2008.

[56] B. He and I. Ounis. Query performance prediction. *Inf. Syst.*, 31:585–594, November 2006.

[57] W. Hersh, A. Cohen, L. Ruslen, and P. Roberts. Trec 2007 genomics track overview. In *TREC*, 2007.

[58] W. R. Hersh, A. M. Cohen, P. M. Roberts, and H. K. Rekapalli. Trec 2006 genomics track overview. In *TREC*, 2006.

[59] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient ir-style keyword search over relational databases. In *VLDB 2003*.

[60] A. Huang. Similarity measures for text document clustering. In *NZCSRSC*. 2008.

[61] N. C. Ide, R. F. Loane, and D. Demner-Fushman. Essie: a concept-based search engine for structured biomedical text. *JAMIA*, 14(3):253–263, 2007.

[62] D. Ikeda, T. Fujiki, and M. Okumura. Automatically linking news articles to blog entries. In *AAAI Spring Symposium: Computational Approaches to Analyzing Weblogs*, pages 78–82, 2006.

[63] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of ir techniques. *TOIS*, 20(4):422–446, 2002.

[64] R. Jones and F. Diaz. Temporal profiles of queries. *TOIS*, 25(3), 2007.

[65] S. M. Katz. Estimation of probabilistic from sparse data for the language model component of a speech recognizer. *IEEE Trans. Signal Process.*, 35(3):400–401, 1987.

[66] J. Kim, X. Xue, and B. Croft. A probabilistic retrieval model for semistructured data. In *ECIR*, 2009.

[67] C. Kohlschütter, P. Fankhauser, and W. Nejdl. Boilerplate detection using shallow text features. In *WSDM*, pages 441–450, 2010.

[68] M. Koniaris, G. Giannopoulos, T. Sellis, and Y. Vasileiou. Diversifying microblog posts. In *Web Information Systems Engineering–WISE 2014*, pages 189–198. Springer, 2014.

[69] A. C. König, M. Gamon, and Q. Wu. Click-through prediction for news queries. In *SIGIR*, pages 347–354, 2009.

[70] A. Kothari, W. Magdy, A. M. Kareem Darwish, and A. Taei. Detecting comments on news articles in microblogs. *ICWSM*, 2013.

[71] A. Kulkarni, J. Teevan, K. M. Svore, and S. T. Dumais. Understanding temporal query dynamics. In *WSDM*, pages 167–176, 2011.

[72] S. Kullback and R. Leibler. On information and sufficiency. *Annals of Mathematics and Statistics*, 22:79–86, 1951.

[73] O. Kurland, A. Shtok, D. Carmel, and S. Hummel. A unified framework for post-retrieval query-performance prediction. In *ICTIR*, 2011.

[74] O. Kurland, A. Shtok, S. Hummel, F. Raiber, D. Carmel, and O. Rom. Back to the roots: a probabilistic framework for query-performance prediction. In *CIKM*, 2012.

[75] H. Kwak, C. Lee, H. Park, and S. Moon. What is twitter, a social network or a news media? In *WWW*, pages 591–600, 2010.

[76] J. Lewis, S. Ossowski, J. Hicks, M. Errami, and H. R. Garner. Text similarity: an alternative way to search medline. *Bioinformatics*, 22(18):2298–2304, 2006.

[77] X. Li and W. B. Croft. Time-based language models. In *CIKM*, pages 469–475, 2003.

[78] D. A. Lindberg, B. L. Humphreys, and A. T. McCray. The unified medical language system. *Methods of information in Medicine*, 32(4):281–291, 1993.

[79] Z. Lu. Pubmed and beyond: a survey of web tools for searching biomedical literature. *Database*, 2011:baq036, 2011.

[80] C. Lucchese, S. Orlando, R. Perego, F. Silvestri, and G. Tolomei. Identifying task-based sessions in search engine query logs. In *WSDM*, pages 277–286, 2011.

[81] Y. Luo, X. Lin, W. Wang, and X. Zhou. Spark: Top-k keyword query in relational databases. In *SIGMOD*, 2007.

[82] G. S. Manku, A. Jain, and A. Das Sarma. Detecting near-duplicates for web crawling. In *WWW*, pages 141–150, 2007.

[83] C. Manning, P. Raghavan, and H. Schutze. *An Introduction to Information Retrieval*. 2008.

[84] E. Meij, W. Weerkamp, and M. de Rijke. Adding semantics to microblog posts. In *WSDM*, pages 563–572, 2012.

[85] D. Metzler, S. Dumais, and C. Meek. Similarity measures for short segments of text. In *Advances in Information Retrieval*, volume 4425, pages 16–27. 2007.

[86] E. Mittendorf and P. Schauble. Measuring the effects of data corruption on information retrieval. In *SDAIR*, 1996.

[87] F. Morstatter, J. Pfeffer, H. Liu, and K. M. Carley. Is the sample good enough? comparing data from twitter's streaming api with twitter's firehose. In *ICWSM*, 2013.

[88] A. Nandi and H. V. Jagadish. Assisted querying using instant-response interfaces. In *SIGMOD*, 2007.

[89] J. Oh, T. Kim, S. Park, H. Yu, and Y. H. Lee. Efficient semantic network construction with application to pubmed search. *Knowledge-Based Systems*, 39:185–193, 2013.

[90] A. Olariu. Clustering to improve microblog stream summarization. In *SYNASC*, pages 220–226, 2012.

[91] M. G. Ozsoy, K. D. Onal, and I. S. Altingovde. Result diversification for tweet search. In *WISE*, pages 78–89. Springer, 2014.

[92] M.-H. Peetz, E. Meij, M. de Rijke, and W. Weerkamp. Adaptive temporal query modeling. In *ECIR*, pages 455–458, 2012.

[93] J. Pérez-Iglesias, J. R. Pérez-Agüera, V. Fresno, and Y. Z. Feinstein. Integrating the probabilistic models BM25/BM25F into Lucene. *CoRR*, abs/0911.5046, 2009.

[94] J. M. Ponte and W. B. Croft. A language modeling approach to information retrieval. In *SIGIR*, pages 275–281, 1998.

[95] L. Qin, J. X. Yu, and L. Chang. Diversifying top-k results. *PVLDB*, 5(11):1124–1135, 2012.

[96] K. Radinsky, K. M. Svore, S. T. Dumais, J. Teevan, A. Bocharov, and E. Horvitz. Modeling and predicting behavioral dynamics on the web. In *WWW*, pages 599–608, 2012.

[97] D. Rafiei, K. Bharat, and A. Shukla. Diversifying web search results. In *WWW*, pages 781–790, 2010.

[98] Z. Ren, S. Liang, E. Meij, and M. de Rijke. Personalized time-aware tweets summarization. In *SIGIR*, pages 513–522, 2013.

[99] C. Sadowski and G. Levin. Simhash: Hash-based similarity detection. Technical report, Google, 2007.

[100] J. Sankaranarayanan, H. Samet, B. E. Teitler, M. D. Lieberman, and J. Sperling. Twitterstand: news in tweets. In *GIS*, pages 42–51, 2009.

[101] R. L. T. Santos, J. Peng, C. Macdonald, and I. Ounis. Explicit search result diversification through sub-queries. In *ECIR*, pages 87–99, 2010.

[102] N. Sarkas, S. Paparizos, and P. Tsaparas. Structured annotations of web queries. In *SIGMOD*, 2010.

[103] X. Shen and C. X. Zhai. Exploiting query history for document ranking in interactive information retrieval. In *CIKM*, pages 377–378, 2003.

[104] L. Shou, Z. Wang, K. Chen, and G. Chen. Sumblr: continuous summarization of evolving tweet streams. In *SIGIR*, pages 533–542, 2013.

[105] A. Shtok, O. Kurland, and D. Carmel. Predicting query performance by query-drift estimation. In *ICTIR*, 2009.

[106] X. Shuai, X. Liu, and J. Bollen. Improving news ranking by community tweets. In *WWW*, pages 1227–1232, 2012.

[107] I. Soboroff. A comparison of pooled and sampled relevance judgments. In *SIGIR*, pages 785–786, 2007.

[108] F. Song and W. B. Croft. A general language model for information retrieval. In *CIKM*, pages 316–321, 1999.

[109] S. Sood and D. Loguinov. Probabilistic near-duplicate detection using simhash. In *CIKM*, pages 1117–1126, 2011.

[110] S. Sriram, X. Shen, and C. Zhai. A session-based search engine. In *SIGIR*, pages 492–493, 2004.

[111] A. Styskin, F. Romanenko, F. Vorobyev, and P. Serdyukov. Recency ranking by diversification of result set. In *CIKM*, pages 1949–1952, 2011.

[112] H. Takamura, H. Yokono, and M. Okumura. Summarizing a document stream. In *Advances in Information Retrieval*, pages 177–188. Springer, 2011.

[113] K. Tao, F. Abel, C. Hauff, G.-J. Houben, and U. Gadiraju. Groundhog day: near-duplicate detection on twitter. In *WWW*, pages 1273–1284, 2013.

[114] A. Termehchy, M. Winslett, and Y. Chodpathumwan. How schema independent are schema free query interfaces? In *ICDE*, 2011.

[115] S. C. Townsend, Y. Zhou, and B. Croft. Predicting query performance. In *SIGIR*, 2002.

[116] T. Tran, P. Mika, H. Wang, and M. Grobelnik. Semsearch'10: the 3th semantic search workshop. In *WWW*, 2010.

[117] TREC 9 Filtering dataset. http://trec.nist.gov/data/t9_filtering.html.

[118] A. Trotman and Q. Wang. Overview of the inex 2010 data centric track. In *Comparative Evaluation of Focused Retrieval*, volume 6932 of *Lecture Notes in Computer Science*. 2011.

[119] M. Tsagkias, M. de Rijke, and W. Weerkamp. Linking online news and social media. In *WSDM*, pages 565–574, 2011.

[120] E. M. Voorhees and D. K. Harman. *TREC: Experiment and Evaluation in Information Retrieval*. The MIT Press, 2005.

[121] X. Wang, H. Liu, P. Zhang, and B. Li. Identifying information spreaders in twitter follower networks. Technical Report TR-12-001, School of Computing, Informatics, and Decision Systems Engineering, Arizona State University, Tempe, AZ 85287, USA, 2012.

[122] L. Wu, W. Lin, X. Xiao, and Y. Xu. LSII: An indexing structure for exact real-time search on microblogs. In *ICDE*, 2013.

[123] L. Wu, Y. Wang, J. Shepherd, and X. Zhao. An optimization method for proportionally diversifying search results. In *Advances in Knowledge Discovery and Data Mining*, pages 390–401. 2013.

[124] B. Xiang, D. Jiang, J. Pei, X. Sun, E. Chen, and H. Li. Context-aware ranking in web search. In *SIGIR*, pages 451–458, 2010.

[125] X. Yang, A. Ghoting, Y. Ruan, and S. Parthasarathy. A framework for summarizing and analyzing twitter feeds. In *SIGKDD*, pages 370–378, 2012.

[126] E. Yom-Tov, S. Fine, D. Carmel, and A. Darlow. Learning to estimate query difficulty: including applications to missing content detection and distributed information retrieval. In *SIGIR*, 2005.

[127] O. Zamir and O. Etzioni. Web document clustering: a feasibility demonstration. In *SIGIR*, pages 46–54, 1998.

[128] H.-J. Zeng, Q.-C. He, Z. Chen, W.-Y. Ma, and J. Ma. Learning to cluster web search results. In *SIGIR*, pages 210–217, 2004.

[129] C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In *SIGIR*, pages 334–342, 2001.

[130] Y. Zhao, F. Scholer, and Y. Tsegay. Effective pre-retrieval query performance prediction using similarity and variability evidence. In *ECIR*, 2008.

[131] Y. Zhou and B. Croft. Ranking robustness: A novel framework to predict query performance. In *CIKM*, 2006.

[132] Y. Zhou and B. Croft. Query performance prediction in web search environments. In *SIGIR*, 2007.