Elevating Search Results from Flat Lists to Structured Expansions

by

Xueyao Liang

B.E. in Software Engineering, Nanjing University, 2009

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES
(Computer Science)

The University Of British Columbia
(Vancouver)

December 2011

Abstract

Keyword based search interfaces are extremely popular as a means for efficiently discovering items of interest from a huge collection, as evidenced by the success of search engines like Google and Bing. However, most of the current search services still return results as a flat ranked list of items. Considering the huge number of items which can match a query, this list based interface can be very difficult for the user to explore and find important items relevant to their search needs. In this work, we consider a search scenario in which each item is annotated with a set of keywords. E.g., in Web 2.0 enabled systems such as flickr and del.icio.us, it is common for users to tag items with keywords. Based on this annotation information, we can automatically group query result items into different expansions of the query corresponding to subsets of keywords. We formulate and motivate this problem within a top-k query processing framework, but as that of finding the top-k most important expansions. Then we study additional desirable properties for the set of expansions returned, and formulate the problem as an optimization problem of finding the best k expansions satisfying all the desirable properties. We propose several efficient algorithms for this problem. Our problem is similar in spirit to recent works on automatic facets generation, but has the important difference and advantage that we don't need to assume the existence of pre-defined categorical hierarchy which is critical for these works. Through extensive experiments on both real and synthetic datasets, we show our proposed algorithms are both effective and efficient.

Preface

This thesis is the outcome of my collaborative research with my PH.D. colleague Min Xie and my supervisor Laks V.S. Lakshmanan. The three of us together published the below listed paper [33], which subsumes the major outcome of this thesis.

1. Xueyao Liang, Min Xie, and Laks V.S. Lakshmanan. 2011. Adding structure to top-k: from items to expansions. In *Proceedings of the 20th ACM international conference on Information and knowledge management* (CIKM '11). ACM, New York, NY, USA, 1699-1708.

The ideas of this paper and my master's thesis are formed through rounds of discussions with these two co-authors. Below lists the contribution of each of the co-authors, including me, in detail.

Dr. Laks V.S. Lakshmanan, my supervisor, motivated the initial brainstorming of the problem solved in this thesis. And later involved in formulating the research problem, designing methods to solve the problem, and revising paper before submission.

Min Xie, a PH.D. colleague of mine, provided generous help on formally defining the research problem, designing algorithms to solve the problem, and designing experiments to evaluate the solution proposed. Min also proposed using path exclusive constraint to semantically optimize the solution, and also involved in writing and revising the aforementioned paper.

I was in charge of summarizing all the thoughts and formulating the ideas at the early stage of the project. Later I was involved in formulating the problem and designing efficient algorithms for solving the problem. I designed the lazy lattice

algorithm with help from the two co-authors. I was also involved in designing methods to semantically optimize the solution, and proposed to use weighted score based on expansion size. Also, I implemented all the experiments included in the aforementioned paper and this thesis, including crawling data from ACM websites, preprocessing, adapting the algorithm, and evaluation. In addition, I was involved in writing, reviewing and revising the published paper and this thesis.

Some paragraphs in the aforementioned paper are duplicated or paraphrased in this thesis, so are some of the figures and tables. ACM grants permission to reprint these materials as long as a proper copyright notice is included. Below is the ACM copyright notice:

Table of Contents

List of Tables

List of Figures

Acknowledgments

I would like to sincerely thank my supervisor Laks V.S. Lakshmanan, for his invaluable guidance on both my master's study and the forming of this thesis. Without his encouragement, support, and insightful suggestions, this work wouldn't be finished.

I also would like to thank my PH.D. colleague Min Xie, who worked with me from the beginning to the end of my Master program. Most ideas of this thesis came from the discussion among Laks, Min and me.

In addition, I would like to thank Professor Raymond Ng for reading this thesis, attending my practise talk for a related publication, and providing many valuable suggestions.

At last, I give my thanks to all my colleagues in Database Management and Mining lab and all staffs in the department for providing support and good research environment during the process.

Chapter 1

Introduction

Nowadays, keyword based search interfaces are widely used as a means for efficiently discovering items of interest from a collection. The success of it has been evidenced by popular search engines like Google [4], Bing [2] and Yahoo! [9]. However, most of the current search services still return a flat ranked list of items as result. Though effective in most cases, this list-based interface can sometimes make it very difficult for the user to explore and find important items relevant to their search needs, as pointed out in a recent study [11]. In this thesis, we claim that *automatically grouping search results into semantically independent "topics" can significantly enhance the usability of the results.*

In section Section 1.1, a detailed discussion about text-based search interfaces and result representation is provided. Several motivating examples are represented in Section 1.1.1 to illustrate the limitations of flat ranked lists. Section Section 1.1.2 briefly introduces faceted search interface. Section Section 1.2 introduces how we model the problem, with a formal definition provided in Chapter 4. The contributions of this thesis are stated in Section 1.3.

1.1 Text-based Search Interface

In the 21st century, it's hard to imagine a life without search engines. Formally, search engines provide an interface to a group of items that enables users to specify criteria about an item of interest and have the engine find the matching items. The

criteria are referred to as a search query[8]. Most current available search engines are texted based search engines, for which the query is typically expressed as a set of words that identify the desired concept that one or more items may contain[31]. Though recently many new forms of search query are proposed, such as Google's search by image[5], which brought tons of interesting problems and opportunities, this thesis only focus on improvements on text-based search interfaces as its still the dominant one.

Almost all text-based search engines nowadays return a flat list of items as the result. An item's position in the result list depends on various criteria like how well its content matches with the search query, its authority within the dataset, and so on. Usually these aspects are combined and subsumed in a ranking function, and the items are represented in descending order of its score according to this ranking function. The most famous function to measure a web page's authority score is called PageRank and proposed by Larry Page [34] in 1999. Within this twenty years, text-based search engines ranking functions have improved tremendously, but the search interface remains almost the same, in the aspect of returning flat item lists as result. Although effective in most cases, there are scenarios where this search interface may not be the most useful one, considering the users are only able to view the first few items returned in the list. This problem is usually exaggerated when the search query is short or general. And according to previous survey and observations, over 90% of the queries are with less than 4 keywords[13][29].

In section Section 1.1.1, we give several examples to illustrate this point of view, and discuss possible better representations. Section Section 1.1.2 introduces faceted interface, which is widely used to alleviate the limitations of flat lists result. Also introduced is attempts of using faceted interface in various domains, and their limitations.

1.1.1 Flat Ranked Lists are not Enough

Consider an academic search engine like Google Scholar [7]. Though it works perfectly for those queries which target a specific paper, when handling a general purpose query like "find all papers which are relevant to the topic histogram", the search engine returns a huge list of papers ranked by their relevance to the query.

The first few items in this ranked list is difficult for user to work from for exploring papers related to the query and efficiently finding the papers they want, especially when the user is new to the field, and is exploring interesting areas within this general topic. It is clear that the user may benefit significantly if the search service can automatically group all the papers into semantically independent "topics".

As a second example, consider search on social annotation websites like Flickr [3]. These websites have rich user generated metadata for each item, however current search engines on these websites only utilize them to generate a ranked list of items for a query based on keyword relevance. For example, when searching for "London", Flickr returns a list containing 13,696,628 images, which made it very hard to further explore the result set like identifying good places of interest for photography in London.

As a third example, consider a community question answering forum like Quora [6]. A user may wish to search the Q&A in Quora using keywords and it is often helpful for the system to present the search results in an automatically grouped form, where different groups somehow correspond to different "subtopics" of the "topic" that the user may have questions about.

1.1.2 Faceted Search Interface

Faceted search interface [12] are widely used to alleviate the previously mentioned limitations of flat lists. In a traditional faceted search interface, item are associated with orthogonal attributes as facets, and the system keeps hierarchical structures among facets, which are usually domain specific. The facets are chosen to filter the original result set of a keyword query to focus on a subset of items containing the specific attributes. Usually, facets are navigated following their position in the hierarchies, from top to the bottom. The facets and their hierarchical relationship are usually predefined for the domain by experts. This usually works fine for specific domains, but heavy human effort is needed, and prevents the utilization of faceted search in general search enabled systems. Based on the domain and pre-assumed knowledge, faceted search interface could be very restricted to very dynamic. Below gives several concrete example of utilizing faceted interface in real search engines, their advantage over flat lists, and some limitations they have.

Fixed Facet Filters:

To help users explore the returned search results, current search engines like Google, Bing and Yahoo! often add to the search result interface a set of facets, like publishing time, size of document, price etc. These facet filters can greatly facilitate user navigation through the results. However, these facets are often pre-defined, and may not capture the attributes of the item which are the most important. E.g., for Google Scholar, current facets for the search results contain only types of publication and publication date, whereas the users may want to explore "topics" of papers among the search results.

Dynamic Facets Representing Interfaces:

In case of Del.icio.us, the ranked list result is affiliated by a faceted interface, by suggestions to expand the user's keyword query with one of a fixed set of tags. E.g., by issuing the query "programming", a list of related tags are provided, with the top three ones being "development", "javascript", and "web". These additional keywords could be useful for providing clues of additional tags highly correlated with "programming". However, only single keywords are provided, and from it is not obvious why they are related to the original query from the search interface. In the previous example, searching on the tag "programming" returns more than 1.3 million hits on Del.icio.us, however, other useful expansions such as "c++ programming tutorial" and "database programming language" cannot be found on the interface.

A more successful adaptation of faceted search interface is in online shopping websites like Amazon[1]. By suggesting a group of "attribute=value" pairs to filter the search result, the user could spend less effort to find the desired product by issuing a general query. E.g., example of facets suggested for the query "Harry Porter" include "Format = DVD", "Genre = Action & Adventure", "Decade = 2010 & Newer", etc. These facets are efficiently and dynamically generated

for each distinct query. However, large amount of "attribute=value" attributes are stored as orthogonal information to products, and usually manually input. Also, hierarchical structure among the attributes are assumed. Moreover, the process of suggesting facets are usually semi-automated, which means human effort is still involved in this process to guarantee the suggestion quality.

These aforementioned examples all illustrate the usefulness of adapting faceted search interface to facilitate the ranked lists based search interfaces. However, these faceted search interfaces are either with limited flexibility or heavily depends on domain specific model and structural information. In this thesis, we are interested in proposing a model which could dynamically group the result items into several facets, with no pre-assumed structural information. The model and problem definition will be introduced in Chapter 4.

1.2 Problem

Motivated by these drawbacks of current search result interfaces, we consider a search scenario in which each item is annotated with a set of keywords. These can be keywords associated with papers or tags assigned to items by users of social annotation systems or keywords occurring in question and answers in Q&A systems. E.g., in social annotation websites such as flickr and del.icio.us, it is common for users to tag items with keywords.

Based on this annotation information, we want to automatically group query result items into different expansions of the query corresponding to subsets of keywords. Items may have a number of attributes, either explicitly stored or computed, which can signify their importance or utility to the user. E.g., papers have citation score, pagerank of their authors, etc. In social annotation systems, popularity of items and pagerank of the annotator can be important attributes. Finally, in a system like Quora, we have attributes like importance scores of questioner or answerer, number of people interested in a question, etc. Intuitively, the expansions that we wish to return to a user should be driven by how important the items are that match an expansion and how many such important items match it. This raises the problem of what expansions of a query should we show the user? We formulate

and motivate this problem within a top-k query processing framework, but as that of finding the top-k most important expansions, where the importance or utility of an expansion is driven by the utility of the items matching it.

Our problem is similar in spirit to recent works on automatic generation of facets [32], but has the important difference and advantage that we don't need to assume the existence of pre-defined categorical hierarchy which is critical for these works. Indeed, in the applications we consider such as above, we cannot assume any prior taxonomy.

1.3 Contributions

In this thesis, we make the following contributions:

- We introduce the problem of finding top-k high quality expansions (Chapter 4).
- We first propose a naïve algorithm for this problem and then discuss its limitations and propose a significant improvement leveraging the lattice structure of expansions (Section 5.1).
- We consider some semantic issues with directly returning the top-k high quality expansions, and propose several different algorithms for improving the quality of the returned results (Section 5.2).
- Through extensive experiments on both real and synthetic datasets, we demonstrate that our proposed algorithms have excellent performance and very good quality (Section 6).

Related work is discussed in Section 3. We summarize the work in Section 7 and discuss open research problems.

1.4 Thesis Structure

In this thesis, we first formally define our top-k query expansion problem in Chapter 4, then we will propose a naïve algorithm and an improved algorithm for solving this problem in Chapter 5. After that in Section 5.2 we will discuss how to handle some semantic issues of the basic algorithms. Our empirical results will be reported in

Chapter 6 and we will summarize and conclude our paper in Chapter 7, along with possible ideas for future works.

Chapter 2

Prerequisites

This chapter is dedicated to give a brief background introduction about concepts and techniques needed to understand technical details of this thesis. As we motivate and formulate the problem within a top-k query processing framework, Section 2.1 gives a general introduction to this framework and NRA algorithm for solving the top-k query processing problem, also how it is taken use of in text based search engines. In Section 2.1.1, an algorithm to rank object instead of items based on the NRA algorithm for top-k query processing is introduced[16], based on which we proposed our baseline naïve algorithm. Though we are able to adapt a similar algorithm with it to our framework, our problem setting is different with theirs. The important difference is that in [16] the candidate object search space is fixed but not exponential, and there is no direct relationship among different candidates. These differences provide space for improvements within our framework.

2.1 Threshold Algorithm for Top-k Query Processing

Top-k query processing is at the core of many search enabled systems. Examples of these systems include not only text based web search engines[19], but also relational databases[27] and distributed middleware systems[21]. In this thesis, the discussion is only limited to text based search engines.

[21] proposed Threshold Algorithm for top-k query processing for middleware systems, which is consisted by n distributed data sources containing values for m

attributes for the objects in the system. And without lose of generality, the items are ranked by a monotonic function F which aggregates the individual attribute values. E.g., min or average. A similar scenario could be mapped to in text based search engines, if we view each item's relevance score to a individual query term as an attribute, for which the most popular way of calculating the score is using BM25 [35] or its variations.

One way of retrieving the top-k answers is using the Document-at-a-time(DAAT) model, by calculating the aggregated score for each item, rank the items in descending order of its score, and return the top k items. However, because of the sparsity of the data and the fact the number of keywords in the corpus is usually much less than the number of items in it, items' relevance score are usually stored in a inverted index for each keyword in the corpus. In each inverted index stores entries for each item with their relevance score to this specific keyword, in descending order of the relevance score. Moreover, due to the sheer volume of datasets, these inverted indexed may be stored on distributed machines, or in disk storage devices. These made randomly accessing the relevance scores of each item rather costly. However, the aforementioned methodology requires extensive random access of the relevance scores from each item, which could be prohibitive given sometime the large size of the corpus.

On the other hand, Threshold Algorithm (TA) enables efficiently finding the top-k items using the Term-at-a-time(TAAT) model. Similar to Threshold Algorithm in middleware system, the query keywords' inverted indexes are scanned sequentially. Since the current scanned score is the always the largest possible score for entries in the remaining part of the list, we could safely keep the upper bound and lower bound updated for each item. The upper bound of an item is non-increasing, while the lower bound is non-decreasing. Thus, at certain depth of scanning, we could identify there are k items' lower bounds are higher than the upper bounds of any other items, and these k items are guaranteed to be the top-k items. Since the algorithm usually achieves this criteria before reaching the end of the inverted indexes, we call it early termination[21]. Depending on whether the algorithm is allowed to randomly access the entries in attribute lists with a relatively higher cost compared with sequential accessing, TA algorithm is further classified into RA (Random Access) algorithm and NRA (Non-Random Access).

In this thesis, we only consider the NRA algorithm family.

It could be proved under sequential accessing model, the threshold algorithm is instance optimal[21]. Specifically, we say an algorithm is instance optimal if for every monotone aggregation function, the algorithm is instance optimal over all algorithms that correctly find the top-k answers and that do not allow random access or make wild guesses, over the class of all dataset.

For details of the proof and NRA algorithm, please refer to [21].

2.1.1 Entity Ranking with Top-k Technique

Traditional threshold algorithm are utilized to find the top-k items, as long as the items are ranked by a monotone function on related attributes. [16] proposed a new class of queries called the "object finder" queries, whose goal is to return the top-k object that best match a given set of keywords. In the scenario they discuss, there are two types of objects in the system:

1. **Search Objects**(SOs): Objects search by the keywords(e.g., papers in expert finder, reviews in the product finder, and previously mentioned items)

2. **Target Objects**(TOs): Objects desired as answers to the query(e.g., authors in expert finder, entities in entity finder)

FigureFigure 2.1 gives an example of the relationship between Search Objects and Target Objects, where Search Objects are product reviews and Target Objects refer to products.

In the figure, there are multiple-to-multiple relationships between Search Objects and Target Objects. This brings challenge to adapting the traditional NRA algorithm for finding top-k objects, because search objects, as the desired answer in traditional search engines, are not the desired answers in this scenario. One more step need to taken to infer the relevance between the query and the target objects, based on the relationship between search and target objects.

An adjusted NRA algorithm is proposed for solving this problem. And the frameworks is showed in FigureFigure 2.2. The system keeps a item-object mapping table, and a ranked list of documents(inverted index) for each keyword. First the inverted indices for the query keywords are retrieved, and entries are sequen-

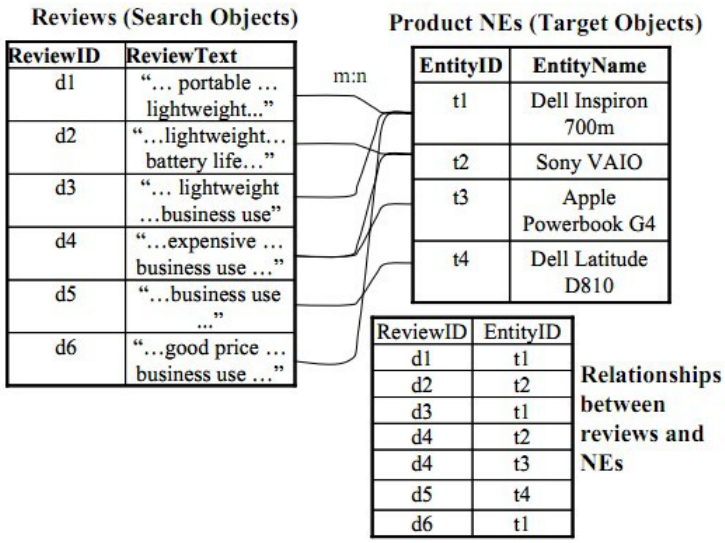

Figure 2.1: Relationship between Search Object and Target Objects

tially scanned in these indexes. Once an entry is scanned, the search object's corresponding target objects are identified in the mapping table, and the lower and upper bounds of corresponding target objects are updated. As discussed in the paper, early termination property holds when the aggregation function satisfies certain criteria. For details of these criteria, please refer to [16].

2.2 Weighted Independent Set Problem

An independent set in a graph is a set of vertices in which no two vertices are adjacent. The (unweighted) independent set problem is that of finding a maximum independent set from a graph. In a weighted version of the problem, each vertex in the graph is assigned a weight. Accordingly, the *weighted* independent set problem is that of finding a maximum independent set[30], where the sum of vertices weight is maximized.

More concretely, for the weighted version of the problem, let G be an undirected graph where each vertex v is associated a positive weight w_v, with $V(G)$

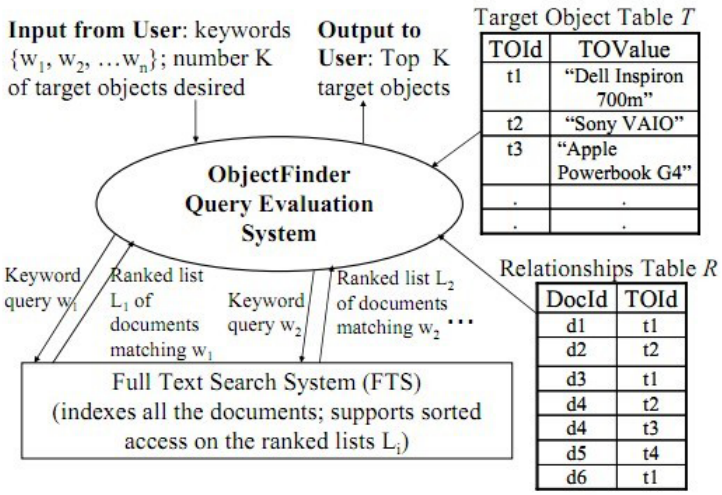

Figure 2.2: Overview of Object Finder Query Evaluation System

and $E(G)$ referring the vertex set and edge set of G accordingly. Let W be the summation of weights of all vertices, n be the number of vertices in G, and $d(v,G)$ as the degree of vertex v in graph G. Moreover, for $X \subseteq V(G)$, $w(X)$ denotes the summation of the weights of vertices in X.The weighted independent set problem is to find a independent set X from G where w(X) is maximized. Several useful definitions needed later are listed below:

1. **Weighted Degree of a Vertex:** $d_w(v,G) = \frac{w(N_G(v))}{w_v}$

   where $N_G(v)$ refers to the set of vertices adjacent to vertex v in G.

2. **Inductiveness:** $\delta_w = \max_{H \subseteq G} \min_{v \in V(H)} d_w(v,H)$

   where $H \subseteq G$ specifies its a subgraph of G.

Both the unweighted and weighted version of independent set problem is NP-Hard. Many approximation algorithms are proposed to solve the unweighted independent set problem, among them a simple greedy solution is proposed, to start from the original graph and repeat these two steps:

1. Select the vertex with the minimal degree from the graph into the independent set result.

2. Remove this vertex and its neighbors from the graph

This process continues until the graph become empty[23, 26].

A weighted version of this greedy algorithm could be adapted on weighted graphs. And the only difference is instead of selecting the vertex with minimum degree in every step, the algorithm selects the vertex with minimum weighted degree. The algorithm is proved to have an approximation ratio $max(\delta_w, 1)$[30]. In Section 5.2, based on heuristic from this algorithm, we adapted a slightly revised version of it for including our Path Exclusive constraints, though in our setting, we don't have full knowledge of the weights of nodes, instead we only keep an upper bound and an lower bound for each of the vertices. Thus our algorithm do not have guarantee of approximation ratio stated here.

Chapter 3

Related Works

The area of top-k query processing has been studied extensively in the past several years [27]. Most of the top-k algorithms are based on the TA and NRA family of algorithms and their variants or enhancements [21]. The majority of them assume a monotone aggregation function for combining scores of items for different attributes. For each attribute, a non-increasing score-sorted list of items is maintained. In [16], Chakrabarti et al. consider the problem of finding top-k entities in a document corpus, where the score of an entity is defined as a weighted aggregation of the scores of its related documents. The proposed algorithm is similar to our naïve algorithm for the top-k expansion problem, and both algorithms are extensions of the NRA algorithm. However, the number of entities arising in [16] can be considered as a constant, whereas in our context, the number of expansions is exponential w.r.t. the total number of keywords, which can be huge. Thus, it is critical to generate the expansions as lazily as possible. A more detailed introduction and discussion is included in Section 2.1.

Our top-k expansion problem is also related to the recent efforts on faceted search [25, 32, 39]. Li et al. [32] propose the problem of automatic generation of top-k facets for query results on Wikipedia. However, their work makes crucial use of a pre-defined category hierarchy in Wikipedia. Zwol et al. [39] suggested to use facets to narrow the result set of image searching. However, their facet selection is based on a pre-constructed facet repository and its mapping to the images, with the facets ranked according to previous query logs. In [18], facets are

modeled as structured information associated with each document, in the form of attribute/value pairs. In their work, facets are selected to present based on their "interestingness", which is based on the attribute/value appearance frequency in the result set.

The above works all assume facets as orthogonal attributes with domain specific hierarchical structure among them. This assumption is not assumed in this thesis. In fact, as mentioned in the introduction, for the applications we consider, this assumption *cannot* be made.

Recently, there are works utilizing unstructured keywords to facilitate faceted search interface construction[17, 36, 37]. Dakka et al. [17] proposes to automatically construct the facet hierarchy for each category, by first classify keywords into pre-defined categories, for which a existed training set is needed, after that construct a hierarchy among keywords fall into the same category based on consumption rules. However, it requires training to the classifier, and the available categories are fixed and predefined.

Simitsis et al. [37] proposed to use dynamic facets extracted from the document contents: frequent phrases appearing in the documents which compose the result set. However, the phrases are picked merely based on frequency, and utilized only as an affiliation to structured attributes, and it could only handle phrases of length up to 5.

In [36], the authors refine the previous paper and propose a way to facilitate the search process by suggesting interesting additional query terms. To measure the interestingness of an additional query term or keyword, they proposed the surprising score which is based on the co-occurrence of two keywords. Compared with our work, they don't consider the overlap between different sets of query keywords. Besides, their score function needs to access all items which are relevant to the query. Since this cannot be calculated at the query time in a scalable manner, they instead propose an approximate solution.

Citation recommendation is another area where previous works [14, 20, 24] often generate results as a ranked list of documents. By contrast, in our work we automatically group all the results into different expansions, and return to the user the top-k interesting expansions along with the relevant items. Ekstrand et al. [20] proposed that the authority of the papers within the literature should be

taken into consideration when recommending papers for new researchers in the respective field. They incorporated importance score (e.g. HITS,PageRank) into the collaborative filtering framework, by adjusting the user-item matrix with paper importance score. They shared some similarity with us in the sense that they also believe the authority of the paper is important for new users, and provide clues for our evaluation.

Chapter 4

Problem Definition

Consider a set of items $S = \{t_1, ..., t_n\}$, each item $t_i \in S$ being associated with a set of m attributes $\{a_1, ..., a_m\}$. We denote the value of t_i on attribute a_j as $t_i.a_j$, and assume without loss of generality that all values on attribute a_j are normalized to $[0,1]$, $j \in 1...m$. Attributes of an item t_i correspond to t_i's "utility" to a user, and in this work, without loss of generality, we assume large attribute values are preferred, so larger the value larger the utility. The overall *utility* of an item t_i, denoted $u(t_i)$, is captured by a weighted sum of its values on all attributes, i.e., $u(t_i) = \sum_{j \in 1...m} w_j \times t_i.a_j$, where w_j is a positive weight associated with attribute a_j. These weights may be chosen by a user or the system may learn the "best" weights from user behavior using models like linear regression [15].

We assume each item $t_i \in S$ is also associated with a set of l keywords or tags $\{k_1, ..., k_l\}$ which summarize the contents of t_i. We denote the set of keywords associated with an item t_i as $Kw(t_i)$. A query Q in our system is a set of keywords, $Q = \{k_1, ..., k_p\}$. An item t *matches* query Q iff $Q \subseteq Kw(t)$. The answer to Q, denoted S_Q, is then the set of matching items in S, i.e., $S_Q = \{t \mid t \in S, Q \subseteq Kw(t)\}$.

As mentioned in the introduction, the number of items matching a query Q can often be huge, and merely returning the top-k matching items may not help the user find the items they are most interested in. So we want to group items into different expansions of Q and return high quality expansions. Let K denote the set of all keywords in the system. We call a subset of keywords $e \subseteq K - Q$ an *expansion*[1]

[1]Note, actually the real expansion is $Q \cup e$ but for technical convenience, we deal with the part of

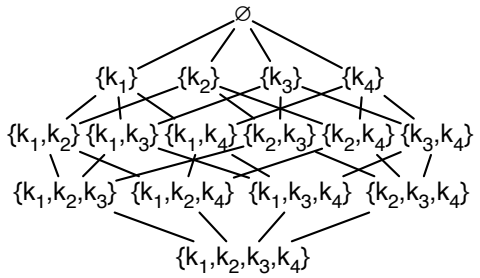

Figure 4.1: Lattice Structure of Expansions.

of Q. The size of an expansion e, $|e|$, is the number of keywords in e. Let the set of all possible expansions for Q be E_Q, then given an expansion $e \in E_Q$, we say an item $t \in S$ *matches* e if $Q \cup e \subseteq Kw(t)$. Let S_e denote the set of matching items for e, then we have $S_e = \{t \mid t \in S, Q \cup e \subseteq Kw(t)\}$.

Given a query Q, all possible expansions of Q can be organized as a lattice based on the *subset-of* relationship, E.g., Figure 4.1 shows the lattice structure of all possible expansions for $K - Q = \{k_1, ..., k_4\}$. We will shortly define the quality of an expansion, and we will show in Section 5.1.2 that this lattice structure can be used to improve the efficiency of our algorithm for determining high quality expansions.

Intuitively, the importance of an expansion e can be captured by aggregating the utilities of items which match e. Let g be a monotone aggregation function which calculates the overall utility or quality of expansion e, i.e., define $u(e) := g(\{u(t) \mid t \in S_e\})$. Then we can define our top-k expansion problem as follows.

Definition 1 (Top-k Expansions)**.** *Given a set S of items and a keyword query Q, find the top-k expansion set $E_k = \{e_1, ..., e_k\}$ s.t. $\forall e \in E_k$ and $\forall e' \in E_Q - E_k$, $u(e) \geq u(e')$.*

The intuition is that in response to a user query, we want to return the best quality expansions where the quality of an expansion is monotonically determined by the utility of the items matching it.

The set of annotations used in this paper is summarized in Table 4.1. Below, we give an example of an aggregation function with certain desirable properties.

the expansion outside Q.

Symbol	Description
t	an item in the system
S	a set of items
$t.a$	attribute a's value of an item t
w	the weight associated with an attribute
$u(t)$	the utility value of an item t
k	a keyword
K	the set of all possible keywords
$\mathscr{L}_K$	the lattice of all possible keywords K
$\mathscr{L}$	the partially materialized lattice
$Kw(t)$	the set of keywords associated with t
Q	a query which is composed of a set of keywords
S_Q	the set of matching items for query Q
e	an expansion
E_Q	the set of all possible expansions for query Q
S_e	the set of matching items for expansion e
$u(e)$	the utility value of an expansion e
V_{S_e}	the multiset of utility values of all items matching e
$\mathscr{L}_K(e)$	the sub-lattice induced by e
E_t	all possible expansions of t
e_t	the largest size expansion in E_t

Table 4.1: Annotations

4.1 Determining Importance of an Expansion

To determine the utility $u(e)$ of expansion e, a natural idea is to use a monotone function to aggregate utility values of all items which can match e. However, this approach means for every expansion, we may need to retrieve all items that are relevant to this expansion which can be prohibitive considering the huge size of matching items. Besides, low quality items matching e intuitively should not determine its importance. So we will only consider top-N matching items for determining importance of an expansion e, where N is a parameter that is tuned for each application.

Let $V_{S_e} = \{u(t) \mid t \in S_e\}$ be the multiset of utility values of all items matching e. We will apply a function top_N to V_{S_e} which retrieves the top-N highest utility values from V_{S_e}. Then to determine the importance of e, we will sum up all values in $top_N(V_{S_e})$, so $g(S_e) := \sum_{v \in top_N(V_{S_e})} v$. It is clear that $g(S_e)$ satisfies the desired properties for determining the importance of an expansion, as $g(S_e)$ will be large when there are high quality items matching e and also when the number of these high quality items is large.

It can be easily shown that the aggregation function g defined is *subset-monotone*, which means for two expansions e_1 and e_2, if $S_{e_1} \subseteq S_{e_2}$, then $g(e_1) \leq g(e_2)$. And it is worth noting that the algorithms proposed in this work can be easily adapted to other subset-monotone score functions.

On top of the basic top-k expansion problem, we will want to impose some additional desirable properties for the k expansions returned by our algorithms. In Section 5.2, we study these properties and propose additional algorithms which can return expansions satisfying these properties.

Chapter 5

Algorithms

5.1 Basic Algorithm

In this section, we present two algorithms for finding top-k expansions. In order to facilitate both algorithms, we assume m inverted lists are materialized, one each for the m attributes, where for each inverted list, items are sorted in the non-increasing order of their value on the corresponding attribute.

5.1.1 Naïve Algorithm

Inspired by the NRA algorithm proposed in [21], a naïve way of generating top-k expansions can be described as follows: 1. access items in the non-increasing order of their attribute value; 2. for each matching item accessed, enumerate all possible expansions and update their lower bound and upper bound utility value; 3. stop the iterative process once top-k expansions have been identified. The pseudo-code for the above process is given in algorithm 1.

In Algorithm 1, all attribute lists are accessed in a round-robin fashion (line 4), and for each matching item t obtained from list I_a, we will enumerate the set E_t of all possible expansions (line 5–7).

Consider an expansion $e \in E_t$, let SS_e be the current set of accessed items which can match e. Then for each $t \in SS_e$, because all attribute values are normalized to $[0,1]$ and items are accessed in the non-increasing order of their attribute values, similar to the NRA algorithm [21], we can determine the lower bound on t's utility

$\underline{u}(t)$ by summing up all current accessed attribute values of t, and we can determine its upper bound utility $\bar{u}(t)$ by summing up all accessed attribute values of t along with the last accessed values of other attributes in the inverted lists.

Then because of the subset-monotonicity of the score function g, the lower bound utility $\underline{u}(e)$ for expansion $e \in E_t$ can be calculated as a sum of all values in $top_{min(N,|SS_e|)}(\{\underline{u}(t) \mid t \in SS_e\})$. And the upper bound utility $\bar{u}(e)$ of an expansion e can be estimated by considering both SS_e and the maximum utility value which can be achieved by any unseen items. Let the last accessed values on each of the m attribute lists be $\bar{v}_1, ..., \bar{v}_m$ respectively, then because items are accessed in the non-increasing order of their attribute value, an "imaginary" item t' with m attribute values $\bar{v}_1, ..., \bar{v}_m$ must have the maximum value that can be achieved by any unseen items. So the upper bound utility of e can be estimated as the sum of all values in $top_N(\{\bar{u}(t) \mid t \in SS_e\} \cup \{u(t'_1), ..., u(t'_N)\})$. where $t'_1, ..., t'_N$ are N imaginary items which have the same utility as t'.

After the lower and upper bounds for all expansions in E_t have been updated (line 8–10), we can estimate the upper bound utility for all possible unseen items by summing up the values of N maximum possible imaginary items (updateUpperBound($\emptyset$) in line 12).

Henceforth, by the value of an expansion, we mean its utility value. Let EQ be the expansion query which contains the set of expansions of Q that have been materialized by the algorithm at a given point. We can rank all expansions in EQ by their lower bound values. Let EQ_k be the top-k expansions in EQ and UB' be the maximum upper bound value of an expansion in $EQ - EQ_k$, then the maximum of UB' and the upper bound value for all unseen items determines the overall upper bound value UB (line 11–12). If the lower bound value for the k^{th} expansion in EQ_k is already larger than UB, the algorithm can be safely terminated, exactly in the spirit of NRA.

The correctness of Algorithm 1 easily follows from the fact that both lower bound and upper bound of an expansion are correct. And to assess the performance of Algorithm 1, we borrow the notion of *instance optimality* as proposed by Fagin et al. in [21].

Definition 2. Instance Optimality: *Let $\mathscr{A}$ be a class of algorithms that make no*

Algorithm 1: TopExp-Naive(Q, I, g, k)

```
1  EQ ← expansion queue;
2  UB ← upper bound threshold;
3  while |EQ| < k OR u(EQ.k^th expansion) < UB do
4      I_a ← getNextListRR();
5      t ← I_a.getNextItem();
6      if Q ⊈ Kw(t) then continue;
7      E_t ← enumerateExpansion(Kw(t) − Q);
8      foreach e ∈ E_t do
9          if e ∉ EQ then EQ.push(e);
10         e.updateLower&UpperBound(t, g);
11     UB′ ← maximum upper bound value of an expansion in EQ − EQ_k;
12     UB ← MAX(UB′, updateUpperBound(∅));
```

random accesses to the m inverted lists, and let $\mathscr{I}$ be a class of problem instances. Given a non-negative cost measure $cost(A,I)$ of running algorithm $A \in \mathscr{A}$ over $I \in \mathscr{I}$, an algorithm $A \in \mathscr{A}$ is instance optimal *over $\mathscr{A}$ and $\mathscr{I}$ if for every $A' \in \mathscr{A}$ and every $I \in \mathscr{I}$ we have $cost(A,I) \leq c \cdot cost(A',I) + c'$, for constants c and c'. Constant c is called the optimality ratio.*

Let the cost of an algorithm for the top-k expansion problem be determined by the number of items accessed, we first show the following result.

Lemma 1. *Given any instance I of the top-k expansion problem and any algorithm A with the same access constraints as TopExp- Naive, assume A accesses x items, then TopExp-Naive will access at most $m \times x$ items.*

Proof. (Sketch) Assume another algorithm $B \in \mathscr{A}$ stops in list I_a after accessing x items, then it must be true that at the time when B stops, the current utility of the k^{th} maximum utility expansion e_k will have its utility larger than or equal to the upper bound utility of all generated non-result expansions and all possible unseen expansions. This is because otherwise, we can come up with an configuration of the remaining unseen items such that we can have an expansion of which the utility is larger than e_k. Then it is clear that our algorithm TopExp-Naive will also stop at the same position in list I_a. By considering the fact there are m lists in total,

we can infer that the total number of items accessed by TopExp-Naive is at most $m \times x$, for the scenario where B accesses x items in list I_a, zero item in other lists, and TopExp-Naive accesses x items in each list. □ □

Theorem 1. *Let $\mathscr{I}$ be the class of all top-k expansion problem instances, and $\mathscr{A}$ be the class of all possible algorithms that find the top-k expansions, that are constrained to access items sequentially in non-increasing order of their attribute values, then TopExp-Naive is instance optimal over $\mathscr{A}$ and $\mathscr{I}$ with an optimality ratio of m.*

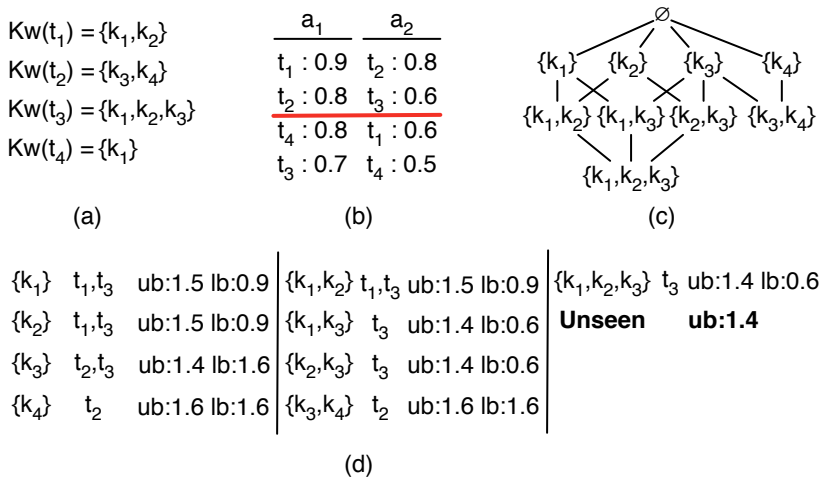

Figure 5.1: Example for TopExp-Naive.

Example 1. We show an example of algorithm TopExp-Naive in Figure 5.1. In this example, for simplicity of presentation, we assume all 4 items $t_1, ..., t_4$ can match the query Q, so Q will be ignored from the keyword list of all items. Furthermore, we assume $N = 1$ which means we are using the best item to determine the importance of each expansion, and $k = 1$ which means we are looking for top-1 expansion. Keywords associated with each item are shown in Figure 5.1 (a), and each item in the example has two attributes a_1 and a_2. The inverted lists for these two attributes are shown in Figure 5.1 (b).

As described in TopExp-Naive, the algorithm will enumerate all possible expansions for each item accessed, e.g., after t_1 is accessed, $\{k_1\}$, $\{k_2\}$ and $\{k_1, k_2\}$

will be generated, and their utility bound values will be updated using the attribute value of t_1. For this case, because only $t_1.a_1$ is known, we know $\bar{u}(t_1) = 1.9$ and $\underline{u}(t_1) = 0.9$, these will be the upper and lower bound utility value for all three expansions. After we have accessed the first two items of both lists, the expansions generated are shown in Figure 5.1 (c), while the items contained in each generated expansion and the utility bound values for each generated expansion are shown in Figure 5.1 (d). It is clear that at this moment, the upper bound utility for all generated expansions is 1.6, the upper bound utility for all unseen expansions is 1.4 (sum of the last accessed utility value from each list) and the lower bound utility for expansions $\{k_3\}$, $\{k_4\}$ and $\{k_3, k_4\}$ are all 1.6, so we can stop the algorithm now and return any one of these three expansions. □

5.1.2 Improved Algorithm

One serious drawback of the naïve algorithm is that every time an item t containing keywords $Kw(t)$ is accessed, all $2^{|Kw(t)-Q|}$ possible expansions for this item are explicitly enumerated and their bounds are maintained. In this section, we propose an efficient algorithm which can leverage the lattice structure of expansions to avoid enumerating and maintaining unnecessary expansions. We also address the challenge of determining the bounds of unseen (unmaterialized) expansions, which is necessary for early termination of the top-k algorithm.

Avoiding Unnecessary Expansions

Given a query Q and a newly accessed item t, let $e_t = Kw(t) - Q$ be the largest size expansion in E_t, then the naïve algorithm will enumerate all possible expansions of t by considering all non-empty subsets of e_t. However, this may not be necessary. E.g., let $K_{<t}$ be the set of keywords which have been seen before t; if $\forall k \in K_{<t}$, $k \notin e_t$, we just need to maintain one single expansion e_t, as all other expansions generated from e_t will have the same current matching itemset as e_t and thus the same lower and upper bounds as e_t. This indicates that there are opportunities to avoid the expansion enumeration process for each newly accessed item.

Let $\mathscr{L}_K$ be the lattice of all possible keywords K. For an expansion $e \in \mathscr{L}_K$, we let $\mathscr{L}_K(e)$ denote the sub-lattice *induced* by e, i.e., $\mathscr{L}_K(e) = \{e' \mid e' \subseteq e\}$. For

a newly accessed item t, the naïve algorithm will enumerate all expansions $e \in \mathscr{L}_K(e_t)$, however, as discussed above that this isn't always necessary.

The idea of the new algorithm can be described as follows. Let $\mathscr{L}$ denote a partially materialized lattice which contains the set of expansions generated so far before the current item t. If $\exists e \in \mathscr{L}$ s.t. $e = e_t$, then it is clear we just need to update the lower bound and upper bound utilities of all existing expansions in $\mathscr{L}$ which are subsets of e_t. Otherwise, all expansions in $\mathscr{L}$ will correspond to different sets of items compared with e_t, so we need to first generate the expansion e_t and update its lower bound and upper bound utility. Then we consider the following two cases: 1. if $\forall e \in \mathscr{L}$, $e \cap e_t = \emptyset$, then all $e' \in \mathscr{L}_K(e_t)$ correspond to the same set of items, their lower bound and upper bound utilities are the same, and we just need to maintain one expansion e_t which can concisely represent all expansions in $\mathscr{L}_K(e_t)$; 2. On the other hand, if there exists an expansion $e \in \mathscr{L}$ s.t. $e \cap e_t \neq \emptyset$, then *for each such expansion* e, we need to further consider the following three sub-cases:

1. if $e \subseteq e_t$, we don't need to generate additional expansions, but we need to update the lower bound and upper bound utility of e since item t also contributes to every sub-expansion e of e_t.
2. if $e_t \subseteq e$, similar to case 1, we don't need to generate additional expansions; and since t cannot contribute to e, we don't need to update utility bounds of e.
3. if $e \nsubseteq e_t$ or $e_t \nsubseteq e$, let $e' = e \cap e_t$, then it is clear that $S_{e'} \neq S_e$ and $S_{e'} \neq S_{e_t}$, which means we need to generate a new expansion e' and update its utility bounds accordingly.

Bounds for Unseen Expansions

Since we don't explicitly maintain all possible expansions for each item accessed, this will create a challenge for determining lower bound and upper bound utilities for all possible expansions. For all expansions which are maintained in $\mathscr{L}$, the lower bound and upper bound are determined as discussed in Section 5.1.1. For

Algorithm 2: TopExp-Lazy(Q, I, g, k)

```
1  UB ← upper bound threshold;
2  𝓛 ← partial materialized lattice structure;
3  while |EQ| < k OR u(EQ.k^th expansion) < UB do
4      I_a ← getNextListRR();
5      t ← I_a.getNextItem();
6      if Q ⊈ Kw(t) then continue;
7      e_t ← Kw(t) − Q;
8      if e_t ∈ 𝓛 then
9          foreach e ∈ {e | e ∈ 𝓛 ∧ e ⊆ e_t} do
10             e.updateLower&UpperBound(t);
11     else
12         TQ ← temporary update expansion queue;
13         TQ.push(e_t);
14         while ¬ TQ.empty() do
15             e = TQ.pop();
16             E_l ← {e' | e' ∈ 𝓛 ∧ (∄e'' ∈ 𝓛 : e' ⊂ e'')};
17             foreach e_l ∈ E_l ∧ e_l ∩ e ≠ ∅ do
18                 E ← {e' | e' ∈ 𝓛 ∧ e' ⊆ e_l};
19                 if ∃e' ∈ E s.t. e' ⊈ e ∧ e ⊈ e' then
20                     Find all e' ∈ E s.t. e' ⊈ e ∧ e ⊈ e' and (∄e'' ∈ E :
                       e' ⊂ e'' ∧ e'' ⊈ e ∧ e ⊈ e'');
21                     TQ.push(e ∩ e');
22             if e ∉ 𝓛 then 𝓛.add(e);
23         foreach e ∈ 𝓛 ∧ e ⊆ e_t do
24             e.updateLower&UpperBound(t);
25     UB ← upper bound value of the (k+1)^th expansion in 𝓛;
26     UB ← MAX(UB, getUpperBound(∅));
```

each remaining expansion e not materialized in $\mathscr{L}$, depending on the position of e in the lattice $\mathscr{L}_{\mathscr{K}}$, we need to consider the following two cases:

- $\nexists e' \in \mathscr{L}$ s.t. $e \subset e'$. This means we haven't accessed any item which corresponds to this expansion. The utility upper bound of e in this case is the same as the maximum possible utility of an unseen expansion, as discussed

in Section 5.1.1.

- $\exists e' \in \mathscr{L}$ s.t. $e \subset e'$. This means we have already accessed some items which correspond to this expansion. Then we must be able to find a smallest such expansion $\hat{e} \in \mathscr{L}$ s.t. $e \subset \hat{e}$, and $\forall e' \in \mathscr{L}$, if $e \subset e'$, then $\hat{e} \subseteq e'$. To see this, suppose $e', e'' \in \mathscr{L}$ are distinct expansions such that $e \subset e'$ and $e \subset e''$. From Section 5.1.2, it follows that for $e''' = e' \cap e''$, we have $S_{e'''} \neq S_{e'}$ and $S_{e'''} \neq S_{e''}$, so e''' should be generated as a new expansion before e, and this expansion has the property that it is a subset of both e' and e'', and it is a superset of e. It follows that the smallest superset expansion $\hat{e}$ must exist in $\mathscr{L}$. So after $\hat{e}$ is found, we know e and $\hat{e}$ currently correspond to the same set of items, then e's utility bound will be the same as $\hat{e}$. This means that we don't need to explicitly consider this expansion when using the utility bounds to determine whether the algorithm can stop.

Example 2. Consider the lattice in Figure 4.1 and assume we have only materialized two expansions $e_1 = \{k_1\}$ and $e_2 = \{k_1, k_2, k_3\}$. Then for an expansion $e_3 = \{k_2, k_4\}$, because there is no such materialized expansion $e' \in \mathscr{L}$ s.t. $e_3 \subset e'$, then we know we haven't accessed any item which corresponds to this expansion. So we only need to consider its upper bound utility, which is the maximum possible utility for all possible expansions. And for another un-materialized expansion $e_4 = \{k_1, k_2\}$, we can find out that $e_4 \subset e_2$, so e_4 and e_2 correspond to the same set of items, and e_4's utility bounds are the same as those of e_2. □

So the general idea of our *lazy expansion generation* based algorithm is that we only need to maintain expansions which correspond to a *unique* set of items. For a set of expansions which are matched by the same set of items, we can simply represent them using the largest expansion in the set.

Lazy Expansion Algorithm

The pseudo-code for the lazy expansion generation based algorithm is given in Algorithm 2. Similar to TopExp-Naive, TopExp-Lazy iteratively retrieves items from the attribute lists (line 4–6). However, unlike TopExp-Naive, *we maintain only necessary expansions in the partially materialized lattice* $\mathscr{L}$. For a newly

accessed item t, if e_t has already been generated by a previous item, we simply update the itemset and lower/upper utility bounds of the corresponding expansions which contain this item (line 8–10). Otherwise, as discussed above, we may need to generate some additional expansions which correspond to a unique matching itemset (line 11–24). The procedure works as follows. First, we identify from $\mathscr{L}$ all *leaf expansions* E_l which are maximal expansions, i.e., they are not included in other expansions in $\mathscr{L}$ (line 16). Then for each leaf expansion $e_l \in E_l$, if e_t doesn't overlap with e_l, we can ignore all expansions which are subsets of e_l as they can't overlap with e_t either (line 17). If e_l overlaps with e_t, then we search the set E of expansions in $\mathscr{L}$ which are subsets of e_l: if there is an expansion $e' \in E$ s.t. $e' \not\subseteq e_t$ and $e_t \not\subseteq e'$, then we need to find all of the largest such expansions e' in E, and as described in Section 5.1.2, for each of them, a new expansion $e' \cap e_t$ needs to be inserted to $\mathscr{L}$ as it corresponds to a unique set of matching items, so we will recursively insert this new expansion into $\mathscr{L}$ (line 19–21).

The procedure for updating lower bound and upper bound utilities for each expansion is very similar to TopExp-Naive. However, because in TopExp-Lazy each expansion may represent more than one expansion, in order to determine which expansions in the expansion buffer $\mathscr{L}$ are current top-k expansions, we need to calculate for each expansion $e \in \mathscr{L}$ the exact number of ungenerated expansions which have the same utility bounds as e. The pseudo-code for this procedure are listed in Algorithm 3 and Algorithm **??**.

The idea of Algorithm 3 is that we first find from $\mathscr{L}$ the expansion set ST_e of which the expansions are subset of e (line 1), we prune away those expansions in ST_e of which a superset is also present in ST_e (line 2–3), then for the pruned expansion set ST_e, we can use the classical *inclusion-exclusion principle* to count the total number of expansions covered by ST_e (line 4). Algorithm **??** is a simple implementation of the counting procedure.

Example 3. Figure 5.2 shows an example that illustrates how Algorithm TopExp-Lazy works. The configuration of this example, including query, item attributes, item attribute values, keywords of each item and parameters k, N, is the same as Example 1. However, because we are using the lazy expansion generation based algorithm, we don't need to enumerate all possible expansions for each item ac-

Algorithm 3: updateCount($\mathscr{L}$, e)

```
1  ST_e ← {e' | e' ∈ L ∧ e' ⊂ e};
2  ST'_e ← {e' | e' ∈ ST_e ∧ ∃e'' ∈ E → e' ⊂ e''};
3  ST_e = ST_e − ST'_e;
4  count ← countGeneratedExpansions(ST_e);
5  e.count ← 2^|e| − count − 1;
```

Algorithm 4: countGeneratedExpansions(ST_e)

```
1  count ← 0;
2  for outeridx from 2 to |ST_e| do
3      ST'_e = ∅;
4      for inneridx from 1 to outeridx - 1 do
5          ST'_e.insert(ST_e[outeridx] ∩ ST_e[inneridx]);
6      ST''_e ← {e' | e' ∈ ST'_e ∧ ∃e'' ∈ E → e' ⊂ e''};
7      ST'_e = ST'_e − ST''_e;
8      if ST'_e.hasOverlap() then
9          count += countGeneratedExpansions(ST'_e);
10     else
11         foreach e' ∈ ST'_e do
12             count += 2^|e'|-1;
```

cessed. E.g., when the first item t_1 is accessed, we only need to generate expansion $\{k_1, k_2\}$ and don't need to generate expansions $\{k_1\}$ and $\{k_2\}$, as they correspond to the same current set of matching items as $\{k_1, k_2\}$. The utility bound values for $\{k_1, k_2\}$ will be the same as in the TopExp-Naive algorithm, and again we don't need to maintain these utility bound values for $\{k_1\}$ and $\{k_2\}$ since they are the same as for $\{k_1, k_2\}$. After accessing two items from each lists, the expansions materialized for TopExp-Lazy are shown as bolded expansions in Figure 5.2 (c). Compared with TopExp-Naive, it's worth noting that 5 expansions don't need to be maintained. At this point, similarly to TopExp-Naive, the algorithm can also stop as the top expansion $\{t_3, t_4\}$'s lower bound utility is already larger than or equal to the maximum upper bound utility for all expansions.

Note that for the lazy expansion generation based algorithm, for each expan-

sion which needs to materialized, we need to use Algorithm 3 to count how many expansions correspond to the same set of items. E.g., after accessing t_3 in the inverted list of a_2, we need to consider how many expansions are "covered by" the current expansion $\{k_1, k_2, k_3\}$. Algorithm 3 will first find all materialized expansions which are subsets of $\{k_1, k_2, k_3\}$, for this case, $\{k_1, k_2\}$ and $\{k_3\}$. Then as in line 4 of Algorithm 3, Algorithm **??** will be called to enumerate the number of non-empty expansions covered by these two expansions. Because there is no overlap between $\{k_1, k_2\}$ and $\{k_3\}$, in line 10–12 of Algorithm **??**, we can simply sum up the number of non-empty expansions covered by these two expansions, which is 4. Then this number will be used to determine the number of non-empty expansions covered by $\{k_1, k_2, k_3\}$ in line 5 of Algorithm 3, which is 3 ($\{k_1, k_2, k_3\}$, $\{k_1, k_3\}$ and $\{k_2, k_3\}$).

Figure 5.2 (d) shows all the expansions that should be considered and whether they are "covered by" some other expansions in the lattice. Compared with Figure 5.1 (d), it is clear that the lazy expansion generation based algorithm will maintain just one expansion for each set of expansions which correspond to the same set of items. □

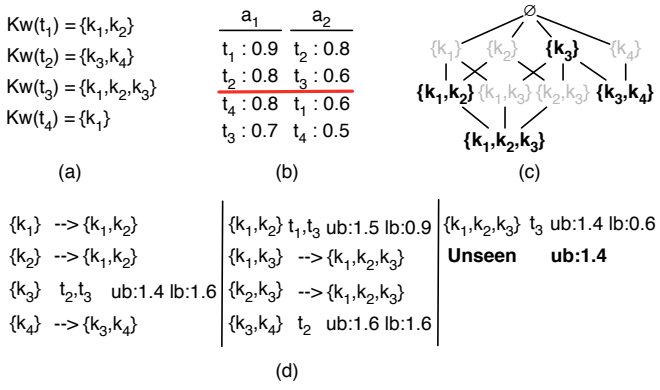

Figure 5.2: Example for TopExp-Lazy.

5.2 Semantic Optimization

Though algorithms described in Section 5.1 can correctly find expansions which have the k highest utility, there are two kinds of issues with these algorithms. First,

the basic algorithm will favor small expansions (i.e., fewer keywords) as these expansions have more matching items than larger expansions. Second, in the returned top-k expansions, it may happen that two expansions have the subset-of relationship, which is not ideal. Indeed, we would like the resulting expansions to have little overlap with each other.

In this section, we propose two solutions to remedy the above drawbacks. In Section 5.2.1, we will study weighting schemes which can penalize expansions that are either too small or too large. Then in Section 5.2.2, we propose to find the k most interesting expansions which don't have overlap with each other.

5.2.1 Weighting Expansions

It is clear that expansions which have small size (e.g., "XML") correspond to "general topics" which are related to the query, whereas expansions which have large size (e.g., "XML, schema, conformance, automata") correspond to "specific topics" which are related to the query. To help users quickly locate interesting information from the returned results, intuitively neither too general topics nor too specific topics should be returned early, so we want to favor those expansions which are neither too large nor too small.

We assume there is a function $f_w : \mathbb{N} \to \mathbb{R}$ (where $\mathbb{N}/\mathbb{R}$ denote the sets of natural/real numbers) which can return the weight $f_w(p)$ for an expansion of size p. The intuition is that f_w penalizes too small and too large expansions for we expect them to be intuitively too general or too specific. Then we can use this function to weight the utility of all expansions under consideration. In this work, we consider the Gaussian function $f_w(p) = e^{-\frac{(x-\mu)^2}{2\sigma^2}}$. The mean μ of f_w is set as the most ideal size of an expansion and the variance σ can be adjusted by the system for different problem instances.

So how are the top-k algorithms impacted? For TopExp-Naive, the weighting function can be simply applied to the utility bounds of each expansion enumerated, and other parts of the algorithm won't be affected. However, for TopExp-Lazy, for each materialized expansion $e \in \mathscr{L}$, the unmaterialized expansions which have the same set of matching items as e have the same utility bounds as e when no weighting is applied. But once weighting is applied, these expansions may have

different utility bounds depending on the size. E.g., consider that there is only one materialized expansion $e = \{k_1, k_2\}$ in $\mathscr{L}$ and let the lower and upper utility bounds of e be $\underline{u}(e) \times f_w(2)$ and $\bar{u}(e) \times f_w(2)$ respectively. Then for expansions $\{k_1\}$ and $\{k_2\}$, though they correspond to the same set of matching items as e, their lower and upper utility bounds are $\underline{u}(e) \times f_w(1)$ and $\bar{u}(e) \times f_w(1)$ respectively.

So for the weighted lazy expansion generation based algorithm, for a set E of expansions which correspond to the same set of matching items, we may need to maintain multiple expansions where the number of expansions to be maintained depends on the size of the largest expansion in E. For an expansion e, Algorithm 5 and Algorithm **??**, which are adapted from Algorithm 3 and Algorithm **??**, can be used to count for each possible expansion size, the number of expansions which correspond to the same set of matching items as e. These counts can be utilized along with the weighting function to determine the corresponding lower and upper utility bounds.

Algorithm 5: updateCount(λ, e)

1 $ST_e \leftarrow \{e' \mid e' \in L \wedge e' \subset e\}$;
2 $ST'_e \leftarrow \{e' \mid e' \in ST_e \wedge \exists e'' \in E \rightarrow e' \subset e''\}$;
3 $ST_e = ST_e - ST'_e$;
4 $vec_c \leftarrow$ a new count vector of size $|e|$;
5 **for** *expsize from 1 to* $|e|$ **do**
6 $\quad vec_c[expsize] = \binom{|e|}{expsize}$;
7 $vec_c \leftarrow$ countGeneratedExpansions(ST_e, vec_c);
8 e.count = vec_c;

5.2.2 Path Exclusion based Algorithm

To lessen the semantic overlap between different expansions returned to the user, intuitively we may not want to return two different expansions e_1 and e_2, such that either $e_1 \subset e_2$ or $e_2 \subset e_1$. Note that there can be many sets of expansions satisfying this pairwize comparability, and the set of highest utility expansions may not satisfy this property. So in order to guarantee the quality of the expansions returned, we want to maximize the sum of the utilities for the set of expansions returned under

Algorithm 6: calculateSameLBExpansionVec(ST_e, vec_c)

```
1  for outeridx from 2 to |ST_e| do
2      ST'_e = ∅;
3      for inneridx from 1 to outeridx - 1 do
4          ST'_e.insert(ST_e[outeridx] ∩ ST_e[inneridx]);
5      ST''_e ← {e' | e' ∈ ST'_e ∧ ∃e'' ∈ E → e' ⊂ e''};
6      ST'_e = ST'_e − ST''_e;
7      if ST'_e.hasOverlap() then
8          countGeneratedExpansions(ST'_e, vec_c);
9      else
10         for expsize from 1 to |e| do
11             foreach e' ∈ ST'_e do
12                 vec_c[expsize] = vec_c[expsize] - (|e'| choose expsize);
```

the constraint that the expansions returned should satisfy the pairwise comparability.

Definition 3. *(Maximum k Path-Exclusive Expansion) Given a set S of items and a keyword query Q, find the top k-expansion set* $E_k = \{e_1, ..., e_k\}$ *s.t.* $\forall e_i, e_j \in E_k$, $i \neq j$, $e_i \not\subset e_j$, $e_j \not\subset e_i$, *and* $\sum_{e \in E_k} u(e)$ *is maximized.*

Let $\mathscr{L} = \{e_1, ..., e_n\}$ be the set of all expansions materialized by the algorithm. Consider a weighted undirected graph $G = (Vtx, Edg)$, with nodes $Vtx = \mathscr{L}$ where each node e_i is associated with a weight $u(e_i)$, i.e., the utility of e_i. Whenever two expansions e_i, $e_j \in Vtx$ are such that either $e_1 \subset e_2$ or $e_2 \subset e_1$, Edg contains the edge (e_i, e_j). Then it is straightforward to show that the maximum k path-exclusive expansion problem is NP-hard by a direct reduction from the *maximum weighted independent set* problem [30].

A simple greedy algorithm for the maximum weighted independent set was proposed by [30]: repeatedly select a node in G with minimum weighted degree in each iteration and add it to the current solution; then delete this node and all of its neighbors from the graph; stop when all nodes are removed from G. It has been proven in [30] that this algorithm gives a $\max(\delta_w, 1)$-approximation, where $\delta_w = \max_{H \subseteq G} \min_{v \in V(H)} d_w(v, H)$, $d_w(v, H)$ is defined as $\frac{ww(N_G(v))}{ww(\{v\})}$, $ww(S)$ denotes

the sum of weights of a set of nodes S, and $N_G(v)$ is the set of neighbors of v in G.

Furthermore, if we rank all generated expansions by their upper bound utility, because items are accessed in the non-increasing order of their attribute values, it is clear that the sum of the top-k expansions' upper bound utilities is an upper bound for the value of all possible k path-exclusive expansions.

So based on this information, we propose the following algorithm called Top-PEkExp, which can be used to calculate an approximate solution for the maximum k path-exclusive expansion problem. In Top-PEkExp, similar to the previous algorithms, we iteratively retrieve items from the attribute lists (line 3–5), then we use TopExp-Lazy to generate necessary expansions in $\mathscr{L}$ (line 6). The set of expansions in $\mathscr{L}$ are sent to the greedy algorithm for the maximum weighted independent set problem (line 7), and if the result is already larger than $\frac{1}{\alpha}$ of the maximum upper bound utility, for some constant $\alpha > 1$ that is chosen by the system, we can stop the algorithm (line 8–11).

Algorithm 7: Top-PEkExp(Q, I, g, k)

```
1  L ← partial materialized lattice structure;
2  while true do
3      I_a ← getNextListRR();
4      t ← I_a.getNextItem();
5      if Q ⊈ Kw(t) then continue;
6      Generate necessary expansions using TopExp-Lazy;
7      R^G ← GreedyMWIS(L);
8      E_topk ← k expansions in L which have the largest upper bound utilities;
9      U* = sum_{e∈E_topk} ū(e);
10     if u(R^G) ≥ 1/α × U* then
11         return
```

It is clear that Algorithm Top-PEkExp can correctly return an α approximate answer for the maximum k path exclusive expansion problem. However, because we are using a greedy algorithm for calculating the k path exclusive expansions in each iteration, there may exist a better algorithm, e.g., which utilizes an exact algorithm, which can find an α approximate answer much earlier compared with our Top-PEkExp algorithm. This would trade more work done per iteration for

achieving early termination. We leave a detailed study of optimal algorithms for the maximum k path exclusive expansion problem for future work.

Chapter 6

Experiment

In this section we will discuss the experiments we have been done to evaluate the performance of our proposed algorithms. We first use synthetic datasets to demonstrate the relative efficiency, scalability, and memory savings of the proposed algorithms with respect to the naive algorithm. After that we use a Del.ici.ous dataset to further evaluate the scalability of the algorithm for real datasets. We also use a partial crawled dump of the ACM digital library to demonstrate the quality of the expansions returned by our algorithms.

Section 6.1 introduces the goals of the experiments conducted, and a description of each of the datasets used. In Section 6.2, the major result of efficiency study on the synthetic datasets are represented and analyzed. Section 6.3 represents scalability evaluation results on Del.icio.us dataset to demonstrate the performance of the algorithms for a large scale real dataset. In addition, Section 6.4 discusses about quality of expansions we generate in our partially crawled ACM dataset.

6.1 Experiment Setup and Data Sets

The goal of our experiments is two-fold: (i) Evaluate the efficiency and scalability of the algorithms proposed in this paper. (ii) Evaluate the quality of the expansions discovered by these various algorithms. The experiments are done on a Xeon 2.93GHz X5570 Linux machine with 60GB RAM. All algorithms are implemented in Java using JDK/JRE 1.6.

We use two kinds of datasets in our experiments. First, we generate synthetic datasets to compare the performance of various algorithms with the naive algorithm. The metrics we use for the comparison include the running time, number of items accessed and number of expansions generated during the process. Second, we use two real datasets to study the real time scalability, and quality of the expansions generated. Details of the construction of each dataset is listed below:

Synthetic Datasets: We generated 5 synthetic datasets with size from 8000 to 12000, and for all these datasets, attributes and keyword values are sampled from a power law distribution $x = e^{-\beta}$ with $\beta = 2$, which is in accordance with our observation in the real datasets.

Del.ici.ous: We took use of a tagging log dataset from Del.ici.ous [10]. Bookmarks and tags were extracted from the tagging log. Each bookmark is referred to as an item we described in our framework, and the tags used to tag this bookmark are viewed as corresponding keywords. For each bookmark, we took the number of users tagged this bookmark and the number of tags given to it as two quality attributes for the bookmarks, both of which are normalized into the range $[0,1]$. We measured a tag's relevance to the bookmark by its BM25 [35] score, and took the tags with top 15 BM25 scores as the topical keywords for that bookmark. The reason for this is not all the keywords for an item represent its major content, and tags with low BM25 score usually are not representative of its topic, thus its useful to eliminate the impact of these unimportant keywords to save unnecessary computation time.

ACM Digital Library: Our third dataset is a partially crawled dump of the ACM Digital Library. We obtained the items by combining result paper lists of 4 queries: (1) "histogram" (2) "privacy" (3)"xml". We chose these queries because they are all representative interesting research fields in the database community and also feature a good number of publications. The quality attributes we use for each paper are the average author publication number and the citation count.

Similarly with in Del.icio.us, values of both attributes are normalized into the range $[0,1]$. For each paper, we extract its keywords from the title, keywords list and abstract. Stop words are removed and also stemming is done on all the keywords obtained. For each paper under consideration, we select the top 15 BM25 scored [35] keywords as its topic keywords. A manually created mapping table is used to map similar keywords into a common keyword, namely the most frequent among them. Note that some recent work on tag clustering/recommendation [38] can be leveraged to automate the creation of the mapping table.After preprocessing, there are in total 48656 papers in the dataset, and 9000 distinct topic keywords.

6.2 Efficiency Study with Synthetic Dataset

First, the efficiency comparison of various algorithms on the synthetic datasets with fixed N and k is presented in Figure 6.1 A ($N = k = 10$). Note that for the path exclusive algorithm, we choose $\alpha = 0.1$ and $\alpha = 0.3$ as two settings for the algorithm. The running time, the number of items accessed and the number of expansions generated during execution of these algorithms are presented in Figure 6.1 A, B, and C, respectively.

For running time, both TopExp-Lazy and TopExp-LazyW (Weighted TopExp-Lazy) run much faster than the baseline TopExp-Naive algorithm. And the running time of PekExp based algorithms highly depend on the α parameter chosen. When α is small, the PekExp based algorithm tend to stop faster than TopExp-Lazy algorithm. However, when α is large, the algorithm could run relatively slow. The reason is because we don't have direct access to the exact optimal score, instead we are using the top upper bounds to guarantee the bound we get is larger than the exact optimal score, and also using lower bounds of the candidate expansions to guarantee the exact score is within the approximation factor α. Using these bounds may postpone the stop of the algorithm, but comparing to calculating the exact utility of each expansion and follow the greedy algorithm introduced in Section 2.2, the running time is still much faster.

Figure 6.1.B shows the number of items accessed for these datasets with grow-

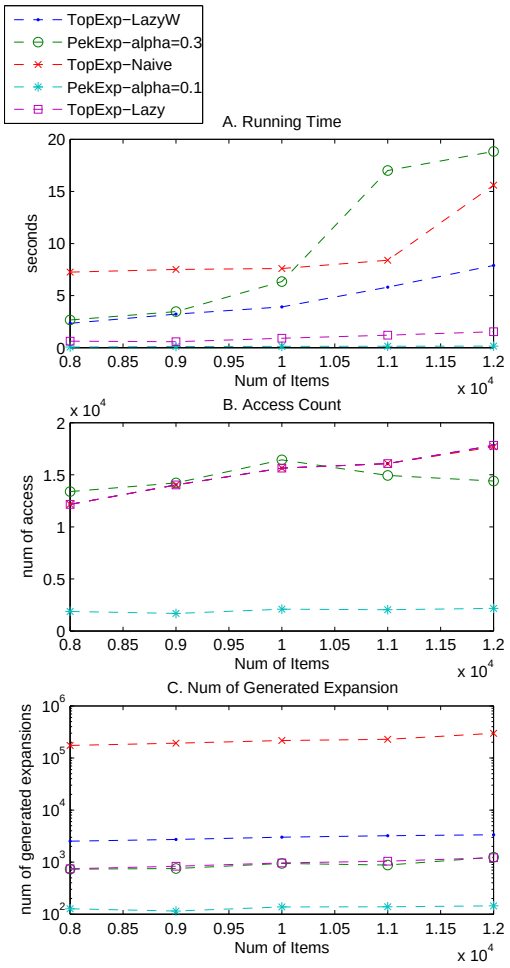

Figure 6.1: Efficiency Experiment with Growing Dataset Size

ing size. TopExp-LazyW, TopExp-Lazy and TopExp-Naive access roughly the same number of items. This coincides with our design of the algorithm, because the lazy lattice based algorithms (TopExp-LazyW and TopExp-Lazy) only avoid generating unnecessary expansions, but keep exactly the same information with TopExp-Naive algorithm. And for PekExp based algorithms, the number of items accessed may vary depending on the parameter α. With relatively small α, PekExp accesses significantly less items compared with other algorithms, hence much more efficient. Through our observation, for small α, PekExp usually stops very

soon, and as we will illustrate in the next section, the quality of the expansions returned by PekExp based algorithms are also comparable compared with other algorithms. If the application needs immediate response in most cases and tolerable for occasion failure, PekExp would be a reasonable choice from the aspect of efficiency.

For the number of expansions generated, as shown in Figure 6.1.C, while accessing the same number of items, both TopExp-LazyW and TopExp-Lazy generate much less expansions than TopExp-Naive does. This could significantly lessen the space required and the related computations of generating and updating expansions. For TopExp-LazyW and TopExp-Lazy, the weighted algorithm takes more time and generates more expansions. This is because the weighted algorithm needs to keep different bounds for expansions which correspond to the same set of seen items but with different number of keywords. However, the space and time cost is still reasonable considering the flexibility we could achieve by customizing the weights.

In Figure 6.2, we show efficiency comparison of the algorithms with k (number of top expansions to return) ranging from 10 to 15(number of items=10000, $N = 10$). As shown in Figure 6.2 A, k slightly affects the running time of the algorithm, in the trend of first increasing then dropping, both in a mild sense. With larger k, more information needs to be kept for the top expansions, so the updating cost is higher. Also, the top-k lowerbound is smaller in every corresponding step compared with settings with a smaller k. But on the other hand, the highest upperbound of expansions outside the current top-k will be smaller in every corresponding step. These three factors affects the running time interatively, thus there is no fixed trend of growing or dropping of the running time. The practical performance depends on the characteristic of the dataset and the top expansions, which coincides with our observation. The number of items accessed and number of expansions generated follows the same observation, as shown in Figure 6.2 B and C.

The only exception to the previous observations is PekExp-alpha= 0.3, for this setting, the number of items accessed and the running time decrease as k increases. This is because that as k grows, the upper bound of the top-k expansions will drop much faster as more low utility value expansions are included.

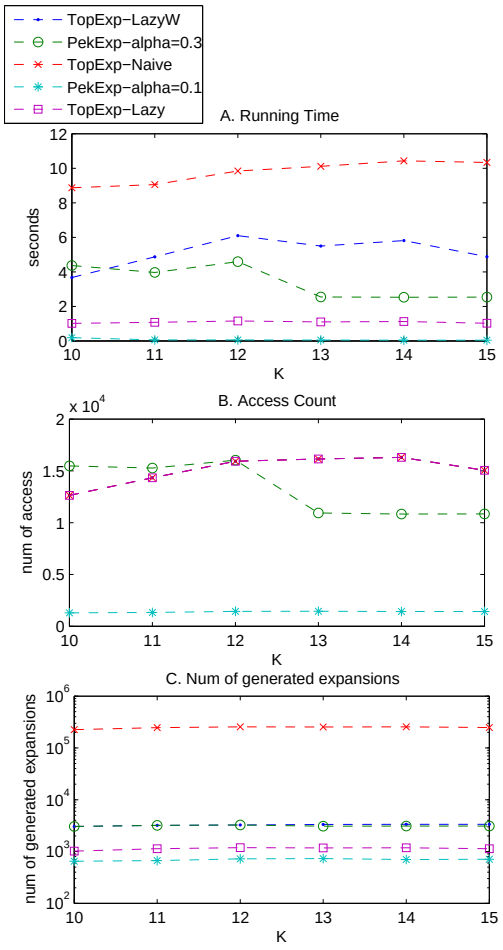

Figure 6.2: Efficiency Experiment with Growing k

Similarly, in Figure 6.3 A - C, we compare the algorithms with N (number of top item utilities to aggregate for expansion utility) ranging from 10 to 15(number of items=10000, $k = 10$). As N grows, the running time, number of items accessed and number of expansions generated all increase. The reason for this is with larger N, we need to keep track of more items to determine the bounds for each expansion, thus we could expect longer running time, more items to be accessed and more expansions to be generated. From another aspect, though the running time increase as N grows, the scalability is good as the lines are linear with a smooth slope. This

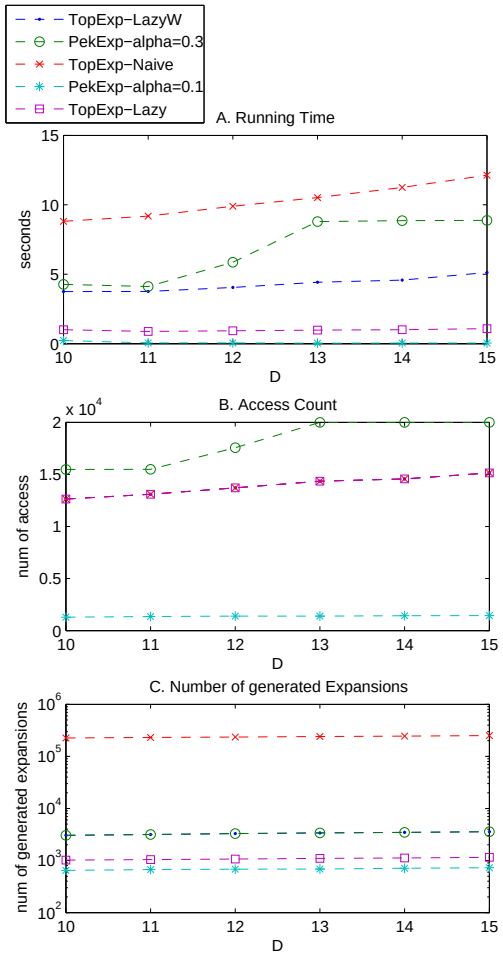

Figure 6.3: Efficiency Experiment with Growing N

is important as N is a tuned parameter, so we will want the algorithm to scale well with different N picked.

We can conclude from these plots that the performance of our proposed algorithms are very robust with respect to different settings, and the performance of the algorithms grows linearly with respect to the size of the dataset, so these algorithms can easily scale to larger datasets. Furthermore, it is clear that the TopExp-Lazy algorithms (weighted and unweighted) always outperform the TopExp-Naive algorithm. This is because the running time largely depends on the number of

expansions generated by the algorithm, which is avoided when necessary in the TopExp-Lazy algorithms. PekExp based algorithms is an exception to these observations. With proper configuration it could give good performance, but generally the performance varies depending on the value selected for the parameter α.

6.3 Scalability Study with Del.icio.us

To further demonstrate the scalability of our proposed algorithms on large scale real datasets. We use Del.icio.us as the experiment dataset, and incrementally sampled 8 itemsets with size ranging from 20,0000 to 160,0000 from the bookmarks we reconstructed from the tagging log between 2003.09 and 2005.07 on Del.icio.us. For each itemset, 30 different query keywords are randomly picked from the top 20% most frequent keywords in the dataset. And the average running time of issuing each of these keywords as the query is shown in Figure 6.4.

TopExp-Naive is not shown in Figure 6.4 because in general the running time is not on the same scale with our proposed algorithms and hard to show in the same graph. As we could see in the graph, the running time of all the algorithms we propose increase as the size of the dataset grows. The running time is growing in a linear scale, which demonstrates the scalability of the proposed algorithms. As we could expect there are randomness in the result shown, because of the sampling and runtime environment differences. Nevertheless, in general the TopExp-Lazy algorithm runs faster than TopExp-LazyW, which is in accordance with our previous observation in Section 6.2.

On the other hand, as we picked queries from the most frequent keywords as we're interested in worst case performance, we could expect the average running time on these datasets could be shorter than what we show. However, the running time we achieve is still far from enough compared with commercial search engines currently available online, even consider it is the worst case.

6.4 Quality of the Generated Expansions

The quality of the expansions generated can be measured by the quality of the keywords of each top expansion. Our expectation is the top keywords returned should to some extent match with important aspects of the research field implied

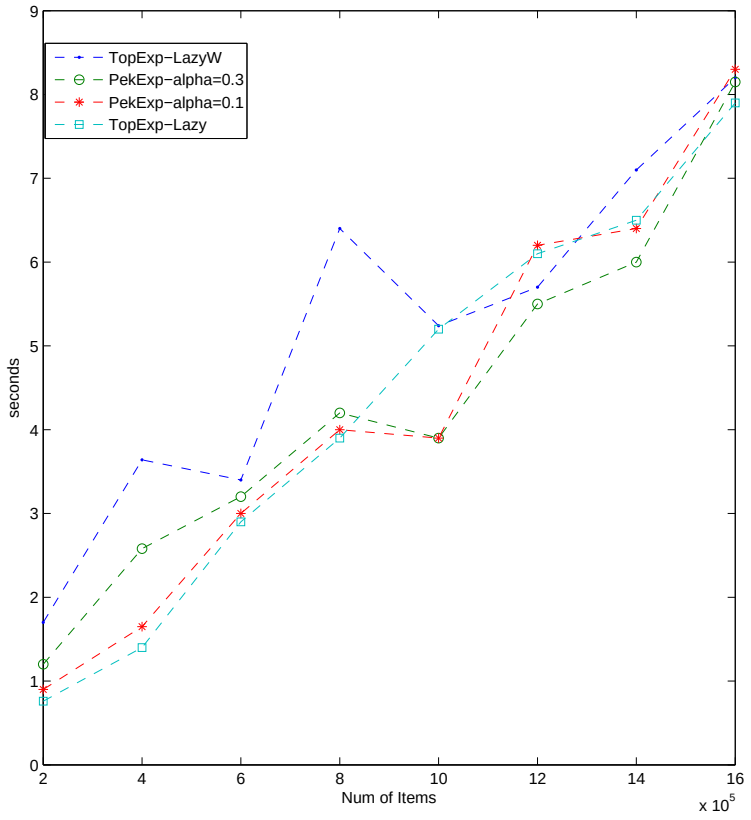

Figure 6.4: Running Time for Del.icio.us with Growing Sampling Size

by the query. And the important aspects of a research field should be included in survey papers about the field. Thus, we expect the top expansion keywords we find could to some extent match with important aspects keywords extracted from a corresponding survey paper. We include the result of TopExp-LazyW and PekExp-alpha=0.1 for comparison between non-Path Exclusive and Path Exclusive algorithms.

For the three queries (histogram,privacy,xml) we have chosen, we are able to find corresponding survey papers for the first two of them[22, 28].

In Table 6.1, we show the top-10 expansions generated by the TopExp-Lazy

TopExp-LazyW	Path-exclusive
approxim	approxim wavelet
select estimate	approxim olap
optim	rang queri
query	multi-dimension
wavelet	attribute optim
bucket	cost select
approxim wavelet	approxim stream
approxim olap	cumul wavelet
approxim optim	approxim summari
multi-dimension	alloc bucket

Table 6.1: Expansions Generated for "histogram"

and PekExp (shown as Path-exclusive in Table 6.1) algorithms on the ACM Digital Library dataset. By manually analyzing the expansions generated, we can see that TopExp-Lazy works fairly well in generating expansions related to major subtopics of the query "histogram", according to the survey [28], the two most important applications of histogram techniques in databases have been "selectivity estimation" and "approximate query answering". For both topics we can find corresponding expansions in the list of expansions generated, e.g. "select estimate", "approxim". Furthermore, Approximation is an established application of histograms, which constitutes a large portion of [28]. Indeed, accordingly we can see various low level expansions like "approxim wavelet", "approxim olap" and so on, which reveals further interesting direction to explore under approximation. Notice that other expansions of "histogram" also correspond to important aspects in the survey. For example, "multi-dimension" histogram is a relatively new and important subfield of histogram, and "optim" refers to an important application of approximation using histogram – query optimization. So in general, the expansions generated can cover most important aspects of the survey. Similar results can be observed for "privacy" as well. For "xml", though we were not able to find a suitable survey paper, through one of the co-author's manual evaluation as an domain expert, the result is quite good as it covers important aspects like "xpath query" and "dtd schema".

While the quality of expansions returned by TopExp-Lazy are quite good, as

privacy		**xml**	
TopExp-Lazy	**Path-Exclusive**	**TopExp-Lazy**	**Path-Exclusive**
preserve mining	mining preserve	dtd conform	twig
anonym protect	anonym protect	twig holist	join
preserve	mining rule	query	ancestor-descend
mining	anonym mining	query xpath	conform
anonym publish	individual	dtd	dtd tractable
anonym	perturb reconstruct	twig pattern	decide
anonym k	disclose disclosure	index	dissemin secure
protect individual	breach	access control	sql store
protect	mining reconstruct	holist pattern	typecheck
mining rule	access	dtd schema	node

Table 6.2: Expansions Generated for "privacy" and "histogram".

can be found in Table 6.1 and Table 6.2, these expansions may contain redundant information. E.g., for query "histogram", the expansions returned include both the high level expansion "approxim" and low level (i.e., more refined) expansions "approxim wavelet", "approxim olap". Similarly, for query "privacy", we have "anonym" returned along with "anonym protect" and "anonym k", and for query "xml", we have "query" returned along with "query xpath" and "query store". Our path exclusive algorithm can avoid this problem by making sure that no expansion in the returned result set is a subset of other expansions in the result, thus avoiding such redundancy. By a manual inspection of the list of expansions in Table 6.2, it is easy to see that such redundancy is avoided by PekExp and also that the set of expansions returned by the that algorithm can also cover most of the important expansions for each query.

6.4.1 Further Discussion

Although the expansion keywords generated has high quality considering we assume no pre-defined structure information, there are limitations on the top expansion keywords generated. These limitations provide us with clues of future improvements. A list of observed limitations and possible reasons is listed below:

1. Generated expansion keywords are hard to interpret. There are two reasons

for this:(1) We did stemming on all the keywords during preprocessing; (2) Our experiment design does not deal with phrases.

2. Noise exists in the top expansions generated, in the sense that some expansion keywords may not necessarily summarize the papers included in the expansion. One possible reason for this is because we directly use keywords of a paper to imply its major content, which does not always hold. Using topic extraction techniques to first extract the topic keywords for indexing may alleviate this problem.

3. Overlap among papers fall into different expansions exist. This is due to the fact we didn't consider overlap among expansions from the aspect of items covered by the expansion.

Chapter 7

Conclusion and Future Work

In this paper, we started with the observation that years after search engines first came into being, most current search services still return results as a flat ranked list of items, which is not ideal for users to easily get to the items they are really interested in. We studied the problem of how to better present search/query results to users. We considered a search scenario in which each item is annotated with a set of keywords and is equipped with a set of quality attributes, and proposed novel ways to automatically group query result items into different expansions of the query, corresponding to subsets of keywords. We proposed various efficient algorithms which can calculate top-k expansions, and we also studied additional desirable properties for the set of expansions returned, from a semantic perspective, whereby certain redundancies in the expansions returned can be avoided. With a detailed set of experiments, we not only demonstrated the performance of the proposed algorithms, we also validated the quality of the expansions returned by doing a study on a real data set. It is interesting to explore more desirable properties of the expansions returned, and to investigate more efficient algorithms which can return high quality expansion set and can handle the web scale.

Bibliography

[1] http://www.amazon.com. → pages 4

[2] http://www.bing.com. → pages 1

[3] http://www.flickr.com. → pages 3

[4] http://www.google.com, . → pages 1

[5] http://images.google.com, . → pages 2

[6] http://www.quora.com. → pages 3

[7] http://scholar.google.com. → pages 2

[8] http://www.wikipedia.org. → pages 2

[9] http://www.yahoo.com. → pages 1

[10] *Analyzing social bookmarking systems: A del.icio.us cookbook*, ECAI 2008, 2008. → pages 38

[11] S. Amer-Yahia. I am structured: Cluster me, don't just rank me. In *BEWEB*, 2011. → pages 1

[12] O. Ben-Yitzhak, N. Golbandi, N. Har'El, R. Lempel, A. Neumann, S. Ofek-Koifman, D. Sheinwald, E. Shekita, B. Sznajder, and S. Yogev. Beyond basic faceted search. In *Proceedings of the international conference on Web search and web data mining*, WSDM '08, pages 33–44, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-927-2. doi:http://doi.acm.org/10.1145/1341531.1341539. URL http://doi.acm.org/10.1145/1341531.1341539. → pages 3

[13] M. Bendersky and W. B. Croft. Analysis of long queries in a large scale search log. In *Proceedings of the 2009 workshop on Web Search Click Data*,

WSCD '09, pages 8–14, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-434-8. doi:http://doi.acm.org/10.1145/1507509.1507511. URL http://doi.acm.org/10.1145/1507509.1507511. → pages 2

[14] S. Bethard and D. Jurafsky. Who should i cite: learning literature search models from citation behavior. In *CIKM*, pages 609–618, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0099-5. doi:http://doi.acm.org/10.1145/1871437.1871517. URL http://doi.acm.org/10.1145/1871437.1871517. → pages 15

[15] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006. → pages 17

[16] K. Chakrabarti, V. Ganti, J. Han, and D. Xin. Ranking objects based on relationships. In *SIGMOD*, pages 371–382, New York, NY, USA, 2006. ACM. ISBN 1-59593-434-0. doi:http://doi.acm.org/10.1145/1142473.1142516. URL http://doi.acm.org/10.1145/1142473.1142516. → pages 8, 10, 11, 14

[17] W. Dakka, P. G. Ipeirotis, and K. R. Wood. Automatic construction of multifaceted browsing interfaces. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, CIKM '05, pages 768–775, New York, NY, USA, 2005. ACM. ISBN 1-59593-140-6. doi:http://doi.acm.org/10.1145/1099554.1099738. URL http://doi.acm.org/10.1145/1099554.1099738. → pages 15

[18] D. Dash, J. Rao, N. Megiddo, A. Ailamaki, and G. Lohman. Dynamic faceted search for discovery-driven analysis. In *Proceeding of the 17th ACM conference on Information and knowledge management*, CIKM '08, pages 3–12, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-991-3. doi:http://doi.acm.org/10.1145/1458082.1458087. URL http://doi.acm.org/10.1145/1458082.1458087. → pages 14

[19] J. Dean. Challenges in building large-scale information retrieval systems: invited talk. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining*, WSDM '09, pages 1–1, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-390-7. doi:http://doi.acm.org/10.1145/1498759.1498761. URL http://doi.acm.org/10.1145/1498759.1498761. → pages 8

[20] M. D. Ekstrand, P. Kannan, J. A. Stemper, J. T. Butler, J. A. Konstan, and J. T. Riedl. Automatically building research reading lists. In *ACM RecSys*,

pages 159–166, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-906-0. doi:http://doi.acm.org/10.1145/1864708.1864740. URL http://doi.acm.org/10.1145/1864708.1864740. → pages 15

[21] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *JCSS*, 66(4):614–656, 2003. → pages 8, 9, 10, 14, 21, 22

[22] B. C. M. Fung, K. Wang, R. Chen, and P. S. Yu. Privacy-preserving data publishing: A survey of recent developments. *ACM Comput. Surv.*, 42: 14:1–14:53, June 2010. ISSN 0360-0300. doi:http://doi.acm.org/10.1145/1749603.1749605. URL http://doi.acm.org/10.1145/1749603.1749605. → pages 45

[23] M. Halldrsson and J. Radhakrishnan. Greed is good: Approximating independent sets in sparse and bounded-degree graphs. *Algorithmica*, 18: 145–163, 1997. ISSN 0178-4617. URL http://dx.doi.org/10.1007/BF02523693. 10.1007/BF02523693. → pages 13

[24] Q. He, J. Pei, D. Kifer, P. Mitra, and L. Giles. Context-aware citation recommendation. In *WWW*, pages 421–430, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-799-8. doi:http://doi.acm.org/10.1145/1772690.1772734. URL http://doi.acm.org/10.1145/1772690.1772734. → pages 15

[25] M. A. Hearst. Clustering versus faceted categories for information exploration. *Commun. ACM*, 49:59–61, April 2006. ISSN 0001-0782. doi:http://doi.acm.org/10.1145/1121949.1121983. URL http://doi.acm.org/10.1145/1121949.1121983. → pages 14

[26] D. Hochbaum. Efficient bounds for the stable set, vertex cover and set packing problems. *Discrete Applied Mathematics*, 6:232–248, 1983. → pages 13

[27] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4), 2008. → pages 8, 14

[28] Y. Ioannidis. The history of histograms (abridged). In *VLDB*, pages 19–30. VLDB Endowment, 2003. ISBN 0-12-722442-4. URL http://portal.acm.org/citation.cfm?id=1315451.1315455. → pages 45, 46

[29] B. J. Jansen, A. Spink, and T. Saracevic. Real life, real users, and real needs: a study and analysis of user queries on the web. *Information Processing and*

Management, 36(2):207 – 227, 2000. ISSN 0306-4573. doi:10.1016/S0306-4573(99)00056-4. URL http://www.sciencedirect.com/science/article/pii/S0306457399000564. → pages 2

[30] A. Kako, T. Ono, T. Hirata, and M. M. Halldórsson. Approximation algorithms for the weighted independent set problem. In *Graph-theoretic Concepts In Computer Science, 31st International Workshop, WG*, pages 341–350, 2005. → pages 11, 13, 34

[31] D. D. Lewis and K. S. Jones. Natural language processing for information retrieval. *Commun. ACM*, 39:92–101, January 1996. ISSN 0001-0782. doi:http://doi.acm.org/10.1145/234173.234210. URL http://doi.acm.org/10.1145/234173.234210. → pages 2

[32] C. Li, N. Yan, S. B. Roy, L. Lisham, and G. Das. Facetedpedia: Dynamic generation of query-dependent faceted interfaces for wikipedia. In *WWW*, pages 651–660, 2010. → pages 6, 14

[33] X. Liang, M. Xie, and V. L. Lakshmanan. Adding structure to top-k: From items to expansions. In *CIKM*, New York, NY, USA, 2011. ACM. → pages iii

[34] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999. URL http://ilpubs.stanford.edu:8090/422/. Previous number = SIDL-WP-1999-0120. → pages 2

[35] S. E. Robertson, S. Walker, and M. M. Hancock-Beaulieu. Okapi at trec-3. In *TREC-3*, pages 109–126. NIST, 1994. → pages 9, 38, 39

[36] N. Sarkas, N. Bansal, G. Das, and N. Koudas. Measure-driven keyword-query expansion. *PVLDB*, 2(1):121–132, 2009. → pages 15

[37] A. Simitsis, A. Baid, Y. Sismanis, and B. Reinwald. Multidimensional content exploration. *Proc. VLDB Endow.*, 1:660–671, August 2008. ISSN 2150-8097. doi:http://dx.doi.org/10.1145/1453856.1453929. URL http://dx.doi.org/10.1145/1453856.1453929. → pages 15

[38] Y. Song, Z. Zhuang, H. Li, Q. Zhao, J. Li, W.-C. Lee, and C. L. Giles. Real-time automatic tag recommendation. In *SIGIR*, pages 515–522, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-164-4. doi:http://doi.acm.org/10.1145/1390334.1390423. URL http://doi.acm.org/10.1145/1390334.1390423. → pages 39

[39] R. van Zwol, B. Sigurbjornsson, R. Adapala, L. Garcia Pueyo, A. Katiyar, K. Kurapati, M. Muralidharan, S. Muthu, V. Murdock, P. Ng, A. Ramani, A. Sahai, S. T. Sathish, H. Vasudev, and U. Vuyyuru. Faceted exploration of image search results. In *WWW*, pages 961–970, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-799-8. doi:http://doi.acm.org/10.1145/1772690.1772788. URL http://doi.acm.org/10.1145/1772690.1772788. → pages 14