KEYWORD-BASED OBJECT SEARCH AND EXPLORATION IN MULTIDIMENSIONAL TEXT DATABASES

BY

BO ZHAO

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2011

Urbana, Illinois

Master's Committee:

Professor Jiawei Han, Director of Research

Abstract

We propose a novel system TEXplorer that integrates keyword-based object ranking with the aggregation and exploration power of OLAP in a text database with rich structured attributes available, *e.g.*, a product review database.

TEXplorer can be implemented within a multi-dimensional text database, where each row is associated with structural dimensions (attributes) and text data (*e.g.*, a document). The system utilizes the *text cube* data model, where a *cell* aggregates a set of documents with matching values in a subset of dimensions. Cells in a text cube capture different levels of summarization of the documents, and can represent objects at different conceptual levels.

Users query the system by submitting a set of keywords. Instead of returning a ranked list of all the cells, we propose a *keyword-based interactive exploration framework* that could offer flexible OLAP navigational guides and help users identify the levels and objects they are interested in. A novel *significance measure* of dimensions is proposed based on the distribution of *IR relevance* of cells. During each interaction stage, dimensions are ranked according to their significance scores to guide drilling down; and cells in the same cuboids are ranked according to their relevance to guide exploration. We propose efficient algorithms and materialization strategies for ranking top-k dimensions and cells. Finally, extensive experiments on real datasets demonstrate the efficiency and effectiveness of our approach.

To Father and Mother.

Acknowledgments

First I would like to express my deepest appreciation to my advisor Professor Jiawei Han, who always motivates me to conduct quality research and inspires my interest in data mining, databases and OLAP techniques with his great insight and knowledge in the area. I would also like to thank Professor Chengxiang Zhai, who always gives constructive feedbacks and suggestions for my thesis research, especially problems related to keyword search and text mining. My gratitude also goes to many colleagues in the DAIS group at UIUC who collaborate with me on the research related to this thesis or provide helpful feedbacks, especially Bolin Ding, Cindy Lin, Tianyi Wu, Hongbo Deng, Duo Zhang, Jing Gao, Yizhou Sun and Yue Lu.

The work was supported in part by HP Lab Innovation Award, NASA NRA-NNH10ZDA001N, and the U.S. Army Research Laboratory under Cooperative Agreement No. W911NF-09-2-0053 (NS-CTA).

Table of Contents

Chapter 1

Introduction

Nowadays the web is deeply integrated with database technologies, and one important trend is that much more free texts generated on the web are associated with structured attributes which are either automatically extracted or recognized by human experts. For example, many shopping websites display technical attributes of their products, *e.g.,* Brand, Model, Price, along with customer reviews about the products, so users can conveniently specify those structured attributes to refine their results. Structured data coexist with texts also in many other domains, such as medical reports, system bug reports, *etc.* Therefore, developing novel systems that can efficiently manage such *multidimensional textual data* and provide effective methods for users to explore and understand the data has become a demanding need.

Intuitively, users should be able to query multi-dimensional text databases by specifying values that match the dimensions and keywords that match the texts. Keyword search has been developed in RDBMSs [4], and traditional keyword search systems are effective if users only care about the top relevant tuples, but when the relevant results are many, it could be difficult for users to digest the results and come up with more accurate queries to refine the results. However, if rich structured dimensions are available, they can be naturally utilized to organize the results and give users possible refinement. One example is faceted search systems [15, 17, 6, 11, 12, 10, 2], which show structured facets that are associated with relevant documents in their interfaces, and facets can be interactively selected by users to further refine the search results. Moreover, to further guide user exploration, some systems [17, 6, 11] rank facets based on their proposed measures to promote those dimensions and attributes that are more effective in helping users navigate to their desired results. For example, DynaCet [17] focuses on minimizing the number of facets users need to select to reach the most relevant documents.

Faceted search offers users much more flexibility to explore the results by specifying various facet refinement, however, we should notice in faceted search attributes are mainly treated as filtering conditions and the results displayed to users are still individual relevant tuples, which means the relevance computation, if there is any, still stays at the tuple level. However, individual tuples may not be exactly what users are looking for in many scenarios, instead, users may be more interested in finding the relevance objects, which can

be represented by certain aggregation of the tuples. For example, when a user searches for "light-weighted laptop" on Amazon, usually she is not looking for specific review documents that mention the keywords, but some laptop brands or models that reviewers generally think are light-weighted. In this case, faceted search that returns the relevant documents may not be very helpful, instead, we need a framework that supports object-level relevance and navigation.

Past works on entity and object search [5, 1, 3] have justified the effectiveness of aggregating relevant tuples into high-level objects. And recently, [7, 23] target on supporting more flexible aggregation in multidimensional databases, *i.e.,* any combination of the structured attributes correspond to an object at a certain level. In data cubes built on top of the databases, [23] searches for the minimum cells covering the query, while TopCells [7] ranks all the cells based on IR relevance functions. However, all these systems output results in a ranked list, which as we have argued is not ideal for user digestion and exploration, and the fact that ranked objects are in different levels could give more difficulties to users. Therefore, an exploration mechanism suitable for higher-level objects needs to be developed.

With such motivation, in this work we propose TEXplorer, *a keyword-based interactive OLAP framework* that incorporates keyword search with the aggregation and exploration functionalities that are essential in OLAP systems (Figure 1.1 compares TEXplorer with existing keyword search systems). In TEXplorer, we adopt the strengths of object search and faceted search by utilizing structured attributes in two aspects: first, as in object/entity search they represent high-level objects which individual tuples can be aggregated into; and second as in faceted search they are ranked and displayed to users to facilitate informative navigation of the results.

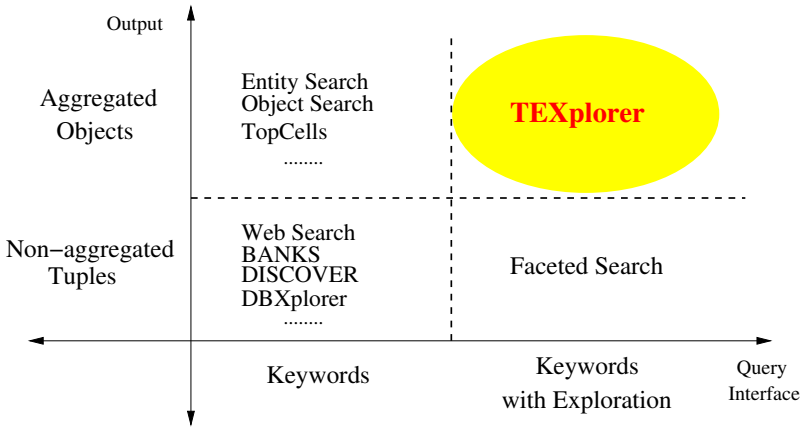


Figure 1.1: Roadmap of Keyword Search Systems

Moreover, to enable smooth exploration experience, TEXplorer employs a *text cube* data model as presented in [7], so that objects, *i.e.,* cells in the cube, at different levels can be efficiently aggregated. When

a keyword query comes, the relevance of each object/cell is aggregated from the relevance scores of tuples belonging to the cell, so that objects in same dimension subspaces can be ordered by relevance, and we further propose a significance measure for ranking different dimension subspaces, which indicates to users which subspaces are more likely to contain desired objects. To interactively explore the results, users can drill down to a dimension with high significance to look at the objects in the corresponding subspace, then they can select an object with high relevance and advance to the next interaction stage, where sub-objects are computed only from tuples belonging to the selected object. Example 1 explains the user interaction with concrete details.

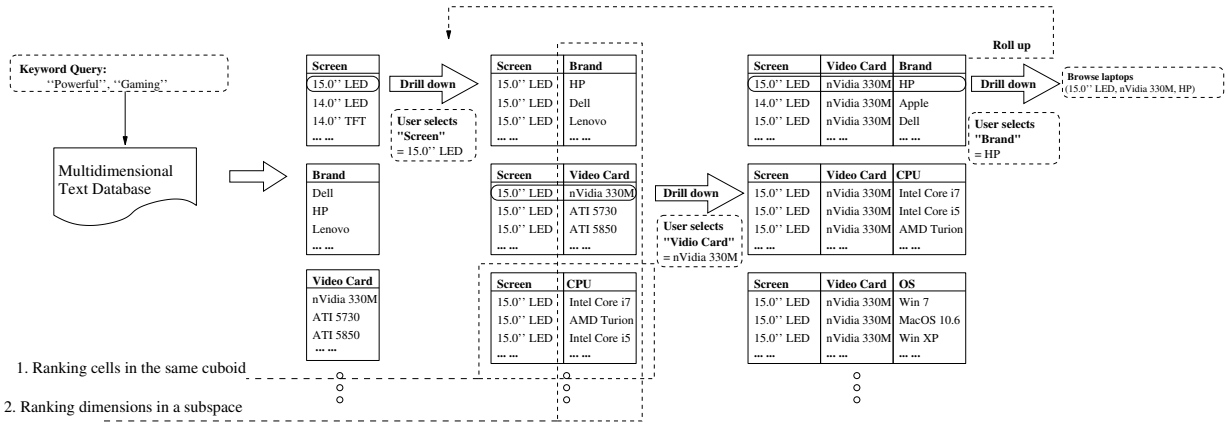


Figure 1.2: Keyword-based Interactive Exploration in TEXplorer

Example 1. *(Motivating Example) Given a customer review database of laptops, each review document is associated with several structural attributes of the laptop being reviewed, e.g., Brand, CPU, Screen, OS, Video Card and Weight, etc.*

Figure 1.2 shows a running example of TEXplorer. A customer Kate wants to buy a powerful laptop suitable for gaming. Using our system, she could submit a keyword query "powerful, gaming".

Suppose at the beginning, the system returns Kate a list of dimensions ordered by significance: Screen, Brand, Video Card, etc. The Screen dimension will be on the top if we infer from review data it is the most important factor for choosing gaming laptops. Within the Screen subspace different screen objects are ranked according to relevance, e.g. 15" LED will be the most relevant one if reviews about laptops with this screen have the highest aggregated relevance score.

Kate can drill down the Screen dimension and select 15" LED, then in the second stage, the rest dimensions and objects in each dimension are re-ranked only based on reviews for laptops that have the selected screen type. Then this time she could drill down to the top 2 "Video Card" dimension if she feels video cards are more important to her, after which she will be prompted to the third stage to make more refinements. At

each stage, Kate can modify her keyword query and the system will re-rank the current records based on the updated query; Kate can also choose to roll back to any previous stages and explore other subspaces.

To summarize, we identify the contributions of this thesis [1] are :

- We propose a *keyword-based interactive exploration framework*, which integrates keyword search with OLAP aggregation and exploration, and supports most state-of-the-art IR models for relevance computation.
- We introduce a novel measure, *significance*, to effectively rank dimensions and facilitate informative user exploration based on relevance scores.
- We identify the major computational challenges in our system and propose pre-computation strategies, as well as ranking algorithms that efficiently retrieve top-k dimensions/cells.

Organization We will introduce the data model, *text cube*, and identify the major tasks of TEXplorer in Chapter 2. Chapter 3 introduces our *significance* measure. Chapter 4 proposes pre-omputation strategies and efficient algorithms for ranking dimensions and cells. Chapter 5 reports experimental study. Chapter 6 discusses related work, and finally Chapter 7 concludes this thesis.

[1] This thesis is based on previously published work [22]

Chapter 2

Problem Definition

In this chapter, we first introduce the *text cube* data model, in Section 2.1, then define the problems we need to address in Section 2.2.

2.1 Data Model: Text Cube

In an *n-dimensional text database* $\mathcal{DB}$, each tuple t can be represented by a *document* d and an object with *n structural dimensions* $\mathcal{A}_1$, $\mathcal{A}_2$, ..., and $\mathcal{A}_n$, *i.e.*, $t = (a_1, a_2, \ldots, a_n, \mathsf{d})$. Each dimension corresponds to one attribute of the object, *e.g.*, Brand, Model, *etc.* We let $t_{\mathcal{A}_i} = a_i$, and $t_{\mathcal{D}} = \mathsf{d}$ denote the *value* of *dimension* $\mathcal{A}_i$ and document of the tuple respectively.

A *text cube* can be built on top of the multidimensional text database in order to support OLAP operations on the text data. The model was first introduced in [16]. Our system utilizes text cube so we briefly introduce several important concepts as follows.

A *cell* in the text cube is denoted as $C = (v_1, v_2, \ldots, v_n : \mathsf{D})$, where $v_i \in \mathcal{A}_i \cup \{*\}$. $v_i = *$ means the dimension $\mathcal{A}_i$ is *aggregated* in C (v_i can take any value in $\mathcal{A}_i$). D is the set of documents in the tuples of $\mathcal{DB}$ whose dimension values match C. We call this set D the *aggregated document* of the cell C. Let $C_{\mathcal{A}_i}$ denote v_i, and $C_{\mathcal{D}}$ denote D. A cell is called *empty* if $C_{\mathcal{D}} = \varnothing$.

A *cuboid* is a group of cells having the same non-$*$ dimensions. A cuboid with m non-$*$ dimensions is called an *m-dim cuboid*. Cell C' is a *child* of cell C iff $\exists i$, s.t. $C_{\mathcal{A}_i} = *$, and $C'_{\mathcal{A}_i} \in \mathcal{A}_i$, and $\forall j \neq i : C_{\mathcal{A}_j} = C'_{\mathcal{A}_j}$. Specifically, if $\mathcal{A}$ is the only one dimension that is aggregated in C but has non-$*$ value in C', we call C' the $\mathcal{A}$-*child* of C. Let $\mathsf{chd}(C')$ denote the set of (non-empty) children of C', and $\mathsf{chd}_{\mathcal{A}}(C')$ denote the set of (non-empty) $\mathcal{A}$-children of C'.

2.2 Tasks of TEXplorer

After introducing our data model, in this section we will formally define the major tasks of TEXplorer. As explained in Example 1 and Figure 1.2, at any interactive stage, if users would like to further drill down to a

more detailed level, they need to decide: (i) which dimension to drill down; and (ii) which cell in the drilled down cuboid to further explore. Hence, two major tasks of TEXplorer are (i) ranking candidate dimensions by the significance measure; and (ii) for each dimension, ranking children cells in the corresponding subspace by relevance. We give formal definitions of these two problems as follows:

Ranking Dimensions for Drilling Down

Given a cell C (being explored by the user), for any aggregated dimension $\mathcal{A}_i$ of C, we define the *significance* of $\mathcal{A}_i$ w.r.t. the keyword query q, denoted by $\mathsf{Sig}_{\mathcal{A}_i}(\mathsf{q}, C)$. Our goal is to rank all the aggregated dimensions of C according to $\mathsf{Sig}_{\mathcal{A}_i}(\mathsf{q}, C)$, or provide the top-$k$ ones.

Example 2. *Consider the example in Figure 1.2. When the cell (Screen = 15.0" LED) is being explored and query is "gaming, powerful", the drilling down dimensions are ranked as Brand, Video Card, CPU, ... This implies, for laptops with 15.0" LED screens, Brand, Video Card and CPU are the most important factors that are related to gaming experience. The user can choose a dimension to drill down based on the system outputs as well as her own intention and preference.*

Providing a ranked list of top-k significant dimensions is important, especially when the number of dimensions is large, to help users explore the data more effectively. Otherwise, users who do not have much knowledge about the domain, *e.g.*, laptops, could waste a lot more time looking into all possible subspaces. Even for domain experts, such data-oriented ranking can also benefit the analysis of the correlation between structured attributes and keywords in the texts. For example, in a TEXplorer built on aviation report databases, aviation safety analysts can query specific problems mentioned in pilot reports by several keywords and find what dimensions, *e.g.*, location, weather, time, *etc*, are the most important factors that are potentially related to the problem.

Note that although the roles are similar, the significance measure in TEXplorer is designed with quite different intuitions from facet ranking measures in faceted search systems. Since we focus on the ranking and exploration of relevant objects rather than individual tuples given a keyword query, our significance measure is more appropriate for this goal by exploiting the distributions of relevance scores of objects and tuples. We will formally define the significance measure in Chapter 3.

Ranking Cells in the Same Cuboid

Given a cell C (being explored by the user) and an aggregated dimension $\mathcal{A}_i$ of C, we want to rank all the $\mathcal{A}_i$-children of C according to the relevance $\mathrm{Rel}(\mathsf{q}, C)$ efficiently, or provide the top-k relevant ones.

Example 3. *Consider the second stage in Figure 1.2, when the user decides to drill down from the cell C: (Screen = 15.0" LED), for C's Brand-children, we need to rank them according to their relevance to the keyword query, in the order of, e.g., (Screen = 15.0" LED, Brand = HP), (Screen = 15.0" LED, Brand = Dell), and so on. Similarly, we also rank C's Video Card-children, CPU-children, and so on. Note we only rank cells within the same cuboid.*

Different from [23, 7], where all the cells in the whole cube are ranked together, we focus on ranking cells (objects) in the same cuboid (subspace), which not only makes the ranking more meaningful in semantics, but also enables our system to adopt much more IR relevance models that could potentially increase accuracy.

Recall a cell C aggregates the documents in $C_{\mathcal{D}}$, so the relevance of C w.r.t. a query q, *i.e.*, $\text{Rel}(\mathsf{q}, C)$, is actually the relevance of a set documents. Previous studies in IR [8] propose two general relevance models for this problem. The first one is called "large document model": simply concatenating all the documents in the set into a pseudo document, and computing the relevance of such pseudo document. The second is "small document model": computing the relevance of each individual document and aggregating the scores using an aggregate function, *e.g.*, the average function:

$$\text{Rel}(\mathsf{q}, C) = \frac{1}{|C_{\mathcal{D}}|} \sum_{\mathsf{d} \in C_{\mathcal{D}}} \text{rel}(\mathsf{q}, \mathsf{d}) \tag{2.1}$$

where $|C_{\mathcal{D}}|$ is the number of documents in $C_{\mathcal{D}}$, and $\text{rel}(\mathsf{q}, \mathsf{d})$ is the relevance of a document d w.r.t. q.

The two relevance models have their own advantages. One limitation of previous methods [23, 7] is that they are restricted to specific relevance functions. For example, TopCells [7] can only support the "small document model" due to its efficiency issue. But in TEXplorer, we can efficiently support both models. In the rest of the thesis, we will use the average model (Equation 2.1) to denote the relevance of a cell only for the ease of explanation.

For the document relevance $\text{rel}(\mathsf{q}, \mathsf{d})$, TEXplorer is not restricted to a specific form of document relevance function either. We support the class of document relevance in a more general form of:

$$\text{rel}(\mathsf{q}, \mathsf{d}) = \sum_{w \in \mathsf{q}} \mathsf{IDF}_w \cdot \mathsf{TFW}_{w,\mathsf{d}} \cdot \mathsf{QTW}_{w,\mathsf{q}} \tag{2.2}$$

where IDF_w is the inverted document frequency factor of term $w \in \mathsf{q}$, $\mathsf{TFW}_{w,\mathsf{d}}$ is the term frequency factor of w in document d, and $\mathsf{QTW}_{w,\mathsf{q}}$ is the query term frequency factor of w in q.

Most state-of-the-art relevance functions fit into the above form, such as Okapi BM25, pivoted normalization, and some language modeling approaches [21]. Moreover, user relevance models [12] can also be

implemented in TEXplorer to take users' interaction as feedback and make the ranking more personalized. For example, documents in a cell may be less relevant if a user rolls up after visiting the cell. We will not explain more details since the relevance function for ranking documents and objects is not the contribution of this thesis, since we intend to design a general framework that can utilize any IR relevance measure.

Chapter 3

Significance of a Dimension

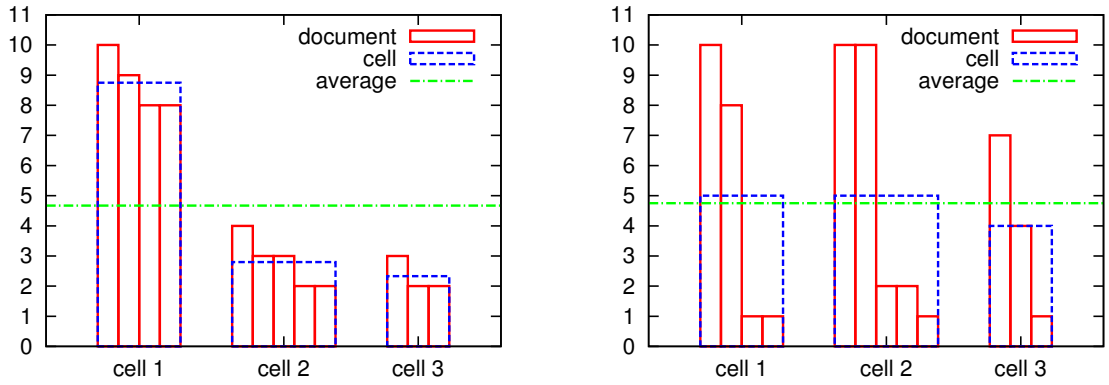
In this chapter, we introduce a novel *significance* measure to help users determine which dimension to drill down when exploring a cell. For a cell C and an aggregated dimension $\mathcal{A}_i$, $\mathcal{A}_i$-children of C are obtained by drilling down $\mathcal{A}_i$ from C. The significance of drilling down dimension $\mathcal{A}_i$ w.r.t. a keyword query q is defined based on the following two intuitions.

i) **How distinctive are top relevance cells?** The overall relevance of documents in C is constant given the query q, then if for some dimension $\mathcal{A}_i$, relevant documents are aggregated in a way that some cells are very relevant while the others are not, it implies $\mathcal{A}_i$ is discriminative w.r.t. q, and therefore $\mathcal{A}_i$ is more interesting to look into. For example, if reviews of (Brand = Lenovo, Model = ThinkPad) laptops are highly relevant to query "long battery life" while reviews of (Brand = Lenovo, Model = IdeaPad) are not, then users might be more interested in drilling down dimension Model from cell (Brand = Lenovo) than those dimensions where top relevance cells are not very distinctive.

ii) **How consistent are documents within each cell?** Now consider the documents in each of the $\mathcal{A}_i$-children of C. If these documents are consistent w.r.t. q, *i.e.,* they are either all relevant to q or all irrelevant to q, then it implies the relevance of this cell is of high confidence. For example, if we drill down dimension Color from a cell (Brand = Lenovo), and documents in every children cell are not very consistent w.r.t. the query {"battery", "long", "time"}, then it probably means Color is not a very meaningful dimension to drill down.

Figure 3.1 illustrates the significance scores for two dimensions. The height of the bars indicate the relevance of the documents and cells. In the first dimension, top cells are more distinguished from others and documents are more consistent within each cell, and therefore the first dimension will have higher significance score.

Now we can formally introduce the significance measure. To capture our first intuition, we define *cell variance* $\mathsf{CV}_{\mathcal{A}_i}(\mathsf{q}, C)$ as how much the relevance of each of C's $\mathcal{A}_i$-children deviates from the relevance of C, weighted by the number of documents in each $\mathcal{A}_i$-child, since intuitively larger cells should have higher



(a) High Significance

(b) Low Significance

Figure 3.1: Example of Significance Scores

weights:

$$\mathsf{CV}_{\mathcal{A}_i}(\mathsf{q}, C) = \frac{\sum_{C' \in \mathsf{chd}_{\mathcal{A}_i}(C)} |C'_{\mathcal{D}}| \cdot (\mathrm{Rel}(\mathsf{q}, C') - \mathrm{Rel}(\mathsf{q}, C))^2}{|\mathsf{chd}_{\mathcal{A}_i}(C)| - 1} \tag{3.1}$$

To capture our second intuition, we define the *document variance* $\mathsf{DV}_{\mathcal{A}_i}(\mathsf{q}, C)$ as how much the relevance of each document in C deviates from the relevance of the $\mathcal{A}_i$-child of C which contains that document. And *inverted document variance* $\mathsf{IDV}_{\mathcal{A}_i}(\mathsf{q}, C)$ is the reciprocal of $\mathsf{DV}_{\mathcal{A}_i}(\mathsf{q}, C)$, so higher IDV implies higher consistency of documents within each $\mathcal{A}_i$-child of C:

$$\mathsf{IDV}_{\mathcal{A}_i}(\mathsf{q}, C) = \frac{|C_{\mathcal{D}}| - |\mathsf{chd}_{\mathcal{A}_i}(C)|}{\sum_{C' \in \mathsf{chd}_{\mathcal{A}_i}(C)} \left(\sum_{\mathsf{d} \in C'_{\mathcal{D}}} (\mathrm{rel}(\mathsf{q}, \mathsf{d}) - \mathrm{Rel}(\mathsf{q}, C'))^2 \right)} \tag{3.2}$$

If the average relevance model (Equation 2.1) is used, treating each cell as a document group, CV is actually the *variance of group means*, and DV is the *mean of with-in group variances*. We define a more general form here so other relevance models can also be plugged in.

We will consider a dimension more significant if the CV and IDV are both high. On the other hand, a dimension with higher CV is also more likely to have lower IDV, since the documents in this cuboid tend to be inconsistent. Therefore, similar to the well-known TF-IDF mechanism, we penalize high CV scores based on IDV scores by using the product of the two as our measure, *i.e.*, we define the significance as:

$$\mathsf{Sig}_{\mathcal{A}_i}(\mathsf{q}, C) = \mathsf{CV}_{\mathcal{A}_i}(\mathsf{q}, C) \cdot \mathsf{IDV}_{\mathcal{A}_i}(\mathsf{q}, C) \tag{3.3}$$

Statistical Meaning of Sig Actually our proposed significance measure $\mathsf{Sig}_{\mathcal{A}_i}$ has some intrinsic relationship to the *F-ratio*, which is often used in *ANOVA* test [13] in statistics. All the documents in a cell C are

partitioned into its $\mathcal{A}_i$-children for a drilling down dimension $\mathcal{A}_i$. If we associate each of the $\mathcal{A}_i$-children with a relevance distribution, we can regard the relevance scores of documents in each of the $\mathcal{A}_i$-children as samples drawn from this distribution. If the average relevance model (2.1) is used, $\mathsf{Sig}_{\mathcal{A}_i}$ as defined in (3.3) is actually the *F-ratio*, which measures how significantly these relevance distributions are different from each other. The higher *F-ratio* is, the more significantly these distributions are different. And in this thesis we generalize its form so other relevance models can also be used to compute $\mathsf{Sig}_{\mathcal{A}_i}$.

Chapter 4

Ranking Algorithms

In this chapter, we focus on designing efficient algorithms for the two computational tasks: ranking most significant dimensions and most relevant cells in each dimension. In the previous chapter, we can see that our significance measure depends on the relevance scores of children cells in each dimension, so the ranking of relevant cells can be naturally implemented within the framework of ranking dimensions. Therefore, in the discussion of our proposed algorithms we mainly focus on computing the top-k significant dimensions. We identify two major computational challenges of this problem and proposed the corresponding solutions:

- Each document belongs to one children cell in each drill-down dimension. Therefore, in order to rank children cells in every dimension and compute the Sig score, every relevant document needs to be aggregated multiple times to different cells. However, we find the cell relevance scores are *pre-computable*, therefore we can materialize some statistics offline in the Text Cube, so that online query-dependent aggregation can be performed very efficiently.
- The Sig measure requires aggregated statistics from all the relevant documents. Hence, to compute the exact scores all relevant documents need to be scanned, which could be costly on large scale corpus. To solve this problem, we focus on finding top-k solutions, *i.e.,* if only top-k significant dimensions are desired we can just scan a very small number of documents. We identify our top-k problem is very different from existing ones, and by exploiting some good properties of our problem, we could achieve stronger pruning power than previous top-k algorithms.

4.1 MultiAccess Algorithm

The first algorithm is the baseline approach. Equation (3.1) and (3.2) show that, given the cell C (currently visited by users) and the keyword query q, the Sig score of dimension $\mathcal{A}_i$ depends on: i) relevance of C, and every $\mathcal{A}_i$-child of C; ii) relevance of every document in C, *i.e.,* rel(q, d); iii) some query independent factors: $|C'_{\mathcal{D}}|$, $|C_{\mathcal{D}}|$, and $|\mathsf{chd}_{\mathcal{A}_i}(C)|$.

For the query independent factors, we can pre-compute and materialize them in the Text Cube, so that

we do not have to scan all the documents in C on-the-fly. However, we still need to scan every *relevant* document when the query comes. Moreover, each relevant document should be aggregated to all the cells C' it belongs to for computing $\text{Rel}(\mathsf{q}, C')$. After we scan all the relevant documents, exact significance scores can be computed and the dimensions with top-k highest scores will be output. Because multiple aggregation operations are performed for each relevant document, we call this algorithm MultiAccess.

Algorithm 1 MultiAccess Algorithm

Input: keyword query q, starting cell C, parameter k.

```
1: d = FetchNextRelevant(q, C);
2: while d is not NULL do
3:    Update C with d;
4:    for each aggregated dimension A_i of C do
5:       Update d's cell C' ∈ chd_{A_i}(C) with d;
6:    d = FetchNextRelevant(q, C);
7: for each aggregated dimension A_i of C do
8:    Compute Sig_{A_i}(q, C);
9: Output dimensions A_i with the top-k highest Sig_{A_i}(q, C);
```

Analysis Suppose there are m different aggregated dimensions in C, and n relevant documents, then the time complexity of the MultiAccess algorithm is $O(m \times n)$. When n is very large, this algorithm runs very slow. Also notice that if we do not pre-compute the query-independent factors like the number of documents in C, then those factors can only be computed by scanning all documents in the cell, meaning the time complexity for every query is $O(m \times |C_{\mathcal{D}}|)$, which is not acceptable.

4.2 OneAccess Algorithm

One major overhead of the MultiAccess algorithm is that multiple aggregation operations are performed for each relevant document. To alleviate this problem, we naturally exploit the pre-computability of the scores by materializing some statistics about the terms offline in the Text Cube, so that when online queries come, the scores can be computed based on materialized results very efficiently.

From Equation (3.3) we can see the only factor that causes multiple aggregation of documents is the relevance of children cells of C, *i.e.,* $\text{Rel}(\mathsf{q}, C')$. Since users could query on any cell in the Cube, we now discuss how to precompute relevance signals for any cell in the Text Cube.

Precomputation of Relevance To compute the relevance of a cell C w.r.t. a keyword query q, without any precomputation, we need to scan each document in $C_{\mathcal{D}}$ at least once. In the following, we show that, for the general form of cell relevance defined in Section 2.2, we need only $O(1)$ space per term in each cell for precomputation, so that the relevance of the cell C, $\text{Rel}(\mathsf{q}, C)$, can be computed in $O(|\mathsf{q}|)$ time.

For the "small document" cell relevance model defined in (2.1), we can rewrite it as follows:

$$\begin{aligned}\mathrm{Rel}(\mathsf{q}, C) &= \frac{1}{|C_{\mathcal{D}}|} \sum_{\mathsf{d} \in C_{\mathcal{D}}} \sum_{w \in \mathsf{q}} \mathsf{IDF}_w \cdot \mathsf{TFW}_{w,\mathsf{d}} \cdot \mathsf{QTW}_{w,\mathsf{q}} \\ &= \frac{1}{|C_{\mathcal{D}}|} \sum_{w \in \mathsf{q}} \mathsf{IDF}_w \cdot \mathsf{QTW}_{w,\mathsf{q}} \cdot \left(\sum_{\mathsf{d} \in C_{\mathcal{D}}} \mathsf{TFW}_{w,\mathsf{d}} \right)\end{aligned}$$

Let $\mathsf{TFW}_{w,C} = \sum_{\mathsf{d} \in C_{\mathcal{D}}} \mathsf{TFW}_{w,\mathsf{d}}$. If for each term w, we precompute IDF_w, and in each cell C, we precompute $\mathsf{TFW}_{w,C}$, then from the above equation, $\mathrm{Rel}(\mathsf{q}, C)$ can be computed in $O(|\mathsf{q}|)$ time.

For the "large document model", the precomputation is straightforward: we concatenate the documents in each cell offline and store the pseudo document in the cell. Computing $\mathrm{Rel}(\mathsf{q}, C)$ is then equivalent to computing the relevance of the pseudo document, which takes $O(|\mathsf{q}|)$ time.

The precomputaion needs O(1) space per term for each cell, so the total space cost of the text cube is $O(|V| \times |Cube|)$, where $|V|$ is the size of the vocabulary, and $|Cube|$ is the number of non-empty cells in the Cube, bounded by $2^m \times \#(\text{Base Cells})$, m is the number of dimensions. For a text database that does not have very high number of dimensions, *e.g.,* 10 dimensions, but have a large number of documents, the number of non-empty cells is comparable to the number of documents, so the extra space needed is comparable to the size of inverted index built on the document collection, which is quite affordable.

Moreover, there have been extensive studies on reducing the space cost and supporting efficient query processing in a *partial materialized* cube [16, 20], where only some cells of the cube are precomputed and the rest ones can be efficiently computed online based on them. Those techniques can be easily implemented in our system to further save the storage, since the measures stored in the text cube satisfy the *distributive* property [16]. In the remainder of the thesis, however, we assume our cube is fully materialized since how to reduce the space is not the main focus of this work.

Online Computation We have showed with precomputation the relevance of cells can be computed very efficiently on-the-fly. And based on those scores we can compute the cell variance $\mathsf{CV}(\mathsf{q}, C)$ (3.1) directly. Next, we further decompose the $\mathsf{IDV}(\mathsf{q}, C)$ and rewrite Equation (3.3):

$$\mathsf{Sig}_{\mathcal{A}_i}(\mathsf{q}, C) = \frac{\mathsf{CV}_{\mathcal{A}_i}(\mathsf{q}, C) \cdot (|C_{\mathcal{D}}| - |\mathsf{chd}_{\mathcal{A}_i}(C)|)}{\sum_{C' \in \mathsf{chd}_{\mathcal{A}_i}(C)} (|C'_{\mathcal{D}}| \mathrm{Rel}(\mathsf{q}, C')^2 - 2 S_{C'} \mathrm{Rel}(\mathsf{q}, C')) + SS_C} \tag{4.1}$$

where $S_{C'} = \sum_{\mathsf{d} \in C'} \mathrm{rel}(\mathsf{q}, \mathsf{d})$, and $SS_C = \sum_{\mathsf{d} \in C} \mathrm{rel}(\mathsf{q}, \mathsf{d})^2$.

$S_{C'}$ is the sum of the relevance scores of documents in cell C', so it can also benefit from the precomputation. In fact, $S_{C'}$ just equals to $|C'_{\mathcal{D}}| \cdot \mathrm{Rel}(\mathsf{q}, C')$ if the average relevance model (2.1) is used. Then the only factor left unknown in Sig is SS_C, the sum of squares of each document's relevance score. And this

factor can not be easily pre-computed unless bigrams are considered, which would cause too much storage overhead. Therefore, in the online query processing we still need to scan all relevant documents, but this time we only need $O(1)$ operation for each document to compute SS_C. We call this algorithm OneAccess.

Algorithm 2 OneAccess Algorithm

Input: keyword query q, starting cell C, parameter k.

```
1: Compute Rel(q, C);
2: for each aggregated dimension A_i of C do
3:    for each C' ∈ chd_{A_i}(C) do
4:       Compute Rel(q, C');
5: d = FetchNextRelevant(q, C);
6: while d is not NULL do
7:    SS_C = SS_C + rel(q, d)^2;
8:    d = FetchNextRelevant(q, C);
9: for each aggregated dimension A_i of C do
10:   Compute Sig_{A_i}(q, C) using SS_C, etc;
11: Output dimensions A_i with the top-k highest Sig_{A_i}(q, C);
```

Analysis Since for each relevant document OneAccess only needs $O(1)$ operation, the time complexity is $O(n + m)$, if there are n relevant documents and m dimensions. When m or n is large, OneAccess will be significantly faster than MultiAccess. Another benefit of OneAccess is that the relevance of cells can be computed without scanning any document, so the ranking of top cells in each dimension can be performed very efficiently.

4.3 OneAccess+ Algorithm

OneAccess needs to scan all relevant documents once in order to compute the exact significance scores. However in the case only top-k dimensions are desired by users, we do not have to compute the exact scores if the top-k is guaranteed.

Specifically, in OneAccess+ we progressively fetch documents in the descending order of their relevance scores, which can be easily supported by the underlying full text search component. Then we can estimate the upper and lower bounds of the scores, and rank all dimensions in two lists: $\mathcal{L}_{UB}$ and $\mathcal{L}_{LB}$, based on their upper bounds and lower bounds respectfully. We update the bounds and the ranked lists every time we fetch new documents. When the top-k dimensions with highest values in $\mathcal{L}_{UB}$ and $\mathcal{L}_{LB}$ are the same, and the k-th lower bound is greater than or equal to the $(k+1)$-th upper bound, it means no dimension in $\mathcal{L}_{UB}$ that ranks lower than k could eventually win any top-k dimension in $\mathcal{L}_{LB}$, *i.e.,* the top-k is guaranteed and the algorithm can stop.

Upper and Lower Bounds of Sig As we have discussed in the OneAccess algorithm, all factors in (4.1)

can be computed efficiently from offline materialized scores without scanning any relevant documents, except the sum of squares of document relevance scores $SS_C = \sum_{\mathsf{d} \in C} \text{rel}(\mathsf{q}, \mathsf{d})^2$. So if we treat factors that have been calculated as constants, it is easy to see the Sig score monotonically decreases with SS_C. Also we know the documents can be fetched in a non-increasing order of their relevance. Based on these facts, we can derive the upper and lower bounds of Sig as follows.

Suppose at the current step we are accessing the i-th document, then an obvious lower bound $\underline{SS_C}$ is given by the current SS_C score we have aggregated:

$$\underline{SS_C} = \sum_{j=1}^{i} \text{rel}(\mathsf{q}, \mathsf{d}_j)^2 \tag{4.2}$$

Then the Sig score computed by using lower bound $\underline{SS_C}$ is the upper bound.

To derive the upper bound of SS_C, we first need to know the number of relevant documents for the current query. While the exact number is not easy to get without scanning all the relevant documents, the total number of documents in the current starting cell $C_{\mathcal{D}}$ is obviously too loose. The solution is we can offline compute and materialize the document frequency of w in C, *i.e.,* number of documents in C that contain w, denoted as $df_{w,C}$. Then for each w in the query q, the sum of $df_{w,C}$ gives an upper bound of the number of relevant documents. And since we know the relevance of future documents is not greater than the current $\text{rel}(\mathsf{q}, \mathsf{d}_i)$. The upper bound of SS_C can be estimated by:

$$\overline{SS_C} = \sum_{j=1}^{i} \text{rel}(\mathsf{q}, \mathsf{d}_j)^2 + (\min\{\sum_{w \in \mathsf{q}} df_{w,C}, |C_{\mathcal{D}}|\} - i) \cdot \text{rel}(\mathsf{q}, \mathsf{d}_i)^2 \tag{4.3}$$

And the lower bound of Sig can be computed using $\overline{SS_C}$.

Analysis OneAccess+ exploits upper and lower bounds of the significance scores to enable early termination of the algorithm when top-k is guaranteed. When the bounds are tight, and k is small, it could lead to more efficient execution than MultiAccess and OneAccess. However, every time we get a new relevant document, the upper and lower bounds for each dimension should be updated, which takes $O(m)$, and the ranked lists may need to be adjusted, which takes expected $O(m \log m)$ time. Hence the time complexity of OneAccess+ is $O(n \cdot m \log m)$ for n relevant documents and m dimensions, which is worse than OneAccess. Several heuristics could be applied to improve the running time, *e.g.,* only updating the bounds after fetching a batch of t documents.

Algorithm 3 OneAccess+ Algorithm

$\mathcal{L}_{UB}$ and $\mathcal{L}_{LB}$: ranked lists of dimensions (in the non-increasing order of upper bound and lower bound of $\mathsf{Sig}(\mathsf{q}, C)$ respectfully)
Input: keyword query q, starting cell C, parameter k.

```
1: Compute Rel(q, C);
2: for each aggregated dimension A_i of C do
3:    for each C' ∈ chd_{A_i}(C) do
4:       Compute Rel(q, C');
5: d = FetchNextRelevant(q, C);
6: while d is not NULL do
7:    Update overline(SS_C) and underline(SS_C);
8:    for each aggregated dimension A_i of C do
9:       Update overline(Sig_{A_i}(q, C)) using underline(SS_C);
10:      Update Sig_{A_i}(q, C) using overline(SS_C);
11:      Update overline(L_UB) and L_LB;
12:   if L_UB[1..k] == L_LB[1..k] and L_LB[k].score ≥ L_UB[k+1].score then
13:      break;
14:   d = FetchNextRelevant(q, C);
15: Output top-k dimensions L_UB[1..k];
```

4.4 OneAccess++ Algorithm

OneAccess+ deploys standard top-k stop conditions which work for general ranking functions [3]. However, our top-k problem is quite different from previous top-k aggregation problems [3, 1, 14]. In previous problems, each object is aggregated from a set of tuples in the database, and to get the top-k objects, the algorithms give higher priorities to the promising objects so that their exact scores can be computed first. However, in our problem, the Sig scores for different dimensions depend on all relevant documents in the cell. Computation of the exact scores requires scanning all the relevant documents, which is what we want to prevent.

More specifically, for the Sig score in Equation (4.1), if all pre-computable factors are already calculated, they can be treated as constant. Therefore, the significance scores for different dimensions can be simplified as a series of functions with different parameters but the same variable, which is the unknown factor SS_C (it is unknown before all relevant documents are scanned):

$$\mathsf{Sig}_{\mathcal{A}_i}(SS_C) = \frac{p_{\mathcal{A}_i}}{q_{\mathcal{A}_i} + SS_C} \tag{4.4}$$

Then we can exploit a good property of this family of functions to achieve stronger pruning power.

Property 1. *f^1, f^2, ..., f^n are a series of functions of x. For any two functions f^s and f^t ($1 \le s < t \le n$), there exists no $x_i < x_k < x_j$, s.t. $f^s(x_i) < f^t(x_i)$, and $f^s(x_j) < f^t(x_j)$, but $f^s(x_k) > f^t(x_k)$.*

Property 1 says if the values of any two functions at x_i, x_j rank in the same order, then such order remains for the values of the two functions at any point between x_i and x_j. It is equivalent to say that any

two functions do not cross more than once between any x_i and x_j.

According to Equation (4.4), $\mathsf{Sig}_{\mathcal{A}_i}(SS_C)$ represents a family of functions of SS_C. And it is easy to verify that any two functions in this family with different parameters can only have at most one intersection, *i.e.*, $\mathsf{Sig}_{\mathcal{A}_i}(SS_C)$ satisfies Property 1.

Now we give the lemma which is the key of this OneAccess++ algorithm.

Lemma 1. *Given a series of functions of x: f^1, f^2, ..., f^n that satisfy Property 1. If x is bounded between $lower(x)$ and $upper(x)$, and f functions with top-k highest values at $lower(x)$ are the same as f functions with top-k highest values at $upper(x)$; then the top-k remain the same at any point x between $lower(x)$ and $upper(x)$.*

Proof. Let us assume the lemma is not true, *i.e.*, there exists a function f', s.t. f' is not in top-k either at $lower(x)$ nor at $upper(x)$, but in top-k at some point x' which is between $lower(x)$ and $upper(x)$. Then there must exist a function f'' which is in top-k at both $lower(x)$ and $upper(x)$, but $f''(x') < f'(x')$. On the other hand, since the values of f'' are greater than f' at both $lower(x)$ and $upper(x)$, and Property 1 is satisfied for all functions in the series, we can derive that $f''(x') > f'(x')$. Contradiction. □

Algorithm 4 OneAccess++ Algorithm

$\mathcal{L}_{UB}$ and $\mathcal{L}_{LB}$: ranked lists of dimensions (in the non-increasing order of upper bound and lower bound of $\mathsf{Sig}(\mathsf{q}, C)$ respectfully)
Input: keyword query q, starting cell C, parameter k.

```
1: Compute Rel(q, C);
2: for each aggregated dimension A_i of C do
3:    for each C' ∈ chd_{A_i}(C) do
4:       Compute Rel(q, C');
5: d = FetchNextRelevant(q, C);
6: while d is not NULL do
7:    Update overline(SS_C) and underline(SS_C);
8:    for each aggregated dimension A_i of C do
9:       Update overline(Sig_{A_i}(q, C)) using underline(SS_C);
10:      Update Sig_{A_i}(q, C) using overline(SS_C);
11:      Update overline(L_UB) and L_LB;
12:   if L_UB[1..k] == L_LB[1..k] then
13:      break;
14:   d = FetchNextRelevant(q, C);
15: Output top-k dimensions L_UB[1..k];
```

Since $\mathsf{Sig}_{\mathcal{A}_i}(SS_C)$ is a series of functions of SS_C satisfying Property 1, and SS_C is bounded by its lower bound $\underline{SS_C}$ and upper bound $\overline{SS_C}$. Therefore the real top-k $\mathsf{Sig}_{\mathcal{A}_i}(SS_C)$ scores are achieved when SS_C reaches the exact value, which is some point between $\underline{SS_C}$ and $\overline{SS_C}$. Lemma 1 tells us, if the top-k dimensions $\mathcal{A}_i$ with highest $\mathsf{Sig}_{\mathcal{A}_i}(\underline{SS_C})$ and $\mathsf{Sig}_{\mathcal{A}_i}(\overline{SS_C})$ are the same, then they are the real top-k dimensions.

Therefore, our OneAccess++ algorithm is generally the same as OneAccess+. The only difference is the stop-condition for top-k. OneAccess++ stops as soon as the top-k dimensions in the lower bound and upper bound lists are the same. The condition that the k-th lower bound is greater than or equal to the $k+1$-th upper bound is no longer required. Since the stop condition is relaxed, OneAccess++ is guaranteed to be more efficient than OneAccess+. Also note that OneAccess++ will generate the exactly same top-k results as OneAccess+.

Analysis Since OneAccess++ is generally the same as OneAccess+ except the stop condition is different, the time complexity of the two algorithms are identical. But in practice, OneAccess++ is guaranteed to be faster.

Chapter 5

Experiments

In this chapter, our goal is to (i) verify the effectiveness of our ranking mechanisms and (ii) analyse the performance of our proposed algorithms, and storage of the text cube.

Dataset We crawled customer reviews for laptops from Google Products[1]. The dataset has 26,418 reviews, 920 laptops, and 11 dimensions: Audio Card, Battery Run Time, Brand, Screen Type, Color, Weight, Operating System, CPU, Hard Drive, Main Memory and Video Card.

Environment Setup All experiments were done on a machine running Windows 7 Professional, with a Inter Core Duo T9300 processor, 4GB main memory, and 80G hard disk. The algorithms were implemented in C++ and compiled with Microsoft Visual C++ 2008.

5.1 Effectiveness

In this section we focus on studying the effectiveness of our proposed significance measure for ranking dimensions. For the relevance functions of documents and cells, there have been extensive studies in the field of information retrieval, and TEXplorer can flexibly adopt most state-of-the-art relevance models. Therefore we will not focus on evaluating different relevance functions. In the following experiments, we use Okapi BM25 [21] for document relevance and the average model (Equation 2.1) for cell relevance, which work quite well on our dataset.

5.1.1 Measures in Faceted Search

We compare our significance measure with the dimension ranking functions in two recent faceted search systems [17, 6] defined as follows:

i) [17] targets on building decision trees with minimum height on the tuples, so that users' efforts to reach each individual tuple can be minimized. The Indg score of dimension $\mathcal{A}_i$ measures the number

[1] http://products.google.com

of *indistinguishable* pairs of tuples if choosing $\mathcal{A}_i$ as the root, and the algorithm greedily selects the dimension with minimum Indg to build the tree.

$$\mathsf{Indg}_{\mathcal{A}_i}(\mathsf{q}, C) = \sum_{C' \in \mathsf{chd}_{\mathcal{A}_i}(C)} \left(\sum_{\mathsf{d}_i, \mathsf{d}_j \in C'_{\mathcal{D}}} \mathrm{rel}(\mathsf{q}, \mathsf{d}_i) \times \mathrm{rel}(\mathsf{q}, \mathsf{d}_j) \right) \tag{5.1}$$

ii) [6] estimates the probability of the query results using a *hyper-geometric distribution*: let $|C_{\mathcal{D}}|$ be the number of documents in $C_{\mathcal{D}}$, and $|C_{\mathcal{D}}(\mathsf{q})|$ be the numbers of relevant documents w.r.t. q in $C_{\mathcal{D}}$. For an $\mathcal{A}_i$-child of C, say C', if we randomly sample $|C_{\mathcal{D}}(\mathsf{q})|$ documents from $C_{\mathcal{D}}$, the p-value of getting at least $|C'_{\mathcal{D}}(\mathsf{q})|$ documents from $C'_{\mathcal{D}}$ in the sample can be written as:

$$p_{\mathcal{A}_i, C'}(\mathsf{q}, C) = 1 - \sum_{s=0}^{|C'_{\mathcal{D}}(\mathsf{q})|-1} \frac{\binom{|C'_{\mathcal{D}}|}{s} \binom{|C_{\mathcal{D}}| - |C'_{\mathcal{D}}|}{|C_{\mathcal{D}}(\mathsf{q})| - s}}{\binom{|C_{\mathcal{D}}|}{|C_{\mathcal{D}}(\mathsf{q})|}} \tag{5.2}$$

A smaller p-value indicates it is less likely to get the results by chance, and therefore the cell C' is more *interesting*. And the overall *interestingness* of a dimension $\mathcal{A}_i$ is the aggregation of p-values of the top-k interesting $\mathcal{A}_i$-children of C:

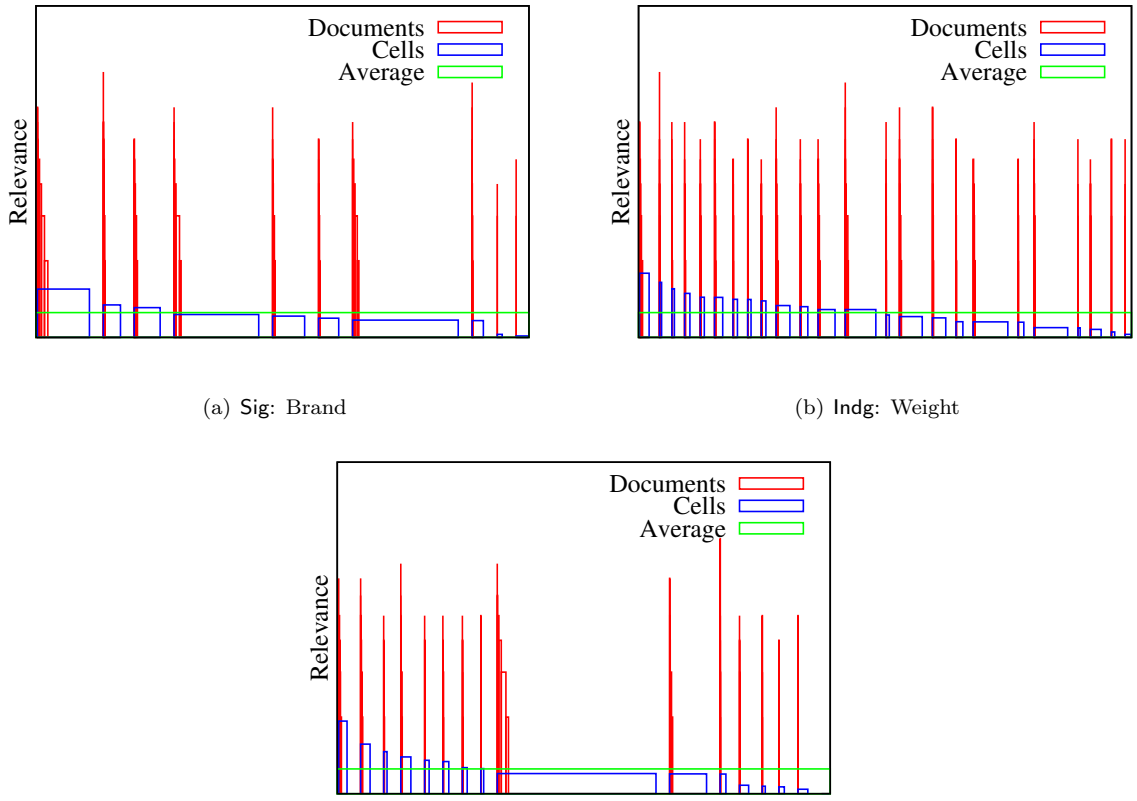
$$\mathsf{Intr}_{\mathcal{A}_i}(\mathsf{q}, C) = -\sum_{t=0}^{k} \log p_{\mathcal{A}_i, C'_t}(\mathsf{q}, C) \tag{5.3}$$

We can see the two measures defined above more focus on the tuple level results. Indg targets on distinguishing individual tuples, which may not be very effective for users to understand the data if they are more satisfied with higher level aggregated results. The p-value and Intr utilize the distribution of result tuples, but the relevance scores of tuples and cells are not considered at all. Therefore, these two measures may not be well suited for our tasks in TEXplorer.

5.1.2 Case Studies

We issue a keyword query "stylish beauty cool fashion" on all documents in our dataset. The top-1 dimension ranked by Sig is Brand, and the corresponding top relevant cell is (Brand = Apple), which is quite reasonable since Apple laptops are well-known for the stylish design. However, the Brand dimension only ranks the 9th and the 10th according to Indg and Intr respectively, while the top-1 dimensions according to those two measures are Weight and CPU, which are quite difficult to interpret.

Figure 5.1 plots the documents and cells for each top-1 dimension. Height of red and blue bars indicates the relevance of documents and cells, and the green bar represents the average relevance of all the documents, which remains the same. Width of the blue bars represents the number of documents in the cells. In



(a) Sig: Brand

(b) Indg: Weight

(c) Intr: CPU

Figure 5.1: Top-1 dimension given by Sig, Indg, and Intr for query "stylish beauty cool fashion"

Figure 5.1(a), we can see the top (Brand=Apple) cell dominates other cells and the documents are rather consistent compared to the other two dimensions, and therefore its Sig score is very high. Figure 5.1(b) shows Indg tends to give higher scores to dimensions that have more cells but each cell has less documents, so that the number of filtered tuples is maximized after users select one cell. However, the results lack semantic interpretation if users prefer higher level objects rather than individual tuples. Intr does not utilize the relevance scores at all. Although in our experiments cell (Brand=Apple) has quite high p-value compared to other Brand-children, more CPU-children get higher p-values and thus the CPU dimension ranks top-1, which is not very meaningful either.

We can show the effectiveness of our method using several more queries. The top-1 dimensions given by the three measures Sig, Indg and Intr are put in Table 5.1. For our proposed Sig measure we also show the top relevant cell in the corresponding subspace. Results of Sig are generally more meaningful than the other two measures. Take the first query for instance, Screen Type is very important to gaming, and the top relevant cell is a fairly large screen, meaning users who bought this screen mentioned gaming experience more

Query	Sig	Indg	Intr
game player	Screen Type (16" TFT)	Weight	Processor
runtime error	OS (MaxOS 10.5)	Weight	Display Type
slow response	Processor (Celeron M353)	OS	Video
long hour	Battery Time (9.5 hour)	Color	Processor

Table 5.1: Top-1 Dimensions given by Sig, Indg and Intr

frequently in their reviews. However, Indg and Intr still rank Weight and Processor as the top-1 dimensions, same as the results of a quite different query in Figure 5.1, which is probably because these two measures do not utilize much the distribution of the relevance scores.

5.1.3 Quantitative Evaluation

To quantitatively justify the effectiveness of our method, we adapt the most popular IR evaluation mechanism: labelling dimensions as relevant or non-relevant for a given query and compute several measures such as precision and MAP against the results. We want to compare with other state-of-the-art systems and see whether our system can provide immediate helpful guides, so in this experiments, we evaluate the first stage of the exploration (immediately after keyword query issued). To get the ground truth, we choose 20 common queries and ask 3 users to label which dimensions are meaningful w.r.t. the queries. Then we take the intersection of their labelled sets as the ground truth, which is quite conservative but can ensure us labelled dimensions are really relevant. Actually, we found the 3 users agree on more than 90 percent of the dimensions, which means the meaningful dimensions are quite obvious given a certain keyword query. For example, for the query "stylish", Brand is obviously a relevant dimension.

Then we compute two most important measures in IR evaluation: mean average precision (MAP) and the precision at top 3 against the dimensions output by Sig, Indg, and Intr measures. Table 5.2 shows the averaged results over 20 queries: unsurprisingly Sig achieves the best MAP and precision among the three measures by a significant margin; the MAP of Intr and Indg are close while the precision of Intr is higher than Indg.

-	Sig	Indg	Intr
MAP	**0.662**	0.430	0.468
Precision@3	**0.467**	0.233	0.367

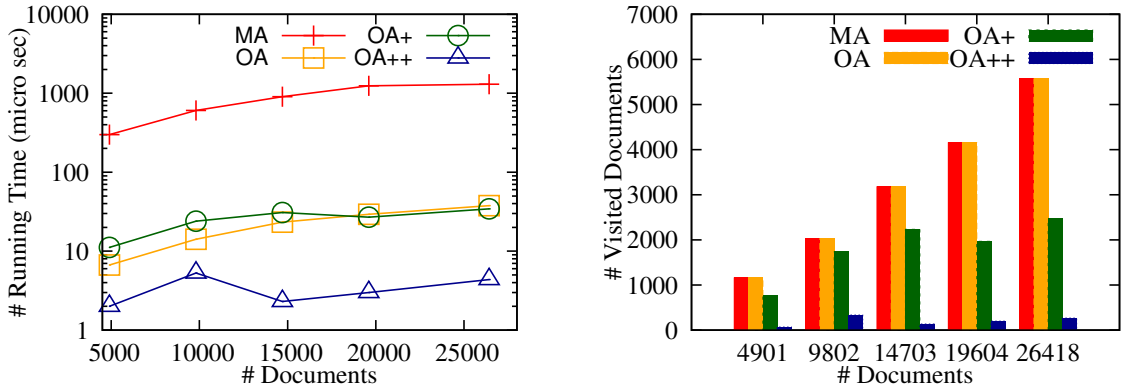
Table 5.2: MAP and Precision@3 of Sig, Indg and Intr

5.2 Efficiency

Now we analyse the performance of the proposed algorithms. In the experiments we issue 10 queries of length 3 and compare the average running time and visited documents of each algorithm. Note that for fair comparison we do not count the time of loading index and fetching relevant documents, since all algorithms assume it is efficiently supported by underlying full text search modules. We also loop the algorithms for multiple times and take the average in order to measure the running time more accurately.

5.2.1 Efficiency vs. Number of Documents

To verify the scalability of the algorithms, we take 4 samples of 4901, 9802, 14703, 19604 documents from our dataset and run the algorithms on the 4 samples and the whole dataset with 26418 documents. All 11 dimensions are considered, and the k is set to 3 for top-k algorithms.



(a) Running Time

(b) Visited Documents

Figure 5.2: Performance for Different # Documents

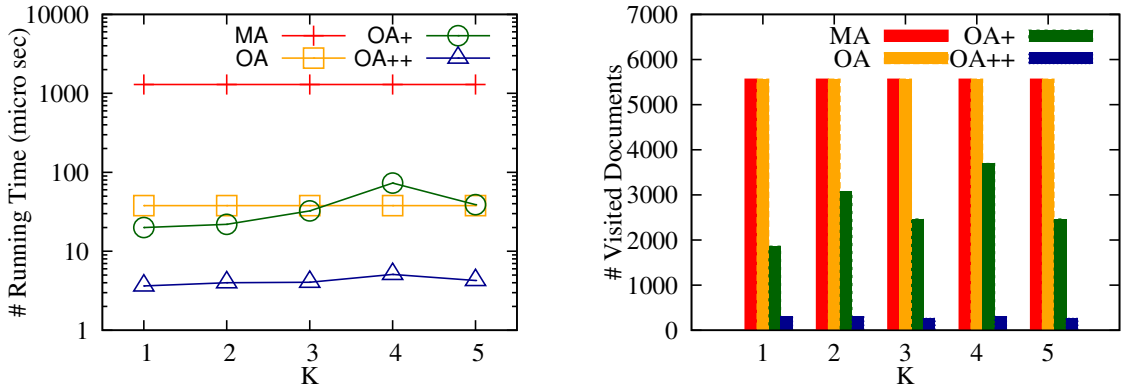
If we look at the number of visited documents in Figure 5.2(b). MultiAccess and OneAccess always visit all the relevant documents as expected, which grow linearly with the number of all documents in the database. OneAccess+ visits less documents by utilizing the upper and lower bounds of Sig to compute the top-k dimensions, and the documents OneAccess+ accesses grow slower than the total number of relevant documents, which means the pruning power of OneAccess+ becomes stronger as the size of the database increases. OneAccess++ visits significantly less documents than all the other three algorithms, and the number is almost stable. This verifies the unique property of our top-k problem does deliver stronger pruning power. Also note that top-k algorithms OneAccess+ and OneAccess++ also has a good property that they may actually visit less documents when the total number of document increases, because when there are more relevant documents it is possible that the gap between top-k results and the rest becomes

larger, so the algorithms can confidently stop earlier.

For the running time shown in Figure 5.2(a), MultiAccess is slower than OneAccess even they visit the same number of documents, since multiple aggregation is performed for each document in MultiAccess. Also notice that OneAccess+ is also slower than OneAccess on the smallest samples because of the overhead for updating the bounds and maintaining the top-k lists. When the size of the database is large, the pruning power begins to dominate and therefore OneAccess+ becomes slightly faster than OneAccess. OneAccess++ is consistently the fastest as expected, and it generally runs more than 10 times faster than OneAccess and OneAccess+ except for the first two smaller samples. This again verifies the good performance of OneAccess++.

5.2.2 Efficiency vs. Top-k

In this experiment, we vary the length of the top list, *i.e.*, k, and evaluate the efficiency for each algorithm. All the documents in the dataset and all 11 dimensions are considered. We evaluate 10 queries of length 3 and report the average running time and number of visited documents in Figure 5.3.



(a) Running Time

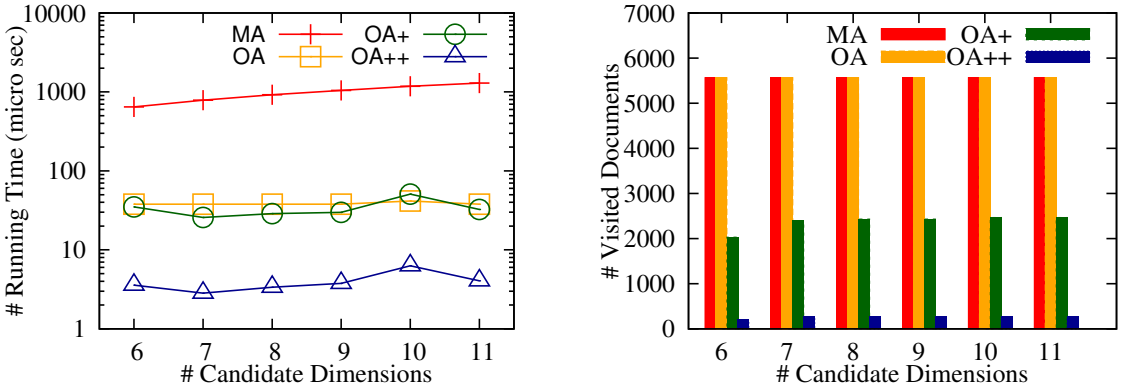
(b) Visited Documents

Figure 5.3: Performance for Different Top K

First Figure 5.3(b) shows MultiAccess and OneAccess always visit all the relevant documents, which do not change for different k. The number of documents OneAccess+ accesses fluctuates, since it may be difficult to predict the stop condition of OneAccess+, but OneAccess+ still visits less documents than MultiAccess and OneAccess. Among all the algorithms, OneAccess++ accesses the least documents, and the performance is quite stable for different k compared to OneAccess+. The running time in Figure 5.3(a) shows the same trend as Figure 5.3(b), and the running time of OneAccess++ is consistently the fastest among all the algorithms.

5.2.3 Efficiency vs. Number of Dimensions

In this experiment, we use all the documents in our dataset, but vary the number of candidate dimensions (*i.e.*, dimensions considered to be ranked). k is set to 3, and the same 10 queries of length 3 as in previous experiments are evaluated. We report the average running time and number of accessed documents in Figure 5.4.



(a) Running Time

(b) Visited Documents

Figure 5.4: Performance for Different # Candidate Dimensions

Figure 5.4(b) shows the number of document visited by each algorithms is quite stable w.r.t. the number of dimensions: MultiAccess and OneAccess always visit all relevant documents as expected, OneAccess+ generally visits less than a half and OneAccess+ consistently visits less than 5% of the relevant documents. The fixed k in this experiment is probably one of the reasons why the pruning power of the two top-k algorithms is stable.

As observed in Figure 5.4(a), the running time of all algorithms follows the similar patterns as in previous experiments: MultiAccess is the slowest, and OneAccess+ is faster than OneAccess when the number of dimensions is small, and with the growth of candidate dimensions the overhead of OneAccess+ increases. And OneAccess++ is consistently around 10 times faster than OneAccess and OneAccess+.

5.3 Storage

We now examine the space overhead of the text cube in our system. We use all the 11 dimensions and vary the number of documents to build the cube, and compare its size with inverted index built on the document collection (Figure 5.5). Although the size of text cube is several times larger, it increases much slower w.r.t. the number of tuples, which means the summarization power of text cube becomes stronger when the size of original databases gets larger. Given in our experiments we are only able to get a small portion of the

real world customer review data, we could image the size of text cube would be comparable with inverted index on much larger datasets, in which sense our solution is quite scalable.

Also notice there have been extensive studies on saving the space of a data cube using partial materialization [16, 20]. Since it is not the main focus in this thesis, we construct the full cube in the experiments. However, those techniques can be easily applied to reduce the space substantially.

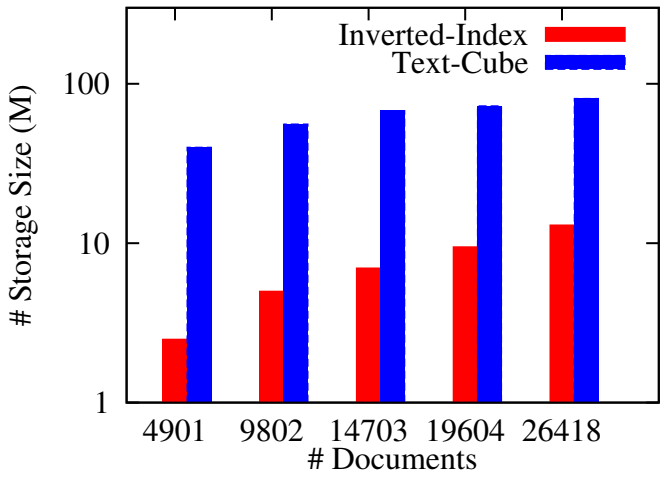


Figure 5.5: Space Overhead vs. Documents

Chapter 6

Related Work

In this chapter we discuss previous work related to this thesis.

6.1 Faceted Search

Faceted search systems [17, 6, 11, 12, 10, 2, 15] allow users to flexibly select structured attributes to refine the search results. Facets are ranked based on number of associated documents [10], estimated user effort [17, 11], or p-value of the results [6]. Some systems dynamically generate facets from data [2, 15] or build personalized relevance models [12]. As we have mentioned, TEXplorer is different from previous faceted search systems because it focuses on keyword-based ranking and exploration of aggregated objects rather than individual documents. None of the previous facet ranking measures consider the distribution of IR relevance scores as we do and our experiments show the proposed Sig measure is more effective for our task.

6.2 Discovery-driven OLAP Systems

In traditional data cubes, discovery driven OLAP mechanisms [18] can find surprising or unexpected cells in the cube. [19] further supports matching candidate subspaces by keywords and then dynamically suggests facets by measuring surprising or correlated aggregates of categorical or numerical attributes. TEXplorer is different in the sense that we care about the relevance of cells and correlated dimensions, rather than the distribution of categorical or numerical values in the database. And we target on the most common search behaviors: we suppose users who issue the keyword queries are always interested in the most relevant objects rather than surprising ones, because objects with rather low relevance could still get high surprise scores.

6.3 Object Search in Database

Ranking of aggregated documents for keyword queries have been developed in relational databases [1, 3] and data cubes [23, 7]. TEXplorer ranks objects in the same subspaces so that the comparison is more

meaningful, and the interactive exploration framework organizes different subspaces and rankings in an effective way.

Our algorithms of ranking top-k dimensions are related to top-k query processing in database [9, 14, 3]. [14] supports ad-hoc ranking aggregation for the group-by operations in SQL. The mechanism of progressively aggregating scores to each group is proposed so that the top-k groups can be computed efficiently. [3] also proposes aggregation-based top-k generation and pruning strategies. We identify the unique property of our top-k problem and our algorithms achieve stronger pruning power.

Chapter 7

Conclusions

7.1 Overview

In this work, we propose a novel system TEXplorer that allows users to perform keyword search and OLAP-style aggregation and exploration of the objects in a text cube built on a multidimensional text database. A novel significance measure is proposed to rank dimensions. Top relevant objects/cells in each dimension subspace are also ranked for users to select. We develop top-k algorithms and materialization strategies to accelerate the ranking. Extensive experimental studies verify the scalability of our methods and the effectiveness of our proposed significance measure. TEXplorer can be potentially implemented on top of many text databases like Amazon to improve user experience.

7.2 Future Work

7.2.1 Clustering of Objects

Currently in TEXplorer all objects or cells are considered separately in computing the relevance and significance, even some objects are very similar, *e.g.,* 16" and 17" screens can be both considered as "large" screens, and users often do not have preferences in selecting a particular one. In this case, if we can automatically cluster similar cells or allow users flexibly group cells, then relevance and significance scores based on these object clusters could probably be more meaningful. In terms of computation, current system can support the selection of a set of objects by aggregating the cells on-the-fly. If some cells are clustered offline with high confidence, the clusters can also be materialized in text cube.

7.2.2 Advanced Relevance Models

Currently TEXplorer supports bag-of-words relevance models. Although these models still work very well in general, we can also consider other signals for computing relevance. For example, to support the proximity of keywords, we can mine frequent phrases from the texts and in text cube cells we can materialize phrases

that are more correlated or more likely to be queried, which could reduce the space overhead. To judge the relevance of a cell we can also give different weights to different documents based on whether they are important, etc.

7.2.3 User Feedback and Query Suggestion

When users interact with the system their behaviors can be utilized to compute the relevance and significance. For example, if the user selects one object then decides to roll up and explore another dimension, it probably means that object is not very relevant and that dimension is not very significant. Weights for terms that appear very frequently in those objects can also be adjusted.

In each interaction stage we can also suggest users keywords that can summarize different candidate objects and dimensions, which helps users choose the most effective query. For instance, when users have drilled down to the very detailed level of the cube, instead of still executing their original query at higher levels, the system should also give users suggestion on what are the most representative keywords of in the current subspace so that users would have an even better understanding of the data.

References

[1] Albert Angel, Surajit Chaudhuri, Gautam Das, and Nick Koudas. Ranking objects based on relationships and fixed associations. In *EDBT*, 2009.

[2] Ori Ben-Yitzhak, Nadav Golbandi, Nadav Har'El, Ronny Lempel, Andreas Neumann, Shila Ofek-Koifman, Dafna Sheinwald, Eugene J. Shekita, Benjamin Sznajder, and Sivan Yogev. Beyond basic faceted search. In *WSDM*, 2008.

[3] Kaushik Chakrabarti, Venkatesh Ganti, Jiawei Han, and Dong Xin. Ranking objects based on relationships. In *SIGMOD Conference*, 2006.

[4] Yi Chen, Wei Wang, Ziyang Liu, and Xuemin Lin. Keyword search on structured and semi-structured data. In *SIGMOD Conference*, 2009.

[5] Tao Cheng, Xifeng Yan, and Kevin Chen-Chuan Chang. Entityrank: searching entities directly and holistically. In *VLDB*, 2007.

[6] Debabrata Dash, Jun Rao, Nimrod Megiddo, Anastasia Ailamaki, and Guy M. Lohman. Dynamic faceted search for discovery-driven analysis. In *CIKM*, 2008.

[7] Bolin Ding, Bo Zhao, Xide Lin, Jiawei Han, and Chengxiang Zhai. Topcells: Keyword-based search of top-k aggregated documents in text cube. In *ICDE*, 2010.

[8] Jonathan L. Elsas, Jaime Arguello, Jamie Callan, and Jaime G. Carbonell. Retrieval and feedback models for blog feed search. In *SIGIR*, 2008.

[9] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4), 2003.

[10] Marti Hearst, Ame Elliott, Jennifer English, Rashmi Sinha, Kirsten Swearingen, and Ka-Ping Yee. Finding the flow in web site search. *Commun. ACM*, 2002.

[11] Abhijith Kashyap, Vagelis Hristidis, and Michalis Petropoulos. Facetor: Cost-driven exploration of faceted query results. In *CIKM*, 2010.

[12] Jonathan Koren, Yi Zhang, and Xue Liu. Personalized interactive faceted search. In *WWW*, 2008.

[13] R Kuehl. *Design of Experiments: Statistical Principles of Research Design and Analysis*. Duxbury Press, 2000.

[14] Chengkai Li, Kevin Chen-Chuan Chang, and Ihab F. Ilyas. Supporting ad-hoc ranking aggregates. In *SIGMOD Conference*, 2006.

[15] Chengkai Li, Ning Yan, Senjuti B. Roy, Lekhendro Lisham, and Gautam Das. Facetedpedia: dynamic generation of query-dependent faceted interfaces for wikipedia. In *WWW*, 2010.

[16] Cindy Xide Lin, Bolin Ding, Jiawei Han, Feida Zhu, and Bo Zhao. Text cube: Computing ir measures for multidimensional text database analysis. In *ICDM*, 2008.

[17] Senjuti Basu Roy, Haidong Wang, Gautam Das, Ullas Nambiar, and Mukesh K. Mohania. Minimum-effort driven dynamic faceted search in structured databases. In *CIKM*, 2008.

[18] Sunita Sarawagi, Rakesh Agrawal, and Nimrod Megiddo. Discovery-driven exploration of olap data cubes. In *EDBT*, 1998.

[19] Ping Wu, Yannis Sismanis, and Berthold Reinwald. Towards keyword driven analytical processing. In *SIGMOD Conference*, 2007.

[20] Dong Xin, Jiawei Han, Xiaolei Li, and Benjamin W. Wah. Star-cubing: Computing iceberg cubes by top-down and bottom-up integration. In *VLDB*, 2003.

[21] Cheng Xiang Zhai and Graeme Hirst. *Statistical Language Models for Information Retrieval*. Morgan & Claypool Publishers, 2008.

[22] Bo Zhao, Cindy Xide Lin, Bolin Ding, and Jiawei Han. Texplorer: keyword-based object search and exploration in multidimensional text databases. In *CIKM*, 2011.

[23] Bin Zhou and Jian Pei. Answering aggregate keyword queries on relational databases using minimal group-bys. In *EDBT*, 2009.