KEYWORD SEARCH IN TEXT CUBE: FINDING TOP-K RELEVANT CELLS

BOLIN DING*, YINTAO YU*, BO ZHAO*, CINDY XIDE LIN*, JIAWEI HAN*, AND CHENGXIANG ZHAI*

Abstract. We study the problem of keyword search in a data cube with text-rich dimension(s) (so-called *text cube*). The text cube is built on a multidimensional text database, where each row is associated with some text data (*e.g.*, a document) and other structural dimensions (attributes). A *cell* in the text cube aggregates a set of documents with matching attribute values in a subset of dimensions. A *cell document* is the concatenation of all documents in a cell. Given a keyword query, our goal is to find the top-k most relevant cells (ranked according to the relevance scores of cell documents w.r.t. the given query) in the text cube.

We define a keyword-based query language and apply IR-style relevance model for scoring and ranking cell documents in the text cube. We propose two efficient approaches to find the top-k answers. The proposed approaches support a general class of IR-style relevance scoring formulas that satisfy certain basic and common properties. One of them uses more time for pre-processing and less time for answering online queries; and the other one is more efficient in pre-processing and consumes more time for online queries. Experimental studies on the ASRS dataset are conducted to verify the efficiency and effectiveness of the proposed approaches.

1. Introduction

The boom of Internet and different database systems has given rise to an ever increasing amount of text data associated with multiple dimensions (attributes), which is usually stored in tables. For example, customer reviews in shopping websites (*e.g.*, Amazon) are always stored and associated with attributes like Price, Model, and Rate. In NASA's ASRS database [15], after each commercial flight in the United States, a report is written to describe how the flight went, with several attributes specified, like Weather, Light, Flight Phase, and Event Anomaly.

We have extended a traditional OLAP *data cube* to summarize and navigate structured data together with unstructured text data in [22]. Such a cube model is called *text cube* [22]. A *cell* in the text cube aggregates a set of documents with matching attribute values in a subset of dimensions.

In this paper, we focus on *cell documents*, each of which is the concatenation of all documents in a cell. We study *how to support keyword-based search in text cube*. More specifically, the goal is to: *find the top-k most relevant cells, ranked according to the relevance scores of cell documents w.r.t. the given query, in the text cube*. It provides insights about the relationship between multidimensional attributes and text data.

Example 1.1 (Motivation)**.** Table 1 shows a tiny sample from the ASRS database. It has both structured data (*e.g.*, Weather, Light, flight Phase, and event Anomaly) and Narrative about an anomalous event written by a pilot or flight attendant after each flight as text data.

Suppose Jim, an analyst for flight safety, wants to know under which condition, the *runway excursion* is likely to happen. He types a set of keywords: {"RWY", "EXCURSION"}. Using traditional IR techniques, the system can rank all the narratives (or reports) and output the most relevant ones. However, as there are many reports relevant to the query, Jim have to spend much time browsing through them one by one and summarizing different conditions by himself.

So, is it more desirable that a system provides users with "aggregated information", such as "in a *rainy night*, the runway excursion is likely to happen in the *landing* phase" (Weather = Rainy, Light = Night, Phase = Landing, Anomaly = $*$), instead of returning individual narratives? This is our intention to study such a new mechanism.

*Department of Computer Science, UIUC, {bding3, yintao, bozhao3, xidelin2, hanj, czhai}@uiuc.edu.

Table 1. Motivation Example

Weather	Light	Phase	Anomaly	Narrative
Rainy	Night	Landing	Equipment	. . . RESULTED IN **RWY EXCURSION** DURING ENGINE FAIL . . .
Rainy	Night	Landing	Excursion	SMA **RWY EXCURSION** STRUCK RWY LIGHT . . .
Cloudy	Night	Landing	Excursion	**RWY EXCURSION** DURING TKOF FROM SNOW-SLUSH COVERED **RWY** . . .
Sunny	Daylight	Descent	Equipment	INITIAL WEIGHT AND BALANCE ERROR . . .

A cell in the text cube is in the form of (Weather = Rainy, Light = Night, Phase = Landing, Anomaly = $*$), which aggregates the first two narratives. Cell (Weather = $*$, Light = Night, Phase = Landing, Anomaly = Excursion) aggregates the second and third narratives. Another cell (Weather = Sunny, Light = Daylight, Phase = $*$, Anomaly = $*$) aggregates only the fourth narratives.

It can be seen that the first two cells are more relevant to Jim's query than the third one. The goal of our system is to rank all cells (in different levels and granularities), instead of individual narratives, according to Jim's query.

Given a database of text data (documents) associated with multidimensional attributes, traditional IR techniques to process keyword queries can be used to rank all the *individual documents*; however, they do not fully utilize the association between documents and attributes. Keyword query has also been extended to RDBMSs to retrieve information from text-rich attributes [2, 4, 7, 25, 13, 14, 16, 10, 11, 21, 23, 24, 19, 17, 18] and provide users with relevant *linked structures*: given a set of keywords, existing methods on keyword search in RDBMSs focus on ranking individual tuples from one table or joins of tuples (*e.g.*, linked by foreign keys) from multiple tables that contain the keywords.

This paper studies the problem of *keyword-based top-k search in text cube*, *i.e.*, *given a keyword query, find the top-k most relevant cells in a text cube*. Different from keyword search in plain documents (ranking individual documents) and RDBMSs (ranking relevant linked structures), our ranking objects are *cells*. In a data cube model (a multidimensional space induced by the attributes), *e.g.*, the text cube, a *cell* aggregates the documents with matching values in a subset of attributes. In particular, when ranking cells, we focus on *cell documents*, each of which is the concatenation of all documents in a cell, and evaluate the relevance of this "big document" to the given keyword query for each cell.

A collection of documents (or a "big document", *i.e.*, the concatenation of these documents) is associated with each cell, corresponding to an analytical object (*e.g.*, "landing phase in a rainy night" in Example 1.1). This facilitates the analysis of the relationship between relational attributes and text data, *e.g.*, exploration of relevant cells (objects) w.r.t. a keyword query. When users want to retrieve information from a text cube using keyword queries, we believe that relevant cells, rather than relevant documents, are preferred as the answers, because: (i) relevant cells are easy for users to browse; and (ii) relevant cells provide users insights about the relationship between relational attributes and text data. While most data cube models can support basic operations like roll-up and drill-down, it is unclear how to find the relevant cells using only these operations.

1.1. Overview of Model and Techniques.

Following is an overview of our work.

Ranking Objects and Relevance Score: Given a keyword query, we want to rank all cells in text cube. The first question is how to compute the "*relevance*" of a cell in a text cube for ranking. Note that a cell corresponds to a collection of documents. Consider the following two different models.

- *Average model*: *Any* IR scoring function (*e.g.*, Okapi) can be used to compute the relevance score of each single document w.r.t. the given keyword query, and the *relevance score of a cell* (a document collection) is the average of relevance scores of documents in this cell.

- *Cell document model*: Documents in a cell are concatenated into a "big document", called a *cell document*. The relevance of the cell is the relevance of this cell document w.r.t. the given keyword query.

The two scoring models above carry different semantics: the average model promotes the cells where many documents contain the given keywords; and, the cell document model promotes the cells which contain as many keywords as possible. Two models are suitable in different scenarios and user preferences. It is important to mention that our previous work [8] focuses on the average model, and we will focus on the cell document model in this work. Also note that the model studied in this work is more general in some sense: to efficiently process keyword queries, we only require the relevance scoring function (in Equation (2)) to be *monotone* w.r.t. the term frequencies and the total length of documents in a cell.

Challenges: The first major computational challenge of this keyword search problem is the huge number (increasing exponentially w.r.t. the dimensionality) of cells in a text cube, as we want to rank all cells in different levels and granularities (cuboids). The second computational challenge is, unlike the scoring formula in the average model (satisfying an apriori-like property called *two-side bound property* [8]), we consider the relevance scoring formula in the cell document model, which does NOT satisfy any monotone or apriori-like property in the cube lattice. This difficulty makes the problem studied in this work significantly harder and different from the one in [8].

Efficient Algorithms: Unlike [8], which utilize an apriori-like property (two-side bound property) of scoring formula in the average model to design ranking algorithms, this work introduces two new ranking algorithms which are appliable in a more general class of scoring formula in the cell document model. We design two efficient approaches for the *keyword-based top-k search in text cube*. The first one, TACell, extends the famous TA algorithm [9] to our problem of keyword search in text cube. It requires moderate pre-processing but is efficient in online query processing. The second one, BoundS, estimates upper bounds and lower bounds of the relevance of cell documents in the search space; upper/lower bounds are compared periodically for the early stop of search process, so as to explore as few cells in the text cube as possible before outputting the top-k answers.

1.2. **Contribution and Organization.** In this paper, we study the problem of *keyword-based top-k search in text cube* (or multidimensional text data): *find the top-k cells relevant to a user-given keyword query.* Flexible keyword-based query language and relevance scoring formula of cells (aggregation of text data) are developed based on the cell document model. We propose two efficient approaches, TACell and BoundS, to support the query language in text cube. We also study the effectiveness of the proposed approach in a case study.

Section 2 introduces the text cube model of multidimensional text data, defines the keyword-based query language in text cube, and introduces the relevance scoring formula based on the cell document model. Two algorithms TACell and BoundS are then introduced in Section 3 and 4, respectively, for finding the top-k most relevant cells given a keyword query. Experimental study is reported in Section 5, followed by related work in Section 6. Section 7 concludes this paper.

2. Keyword Queries in Text Cube

We first review our data cube model for multidimensional text data (Section 2.1), then formally define the problem of keyword search in text cube together with the relevance scoring formula based on cell document model (Section 2.2), and analyze the computational challenges (Section 2.3).

2.1. **Preliminary: Text Cube, a Data Cube Model for Text Data.** We first review the *text cube* model introduced in [22, 8], and formally define the cell document model.

A set $\mathbf{D}$ of documents is stored in an *n-dimensional database* $\mathbf{DB} = (\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_n, \mathbf{D})$. An *attribute* $\mathbf{A}_i$ is also said to be a *dimension* in data cube. Each row of $\mathbf{DB}$ is in the form of $r = (a_1, a_2, \ldots, a_n, \mathsf{d})$: let $r[\mathbf{A}_i] = a_i \in \mathbf{A}_i$ be the *value* of *dimension* $\mathbf{A}_i$, and $r[\mathbf{D}] = \mathsf{d}$ be the *document*

Dimensions				Text Data
M	P	T	S	d (Document)
m1	p1	t1	s1	$\mathsf{d}_1 = \{$w1, w1, w2, w2$\}$
m1	p2	t1	s2	$\mathsf{d}_2 = \{$w2, w5, w6$\}$
m1	p3	t2	s2	$\mathsf{d}_3 = \{$w3, w4, w3, w6$\}$
m2	p1	t2	s2	$\mathsf{d}_4 = \{$w1, w1, w1$\}$
m2	p2	t2	s2	$\mathsf{d}_5 = \{$w5, w6, w7, w8$\}$
m2	p3	t1	s1	$\mathsf{d}_6 = \{$w4, w5, w8, w9$\}$

(a) A 4-Dimensional Text Database **DB**

Cell	M	P	T	S	D (Cell Document $C[\mathbf{D}]$)
C_0	*	*	*	*	$\mathsf{d}_1 \circ \mathsf{d}_2 \circ \mathsf{d}_3 \circ \mathsf{d}_4 \circ \mathsf{d}_5 \circ \mathsf{d}_6$
C_1	m1	*	*	*	$\mathsf{d}_1 \circ \mathsf{d}_2 \circ \mathsf{d}_3$
C_2	m2	*	*	*	$\mathsf{d}_4 \circ \mathsf{d}_5 \circ \mathsf{d}_6$
C_3	m1	*	*	s1	d_1
C_4	m1	*	*	s2	$\mathsf{d}_2 \circ \mathsf{d}_3$
C_5	m2	*	t1	*	d_6
C_6	m2	*	t2	*	$\mathsf{d}_4 \circ \mathsf{d}_5$
C_7	m1	*	t1	s1	d_1
C_8	m2	p2	t2	s1	d_1

(b) Some Cells in the Text Cube

in this row. A document d is a multi(sub)set of the *term set* $\mathbf{W} = \{w_1, \ldots, w_M\}$, *i.e.*, a term w_i may appear multiple times in d.

The *data cube* model extended to the above multidimensional text database is called *text cube* [22]. Several important concepts are introduced as follows.

In the text cube built on **DB**, a *cell* is in the form of $C = (v_1, v_2, \ldots, v_n : \mathsf{D})$, where v_i could either be a value of dimension $\mathbf{A}_i$ or be a meta symbol $*$ (*i.e.*, $v_i \in \mathbf{A}_i \cup \{*\}$). If $v_i = *$, the dimension $\mathbf{A}_i$ is aggregated in C. D is the concatenation of the documents in the rows (of the database **DB**) having the same dimension values as C on the non-$*$ dimensions. This "big document" D is called the *cell document* of C. Formally, for a cell $C = (v_1, v_2, \ldots, v_n : \mathsf{D})$,

$$\mathsf{D} = \text{the concatenation of } r[\mathbf{D}]'s, \text{ where for } r \in \mathbf{DB} \text{ s.t. } r[\mathbf{A}_i] = v_i \text{ if } v_i \neq *.$$

We use $C[\mathbf{A}_i]$ to denote the value v_i of dimension $\mathbf{A}_i$ in the cell C, and $C[\mathbf{D}]$ to denote the cell document D of C. To distinguish a document $r[\mathbf{D}]$ in a row of the database **DB** from the cell document $C[\mathbf{D}]$ ($C[\mathbf{D}]$ is the concatenation of some $r[\mathbf{D}]$'s), a document $r[\mathbf{D}]$ in a database row is said to be a *row document* (distinguished from *cell document*). For simplicity, a cell is also written as $C = (v_1, v_2, \ldots, v_n)$. A cell is said to be *empty* if $C[\mathbf{D}] = \varnothing$.

A *cuboid* is a set of cells with the same set of non-$*$ dimensions. A cuboid with m non-$*$ dimensions is an *m-dim cuboid*. The n-dim cuboid (all dimensions are non-$*$) is called the *base cuboid*. Cells in an m-dim cuboid are called *m-dim cells*, and cells in a base cuboid are called *base cells*.

Cell C' is an *ancestor* of C (or C is a *descendant* of C') iff "$\forall i : C'[\mathbf{A}_i] \neq * \Rightarrow C'[\mathbf{A}_i] = C[\mathbf{A}_i]$". Note cell C is an ancestor (or descendant) of itself. We use $\mathsf{ans}(C)$ to denote the set of ancestors of a cell C, and $\mathsf{des}(C)$ to denote the set of descendants of a cell C'. It is well-known that all the cells in a data cube (or text cube) form a *lattice*, according to the ancestor-descendant relationship.

Example 2.1 (Text Cube)**.** Table 1(a) shows a text database **DB**, with four dimensions, M, P, T, and S. Term set $\mathbf{W} = \{$w1, w2, ..., w8$\}$. A total of six documents are stored.

Table 1(b) shows some cells and the corresponding cell documents in the text cube generated from **DB**. Among them, C_3, C_4 are descendants of C_1, and C_5, C_6 are descendants of C_2. Note that C_1, C_2 has some other descendants that are not listed in this table.

We use $\mathsf{d}_1 \circ \mathsf{d}_2$ to denote the concatenation of documents d_1 and d_2. In this example, we have $\mathsf{d}_1 \circ \mathsf{d}_2 = \{\mathrm{w1, w1, w2, w2, w2, w5, w6}\}$. C_0 is the 0-dim cell and C_8 is one of the base cells.

2.2. **Keyword Search Problem in Text Cube.** In traditional data cubes, operations like *drill-down* and *roll-up* suffice for users to explore multidimensional data. However, in text cube, a large portion of data is text. Since keyword query is an effective way for users to explore the text data, we propose the keyword search problem in text cube.

Keyword Search Problem. A *keyword query* is a set of terms, *i.e.*, $\mathsf{q} = \{t_1, t_2, \ldots, t_{|\mathsf{q}|}\} \subseteq \mathbf{W}$. Given a keyword query q, the goal is to *find k cells C's with the top-k highest relevance scores in the text cube of* **DB**.

Note that a cell relevant to the query q may contain all or some of the terms $t_1, \ldots, t_{|\mathsf{q}|}$. The *relevance score* of a cell C w.r.t. the query q is defined as a function $\mathrm{rel}(\mathsf{q}, C[\mathbf{D}])$ of the cell document $C[\mathbf{D}]$ and the query q. For brevity, it is also written as $\mathrm{rel}(\mathsf{q}, C)$. We return the top-$k$ cells in the non-increasing order of relevance scores, because the total number of cells in the text cube could be huge and it is not possible for a user to browse all of them. k can be specified by the user.

Relevance Scoring Formula. To rank all the cells and find the top-k ones, we need to define the relevance scoring function $\mathrm{rel}(\mathsf{q}, C[\mathbf{D}])$ (or $\mathrm{rel}(\mathsf{q}, C)$ for brevity). Here, we treat the cell document $C[\mathbf{D}]$ as a "big document". We compute the relevance score of the cell C w.r.t. a keyword query q as the relevance of this big document w.r.t. q. For example, we can simply apply the Okapi weighting:

$$(1) \quad \mathrm{rel}(\mathsf{q}, C) = \sum_{t \in \mathsf{q}} \ln \frac{N - \mathrm{df}_t + 0.5}{\mathrm{df}_t + 0.5} \frac{(k_1 + 1)\mathrm{tf}_{t,\mathsf{D}}}{k_1((1-b) + b\frac{\mathrm{dl}_{\mathsf{D}}}{\mathrm{avdl}}) + \mathrm{tf}_{t,\mathsf{D}}} \frac{(k_3 + 1)\mathrm{qtf}_{t,\mathsf{q}}}{k_3 + \mathrm{qtf}_{t,\mathsf{q}}}, \text{ (Okapi weighting [27])}$$

where $N = |\mathbf{DB}|$, D is the cell document of C, $\mathrm{tf}_{t,\mathsf{D}}$ is the *term frequency* of term $t \in \mathsf{q}$ in D (the number of times t appearing in D), df_t is the number of documents in $\mathbf{DB}$ containing t, dl_{D} is the length of D, avdl is the average length of cell documents, $\mathrm{qtf}_{t,\mathsf{q}}$ is the number of times t appearing in q, and, k_1, b, k_3 are constants.

Our ranking algorithm can handle a more general form of relevance scoring formula:

$$(2) \quad \mathrm{rel}(\mathsf{q}, C) = s(\mathrm{tf}_1, \mathrm{tf}_2, \ldots, \mathrm{tf}_{|\mathsf{q}|}, |\mathsf{D}|).$$

where tf_i is the term frequency of the i^{th} term of q in the cell document $\mathsf{D} = C[\mathbf{D}]$ of C, and $|\mathsf{D}|$ (dl_{D}) is the length of D. In principle, df_t and $\mathrm{qtf}_{t,\mathsf{q}}$ should also be parameters of the function s; but since they are not critical in our ranking algorithm, we just omit them from (2) for the simplicity.

Note that (1) is a special case of (2), and our ranking algorithms introduced later can handle the general form (2). We require only two basic property of the function s in (2):

- *Monotone w.r.t.* tf_i: From any i,
$$\mathrm{tf}_i \le \mathrm{tf}'_i \;\Leftrightarrow\; s(\mathrm{tf}_1, \ldots, \mathrm{tf}_i \ldots, \mathrm{tf}_{|\mathsf{q}|}, |\mathsf{D}|) \le s(\mathrm{tf}_1, \ldots, \mathrm{tf}'_i, \ldots, \mathrm{tf}_{|\mathsf{q}|}, |\mathsf{D}|).$$
- *Monotone w.r.t.* $|\mathsf{D}|$:
$$l \ge l' \;\Leftrightarrow\; s(\mathrm{tf}_1, \mathrm{tf}_2, \ldots, \mathrm{tf}_{|\mathsf{q}|}, l) \le s(\mathrm{tf}_1, \mathrm{tf}_2, \ldots, \mathrm{tf}_{|\mathsf{q}|}, l').$$

It is important to notice that even though the above two properties about term frequency and document length, which are quite natural for relevance scores, are satisfied, the function rel does not have any monotone or apriori-property in the cube lattice. A simple example is as follows.

Example 2.2 (No Monotone/Apriori Property in the Cube Lattice)**.** Suppose the query q has three terms t_1, t_2, t_3, and there are three cells C_1, C_2, C_3, each of them C_i contains exactly one term t_i. Suppose C is the ancestor of C_1, C_2, C_3, and the cell document $C[\mathbf{D}]$ is the concatenation of cell documents $C_1[\mathbf{D}], C_2[\mathbf{D}], C_3[\mathbf{D}]$. So $C[\mathbf{D}]$ contains exactly the three terms $\{t_1, t_2, t_3\}$.

Now we use the relevance scoring formula in (1), and let $k_1 = 1, b = 1, \mathrm{avdl} = 1$. Then we have $\mathrm{rel}(\mathsf{q}, C) > \mathrm{rel}(\mathsf{q}, C_1), \mathrm{rel}(\mathsf{q}, C_2), \mathrm{rel}(\mathsf{q}, C_3)$. However, if $C[\mathbf{D}]$ has a document length 10, then we have $\mathrm{rel}(\mathsf{q}, C) < \mathrm{rel}(\mathsf{q}, C_1), \mathrm{rel}(\mathsf{q}, C_2), \mathrm{rel}(\mathsf{q}, C_3)$. So rel does NOT satisfy monotone or Apriori property in the cube lattice.

Extended Form of Keyword Query. Users may want to retrieve answers from a certain part of the text cube, by specifying a subset dimensions of interests and/or values of some dimensions, together with a support threshold. The *support* of a cell C, denoted by $|C|$, is the number of documents that are concatenated in the cell document $C[\mathbf{D}]$. Motivated by this, the simplest form of keyword queries q can be extended by adding *dimension-value constraints* and *support threshold.*

In an n-dimensional text cube, *an extended keyword query* is in the form of $Q = (u_1, u_2, \ldots, u_n : \mathsf{q})$, where $u_i \in \mathbf{A}_i \cup \{*, ?\}$. We also use $Q[\mathbf{A}_i]$ to denote u_i. $Q[\mathbf{A}_i] \in \mathbf{A}_i$ specifies the value of dimension $\mathbf{A}_i$ in a cell C; $Q[\mathbf{A}_i] = *$ means the dimension $\mathbf{A}_i$ in a cell C must be aggregated; and $Q[\mathbf{A}_i] = ?$ (question mark) imposes no constraint on the dimension $\mathbf{A}_i$ of a cell C. A cell C is said to be *feasible* w.r.t. the query Q iff

(i) for dimension $\mathbf{A}_i$ s.t. $Q[\mathbf{A}_i] = *$, we have $C[\mathbf{A}_i] = *$ ($\mathbf{A}_i$ is aggregated in C);
(ii) for dimension $\mathbf{A}_i$ s.t. $Q[\mathbf{A}_i] \in \mathbf{A}_i$, we have $C[\mathbf{A}_i] = Q[\mathbf{A}_i]$; and
(iii) for dimension $\mathbf{A}_i$ s.t. $Q[\mathbf{A}_i] = ?$, we have no constraint on $C[\mathbf{A}_i]$.

Given an *extended keyword query* $Q = (u_1, u_2, \ldots, u_n : \mathsf{q})$ and a minimum support minsup, our goal is to find the top-k *feasible* cells C's s.t. supports $|C| \geq \mathsf{minsup}$ with the top-k highest *relevance scores* $\mathrm{rel}(\mathsf{q}, C)$'s in the text cube of $\mathbf{DB}$.

In the rest part of this paper, we will first describe our algorithms for the simple form of keyword query (*i.e.*, a set of keywords without dimension-value constraints and support threshold), and then discuss how our algorithm can be simply modified to handle extended form of keyword query.

2.3. **Computational Challenges.** There are two major challenges of this keyword search problem:

First, as shown in [8], the size of a text cube could be huge, increasing exponentially w.r.t. the dimensionality of the text cube. There is an n-dimensional database $\mathbf{DB} = (\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_n, \mathbf{D})$ with N rows, s.t. the non-empty cells in the text cube of $\mathbf{DB}$ is $\Omega(N \cdot 2^n)$ or $\Omega(\prod_{i=1}^{n}(|\mathbf{A}_i| + 1))$, where $|\mathbf{A}_i|$ is the number of different values in dimension $\mathbf{A}_i$.[1] Therefore, *only when* the number of dimensions is small (2 to 4), we can compute the relevance scores of all cells and then sort them to find the top-k cells efficiently.

Second, as shown in Example 2.2, the relevance scoring formula rel considered here does NOT satisfy monotone/Apriori property in the cube lattice. This increases the difficulty of our problem further, as the early-stop condition for searching top-k is not easy to be obtained.

3. Threshold Algorithm for Finding Top-k Cells

Our first approach TACell naturally extends the famous *threshold algorithm* (TA) [9] for finding the top-k relevant cells w.r.t. a given keyword query q. The basic idea is to treat each cell as a ranking object in TA. Some preprocessing steps and additional space are needed.

Preprocessing: In the preprocessing stage, for each term t in $\mathbf{W}$ (the set of all terms), we build a sorted list of cells L_t, where cells are sorted in the descending order of term frequency of t in each cell document; then we have another sorted list L_{len}, where cells are sorted in the ascending order of the lengths of cell documents. So there are a total of $|\mathbf{W}| + 1$ sorted list.

Online-processing: When a keyword query $\mathsf{q} = \{t_1, t_2, \ldots, t_l\}$ is given, our goal is to find the top-k cells with the highest relevance scores $\mathrm{rel}(\mathsf{q}, C)$. In this step, we apply the TA algorithm [9] on the $l+1$ sorted list $L_{t_1}, L_{t_2}, \ldots, L_{t_l}, L_{len}$. Recall $\mathrm{rel}(q, C)$ is defined in (2), and TA algorithm is applied to output the top-k cells with the highest $s(\mathrm{tf}_1, \mathrm{tf}_2, \ldots, \mathrm{tf}_l, |C[\mathbf{D}]|)$ (refer to Algorithm 2).

```
1: For each term $t \in \mathbf{W}$, construct a sorted list $L_t$ of cells. Cells $C$'s in $L_t$ are sorted in the
   descending order of term frequency of $t$ in the cell document $C[\mathbf{D}]$.
2: Construct another sorted list $L_{len}$ of cells. Cells $C$'s in $L_{len}$ are sorted in the ascending
   order of the length of cell document $|C[\mathbf{D}]|$.
```

Algorithm 1: Preprocessing for TACell

Two points to be clarified in Algorithm 2:

First, in line 3, when a cell C is retrieved from some list, there are three possibilities: i) C is in CAN; ii) C is not in CAN and it is the first time that C is touched; iii) C is not in CAN but C

[1]To construct a text cube with $N \cdot (2^n - 1) = \Omega(N \cdot 2^n)$ non-empty cells, consider $\mathbf{DB} = \{(a_1^{(i)}, a_2^{(i)}, \ldots, a_n^{(i)}, \mathsf{d}_i) \mid i = 1, \ldots, N\}$ with N rows and $a_j^{(i)} \neq a_j^{(i')}$ for any j and $i \neq i'$. To construct a text cube with $\prod_{i=1}^{n}(|\mathbf{A}_i| + 1)$ non-empty cells, let $N = \prod_{i=1}^{n} |\mathbf{A}_i|$ and consider a database whose rows enumerate all possible configurations of the n attributes. Even when the support threshold minsup (> 0) is nontrivial, the number of cells to be considered (those with support $\geq \mathsf{minsup}$) is still huge, since a data cube is "fat" in the middle. For example, suppose minsup is large enough s.t. only d-dim cells with $d \leq n/2$ have support $\geq \mathsf{minsup}$, we may still need to consider $N \cdot \binom{n}{n/2} = \Omega(N \cdot 2^{n/2})$ cells to select the top-k relevant ones w.r.t. a keyword query.

```
1: Candidates of top-$k$ $CAN \leftarrow \emptyset$. Pointer $i \leftarrow 1$.
2: Do a parallel scan of $L_{t_1}, L_{t_2}, \ldots, L_{t_l}, L_{len}$:
3:     In each iteration, retrieve the $i^{th}$ cell of each list of $L_{t_1}, \ldots, L_{t_l}, L_{len}$ (totally $l+1$ cells),
       compute its relevance score rel($\mathsf{q}, C$), and put it into $CAN$.
4:     At any time, $CAN$ keeps only the top-$k$ cells with the highest score rel($\mathsf{q}, C$) and
       the $(k+1)^{th}$ cell will be deleted from $CAN$ (if any).
5:     Let the threshold $TA \leftarrow s(\text{tf}'_1, \text{tf}'_2, \ldots, \text{tf}'_l, l)$, where $\text{tf}'_j$ is the term frequency of $t_j$ in
       the $i^{th}$ cell of the list $L_{t_j}$, and $l$ is the cell document length of the $i^{th}$ cell of the list $L_{len}$.
6:     If $CAN$ has $k$ cells in it and $TA <$ the lowest score in $CAN$, then
          **output** $CAN$ and **end**;
7:     Else $i \leftarrow i+1$ and **goto** line 2.
```

Algorithm 2: Online Processing of TACell

has been touched previously. Besides CAN, we do not keep track of whether C has been touched before. And, we can observe that a cell would be put in and pop out from CAN for at most once.

Second, in line 5 of Algorithm 2, $(\text{tf}'_1, \text{tf}'_2, \ldots, \text{tf}'_l, l)$ may not be the term frequency and cell document length for the same cell. $TA = s(\text{tf}'_1, \text{tf}'_2, \ldots, \text{tf}'_l, l)$ is nothing but the upper bound of relevance score of any cell untouched by the parallel scan. It is used as a threshold for early-stop.

From Theorem 4.1 in [9] and how the lists L_t's and L_{len} are sorted in Algorithm 1, we have the correctness of Algorithm 2.

Corrollary 1. Given any query q against the text cube of **DB**, Algorithm 2 outputs the top-k cells with the highest relevance scores rel(q, C)'s (the ones in CAN), when it terminates.

Handling Extended Form of Keyword Query: TACell (Algorithm 1&2) can be easily adapted to handle extended keyword query $Q = (u_1, \ldots, u_n : \mathsf{q})$ with support threshold minsup specified. The idea is as follows. In line 3, a cell is put into CAN if and only if it is feasible and has support no less than minsup. Moreover, to speed up the processing, in each list, we can prune the first p cells (*i.e.*, start from the $(p+1)^{th}$ cell), if all of them have supports less than minsup.

Complexity: The TA algorithm (extended as Algorithm 2 in our problem) is proved to be optimal [9] in the sense that any other algorithm based on $L_{t_1}, \ldots, L_{t_l}, L_{len}$ cannot stop sooner than Algorithm 2. On the other hand, the actual running time of Algorithm 2 depends on the input and parameters (although in the worst case, all the entries in every sorted list need to be scanned).

Moreover, each iteration of Algorithm 2 (line 2-7) can be efficiently implemented: i) if random accesses of sorted lists are supported, the relevance score of retrieved cells can be efficiently computed (line 3); and CAN is maintained in a priority queue with the relevance scores of cells in it as the keys, so any operation (*e.g.*, adding a cell into and deleting the $(k+1)^{th}$ cell from CAN) (line 3-4) can be done in $O(\log k)$ time [6].

The bottleneck of the TACell algorithm is the space consumption. As discussed in Section 2.3, the total number of non-empty cells in a text cube is huge. So each list of L_t's and L_{len} is very long, and it might be impossible to put all of them into the main memory if the database is large and the dimensionality is high (recall there is a list L_t for each term in $\mathbf{W}$). If these lists are stored in the disk, accessing them (especially the random accesses) in the online processing could be expensive, not to mention that time consumed in the preprocessing stage could also be long.

So in the next section, we aim to design an algorithm which is more efficient in the preprocessing.

4. Bound-checking Search Algorithm

Our second approach BoundS does not require as much preprocessing as TACell. In the preprocessing, it computes nothing more than the inverted indexes for all terms w.r.t. the documents (not the cells). In the online processing, given a keyword query $\mathsf{q} = \{t_1, t_2, \ldots, t_l\}$, the top-$k$ cells with highest relevance scores are output. The basic ideas of online processing in BoundS are as follows.

- *Heuristic Ordering of Row Documents*: First, we order the row documents (the ones in the rows of the database, rather than the cell documents in the cells) in the descending order of relevance scores (w.r.t. the given keyword query). We later process the documents in this order. The intuition is: A highly relevant cell documents is likely to consist of highly relevant row documents; so starting from highly relevant row documents, we can touch the highly relevant cell documents sooner. We also note that the ordering of these row documents does NOT affect the correctness of our algorithm introduced in this section.
- *Partial Cells and Finalized Cells*: Initially, all cell documents are unseen/empty. As we scan each row document, it is concatenated to the cell documents of cells that contain this row. Before all the row documents in a cell are concatenated to the cell document, this cell is said to be *partial*; and after that, it is said to be *finalized*.
- *Lower Bounds and Upper Bounds*: When we scan the row documents, we want to estimate the lower bounds and upper bounds of the relevance scores of the cells we have seen. For each cell C, let $\mathrm{tf}(C)_i$ be the term frequency of $t_i \in \mathsf{q}$ in the current cell document of C. As we scan more row documents, $\mathrm{tf}(C)_i$ will possibly increase before C is finalized, so a *lower bound* of $\mathrm{rel}(\mathsf{q}, C)$ is: (assume that the cell document length $|C[\mathbf{D}]|$ is precomputed)

$$\mathrm{rel}(\mathsf{q}, C) \geq \mathrm{rel}(\mathsf{q}, C)_{\mathrm{lb}} = s(\mathrm{tf}(C)_1, \mathrm{tf}(C)_2, \ldots, \mathrm{tf}(C)_l, |C[\mathbf{D}]|). \tag{3}$$

  Let Δ_i be the total term frequency of $t_i \in \mathsf{q}$ in the unseen row documents. We assume in the extreme case, all the rest instances of t_i in the unseen row documents are in the cell C, and from the monotonicity of s w.r.t. $\mathrm{tf}(C)_i$, we have an *upper bound* of $\mathrm{rel}(\mathsf{q}, C)$ is:

$$\mathrm{rel}(\mathsf{q}, C) \leq \mathrm{rel}(\mathsf{q}, C)_{\mathrm{ub}} = s(\mathrm{tf}(C)_1 + \Delta_1, \mathrm{tf}(C)_2 + \Delta_2, \ldots, \mathrm{tf}(C)_l + \Delta_l, |C[\mathbf{D}]|). \tag{4}$$

- *Condition for Output*: Note that $\mathrm{rel}(\mathsf{q}, C)_{\mathrm{lb}}$'s and $\mathrm{rel}(\mathsf{q}, C)_{\mathrm{ub}}$'s are updated as we scan row documents. Suppose all the cells are maintained in the descending order of $\mathrm{rel}(\mathsf{q}, C)_{\mathrm{lb}}$, and let θ be the k^{th} highest value of $\mathrm{rel}(\mathsf{q}, C)_{\mathrm{lb}}$. A cell can be *pruned* if $\mathrm{rel}(\mathsf{q}, C)_{\mathrm{ub}} \leq \theta$. An obvious condition for output the top-k is that *all cells except the top-k ones can be pruned.*

In the following part, we present our BoundS in Algorithm 3, and then prove its correctness.

```
1:  (Heuristic Ordering)
    Sort all the row documents in the descending order of relevance scores.
2:  (Initialization)
    Before scanning the row documents:
3:      Let $\mathrm{tf}(C)_i \leftarrow 0$ for any cell $C$ and any term $t_i \in \mathsf{q}$ (all cells are unseen);
4:      Let $\Delta_i \leftarrow$ the total term frequency of $t_i$ in all row documents;
5:  (Scanning Row Documents)
    For each row document $r = (a_1, a_2, \ldots, a_n, \mathsf{d})$ (in the descending order of relevance scores) do:
6:      For each term $t_i \in \mathsf{q} \cap \mathsf{d}$ do: (let $\delta_i$ be the term frequency of $t_i$ in $\mathsf{d}$)
7:          $\Delta_i \leftarrow \Delta_i - \delta_i$;
8:          For each cell $C$ containing the row $r$ do:
9:              $\mathrm{tf}(C)_i \leftarrow \mathrm{tf}(C)_i + \delta_i$;
10:     For each cell $C$ containing the row $r$, update its relevance lower bound $\mathrm{rel}(\mathsf{q}, C)_{\mathrm{lb}}$, as in (3);
11:     (Bound Checking and Output Condition Checking)
        Compute $\theta$ as the $k^{th}$ highest value of $\mathrm{rel}(\mathsf{q}, C)_{\mathrm{lb}}$;
12:     Compute $CAN$ as the top-$k$ cells with highest values of $\mathrm{rel}(\mathsf{q}, C)_{\mathrm{lb}}$;
13:     Let $CAN'$ be the cells not in $CAN$ and with $\mathrm{rel}(\mathsf{q}, C)_{\mathrm{ub}} > \theta$ ($\mathrm{rel}(\mathsf{q}, C)_{\mathrm{ub}}$ is defined in (4));
14:     If $CAN' = \emptyset$, then **output** $CAN$ and **end**.
```

Algorithm 3: Bound-checking Search Algorithm

Starting from line 5, we scan the row documents one by one, and the lower bounds of relevance scores are updated for each C containing the row r (line 10) after we update the term frequency

$\mathrm{tf}(C)_i$'s (line 9). Note that when the dimensionality is n, there are 2^n cells containing r. And, we do not need to maintain the relevance lower bounds for the unseen cells.

Bound checking and output condition checking (line 11-14) are NOT executed in every iteration, as computing the upper bounds of relevance scores in line 13 is expensive (line 11 can be implemented using the famous Hoare's selection algorithm [12] in linear time). So in the implementation, we execute line 11-14 after, *e.g.*, every 1000 iterations of line 6-10.

A relaxation of the output condition in line 14 is: instead of checking whether $CAN' = \emptyset$, we check if $|CAN'| \leq M$ for a threshold $M = 10k$; if yes, we compute the relevance score of cells in $CAN \cup CAN'$ and sort them for finding the top-k.

Theorem 1. Given any query q against the text cube of **DB**, Algorithm 3 outputs the top-k cells with the highest relevance scores $\mathrm{rel}(\mathsf{q}, C)$'s, when it terminates.

Proof. First, from the above analysis, we know lower bounds $\mathrm{rel}(\mathsf{q}, C)_{\mathrm{lb}}$ and upper bounds $\mathrm{rel}(\mathsf{q}, C)_{\mathrm{ub}}$ of relevance scores are correctly computed as (3) and (4), respectively. Line 6-9 of Algorithm 3 update the parameters of these bounds as required.

For the output condition, let θ^* be the k^{th} highest value of $\mathrm{rel}(\mathsf{q}, C)$. From the definition, we know $\theta \leq \theta^*$ (θ is computed in line 11). So for a cell C, $\mathrm{rel}(\mathsf{q}, C)_{\mathrm{ub}} \leq \theta$ implies $\mathrm{rel}(\mathsf{q}, C) \leq \mathrm{rel}(\mathsf{q}, C)_{\mathrm{ub}} \leq \theta \leq \theta^*$, and such a cell can be pruned. CAN' keeps the cells that are cannot be pruned and not in CAN. So if $CAN' = \emptyset$, then CAN is the real top-k and can be output.

We also need to prove this algorithm will eventually terminate. This is because, sooner or later, all the cells will be finalized (*i.e.*, all the row document in each of them have been concatenated to the cell document), and the real top-k ones must be kept in CAN. Then the output condition is satisfied and the top-k are output.

In the relaxed version, we compute and sort relevance scores of cells in $CAN \cup CAN'$; this is because all the other cells can be pruned. □

Handling Extended Form of Keyword Query: To handle a extended keyword query $Q = (u_1, u_2, \ldots, u_n : \mathsf{q})$ with support threshold minsup, we can simply modify line 8,10,12,13 in Algorithm 3 to filter in only the feasible cells with support no less than minsup. Adding this step cannot deteriorate the performance of BoundS, as we are now focusing on a smaller number of cells.

5. Experimental Study

In this section, we evaluate the effectiveness of the two algorithms using a real dataset.

5.1. **Datasets and Environment Setup.** A real dataset NASA's ASRS database (Aviation Safety Reporting System) [15] is used in the experiments. We select 10 dimensions in the database together with the narrative information in each row to form a multidimensional text cube. The 10 dimensions are: Year, State, Person, Weather, Light, Make/Model, Flight Phase, Primary Area, Event Anomaly, Resolutory Action. There are two dimensions with too many empty (sensitive) values, so in the efficiency testing below, we consider only 8 dimensions.

In this database, we have a total of 34873 documents (each associated with 10 dimensions). After all the stop words are removed, there are 39453 terms remaining (the number of terms may affect the preprocessing time of TACell). The text cube constructed based on this database has 2634490 nonempty cells. More information about this database can be found in [15].

A demo system (`http://inextcube.cs.uiuc.edu/nasa/Default.aspx?func=topcell`) is constructed to conduct the case study.

All the experiments were conducted on a PC running the Microsoft Windows XP SP2 Professional OS, with a 2.5 GHz Intel Core 2 Duo T9300 CPU, 3.0 GB of RAM, and 150 GB hard drive. Our algorithms were implemented in C/C++ and compiled on Microsoft Visual Studio 2008.

q_1	RWY EXCURSION
q_2	DOWNWIND RWY
q_3	SHUT DOWN ENG
q_4	TOOK EVASIVE ACTION
q_5	GEAR NOT RETRACT
q_6	VISIBILITY LIGHT FOG
q_7	SAW OTHER ACFT
q_8	RADIO MIDAIR COLLISION
q_9	SMOKE FROM ENG GEAR
q_{10}	CALLBACK CONVERSATION REPORTER HAT

TABLE 2. Example queries

5.2. **Efficiency in Preprocessing and Online Processing.** We first test the efficiency of our algorithms in the preprocessing and online processing stages using our real dataset ASRS. Table 2 shows ten example queries used in the following experiments.

5.2.1. *Exp-I: Varying the Number (n) of Dimensions.* In this experiment, we study the effect of dimensionality on the efficiency of preprocessing and online processing of TACell and BoundS. We pick the first 4,6,8 dimensions of the ASRS database, and construct the corresponding text cubes with 4,6,8 dimensions, respectively. We report preprocessing time and online processing time for both algorithms. For online processing time, we report the time of outputting top-10 answers (average of the ten queries in Table 2). In all the following experiments, we report the preprocessing/online processing time in terms of seconds, if not specified.

TACell		
Dimensionality	Sorting cells w.r.t. cell document length	Sorting cells w.r.t. term frequency
8	82.41	$0.982 \times 39453 \approx 10$ hours
6	20.24	$0.091 \times 39453 \approx 1$ hour
4	5.60	$0.003 \times 39453 \approx 2$ minutes
BoundS		
Dimensionality	Computing cell document lengths	
8	20.66	
6	5.14	
4	2.00	

TABLE 3. Varying the Number (n) of Dimensions (preprocessing)

Dimensionality	8	6	4
TACell	24.90333333	1.036666667	0.02
BoundS	23.28333333	3.68	1

TABLE 4. Varying the Number (n) of Dimensions (online processing)

The result is reported in Table 3-4. As we expected, BoundS performs similarly as TACell in online processing, but is much more efficient than TACell in preprocessing. Note that in the preprocessing of TACell, sorting cells in the descending order of term frequency (line 1 of Algorithm 1–the third column in Table 3) can also be done online; but with this modification, the online processing of TACell will be much slower than BoundS.

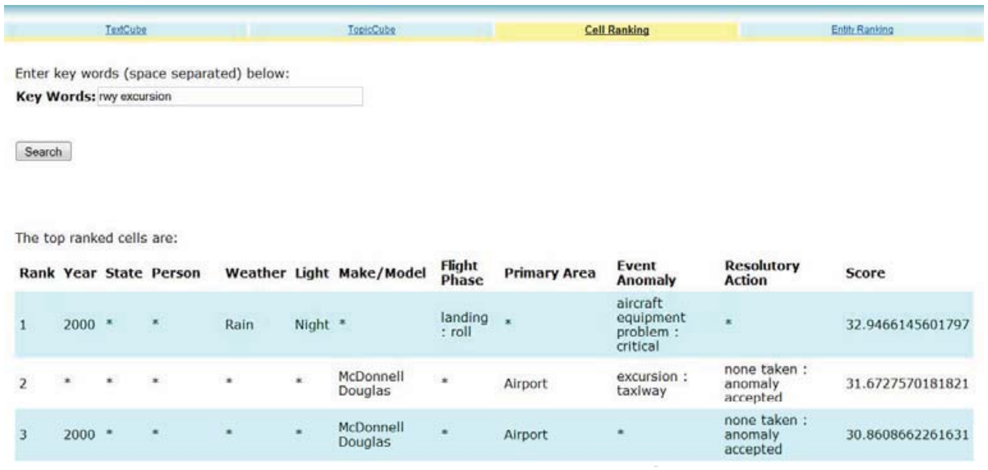
Threshold γ	Time	# doc accessed	total # doc	# cells accessed	total # cells
5	54.87	14833.33	34873	1356582	2634490
10	23.28	6400	34873	611773.33	2634490
20	5.67	1666.67	34873	219973.33	2634490
40	5.66	1666.67	34873	219973.33	2634490

TABLE 5. Varying the Parameter in BoundS (online processing)

5.2.2. *Exp-II: Varying the Parameter in* BoundS. Recall a relaxation of the output condition in line 14 of Algorithm 2 (BoundS) is: "instead of checking whether $CAN' = \emptyset$, we check if $|CAN'| \leq M$ for a threshold $M = \gamma \cdot k$ (we use $\gamma = 10$ in Exp-I); if yes, we compute the relevance score of cells in $CAN \cup CAN'$ and sort them for finding the top-k."

It is also interesting to verify the effect of the threshold γ on the performance of BoundS. We use the text cube with 8 dimensions and report the performance of BoundS for outputting top-10 results, while varying γ from 5 to 40. The result is reported in Table 5. It can be found that a reasonably large γ allows BoundS to access less documents and less cells. However, too large γ (*e.g.* > 20) does not help much while more time is needed for the sorting $CAN \cup CAN'$. So $\gamma = 10$ or 20 is a proper threshold for outputting top-10 answers.

5.3. **Case Study.** In this section, we verify the effectiveness of our model and algorithms by showing a few example queries and the meaningful retrieval results.



Enter key words (space separated) below:
Key Words: rwy excursion

The top ranked cells are:

Rank	Year	State	Person	Weather	Light	Make/Model	Flight Phase	Primary Area	Event Anomaly	Resolutory Action	Score
1	2000	*	*	Rain	Night	*	landing : roll	*	aircraft equipment problem : critical	*	32.9466145601797
2	*	*	*	*	*	McDonnell Douglas	*	Airport	excursion : taxiway	none taken : anomaly accepted	31.6727570181821
3	2000	*	*	*	*	McDonnell Douglas	*	Airport	*	none taken : anomaly accepted	30.8608662261631

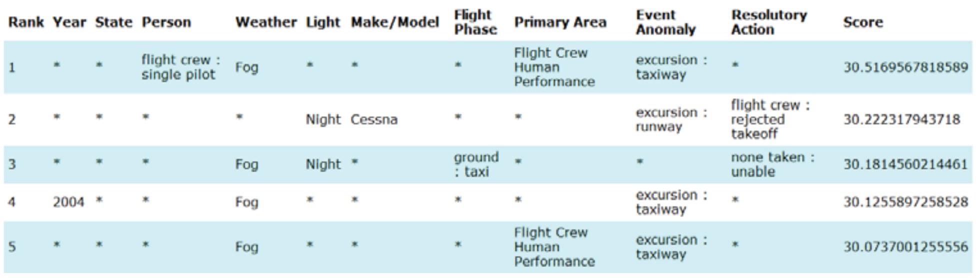
FIGURE 1. Query results of {"RWY", "EXCURSION"} in our demo system

5.3.1. *Runway Excursion.* Suppose we want to find out under which condition, the "runway excursion" is likely to happen. With these two keywords typed into our system, the result is shown in Figure 1 (a screen shot from our demo system–each row above represents a cell in the text cube).

The top-1 result implies that *this situation is likely to happen in a rainy night, during the phase of landing roll, when there is a critical equipment problem detected*. Moreover, nearly all the top-10 results are related to "*rain*" or "*night*". And, three of the top-5 results are related to some model of "*McDonnell Douglas*".

5.3.2. *Fog Weather.* We are also interested in what will happen in a fog weather. So we type three keywords "visibility", "light", and "fog", and the result is shown in Figure 2.

There are four observations from the top-5 results: i) The fog weather is usually reported in the night (maybe because it is more critical in the night). ii) The fog weather is usually reported during

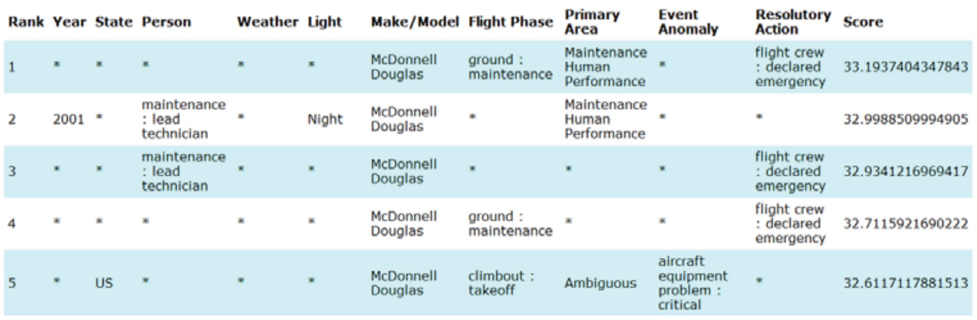


Rank	Year	State	Person	Weather	Light	Make/Model	Flight Phase	Primary Area	Event Anomaly	Resolutory Action	Score
1	*	*	flight crew : single pilot	Fog	*	*	*	Flight Crew Human Performance	excursion : taxiway	*	30.5169567818589
2	*	*	*	*	Night	Cessna	*	*	excursion : runway	flight crew : rejected takeoff	30.222317943718
3	*	*	*	Fog	Night	*	ground : taxi	*	*	none taken : unable	30.1814560214461
4	2004	*	*	Fog	*	*	*	*	excursion : taxiway	*	30.1255897258528
5	*	*	*	Fog	*	*	*	Flight Crew Human Performance	excursion : taxiway	*	30.0737001255556

Figure 2. Query results of {“VISIBILITY”, “LIGHT”, “FOG”} in our demo system

the early phase of flight, *e.g.*, the “ground: taxi” phase. iii) The anomaly of excursion (either on the runway or the taxiway) is usually caused by the fog weather. iv) The resolutory action to be taken could be “rejected takeoff” (that might be the reason why fights are usually delayed by fog).

5.3.3. *Gear Does Not Retract.* Now we want to know more about the situation when the gear does not retract. With the three keywords “gear”, “not”, and “retract” typed into our system, the result is shown in Figure 3.



Rank	Year	State	Person	Weather	Light	Make/Model	Flight Phase	Primary Area	Event Anomaly	Resolutory Action	Score
1	*	*	*	*	*	McDonnell Douglas	ground : maintenance	Maintenance Human Performance	*	flight crew : declared emergency	33.1937404347843
2	2001	*	maintenance : lead technician	*	Night	McDonnell Douglas	*	Maintenance Human Performance	*	*	32.9988509994905
3	*	*	maintenance : lead technician	*	*	McDonnell Douglas	*	*	*	flight crew : declared emergency	32.9341216969417
4	*	*	*	*	*	McDonnell Douglas	ground : maintenance	*	*	flight crew : declared emergency	32.7115921690222
5	*	US	*	*	*	McDonnell Douglas	climbout : takeoff	Ambiguous	aircraft equipment problem : critical	*	32.6117117881513

Figure 3. Query results of {“GEAR”, “NOT”, “RETRACT”} in our demo system

It can be observed from the results that this problem is usually discovered by the lead technician during the ground maintenance phase, and sometimes, during the “climbout: takeoff” phase. It is usually categorized as a critical equipment problem.

Besides the above three, there could be many interesting and meaningful cases unreported and unobserved. We believe that domain experts can better utilize our system than us.

6. Related Work

Keyword Search in RDBMs. Although based on different applications and motivations, keyword search in text cube is related to keyword search in RDBMSs, which has attracted a lot of attention recently [5, 3, 30]. Most previous studies on keyword search in RDBMSs model the RDB as a graph (tuples/tables as nodes, and foreign-key links as edges) and focus on finding minimal connected tuple trees that contain all the keywords. They can be categorized into two types. The first type uses SQL to find the connected trees [2, 14, 13, 11, 23, 24]. The second type materializes the RDB graph and proposes algorithms to enumerate (top-k) subtrees in the graph [4, 7, 16, 19, 10]. Some of these studies model the keyword search problem as *the group (or direct) Steiner tree problem* [26] (an NP-hard problem), and propose parameterized algorithms to find the optimal top-1 answer [7, 20], and top-k answers (or approximate top-k answers) [19].

Different from these two types of works, two recent studies [21] and [25] find single-center subgraphs from the RDB graph, and multi-center induced subgraphs, respectively.

OLAP on Multidimensional Text Data. The text cube model is firstly proposed in [22]. [22] mainly focuses on how to partially materialize inverted indexes and term frequency vectors in cells of text cube, and how to support OLAP queries (not keyword query) efficiently in this partially-materialized cube.

The topic cube model is proposed in [32]. Different from the text cube, the topic cube materializes the language model of the aggregated document in each cell. Efficient algorithms are proposed to compute this topic cube.

The techniques in [22] and [32] cannot be used directly to support keyword search, because the information materialized in text cube (term frequencies and inverted indexes) and in topic cube (language model) is query-independent.

Analysis of Text Data with Multiple Attributes. Besides [32], there are some other works on analyzing text data with multiple attributes, *e.g.* [28, 29]; though they cannot be directly applied in our keyword search problem (as the models they focus on are query-independent). [28] introduces a generative model of entity relationships and their attributes which can simultaneously discovers groups among the entities and topics among the corresponding textual attributes. [29] generalizes techniques such as principal component analysis to text with heterogeneous attributes. Note that, unlike our ranking techniques in the text cube model, [28] and [29] do not start with the aggregation of entities/rows on subsets of dimensions (*i.e.*, cells and cuboids).

Keyword-based Search and OLAP in Data Cube. [1] studies answering keyword queries on RDB using minimal group-bys, which is the work most relevant to ours. For a keyword query against a multidimensional text database, it aims to find the minimal cells containing all (or some of) the query terms in the aggregated text data. "Minimal" here means there is no descendant of this cell containing more query terms. But, it unnecessary that documents (cells) with more query terms are more relevant. And, [1] does not score or rank the answers. So when the number of returned answers is large (e.g., a thousand), it is difficult for the user to browse all the answers.

Another relevant work is keyword-driven analytical processing (KDAP) [31]. Motivated by an application scenario different from [1] and our work, it proposes a two-phase framework for effective OLAP based on user-given keyword queries. In the first phase, *differentiate phase*, candidate subspaces (*i.e.*, possible join paths between the dimensions and the facts in a data warehouse schema) are generated and ranked based on the given keyword query. The user is asked to select one of candidate subspaces. Then it comes to the second phase, *explore phase*. The system computes the group-by aggregates from all qualified fact points. Group-by attributes are ranked, and an interface is provided to explore detailed aggregation. KDAP supports interactive data exploration using keywords. Candidate subspaces are output to disambiguate the keyword terms. But, [1] and our work focus on efficient answering of keyword queries. Efficiency is not a major concern in KDAP ([31] does not report any experiment on the efficiency).

Our previous work [8] solves the same problem as this work, but focuses on another relevance scoring model, *average model*. As discussed in Section 1.1, the properties of this model are different from the ones of the model, *cell document model*, we are focusing on in this work. The algorithms designed in [8] cannot be applied in this work. Moreover, two models have different semantics, and are appliable in different scenarios and different user preferences.

7. Conclusions and Future Work

In this paper, we study the problem of keyword-based top-k search in text cube (*i.e.*, multidimensional text data). Flexible query language and relevance scoring formula are developed based on *cell document model*. We design two efficient approaches for this problem. The first one extends the famous TA algorithm to our problem, which are efficient but requires a large amount of space in the preprocessing. The second one is based on lower/upper bound estimation and checking to find the top-k cells before exploring the whole text cube. It is efficient in both preprocessing and online processing of keyword queries. We conduct extensive performance studies to verify the effectiveness of the proposed approaches.

An interesting direction for future work is to evaluate and compare the effectiveness of the two models *average model* (studied in [8]) and *cell document model* (studied in this paper). For this purpose, user-studies need to be conducted among domain experts. We also believe that domain experts can better utilize our system than us. It is helpful to know which one performs better in which situation, when our methods are applied in practice.

References

[1] B. Z. 0002 and J. Pei. Answering aggregate keyword queries on relational databases using minimal group-bys. In *EDBT*, pages 108–119, 2009.

[2] S. Agrawal, S. Chaudhuri, and G. Das. Dbxplorer: A system for keyword-based search over relational databases. In *ICDE*, pages 5–16, 2002.

[3] S. Amer-Yahia, P. Case, T. Rölleke, J. Shanmugasundaram, and G. Weikum. Report on the db/ir panel at sigmod 2005. *SIGMOD Record*, 34(4):71–74, 2005.

[4] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using banks. In *ICDE*, pages 431–440, 2002.

[5] S. Chaudhuri, R. Ramakrishnan, and G. Weikum. Integrating db and ir technologies: What is the sound of one hand clapping? In *CIDR*, pages 1–12, 2005.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithm (2nd Edition)*. The MIT Press, USA, 2001.

[7] B. Ding, J. X. Yu, S. Wang, L. Qin, X. Zhang, and X. Lin. Finding top-k min-cost connected trees in databases. In *ICDE*, pages 836–845, 2007.

[8] B. Ding, B. Zhao, C. X. Lin, J. Han, and C. Zhai. Topcells: Keyword-based search of top-k aggregated documents in text cube. In *ICDE*, pages 381–384, 2010.

[9] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4):614–656, 2003.

[10] K. Golenberg, B. Kimelfeld, and Y. Sagiv. Keyword proximity search in complex data graphs. In *SIGMOD Conference*, pages 927–940, 2008.

[11] H. He, H. Wang, J. Yang, and P. S. Yu. Blinks: ranked keyword searches on graphs. In *SIGMOD Conference*, pages 305–316, 2007.

[12] C. A. R. Hoare. Algorithm 65: find. *Commun. ACM*, 4(7):321–322, 1961.

[13] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient ir-style keyword search over relational databases. In *VLDB*, pages 850–861, 2003.

[14] V. Hristidis and Y. Papakonstantinou. Discover: Keyword search in relational databases. In *VLDB*, pages 670–681, 2002.

[15] http://asrs.arc.nasa.gov/.

[16] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar. Bidirectional expansion for keyword search on graph databases. In *VLDB*, pages 505–516, 2005.

[17] B. Kimelfeld and Y. Sagiv. Efficient engines for keyword proximity search. In *WebDB*, pages 67–72, 2005.

[18] B. Kimelfeld and Y. Sagiv. Efficiently enumerating results of keyword search. In *DBPL*, pages 58–73, 2005.

[19] B. Kimelfeld and Y. Sagiv. Finding and approximating top-k answers in keyword proximity search. In *PODS*, pages 173–182, 2006.

[20] B. Kimelfeld and Y. Sagiv. New algorithms for computing steiner trees for a fixed number of terminals. Manuscripts, 2006.

[21] G. Li, B. C. Ooi, J. Feng, J. Wang, and L. Zhou. Ease: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In *SIGMOD Conference*, pages 903–914, 2008.

[22] C. X. Lin, B. Ding, J. Han, F. Zhu, and B. Zhao. Text cube: Computing ir measures for multidimensional text database analysis. In *ICDM*, pages 905–910, 2008.

[23] F. Liu, C. T. Yu, W. Meng, and A. Chowdhury. Effective keyword search in relational databases. In *SIGMOD Conference*, pages 563–574, 2006.

[24] Y. Luo, X. Lin, W. Wang, and X. Zhou. Spark: top-k keyword query in relational databases. In *SIGMOD Conference*, pages 115–126, 2007.

[25] L. Qin, J. X. Yu, L. Chang, and Y. Tao. Querying communities in relational databases. In *ICDE*, pages 724–735, 2009.

[26] G. Reich and P. Widmayer. Beyond steiner's problem: A vlsi oriented generalization. In *WG*, pages 196–210, 1989.

[27] S. E. Robertson, S. Walker, and M. Hancock-Beaulieu. Okapi at trec-7: Automatic ad hoc, filtering, vlc and interactive. In *TREC*, pages 199–210, 1998.
[28] X. Wang, N. Mohanty, and A. McCallum. Group and topic discovery from relations and their attributes. In *NIPS*, 2005.
[29] X. Wang, C. Pal, and A. McCallum. Generalized component analysis for text with heterogeneous attributes. In *KDD*, pages 794–803, 2007.
[30] G. Weikum. Db&ir: both sides now. In *SIGMOD Conference*, pages 25–30, 2007.
[31] P. Wu, Y. Sismanis, and B. Reinwald. Towards keyword-driven analytical processing. In *SIGMOD Conference*, pages 617–628, 2007.
[32] D. Zhang, C. Zhai, and J. Han. Topic cube: Topic modeling for olap on multidimensional text databases. In *SDM*, pages 1123–1134, 2009.