Text Cube: Computing IR Measures for Multidimensional Text Database Analysis

Cindy Xide Lin, Bolin Ding, Jiawei Han, Feida Zhu, Bo Zhao
Department of Computer Science
University of Illinois at Urbana-Champagin, USA
{xidelin2, bding3, hanj, feidazhu, bozhao3}@uiuc.edu

Abstract

Since Jim Gray introduced the concept of "data cube" in 1997, data cube, associated with online analytical processing (OLAP), has become a driving engine in data warehouse industry. Because the boom of Internet has given rise to an ever increasing amount of text data associated with other multidimensional information, it is natural to propose a data cube model that integrates the power of traditional OLAP and IR techniques for text. In this paper, we propose a Text-Cube model on multidimensional text database and study effective OLAP over such data. Two kinds of hierarchies are distinguishable inside: dimensional hierarchy and term hierarchy. By incorporating these hierarchies, we conduct systematic studies on efficient text-cube implementation, OLAP execution and query processing. Our performance study shows the high promise of our methods.

1 Introduction

Data Cube has been proved a powerful model to efficiently handle Online Analytical Processing (OLAP) queries on large data collections that are multi-dimensional in nature. An OLAP data cube organizes data with categorical attributes (called *dimensions*) and summary statistics (called *measures*) from lower conceptual levels to higher ones. By offering users the ability to access data collections of any dimension subsets (called *cuboid*), a data cube provides the ease and flexibility for data navigation by different granularity levels and from different angles, without losing the overall picture of the data in its integrity.

Traditional OLAP cube studies[8, 1] focus on numeric measures, such as *count*, *sum* and *average*. Recent years have seen OLAP cubes extended to new domains, such as OLAP on graphs[7], sequences[6], spatial data[3] and mobile data[5]. In view of the boom of Internet and the ever increasing business intelligence applications, a domain of particular interest is that of text data. In this paper, we propose *Text Cube*, a general cube model on text data, to summarize and navigate structured data together with unstructured text data for efficient IR applications in such a way that these two kinds of information can mutually enhance knowledge discovery and data analysis.

Supposing a collection of documents $\mathcal{DOC}$ is stored in a database with dimensions, we are given (i) an IR query q (i.e. a set of terms/key), and (ii) constraints on dimensions, to retrieve relevant documents. Following is an example.

Example 1.1 *Consider a database of user reviews about some products (Table 1(a)), which consists of four dimensions* `M` *(Model),* `P` *(Price),* `T` *(Time), and* `S` *(Score). Each row stores a user review, $d_i \in \mathcal{DOC}$, with values of these four dimensions specified. Note we treat dimensions Price and Score as categorical dimensions: let* `p1` = cheap, `p2` = median, `p3` = high, `s1` = negative, *and* `s2` = positive.

An IR query q ("Dell notebook fast CPU"*) seeks reviews on Dell notebooks with fast CPUs. With database support, a more complicated IR query could be* (q = "Dell notebook fast CPU": `P` = median, `T` $>$ 2006)*, specifying constraints on two dimensions,* "median price" *and* "later than 2006".

The following contributions can be claimed in this paper.

1. A new data cube model called *text cube* is proposed, with *term hierarchy*, a new concept hierarchy for sematic navigation of the text data, defined and being integrated into the text cube by two new OLAP operations: *pull-up* and *push-down*.
2. Two important measures are supported in text cube upon which other IR techniques and applications can be efficiently built. We studied their efficient aggregation and storage.
3. Methods are worked out for optimal query processing in a partially materialized text cube, and algorithms are designed for partially materializing the text cube that greatly reduces the storage cost while bounding the query processing cost.
4. Experimental results on real Dell customer review datasets have been given to illustrate both the efficiency and effectiveness of our text cube model.

(a) A 4-Dimensional Text Database

Dimensions				Text Data
M (Model)	P (Price)	T (Y/M/D) (Time)	S (Score)	$\mathcal{DOC}$ (Documents)
m1	p1	2007/07/01	s1	$d_1 = \{$w1, w1, w2, w6, w8$\}$
m1	p1	2007/07/01	s2	$d_2 = \{$w1, w3, w6, w6, w7$\}$
m1	p2	2007/08/01	s2	$d_3 = \{$w2, w3, w6, w6$\}$
m2	p2	2007/08/01	s2	$d_4 = \{$w4, w5, w6, w7$\}$
m2	p3	2008/06/01	s1	$d_6 = \{$w4, w4, w5, w8$\}$

(b) 2-D Cuboid MS

Relevant Dimensions		Aggregated Text Data	IR Measure
M	S	D	$\mathsf{F}(D)$
m1	s1	$\{d_1\}$	...
m1	s2	$\{d_2, d_3\}$	...
m2	s2	$\{d_4\}$	...
m2	s1	$\{d_5, d_6\}$	...

Table 1. The Original Text Database and its 2-D Cuboid MS

The remaining part is organized as: Section 2 introduces the whole text cube model. Section 3 discusses computational issues, including online query processing and partially materialization. Section 4 gives experiments. Finally, Section 5 concludes this paper.

2 Text Cube: Hierarchy and Measure

Section 2.1 is the overview of the whole model. Section 2.2 introduces two types of concept hierarchies. Section 2.3 discusses essential IR measures supported.

2.1 Model Overview

The capacity of *text cube* to facilitate IR techniques on multidimensional text data is based on the following features:

- **A multidimensional data model:** Users have a flexibility to aggregate measures for any subsets of dimensions. Two types of *concept hierarchies* are supported:
  1. **dimension hierarchy:** it is the same as traditional data cubes.
  2. **Term hierarchy:** newly introduced in text cube is a hierarchy to specify the semantic levels of and relationships among text terms.
- **Measures supported for efficient IR:** For aggregated text data, two measures, *term frequency* and *inverted index*, are materialized. Consequently, IR queries on aggregated text data can be efficiently answered.

A document collection $\mathcal{DOC}$ is stored in a n-dimensional database $\mathcal{DB} = (\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n, \mathcal{DOC})$. Each row of $\mathcal{DOC}$ is in the form of $(a_1, a_2, \ldots, a_n, d)$, where $a_i \in \mathcal{A}_i$ is a *dimension* value for $\mathcal{A}_i$ and $d \in \mathcal{DOC}$. Supposing $\mathcal{W} = \{w_1, w_2, \ldots, w_m\}$ is the set of terms in $\mathcal{DOC}$, a document d is a multiset of $\mathcal{W}$. [1]

A *cell* is in the form of $(a_1, a_2, \ldots, a_n : \mathsf{F}(D))$ $(a_i \in \mathcal{A}_i \cup \{*\}$. $a_i = *$ means text data is *aggregated* on dimension $\mathcal{A}_i$. $D \subseteq \mathcal{DOC}$ is the document subset whose dimension values match to $a_1, a_2, \ldots, a_n$. $\mathsf{F}(D)$ is IR *measures* of D. A *cuboid* is a set of cells with the same inquired dimensions, denoted by $(a_1, a_2, \ldots, a_n : \mathsf{F}(D))$ $(a_i \in \{?, *\}$. $a_i = ?$ means $\mathcal{A}_i$ is an inquired dimension). A cuboid with m inquired dimensions is a *m-D cuboid*. A *subcube* is a subset of cells in a cuboid, in the form of $(a_1, a_2, \ldots, a_n : \mathsf{F}(D))$ $(a_i \in \mathcal{A}_i \cup \{*\} \cup \{?\})$. A cell, a cuboid, or a subcube is also referred to as $(a_1, a_2, \ldots, a_n : D)$ when the measure is not specified. Note that the aggregated text data D is not computed(stored) in cells of a text cube. Instead *each cell stores the IR measure of the aggregate text data*.

Example 2.1 *Table 1 has six documents with term set* $\mathcal{W} = \{$w1, w2, ..., w8$\}$. *The cuboid* MS $(?, *, *, ? : \mathsf{F}(D))$ *is shown in Table 1(b). Subcube* $(\mathtt{m1}, *, *, ? : \mathsf{F}(D))$ *contains the first two cells. In the cell* $(\mathtt{m1}, *, *, \mathtt{s2} : \mathsf{F}(D))$ *of cuboid* MS*, two documents are aggregated in* $D = \{d_2, d_3\}$. *In the cell* $(\mathtt{a2}, *, *, \mathtt{d1} : \mathsf{F}(D))$, $D = \{d_5, d_6\}$. *A simple measure is, for example,* $\mathsf{F} = \mathsf{CNT}$*, the number of documents. Then in the cell* $(\mathtt{m1}, *, *, \mathtt{s2} : \mathsf{CNT}(D))$, $\mathsf{CNT}(D) = 2$.

2.2 Hierarchy and Operations

2.2.1 Dimension Hierarchy

The same as traditional data cubes, each dimension may consist of multiple attributes, and be organized as a tree or a DAG, called *dimension hierarchy*. There are four related OLAP operations *Roll-up*, *Drill-down*, *Slice* and *Dice* [4].

2.2.2 Term Hierarchy

Term hierarchy $\mathcal{T}$ is built upon a term set $\mathcal{W}$ to specify terms' semantic levels and relationship. Each node v in $\mathcal{T}$ is called a *generalized term*, represented by a subset of terms, i.e., $v \subseteq \mathcal{W}$. In particular, each *leaf* is a size-1 set $\{w_i\}$ containing one term $w_i \in \mathcal{W}$, and the *root* of $\mathcal{T}$ is the set of all terms, $\mathcal{W}$. Let $chd(v)$ denote the set of v's *child nodes*, $des(v)$ denote the set of v's *descent nodes*, and $par(v)$ denote v's *parent node*.

Example 2.2 *Term hierarchy* $\mathcal{T} = \{$v1, ..., v14$\}$ *(Figure 1) is built upon the term set* $\mathcal{W} = \{$w1, ..., w8$\}$. *For example, the generalized term* v9 $= \{$Memory, CPU, Disk$\}$ *represents "internal device".*

Term level and related OLAP operations A *term level* is a subset of nodes in $\mathcal{T}$, denoted by $L \subset \mathcal{T}$. The set of all leaf nodes, denoted by L_0, is a term level called *base level*. The set $L_{\mathsf{top}} = \{v_{\mathsf{root}}\}$ is a term level called *top level*. Other term levels can be generated by:

[1] Our cube model and algorithms can be easily extended for the text database, where each row contains more than one document.

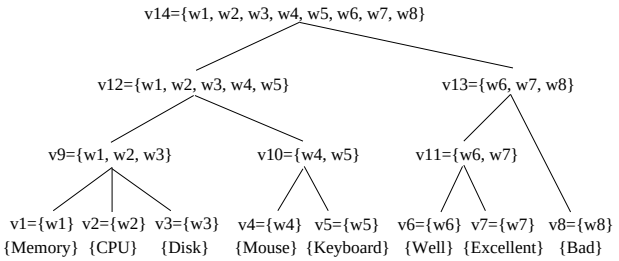


Figure 1. Term Hierarchy $\mathcal{T}$

- Pull-up L on v: Given term level L and generalized term $v \in L$, add v's parent node $u = par(v)$ in and delete u's descent nodes $des(u)$ from L. The result is a higher term level L'.
- Push-down L on v: the reverse of *push-down*.

Example 2.3 *In Figure 1,* $L_0 = \{\text{v1}, \text{v2}, \ldots, \text{v8}\}$.
Pull-up L_0 *on* $\text{v1} \rightarrow L_1 = \{\text{v9}, \text{v4}, \text{v5}, \text{v6}, \text{v7}, \text{v8}\}$.
Push-down $L_{\mathsf{top}} = \{\text{v14}\}$ *on* $\text{v14} \rightarrow L_4 = \{\text{v12}, \text{v13}\}$.

2.3 Measures Suppported for IR

Which measures are essential for IR tasks? After a survey, we select *term frequency* TF and *inverted index* IV.

Let $des'(w) = w \cup des(w)$. For term w and document d, $tf_{w,d}$ denotes the total times terms in $des'(w)$ appear in d, $\mathsf{TF}(w, D) = \sum_{d \in D} \{tf_{w,d}\}$ denotes the total times terms in $des'(w)$ appear in D, and $\mathsf{IV}(w, D) = \{(d, tf_{w,d}) \mid tf_{w,d} > 0\}$ denotes the list of documents in D containing terms in $des'(w)$. Suppose D is the aggregated text data for a cuboid cell and $\mathcal{W} = \{w_1, w_2, \ldots, w_m\}$ is the set of all terms, *term frequency vector* is an m-dimensional vector

$$\mathsf{TF}(D) = \langle \mathsf{TF}(w_1, D), \mathsf{TF}(w_2, D), \ldots, \mathsf{TF}(w_m, D) \rangle \quad (1)$$

and *inverted index* is an m-dimensional vector

$$\mathsf{IV}(D) = \langle \mathsf{IV}(w_1, D), \mathsf{IV}(w_2, D), \ldots, \mathsf{IV}(w_m, D) \rangle. \quad (2)$$

Example 2.4 *See Table 1(a). Cell* $(\texttt{m1}, *, *, \texttt{s2} : \mathsf{TF}(D))$ *has* $D = \{d_2, d_3\}$ *and* $\mathsf{TF}(D) = \langle 1, 1, 2, 0, 0, 4, 1, 0 \rangle$ *(*w1 *appears 1 time,* w2 *appears 1 time, ...); cell* $(\texttt{m1}, *, *, \texttt{s2} : \mathsf{IV}(D))$ *has* $\mathsf{IV}(w_6, D) = \{(d_2, 2), (d_3, 2)\}$ *(*w6 *appear in* d_2 *and* d_3 *both for two times).*

3 Text Cube: Computational Aspects

In this section, we first discuss how to compute the full cube and analyze the storage cost in Section 3.1. It will be shown although our two measures are *distributive*, the storage cost is prohibitive. So in Section 3.2, we focus on how to process online queries in a *partially materialized* text cube, and in Section 3.3, we introduce how to optimize the storage size of partially materialized text cubes while the *query processing cost* is bounded.

3.1 Full Cube Computation

The basic algorithm to do full text cube computation is: first compute all cells in n-D cuboids; then compute all cells in i-D cuboids from cells in $(i-1)$-D cuboids. The key points of this algorithm are: (i) how much the storage cost is; and (ii) how to aggregate cells in a r-D cuboid into a cell in a $(r-1)$-D cuboid without looking the original database.

Storage cost. If term set is $\mathcal{W} = \{w_1, w_2, \ldots, w_m\}$, $O(m)$ space is required by TF for each cell, and $O(m|D|)$ space is required by IV for a cell aggregating document set D.

Aggregation of TF and IV. The two measures under our consideration, TF and IV, are *distributive* [2].

Let $(a_1, a_2, \ldots, a_n : D)$ be a cell in a $(r-1)$-D cuboid. W.o.l.g. suppose $a_n = *$ and $\mathcal{A}_n$ has k distinct values $a_n^{(1)}, a_n^{(2)}, \ldots, a_n^{(k)} \in \mathcal{A}_n$. Consider k cells in a r-D cuboid, namely, $(a_1, a_2, \ldots, a_n^{(j)} : D_j)$ for $j = 1, 2, \ldots, k$. We show $\mathsf{TF}(D)$ and $\mathsf{IV}(D)$ can be efficiently computed from $\mathsf{TF}(D_1), \mathsf{TF}(D_2), \ldots, \mathsf{TF}(D_k)$ and $\mathsf{IV}(D_1), \mathsf{IV}(D_2), \ldots, \mathsf{IV}(D_k)$, respectively.

Suppose $L = \{v_1, v_2, \ldots, v_p\}$. Because we have $D = \bigcup_{1 \le i \le k} D_i$ and $D_i \bigcap D_j = \varnothing$ for $i \neq j$, $\mathsf{TF}(D)$ and $\mathsf{IV}(D)$ can be computed as follows.

$$\mathsf{TF}(v_i, D) = \sum_{1 \le j \le k} \mathsf{TF}(v_i, D_j), \text{ for } i = 1, 2, \ldots m, \quad (3)$$

and $\mathsf{TF}(D) = \langle \mathsf{TF}(v_1, D), \mathsf{TF}(v_2, D), \ldots, \mathsf{TF}(v_m, D) \rangle$.

$$\mathsf{IV}(v_i, D) = \bigcup_{1 \le j \le k} \mathsf{IV}(v_i, D_j), \text{ for } i = 1, 2, \ldots m, \quad (4)$$

and $\mathsf{IV}(D) = \langle \mathsf{IV}(v_1, D), \mathsf{IV}(v_2, D), \ldots, \mathsf{IV}(v_m, D) \rangle$.

To sum up, we can conclude that the time required to compute TF and IV through aggregation is linear w.r.t. their sizes, and these two measures are *distributive*.

3.2 Query Processing in Partially Materialized Cube

Although TF and IV can be efficiently aggregated, they consume a huge amount of space if materialized for all cells. Our solution is to recompute a *subset* of cells instead all cells, called *partially materialized*. We introduce how to process OLAP queries in a text cube where only a *subset* of cells are precomputed; and in Section 3.3, we discuss how to choose this subset.

Two types of OLAP queries: (1) *point query* (seek a cell) (2) *subcube query* (seek a subcube). Since (2) can be processed by calling (1) as a subroutine, we will focus on (1).

Formally, in a n-dimensional text database $\mathcal{DB} = (\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n, \mathcal{DOC})$, A *point query* is in the same form as a cell, i.e., $(a_1, a_2, \ldots, a_n : \mathsf{F}(D))$, where $a_i \in \mathcal{A}_i \cup \{*\}$.

For a point query on a cell which is not precomputed, there are different ways of choosing the set of precomputed ones to obtain the result. To formally define the *query processing problem*, we first need to introduce the *decision space* and the *cost model*.

Decision space. Let us first explore what kind of choices we have to aggregate precomputed cells to process a point

query. Given a point query, $Q = (a_1, a_2, \ldots, a_n : \mathsf{F}(D))$, w.o.l.g. suppose $a_i = *$ for $i = 1, 2, \ldots, n'$ and $a_i \in \mathcal{A}_i$ for $i = n'+1, n'+2, \ldots, n$. We have n' choices; i.e. aggregating cells in one of the following sets:

$$S_1 = \{(a, a_2, \ldots, a_{n'}, \ldots, a_n : \mathsf{F}(D)) \mid a \in \mathcal{A}_1 \wedge D \neq \varnothing\},$$
$$\ldots\ldots$$
$$S_{n'} = \{(a_1, a_2, \ldots, a, \ldots, a_n : \mathsf{F}(D)) \mid a \in \mathcal{A}_{n'} \wedge D \neq \varnothing\}.$$

on one of dimensions $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_{n'}$, respectively. Suppose we choose to aggregate cells in S_1 on $\mathcal{A}_1$, for each cell in S_1, $(a, a_2, \ldots, a_{n'}, \ldots, a_n : \mathsf{F}(D))$, if it is precomputed, it can be directly retrieved for processing Q; otherwise, recursively, to obtain this cell, we have $n'-1$ choices to aggregating other cells on one of dimensions $\mathcal{A}_2, \mathcal{A}_3, \ldots, \mathcal{A}_{n'}$.

Cost model. Given a point query Q, the cost of processing Q with a set of precomputed cells, C, is $|C|$ (i.e. the number of precomputed cells we need to access).

Query processing problem. Since in the decision space, we have different choices of the set of precomputed cells for processing Q, the *query processing problem* is:

- *Given a point query Q, choose the minimum number of precomputed cells for processing Q.*

3.2.1 Optimal Query Processing

Now we focus on the *query processing problem*. We use dynamic programming to compute the optimal cost/decision.

In a n-dimensional text cube, we use $Q = \{a_1, a_2, \ldots, a_n : \mathsf{F}(D)\}$ to denote both a point query and a cell. For each $a_i = *$, we use $Q_{i \to a}$ to denote a point query/cell, which replace $a_i = *$ with $a \in \mathcal{A}_i$; i.e.

$$Q_{i \to a} = \{a_1, \ldots, a_{i-1}, a, a_{i+1}, \ldots, a_n : \mathsf{F}(D)\}. \quad (5)$$

In our cost model, once the precomputed cells are fixed, for each cell with value $*$ in n' dimensions, there is an "optimal" choice among the n' possible ones. This is because the optimal choice for $Q_{i \to a}$ is irrelevant to the one for Q (i.e. the *optimal substructure* in dynamic programming).

So, let $\mathsf{cost}(Q)$ be the optimal cost for processing Q, i.e., the minimum number of precomputed cells needed to be accessed, we have the following recursive relationship:

$$\mathsf{cost}(Q) = \min_{i: a_i = *} \left\{ \sum_{a \in \mathcal{A}_i} \mathsf{cost}(Q_{i \to a}) \right\}. \quad (6)$$

Note for a precomputed cell Q, as in our cost model, define

$$\mathsf{cost}(Q) = 1 \quad (Q \text{ is a precomputed cell}). \quad (7)$$

If Q is *empty*, i.e., $D = \varnothing$, we can avoid accessing it, so

$$\mathsf{cost}(Q) = 0 \quad (Q \text{ is empty}). \quad (8)$$

If the precomputed cells in a text cube are fixed, the optimal cost/decision for processing each Q can be computed from (6)-(8). Recall in our decision space, for a point query Q with $*$ in n' dimension, we have n' choices to obtain it (each choice is to aggregate $\{Q_{i \to a} \mid a \in \mathcal{A}_i\}$ on dimension $\mathcal{A}_i$ if $a_i = *$). So we take the minimum cost over all choices as in (6). But it is important to notice this process is recursively repeated: if a cell, say $Q_{i \to a}$, is not precomputed, we need to find the set of precomputed ones to obtain $Q_{i \to a}$.

Let $\mathsf{desc}(Q)$ be the optimal choice of the aggregating dimension for a cell Q that is not precomputed. From (6),

$$\mathsf{desc}(Q) = i_0 \text{ iff } a_{i_0} = * \wedge \mathsf{cost}(Q) = \sum_{a \in \mathcal{A}_{i_0}} \mathsf{cost}(Q_{i_0 \to a}). \quad (9)$$

For each cell Q, $\mathsf{cost}(Q)$ and $\mathsf{desc}(Q)$ can be precomputed from the recursive relationship (6)-(8), and stored in the text cube (as they only consume $O(1)$ space). When a point query Q arrives online, it can be processed by first tracing $\mathsf{desc}(Q)$ to get a set of precomputed cells, C, and then aggregating them to obtain Q's measure (TF or IV). C can be gotten using algorithm **Process**.

Process$(Q = (a_1, a_2, \ldots, a_n : \mathsf{F}(D)))$
1: **if** Q **is precomputed then** $C \leftarrow C \cup \{Q\}$;
2: **else**
3: $\quad i_0 \leftarrow \mathsf{desc}(Q)$;
4: $\quad$ **for each** $a \in \mathcal{A}_{i_0}$ **do**
5: $\quad\quad$ **if** $Q_{i_0 \to a}$ **is nonempty then Process**$(Q_{i_0 \to a})$;

The following theorem formalized the correctness of our algorithms, but we omit the proof for the space limit.

Theorem 1 *Given a point query Q in a partially materialized text cube, the optimal cost* $\mathsf{cost}(Q)$ *and choice* $\mathsf{desc}(Q)$ *for processing Q can be computed from* (6)-(9) *using dynamic programming algorithm.*

From the optimal choice $\mathsf{desc}(\cdot)$ *of each cell, the set of precomputed cells needed to be aggregated to obtain Q's measure can be computed with algorithm* **Process**.

3.3 Optimizing Cube Materialization with Bounded Query Processing Cost

Cube materialization problem. The remaining question is how to choose a subset of cells to precompute, s.t.

(i) Any point query can be answered by aggregating (some) precomputed cells.

(ii) For any point query Q, its optimal processing cost $\mathsf{cost}(Q)$ is bounded by *a user-specified threshold* Δ.

(iii) The *storage cost* (i.e. the total number of precomputed cells) is as small as possible.

For a n-dimensional text database $(\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n, \mathcal{DOC})$, from (i), all cells in the n-D cuboid must be precomputed, because a cell in the n-D cuboid cannot be obtained by aggregating other cells. On the other hand, if all cells in the n-cuboid are precomputed, then any point query can be answered, by aggregating a subset of these n-cuboid cells. Given a threshold Δ, in the following part, we focus on how to reduce the storage cost while promising the optimal cost

of processing any point query is bounded by Δ. We define a *partial order*, $\preceq$, on the set of all nonempty cells in a text cube. Two cells $Q = (a_1, a_2, \ldots, a_n : D) \preceq Q' = (a'_1, a'_2, \ldots, a'_n : D')$ iff "$a'_i \in \mathcal{A}_i \Rightarrow a'_i = a_i$". Consider a *topological sort* based on this partial order of all nonempty cells, $Q^{(1)}, Q^{(2)}, \ldots, Q^{(N)}$, where N is the total number of cells, we have $Q^{(i)} \preceq Q^{(j)} \Rightarrow i \leq j$.

Recall the definition of $Q_{i\to a}$ in (5), we must have $Q_{i\to a} \preceq Q$ according to the definition of "$\preceq$". So we must have the following property.

Property 1 *Suppose $Q_{i\to a} = Q^{(j)}$ and $Q = Q^{(k)}$ in the topological sort, we must have $j \leq k$.*

Our method to partially materialized a cube works as follows: linearly scan the topological sort from $Q^{(1)}$ to $Q^{(N)}$; in this process for $j = 1, 2, \ldots, N$, from Property 1, we can correctly compute $\mathsf{cost}(Q^{(j)})$ and $\mathsf{desc}(Q^{(j)})$; if $\mathsf{cost}(Q^{(j)}) > \Delta$ for some j, we precompute and store cell $Q^{(j)}$ in the text cube, and let $\mathsf{cost}(Q^{(j)}) = 1$. In this way, after all cells are scanned, the optimal cost $\mathsf{cost}(\cdot)$ and decision $\mathsf{desc}(\cdot)$ are computed for each cell, and $\mathsf{cost}(Q)$ for any query Q is no larger than Δ.

The intuition of why our method can reduce the storage cost is: we precompute cells as later as possible in the topological order. If the processing cost $\mathsf{cost}(Q)$ of a cell Q does not exceed the threshold Δ, we delay its computation to the online query processing, because it can be aggregated from the already precomputed ones with processing cost no more than Δ. Therefore, we reduce the redundancy in the precomputation. Given $Q^{(1)}, Q^{(2)}, \ldots, Q^{(N)}$ in a n-dimensional text cube and Δ, we formalize our cube materialization method as algorithm $\mathbf{GreedySelect}$.

$\mathbf{GreedySelect}(\Delta)$
1: **for** $j = 1$ **to** N **do**
2: **if** $Q^{(j)}$ is a n-D cuboid **then**
3: **precompute** $Q^{(j)}$; $\mathsf{cost}(Q^{(j)}) \leftarrow 1$;
4: **else**
5: $\mathsf{cost}(Q^{(j)}) \leftarrow \min_{i:a_i=*} \left\{\sum_{a\in\mathcal{A}_i} \mathsf{cost}(Q^{(j)}_{i\to a})\right\}$;
6: $\mathsf{desc}(Q^{(j)}) \leftarrow \operatorname{argmin}_{i:a_i=*} \left\{\sum_{a\in\mathcal{A}_i} \mathsf{cost}(Q^{(j)}_{i\to a})\right\}$;
7: **if** $\mathsf{cost}(Q^{(j)}) > \Delta$ **then**
8: **precompute** $Q^{(j)}$; $\mathsf{cost}(Q^{(j)}) \leftarrow 1$;

Theorem 2 *The optimal query-processing cost $\mathsf{cost}(Q)$ and choice $\mathsf{desc}(Q)$ for every cell Q is correctly computed in* $\mathbf{GreedySelect}$*, in $O(N \cdot \max_i\{|\mathcal{A}_i|\} + P_{\mathsf{time}})$ time, where P_{time} is the time for precomputing cells.*

Although we cannot promise $\mathbf{GreedySelect}$ minimizes the storage cost of a partially materialized text cube, s.t. conditions (i) and (ii) in cube materialization problem are satisfied, we will show the effectiveness of $\mathbf{GreedySelect}$ in our performance study (Section 4.1). Minimizing the storage cost and its hardness are open questions.

4 Experimental Studies

We did experiments on a real dataset. $\mathbf{GreedySelect}$ partially materializes text cubes; $\mathbf{Process}$ processes online queries; $\mathbf{Basic}$ answers queries without the support of text cube. Algorithms are implemented in C++ 2005 and SQL 2005, conducted in 3.40GHz CPU and 1G memory PC.

Dataset The dataset is the complete set of customer reviews on Dell laptops crawled from *www.dell.com* before 2008/05/15, which contains 2,013 records with 232,924 text words. We extracted 26 attributes, among which we use 14 categorical attributes dimensions, and customer comments on *pros* and *cons* as the documents.

4.1 Performance Study

Exp-1 (Storage cost). We report the storage costs while varying (i) the threshold of query processing cost Δ in $\mathbf{GreedySelect}$ and (ii) the number of dimensions of our text cube. When the number of dimensions is 14, there are 16,384 cuboids and 1,269,043,200 cells in the text cube.

In Figure 2, the three curves, Cube20, Cube60, and Cube100, shows the storage costs of text cube when $\Delta =$ 20, 60, and 100, respectively.

In principle, the smaller Δ is, the more cells need to be precomputed, and the larger the storage cost is. This fact can be verified by Figure 2, since Cube20 is always the top curve, and Cube100 is always the bottom one. Also, Figure 2 shows that the storage cost increases as the number of dimensions increases. But even when the number of dimensions reaches 14, the storage cost is no more than 70(MB).

Exp-2 (query processing time). We compare the processing time of $\mathbf{Process}$ with the one of $\mathbf{Basic}$. We conclude that a tradeoff between storage cost and query processing time can be controled it tuning Δ in $\mathbf{GreedySelect}$.

In Figure 3(a), the average query processing time is curved as a function of the number of aggregated documents (i.e. if the corresponding cell of a point query is $(a_1, a_2, \ldots, a_n : D)$, this quantity is $|D|$). It is shown $\mathbf{Basic}$ increases approximately linearly w.r.t. $|D|$; but Cube20, Cube60, and Cube100 is shorter than $\mathbf{Basic}$ when $|D|$ is large, and is independent of $|D|$. Actually, it vibrates as $|D|$ increases. The reason can be explained by the behavior of $\mathbf{GreedySelect}$: at first, all base cells (cells in n-D cuboids) are precomputed; then, as more and more documents are aggregated in one cell, the cost of query processing increase; when the cost reaches the threshold Δ, we begin to precompute cells again. This behavior repeats periodically, and so that the query processing time vibrates in periods as $|D|$ increases. Moreover, the larger Δ is, the more sharply it vibrates. The query processing time in Cube20 is the shortest on average and is the most smooth one.

In Figure 3(b), the average query processing time is plotted as a function of the number of dimensions. For similar reason, $\mathbf{Basic}$ increases linearly, but Cube20, Cube60, and

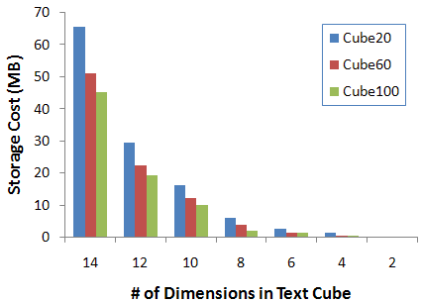
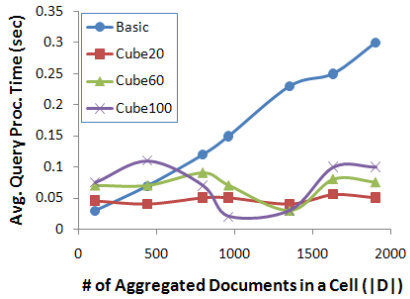
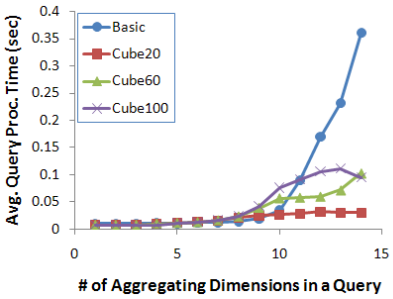


Figure 2. Storage Cost



(a) Varying $|D|$



(b) Varying # of aggregateing dimensions

Figure 3. Query Time

Cube100 also vibrates a bit.

In both Figure 3(a) and 3(b), the average query processing time in Cube20 is the shortest among Cube20, Cube60, and Cube100. Recall Figure 2 in Exp-1, Cube20 requires the most storage cost, so we have a *tradeoff* between storage cost and query processing time, and we can control it by tuning threshold Δ in algorithm **GreedySelect**.

4.2 Case Study

A case study in Table 2 shows the utility of text cube.

Dell XPS M1730 is marketed to gamers, but criticized for its looks, increasing weight and size [2]. We *roll up* all dimensions except *model* to $*$, and slice on $model =' M1730'$, obtaining the top 10 most frequent terms in of *pros* and *cons*. It can be seen that the customers basically agree with Wikipedia, except that customers do not complain much on its looks but care about the high price [3].

Another example is Inspiron 1420 N. which is known for portable models, Ubuntu OS, 8 colors, but limitation on hardware capacity [4].

	XPS M1730	Inspiron 1420N
pros	game great keyboard work video nice design amaze display speed	nice love color light wireless cheap driver webcam quick speedy
cons	hard expensive battery problem keyboard carry game heavy screen need	battery problem gb hibernate ram virus button pad ubuntu g

Table 2. Dell XPS M1730 and Inspiron 1420 N

5 Conclusion

In this paper, we proposed a novel cube model called *text cube* with OLAP operations in *dimension hierarchy* and *term hierarchy* to analyze multi-dimensional text data. It supports two important measures, term frequency vector TF and inverted index IV, to facilitate IR techniques. Algorithms are designed to process OLAP queries with the optimal processing cost, and to partially materialize the text cube provided that the optimal processing cost of any query is bounded. Experimental results on a real dataset show the efficiency and effectiveness of our text cube model.

6 Acknowledgement

The work was supported in part by NASA grant NNX08AC35A, the U.S. National Science Foundation grants IIS-08-42769 and BDI-05-15813, and Office of Naval Research (ONR) grant N00014-08-1-0565.

References

[1] K. S. Beyer and R. Ramakrishnan. Bottom-up computation of sparse and iceberg cubes. In *SIGMOD Conference*, pages 359–370, 1999.

[2] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-total. In *ICDE*, pages 152–159, 1996.

[3] J. Han. Olap, spatial. In *Encyclopedia of GIS*, pages 809–812. 2008.

[4] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *SIGMOD Conference*, pages 205–216, 1996.

[5] J. Li, H. Zhou, and W. Wang. Gradual cube: Customize profile on mobile olap. In *ICDM*, pages 943–947, 2006.

[6] E. Lo, B. Kao, W.-S. Ho, S. D. Lee, C. K. Chui, and D. W. Cheung. Olap on sequence data. In *SIGMOD Conference*, pages 649–660, 2008.

[7] Y. Tian, R. A. Hankins, and J. M. Patel. Efficient aggregation for graph summarization. In *SIGMOD Conference*, pages 567–580, 2008.

[8] Y. Zhao, P. Deshpande, and J. F. Naughton. An array-based algorithm for simultaneous multidimensional aggregates. In *SIGMOD Conference*, pages 159–170, 1997.

[2] http://en.wikipedia.org/wiki/Dell_XPS

[3] From http://www.dell.com/, its price is more than $ 1300.

[4] http://en.wikipedia.org/wiki/Dell_Inspiron