Query Processing on Cubes Mapped from Ontologies to Dimension Hierarchies

Carlos Garcia-Alvarado
University of Houston
Greenplum/EMC

Carlos Ordonez
University of Houston
Dept. of Computer Science

ABSTRACT

Text columns commonly extend core information stored as atomic values in a relational database, creating a need to explore and summarize text data. OLAP cubes can precisely accomplish such tasks. However, cubes have been overlooked as a mechanism for capturing not only text summarizations, but also for representing and exploring the hierarchical structure of an ontology. In this paper, we focus on exploiting cubes to compute multidimensional aggregations on classified documents stored in a DBMS (keyword frequency, document count, document class frequency and so on). We propose CUBO (CUBed Ontologies), a novel algorithm, which efficiently manipulates the hierarchy behind an ontology. Our algorithm is optimized to compute desired summarizations without having to search all possible dimension combinations, exploiting the sparseness of the document classification frequency matrix. Experiments on large text data sets show CUBO can explore faster more dimension combinations than a standard cube algorithm, especially when the cube has a large number of dimensions. CUBO was developed entirely inside a DBMS, using SQL queries and extensibility features.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Query processing*

General Terms

Algorithms, Experimentation, Measurement

Keywords

Data Cubes, Text, Hierarchies, DBMS, Ontologies

1. INTRODUCTION

Information retrieval has exhaustibly explored ontologies for document indexing, query recommendation, and query expansion, among many other information retrieval techniques [13, 17, 3]. Moreover, ontology usage has been extended to other domains such as social networks [14], ontology-based databases [2], geographic information systems [6], and even for XML Data and unstructured data analysis [5]. Special interest has arisen in the latter due to the advantages of analyzing the semantics of a set of documents (corpora) beyond a set of keywords. This summarization tasks are remarkably valuable in domains with existing ontologies, such as the medical and digital libraries domains (e.g. ACM Digital Library), where the ability to explore efficiently in different levels of granularity their large data sets of unstructured data is a pressing need. As the amount of text data continues to grow in database management systems (DBMSs), the data warehousing community has realized the potential of existing methods, such as Online Analytical Processing (OLAP), to provide efficient and compact data structures for analyzing large amounts of data. While traditional applications of OLAP include business reports and financial analysis [4], OLAP provides clear advantages for computing efficient summarization structures that can manage the hierarchical model that is the result of including an ontology. This combination of OLAP techniques for analyzing text is explored in only a few works [15, 16], that have proposed extensions for adapting OLAP to summarize data sets and take advantage of a given ontology. Unfortunately, the scalability of the current proposals and adaptability of the sparse nature of text data remains unexplored.

In this research, we propose CUBO (CUBed Ontologies), a novel algorithm and data structure that integrate a multilevel data cube into a single data structure. The main objective of CUBO is to allow an efficient summarization of all the different levels that are part of a given ontology. More specifically, we show that OLAP techniques can be used to efficiently obtain the aggregations of classes, represented as dimensions (in OLAP terminology) in a set of documents. As a result, the user will be able to obtain a data cube containing aggregations (COUNT, SUM, AVERAGE, MIN, MAX) on several measurements (e.g. term frequency, number of documents) that are related to a set of classes in different hierarchies. An example might be, a data cube with the different levels of the hierarchy, as shown in Figure 1. This data cube represents a set of articles summarized based on the ACM Computing Classification System with a hierarchy of height three. Notice that within each level,

i	D_j	A_1
1	Parallel Processors	30
1	Array and Vector Processors	30
2	Parallel Processors	40
2	Distributed databases	40
3	Array and Vector Processors	50
3	Distributed databases	50
3	Parallel Processors	50
4	Parallel Processors	180
5	Array and Vector Processors	40
6	Distributed databases	310
7	Distributed databases	400

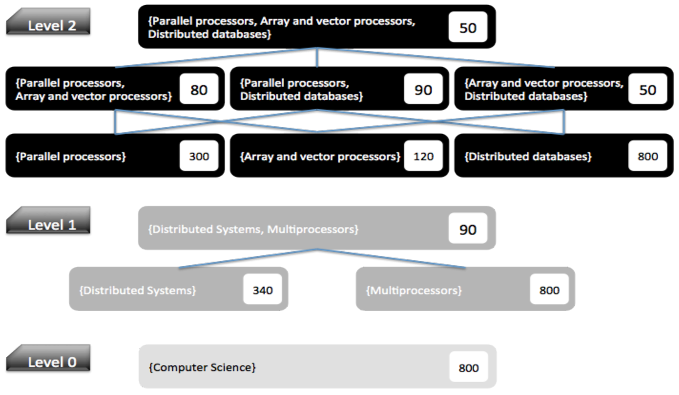
Level 2
{Parallel processors, Array and vector processors, Distributed databases} 50
{Parallel processors, Array and vector processors} 80
{Parallel processors, Distributed databases} 90
{Array and vector processors, Distributed databases} 50
{Parallel processors} 300
{Array and vector processors} 120
{Distributed databases} 800
Level 1
{Distributed Systems, Multiprocessors} 90
{Distributed Systems} 340
{Multiprocessors} 800
Level 0
{Computer Science} 800

Figure 1: Data cube Q and Fact Table F based on The ACM Computing Classification System (1998) with $\{D_1$ =Parallel processors, D_2 =Array and vector processors, D_3=Distributed databases$\}$.

the resulting summarization can be seen as a smaller data cube. This research formalizes and introduces algorithms for processing text data that have not previously been proposed, where summarization and hierarchies are not taken into consideration. The resulting algorithm and data structure are a compact and efficient tool adapted to a database management systems (DBMS) tailored for the sparse nature of text corpora.

This paper is organized as follows: Section 2 presents the notation used throughout this paper. Even though this paper deals with ontologies, the entire notation is focused on the OLAP common notation for mapping ontologies to dimension hierarchies. In addition to this, the main concepts of ontologies and OLAP are described. Section 3 shows our novel algorithm for building CUBO using ontologies. Also, we present a complexity analysis of how the complexity changes from a traditional OLAP data cube. In Section 4, the performance evaluation and scalability experiments on two collections are presented. Section 5, discusses closely related work. Finally, in Section 6, we make our final remarks and discuss future work.

2. DEFINITIONS

Let us focus on defining the required OLAP and ontology notation that will be used to obtain a dimension hierarchy. Due to the interdisciplinary nature of this research, we propose a combination of data cubes and information retrieval definitions. Let a collection, or corpus, of n documents, where each document t_i is composed of a set of dimensions $\{D_1, D_2, \ldots D_j, \ldots, D_k\}$, where k is the number of the dimensions required for building the data cube. Examples of dimensions in a corpus are the topics "Parallel Processor" or "Distributed Database". Moreover, let every set of dimensions in a document be accompanied by a set of attributes $\{A_1, A_2, \ldots, A_m, \ldots, A_e\}$, where e represents the total number of attributes and A_m is a specific attribute (e.g. minimum common frequency of a set of attributes). Moreover, let the collection be stored in a relational table inside a DBMS. Thus, F is a table defined as a vertical fact table $F(i, D_j, A_1, \ldots, A_e)$ of size $e+2$ by $\sum_{i=1}^{n} |t_i|$, where i represents the document number, D_j is a dimension, and A_m is a summarization attribute. A_1 to A_e are constant within every D_j of a document to avoid generating a much larger fact table. Notice that F differs from a traditional fact table for OLAP, because the dimensions are not represented in a horizontal manner and all the attributes are replicated within every D_j. This is due to the sparse nature of text data and the limitation of the number of columns that is present in a traditional row store DBMS. Additionally, if the attributes are not considered to be constant within the same document, additional processing within a document must be performed to obtain a unique aggregation measurement. The backbone for OLAP data cubes is the dimensional lattice, which has a size of 2^k combinations for a set of dimensions, where k is the number of dimensions. Furthermore, a user-query Q is a subset of dimensions from F which builds an OLAP data cube for every level in a given dimension hierarchy represented by the l subscript.

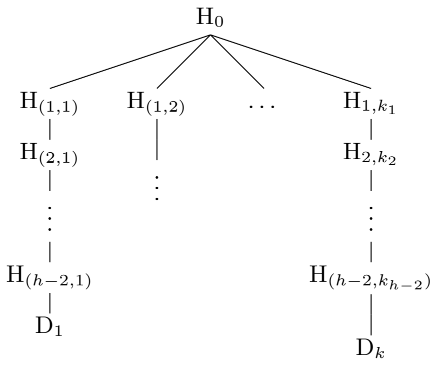

Figure 2: Ontology.

An ontology $\mathcal{O}$, on the other hand, is mapped to a dimension hierarchy as a tree-like structure where the nodes represent dimensions and the branches model the relationships between them. In Figure 2, the k leaf nodes D_j represent the ground level dimensions and H_{h-2} to H_0 represent the parent dimensions, where h is the depth or height of the ontology. In addition, the number of dimensions per level k_l is always less than or equal to the following $k_{(l+1)}$. Lastly, we will consider an ontology to have an overall H_0 root class (e.g "thing"), and every dimension to have only a single parent (except for the root that has none). In relational terms, every concept reference between levels (e.g. D_j and H_{h-2}) can be seen as a dimension table. However, in this approach, the entire data structure is kept in main memory (to be de-

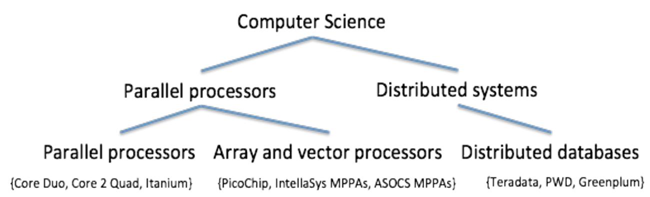
Computer Science
Parallel processors
Distributed systems
Parallel processors
Array and vector processors
Distributed databases
{Core Duo, Core 2 Quad, Itanium}
{PicoChip, IntellaSys MPPAs, ASOCS MPPAs}
{Teradata, PWD, Greenplum}

Figure 3: Ontology Example.

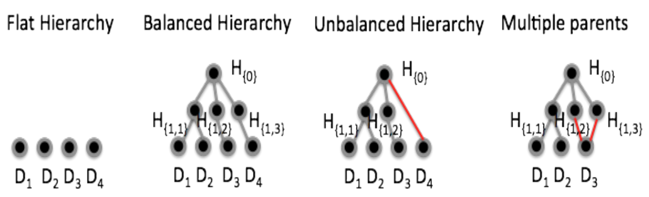

Figure 4: Hierarchies.

tailed in sections to follow) as a tree during the execution of the algorithm.

2.1 Ontologies

Ontologies embody formal representations of knowledge. As such, several language proposals have been used to write explicit formal conceptualizations of domain models [1]. These proposals include the popular RDF/RDFS, as well as some other representations such as DAML +OIL and OWL languages, among many others. RDF/RDFS allows some basic knowledge representation. Despite this, features such as the local scope of properties, the disjointness of classes, Boolean combinations of classes, cardinality restrictions and special characteristics or properties are left out of this language [1]. In order to model more complex relations between classes, DAML+OIL took an object oriented approach [10]. Finally, OWL was developed to standardize all the previous language approaches in a more robust knowledge representation.

Regardless of the ontology language, all these knowledge representations rely on the specification of the main concepts in a given context. The representation of every concept is defined as a class with a set of properties and interactions between the classes, subclasses, and their properties. A class (e.g. owl:Class) is a classification of individuals that share common characteristics. This class classification is obtained through a taxonomy (hierarchy). In this work, CUBO is not tied to a specific language, but on the classes (dimensions) and their relationships. An example of a class taxonomy is given in Figure 3, in which the ontology used for the data cube example is shown. The first level corresponds to concepts D_1 =Parallel processors, D_2 =Array and vector processors, D_3=Distributed databases; the second level to concepts $H_{(1,1)}$ =Parallel processors and $H_{(1,2)}$ =Distributed systems; and the root level (H_0) is for the Computer Science concept. Each node in the taxonomy is a class, and the branches in the tree indicate a type of relationship (e.g. subClassOf). The instances represented in the leaf classes are specific to a particular class (which may or may not be considered as part of the data cube).

2.2 OLAP

OLAP techniques are generally used to quickly process complex queries involving multiple aggregations [9]. While traditional applications of OLAP include summarizations in the form of business reports and financial analysis of large data sets [4], we believe that the OLAP dimensional lattice can be used to efficiently summarize classes and their relations present within a corpus. The main benefit of OLAP is the ability to analyze large amounts of data at various levels of aggregation to obtain additional information. In this case, the dimensions are represented by the collection of classes, the attributes are measurements relating a document and a class, while the level of aggregations (ontology) allows us to obtain various combinations of the parents of the leaf dimensions.

Traditional OLAP accepts inputs that are horizontal, which means that each row contains all dimensions used in the analysis. However, in our system, the fact table has a vertical layout, which translates into each row containing a single dimension. This presents a challenge for our algorithm because we are unable to use normal techniques, such as slicing, with the vertical layout. Despite this, we are only analyzing a small subset (the subset of dimensions in Q) of all dimensions in the corpus. It is nevertheless time consuming to pivot the dimension into a horizontal layout. This is especially costly when the number of documents is large and the dimensions are sparse. As a result, we developed an algorithm that can obtain the necessary aggregations and perform the required auxiliary computations by taking only the desired dimensions. Likewise, the entire data cube for all existing combinations is computed with no minimum threshold required.

OLAP data cubes are able to capture hierarchies in the data. These hierarchies can be described as flat, balanced, or unbalanced (see Figure 4). The most basic type of hierarchy is the flat one, which is the one that we captured in our early works where only the leaf nodes are present. A second type of hierarchies is the balanced ones, where all the nodes are present for every level of the tree. Finally, unbalanced hierarchies are those that do not have nodes in every level of the hierarchy. All these types of hierarchies are able to be represented in an OLAP data cube. However, unbalanced hierarchies require additional information in the ontology, such as providing a hierarchy level for each class as a property. Despite these advantages for managing hierarchies, it is not possible to model every hierarchy variant, such as a node with multiple parents. They cannot be represented in a transparent manner. In this case, a separate hierarchy is needed to model every parent-child relation.

3. CUBO

CUBO is an array of data cubes that contain only the existing dimensions within the fact table F given Q. However, the notion of CUBO is based on the premise that a set of documents does not contain all the dimensions in every document. Thus, only a set of "on-demand" computations of the dimensions are required for every document. Moreover, our algorithm takes advantage of this property when computing the aggregation on superior levels in the hierarchy. In other words, CUBO has a lazy policy that waits to perform the computations until it is absolutely necessary. Furthermore, the algorithm avoids redundant computations by focusing only on storing those dimensions that contain attributes to aggregate. If the entire data cube is desired,

Algorithm 1: CUBO

```
Input: $\mathcal{O}$,$F$,$Q$,$T$,$\{A_1,\dots\}$
Output: R
R $\leftarrow \emptyset$;
$\mathcal{O} \leftarrow$ LoadOntologyFromOWL();
$\hat{F} \leftarrow \{t_i | t_i \in F \wedge \exists D_j \ s.t. D_j \in \ t_i \wedge D_j \in Q\}$ ;
t $\leftarrow \emptyset$;
while  row in $\hat{F}$ do
    if document changed then
        R $\leftarrow$ R $\cup$ BuildCube(t,$\mathcal{O}$,T,R,$\{A_1,\dots\}$);
    end
    t $\leftarrow$ t $\cup$ $\{D_j\}$; /* $D_j \in row$.                    */
end
BuildCube(t,$\mathcal{O}$,T,R,$\{A_1,\dots\}$);/* Adds last doc.   */
```

a post-processing phase can be executed to compute all the missing combinations.

The algorithm for obtaining the CUBO given an ontology $\mathcal{O}$, fact table F, dimensions Q and a maximum ontology depth T is shown in Algorithm 1. The result is temporarily stored in R that contains a level, combination of classes (combo) and the aggregation measurements. Furthermore, the entire computation of all the data cubes is computed through a single scan over the filtered data set. CUBO is composed of three major steps: load ontology, computation of on-demand combinations and the data cube aggregation. The final step of our algorithm stores the resulting data cubes into a relational table. **Example 2:** for an ontology with $h = 3$, $k_3 = 2$, $k_2 = 1$, and given that all the dimensions share parent $H_{(1,1)}$. The resulting CUBO is $\{\{\{D_1, D_2\}, \{D_1\}, \{D_2\}\}, \{\{H_{(1,1)}\}\}, \{\{H_0\}\}\}$.

3.1 Load Ontology

Loading the ontology in main memory is the initial step for our algorithm. The format of the ontology is language-specific (in our case we use OWL). However, our algorithm builds a balanced internal tree-like representation of the given ontology stored as a set of linked lists (every node has a parent pointer). This in-memory loading is important for avoiding joins with dimension tables and allows a fast hierarchy traversal. Due to the fact that we are only interested in capturing class and relationships, all the properties and types of relationships are ignored in this process. Ontologies that are equivalent to flat or balanced hierarchies are easy to capture and traverse. By counting the number of parents, it is possible to know the level. However, an unbalanced ontology must be captured in a tree representation that preserves the level for each node in the tree. Otherwise, the algorithm will return incorrect results by pairing dimensions that are not within the same level. In order to avoid this problem, dummy nodes must be included in the tree data structure, and must be ignored during the combination computation and aggregation. For example, if a dimension D_1 does not have a parent in the immediate superior level $h-2$, in which dimension D_2 has parent $H_{(h-2,1)}$, a dummy parent node must be added for D_1 in order to allow our algorithm to traverse the tree all the way to the root. As a result, if the user desires to support unbalanced ontologies, the given ontology should have a way to provide a property that specifies the level of a dimension in the tree. In this algorithm, hierarchies with multiple parents are not covered, but it is possible to extend our algorithm to handle such cases. Extensions to our algorithm also include adding a new indirection level in the $h-1$ level of the R structure to manage the summarization of instances.

3.2 On-demand Combinations

The lazy-policy of only working with those combinations that are present in the data set is critical for efficient performance in a space data set. Once the ontology has been loaded into main memory, the fact table F is filtered to only focus on those dimensions that are in Q. A single scan through the data set is performed to read all the entries containing a document t_i and a dimension D_j. In order to process a document t_i, all the dimensions must be collected before proceeding. With all the dimensions of a document collected, our algorithm computes the combinations possible with this set of dimensions and stores them in a list for accumulating all the desired measurements. The incoming attributes are always sorted in order to guarantee that the combination will be unique when storing the resulting combination in R. The sorting, as well as the combination computations, are relatively inexpensive operations due to the limited amount of dimensions to consider per document. However, if the data set is dense, these tasks will represent the bottleneck of the algorithm.

3.3 Data Cube Aggregation

The data cube aggregation will cover storing every combo in the corresponding data cube (level) and traversing the loaded hierarchy all the way to the root H_0. This function is covered in the `BuildCube` function shown in Algorithm 2.

The `BuildCube` algorithm takes the result of the on-demand combinations of each document and stores the result in main memory. Then, it will traverse for every dimension D_j in t_i the ontology tree. The corresponding combinations are accumulated in R, in which if a corresponding combo does not exist, a new entry will be created. Otherwise, the corresponding measurements are aggregated. For each D_j that exists in each document, all the superior H are extracted, and their combinations are computed and stored in $s_{0,\dots,h-2}$. Finally, the corresponding value is accumulated for each existing combination in every level. The `CombosForOntologyLevel` function is a recursive function that will stop after a certain number of levels T has been reached, or the parent is null. Due to the fact that we always consider balanced trees, it is possible to perform this operation in a simple manner. Support for dummy nodes should be considered in this section of the algorithm. Once the resulting data cube R is computed, R can be stored as relational tables inside a database management system.

3.4 Complexity

The complexity of our algorithm is given by the time it takes to compute all the combinations of the dimensions for each document $n2^k$. Moreover, because our ontology representation is always the result of balanced hierarchies (remember the introduction of dummy nodes), the time it takes to traverse an entire branch of the tree for each dimension requires h steps. Thus, the total complexity of our algorithm is $O(n2^{kh})$. Despite the time complexity of our algorithm being larger than the traditional data cube computation that has a complexity of $hn2^k$ (because there is a

Algorithm 2: BuildCube

```
Input: $t$,$\mathcal{O}$,$T$,$R$,$\{A_1,\dots\}$
Output: R
$s_h \leftarrow$ Combos();
/* Aggregate all the existing combos of the h-1
   level.                                        */
foreach combo do
    R ← R ∪ $\{1, combo, \{A_1,\dots\}\}$;
end
/* Recursive function to extract all unique
   concepts by level h-2 to 0.                   */
$s_{0,\dots,h-2} \leftarrow$ CombosForOntologyLevel(s,$\mathcal{O}$,T);
/* Increments found combos by level.             */
foreach $l$ in $s_{0,\dots,h-2}$ do
    foreach combo in $s_l$ do
        R ← R ∪ $\{l, combo, \{A_1,\dots\}\}$;
    end
end
return R;
```

need to compute a data cube for every level), in practice, the average size of k per document is small (around 1 or 2) and h is almost always small (less than 5). This results in a faster performance because there is no need to concentrate on computing dimensions that are not within the data set.

The space complexity of our algorithm is limited only by the number of dimensions per level in h. If the data set is dense, the space required by our algorithm is given by $\sum_{l=0}^{h} 2^{k_l}$, where k_l is the number of dimensions per level. Therefore, in the worst case scenario, the space complexity of our algorithm is of the order $O(2^k)$.

3.5 Integration with a DBMS

We assume that the fact table F and the filtered table are cluster indexed by i. This is an important assumption for guaranteeing that all the dimensions of a document are contiguous in one block. Extending a relational database management system to support our algorithm requires injecting our algorithm as a routine programmed in a procedural language (e.g. C or C#). In order to do so, database extensibility mechanisms (e.g stored procedures or table-valued functions) can be used to achieve this goal. The main advantage of using an extensibility mechanism is that it is possible to maintain the ontology, as well as the CUBO structure, in main memory. They provide a framework to hold data structures in main memory while scanning a data set in a cursor fashion. Unlike other types of user-defined functions, such as user-defined aggregates (UDAs), a stored procedure or table-valued function that requires processing every row will not offer the parallelism that is native to UDAs by default. In Figure 5, we show a stored procedure call to execute the CUBO algorithm in a DBMS.

```
EXEC CUBO 'D:\ontology.owl', 'dataset','D1,D2,D3'
```

Figure 5: Stored Procedure SQL Call.

The extensibility mechanism contains a SELECT statement that filters the fact table F using a WHERE/IN clause to consider only those dimensions in Q. This will be the only pass through the data set. Notice that this query is represented as two steps in the algorithm. The backbone data structure R that contains all the aggregated attributes $A_1,\dots,A_e$ for all the levels in the hierarchy is an array list of hash tables, where each hash table contains a data cube.

Unfortunately, there are limitations associated with every relational database management system (which limits the portability of a stored procedure or user-defined function implementation). For example, the amount of data that can be maintained in main memory at a time, the ability to access and read external files (used to load the ontology), or the possibility to create and store the resulting data in a relational table.

Table 1: TPCH Corpora.

n	Avg k	Max k	Min k	Total k
1K	1	3	1	1038
10K	1	3	1	6589
100K	1	5	1	9702
1M	1	5	1	9702
10M	1	5	1	9702

Table 2: dbpedia Corpora.

n	Avg k	Max k	Min k	Total k
1K	2	9	1	156
10K	2	14	1	231
100K	2	16	1	263
1M	2	26	1	302
10M	2	46	1	308

Table 3: Performance of Traditional Cube and CUBO (* unable to compute)

d	Traditional Single Level	CUBO
2	36	5
4	36	8
8	37	9
16	*	15
32	*	44
64	*	96

4. EXPERIMENTS

To verify that our algorithm performs better than a traditional approach, as well as to scale to large datasets and dimensions, we tested our algorithm on two databases. Our experiments were run on an Intel Xeon E3110 server at 3.00 GHz with a 750 GB in hard drive and 4 GB of RAM. The server was running an instance of SQL SERVER 2005. The application was developed entirely as a stored procedure in C#, as part of an extensibility mechanism from SQL SERVER 2005.

The databases that we used for testing our application included an ontology in OWL format and a real and a synthetic data set stored in a relational table F. The synthetic data set is a materialized view from the TPCH data set. The view is the result of $Lineitem \bowtie Part$, where each row represents a dimension in a document (in fact, a document

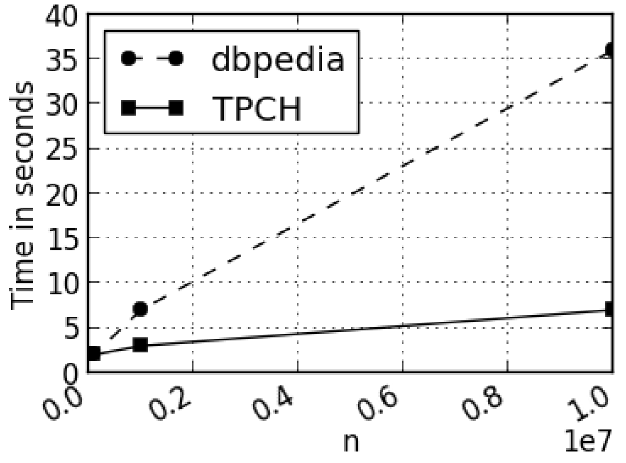

Figure 6: Varying Corpus Size.

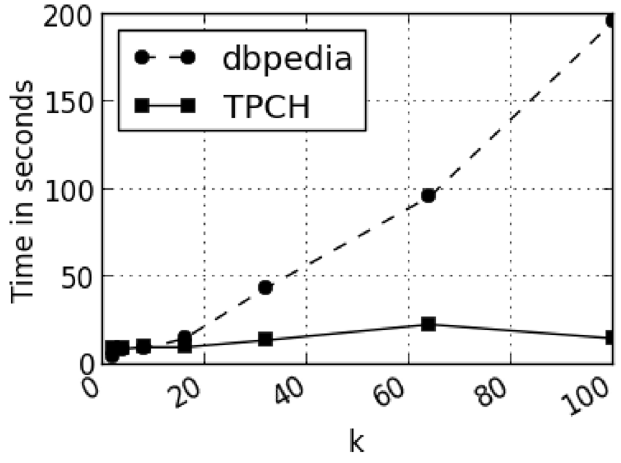

Figure 7: Varying Number of Dimensions.

can be seen as a transaction). The real data set is the dbpedia project, which contains 3 million abstracts with 308 dimensions extracted from the Wikipedia project. Every D_j in this project represents a classification topic for each document in dbpedia. Furthermore, A_1 was assigned to be one for all the D_j as our purpose is to count the number of documents per topic. This data set was preprocessed and stored in a relational table F. The preprocessing time required almost two days to extract the classes existing in the original abstracts. Both databases were given an OWL ontology of 2MB and 1MB in size, respectively. Both ontologies had a depth of 3 levels. A full description of the databases is given in Table 1 and Table 2. All the performance experiments were repeated five times and the results were averaged after removing the upper and lower quantities. The dimensions in Q were obtained randomly for every run from the pool of all possible classes in the $h-1$ level of the ontology. In addition, the default databases for testing our algorithms were the original 3M dbpedia data set and the 10M TPCH data set. All the times are in seconds unless otherwise specified.

Initially, we focused on studying the performance of our algorithm as compared to the traditional cube implementation from SQL SERVER 2005. In order to perform this experiment, the vertical dbpedia data set F had to be modified to have the classes as column names and the documents as rows. Hence, the experiments were set to use this horizontal layout for the traditional cube operator and the vertical layout was used for CUBO. In addition, the traditional cube was only performed in the lowest hierarchical level due to the lack of native support for hierarchies. Obtaining a similar result with the cube (or SQL queries) will require loading the ontology in a star schema that should be known upfront, and then running the cube operator several times for every level. The results, presented in Table 3, show that our algorithm performs an order of magnitude better than the traditional data cube operator in only the lowest level of the hierarchy (level $h-1$). Moreover, the traditional data cube operator cannot scale to a larger number of dimensions to compute. This is due to the fact that the traditional data cube is also computing those dimensions that have zeroes. We did not perform this experiment in the TPCH data set

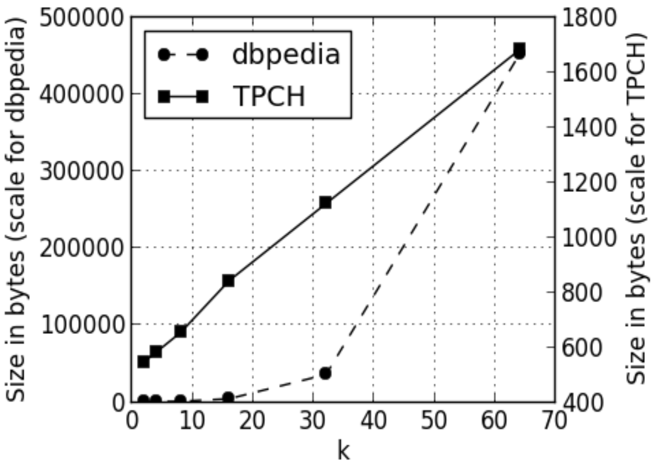

Figure 8: CUBO Size when Varying Number of Dimensions.

because this data set exceeds the maximum number of dimensions allowed in a relational table.

In Figure 6, we present scalability experiments by varying the size of the corpus in the synthetic and real databases. The size of Q was fixed to ten, and the corpora included a range of collections from 1K to 10M of documents. The experiments showed that CUBO scales linearly based on the number of documents in the collection. However, the speed of the increase is related to the average number of dimensions per document. Therefore, a data set with a smaller average of dimensions per document will be processed faster than one with a larger average of dimensions.

Additional scalability experiments for our algorithm include modifying the number of dimensions in Q. In Figure 7, we show the results for the TPCH and dbpedia databases. The results perform as expected in the complexity analysis discussed in subsection 3.4. The plot shows an exponential increase related to the average number of dimensions per document. Hence, dbpedia has a larger exponential growth than the TPCH data set. This is due to the larger average number of sparse dimensions per document in dbpedia than

TPCH. A lower number of average dimensions per document results in an avoidance to compute large combos and a decrease in the exponential growth (shown in the TPCH data set). The inflection point of the function is the result of the selectivity of dimensions in the corpora, hence, this point depends on each data set.

Experiments showing the space complexity of our algorithm were also run. Figure 8 presents the results of this experiment for both collections. Notice that the axes have different scales due to the sparsity of the databases (TPCH is more sparse). The dbpedia data set shows a clear exponential growth of the data structure size. In a similar manner, TPCH has a similar growth that is almost undetectable due to the sparsity of the data set. We also found that the maximum k that we were able to compute was around 100 dimensions and an ontology input file of around 2 MB. However, this limitation is DBMS-specific.

Table 4: Varying Ontology Levels in dbpedia (time in seconds).

n	ALL	MAX 2	MAX 1
1K	2	2	1
10K	2	3	1
100K	2	3	1
1M	7	6	5
10M	36	28	25

Table 5: Varying Ontology Levels in TPCH (time in seconds).

n	ALL	MAX 2	MAX 1
1K	2	2	2
10K	2	2	2
100K	2	2	2
1M	3	2	2
10M	7	7	7

Further experimental analysis of complexity of the algorithm is shown in Tables 4 and 5. These tables show the performance results of modifying the number of levels that are taken into consideration from both ontologies and databases. The results showed that in both databases, the overall impact of modifying the level in the hierarchy is minimal when the average k is small in all the text collections. However, there is an almost linear increase for the real data set in the largest of the collections. It is then possible to observe that our experimental results support our theoretical upper bound of $O(n2^{kh})$, in which there is an exponential growth. Due to our lazy policy of only computing those dimensions that are present, in practice, the complexity of the algorithm is followed by an almost constant performance (e.g. there are no combinations involving all the k dimensions).

Finally, a profiling analysis of our algorithm in both databases is shown in Tables 6 and 7. The experiments are consistent regardless of the data set and show that obtaining the combinations is the least expensive step, along with storing the results in the hash table. However, reading and building the ontology in main memory takes longer than

Table 6: Profiling dbpedia (time in seconds).

Task	Time	Percentage
Load Ontology	0.017	0%
On-demand Combinations	0.030	1%
Data Cube Aggregation	0.061	1%
I/O	5.891	98%
Total	6.000	100%

Table 7: Profiling TPCH (time in seconds).

Task	Time	Percentage
Load Ontology	0.068	1%
On-demand Combinations	0.001	0%
Data Cube Aggregation	0.010	0%
I/O	6.921	99%
Total	7.000	100%

building the actual CUBO. As expected, the majority of the processing is spent in access to secondary storage.

5. RELATED WORK

Using OLAP for summarizing text corpora has been presented in several works [11, 12, 18]. Early in [11], the authors presented an approach to load all the documents inside a DBMS. The documents were all loaded into a star schema that allowed the users to use traditional OLAP algorithms in this scenario. The computations were focused on keyword frequency. The main disadvantage of this early work is the requirement to load all the data in a predefined schema that will allow exploring using traditional OLAP techniques. This approach is prohibitive when we have a data set with a large number of dimensions unlike our vertical layout approach that allows storing large dimensional data sets. In [12], the authors propose Text Cube. This approach focuses on computing a partial cube by using only a partial materialization. Their algorithms are focused on obtaining "on-demand" OLAP queries with the optimal processing cost using a greedy algorithm. Unlike our approach, we focus on computing all the existing dimensions of the data cube at once by taking advantage of the text sparsity. More recently in [18], Topic Cube is proposed to obtain a more complex analysis than just data summarization. The purpose of this new OLAP model is to extract dimensions for existing text data using the probabilistic latent semantic analysis. Despite the fact that their model and results are interesting for generating a possible ontology, the authors do not to propose a new data structure for the multidimensional data cube (they compute the SQL queries as required), and focus on the quality of the built hierarchy.

In [7] and [8], we were able to adapt a traditional OLAP data cube to efficiently process a sparse vertical fact table. However, we were only computing a set of dimensions, and the scalability of the algorithm was limited to only a few dimensions. In addition, the papers were focused on the aggregation of several measurements inside text corpora to produce the most frequent cuboids for generating an unsupervised ontology. This ontology was the result of a post-processing phase on the resulting cuboids. In this paper, we assume a given ontology, and it is used to summarize a set of documents. Thus, the main contributions of this research focus on a new algorithm that integrates the hierarchy given by an ontology into a single data structure called CUBO. In addition, we formalize and study in depth, the theoretical and experimental complexity of our algorithm. Finally, our new algorithm considers a way to manage several types of hierarchies that can be present in an ontology.

6. CONCLUSIONS

In this paper, we proposed a novel approach, CUBO, for summarizing text corpora based on a given ontology using OLAP data cubes. CUBO represents an efficient, compact, and scalable data structure and algorithm that generates a data cube for every level of an ontology. Unlike traditional OLAP approaches, CUBO takes advantage of the sparsity of text corpora and generates the combinations for only those dimensions that are present in each document.

All the data cubes are stored in an array with all the levels of a given ontology for efficiency purposes. Every level in the ontology that is analyzed is stored in a hash table and computed for only those existing dimensions in each document. CUBO was adapted into a relational database system using extensibility mechanisms, such as store procedures. We tested our algorithm in real and synthetic databases, and showed that our algorithm has linear scalability when the number of documents increases in the collection. In addition, we showed that unlike traditional algorithms, CUBO is able to scale to a larger number of dimensions due to the avoidance of computing and storing unnecessary combinations. Moreover, we showed in our experiments that the running time of our algorithm in real and synthetic databases is significantly less than the expected exponential upper bound. This was also appreciated when we analyzed the overhead of computing the lattice in higher levels of the hierarchy, which resulted in only a small difference (almost negligible). Hence, the complexity of the algorithm is still highly dependent on the average number of dimensions in a document. Further analysis of our algorithm in both databases showed that the processing time for building and storing the CUBO is mostly spent on the latter. As a result, CUBO showed to be an efficient and scalable algorithm for summarizing text data with hierarchies.

Future work on this topic includes testing the performance of our algorithm when a new level of indirection is added for capturing instances under the $h-1$ level. Also, we would like to test our algorithm with larger ontologies that do not fit in main memory and have to be loaded in a relational table, as well as more complex real ontologies that have a large depth. Finally, we would like to extend our algorithm to cover more complex ontologies, that include unbalanced hierarchies and multiple parents.

Acknowledgments

This work was supported by NSF grant IIS 0914861.

7. REFERENCES

[1] G. Antoniou and F.v. Harmelen. Web ontology language: OWL. In Peter Bernus, Jacek Blazewics, Günter Schmidt, Michael Shaw, Steffen Staab, and Rudi Studer, editors, *Handbook on Ontologies*, International Handbooks on Information Systems, pages 91–110. Springer Berlin Heidelberg, 2009.

[2] L. Bellatreche and S. Khouri. A methodology and tool for conceptual designing a data warehouse from ontology-based sources. In *Proc. ACM DOLAP Workshop*, 2010.

[3] C. Chakroun, L. Bellatreche, and Y.A. Ameur. Ontological concept dependencies driven approach to design ontology-based databases. *LISI/ENSMA*, 3, 2011.

[4] Z. Chen and C. Ordonez. Efficient OLAP with UDFs. In *Proc. ACM DOLAP Workshop*, pages 41–48, 2008.

[5] R. Feldman and J. Sanger. *The text mining handbook: advanced approaches in analyzing unstructured data*. Cambridge Univ. Press, 2007.

[6] F.T. Fonseca, M.J. Egenhofer, P. Agouris, and G. Câmara. Using ontologies for integrated geographic information systems. *Transactions in GIS*, 6(3):231–257, 2002.

[7] C. Garcia-Alvarado, Z. Chen, and C. Ordonez. OLAP-based query recommendation. In *Proc. of ACM CIKM*, pages 1353–1356, 2010.

[8] C. Garcia-Alvarado, Z. Chen, and C. Ordonez. ONTOCUBE: efficient ontology extraction using OLAP cubes. In *Proc. of ACM CIKM Conference*, pages 2429–2432, 2011.

[9] 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 Conference*, pages 152–159, 1996.

[10] I. Horrocks. DAML+ OIL: a description logic for the semantic web. *Bulletin of the Technical Committee on*, 51(4), 2002.

[11] J. Lee, D. Grossman, O. Frieder, and M.C. McCabe. Integrating structured data and text: A multi-dimensional approach. In *Proc. of IEEE International Conference on Information Technology: Coding and Computing*, pages 264–269, 2000.

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

[13] P. Martin and P. Eklund. Embedding knowledge in web documents. *Computer Networks*, 31(11):1403–1419, 1999.

[14] P. Mika. Ontologies are us: A unified model of social networks and semantics. In *Proc. of ISWC*, pages 522–536, 2005.

[15] T. Priebe and G. Pernul. Ontology-based integration of OLAP and information retrieval. In *Proc. of IEEE DEXA*, pages 610–614, 2003.

[16] F. Ravat, O. Teste, and R.Tournier. OLAP aggregation function for textual data warehouse. In *Proc. of ICEIS)*, pages 151–156, 2007.

[17] U. Shah, T. Finin, A. Joshi, R.S. Cost, and J. Matfield. Information retrieval on the semantic web. In *Proc. of ACM CIKM Conference*, pages 461–468, 2002.

[18] D. Zhang, C. Zhai, and J. Han. Topic cube: Topic modeling for olap on multidimensional text databases. In *Proc. of SIAM SDM Conference*, 2009.