2010 IEEE International Conference on Data Mining Workshops

Enhancing Document Exploration with OLAP

Zhibo Chen
University of Houston
Houston, TX, USA

Carlos Garcia-Alvarado
University of Houston
Houston, TX, USA

Carlos Ordonez
University of Houston
Houston, TX, USA

***Abstract*—Finding relevant documents in digital libraries has been a well studied problem in information retrieval. It is not uncommon to see users browsing digital collections without having a clear idea of the keyword search that they should perform. However, we believe that such initial query search is not totally independent from the target search. Therefore, we use these initial document selections to further explore these documents. In the following demonstration, we exploit On-line Analytical Processing (OLAP) for knowledge discovery in digital collections to achieve query refinement. Such refinement is the result of applying a traditional ranking technique, based on the vector space model, selecting the top keywords in the resulting subset of documents, and then displaying certain cuboids of the keywords. Based on these cuboids, which are ranked by their frequency, the users can select a query that can better represent their actual target search. We show that this document exploration can be done efficiently within the DBMS and exploit in-database extensions, such as User-Defined Functions, as well as standard SQL. Additionally, we demonstrate a novel approach to obtaining query refinement through OLAP data cubes.**

***Keywords*-OLAP;Information Retrieval;UDF;**

I. Introduction

Information retrieval is a vast field of research whose main task is centered on returning relevant documents to users and has an implicit knowledge discovery task in documents and text. One of the pillars of this field involves the retrieval of documents containing user-defined keywords. Such area generally includes the exploration of stored documents and text to resolve a user-defined query. These filtered documents can also be ranked according to a variety of scoring functions such as the vector space model (VSM). There also exists a variety of features, such as phrase completion and suggestions, that assist the users in selecting the correct set of keywords to best narrow down the documents returned.

The belief is that users will often start their searches with naive queries, which implies that the selection of keywords may not be entirely related to the target search. Hence the result can often return an overwhelming amount of hits and a low number of relevant documents. Subsequent searches should narrowed down the search space to be in more manageable levels for the users to browse. There exists a variety of features that have been developed to assist the users in creating more specific queries. Features such as automatic completion and keyword suggestions can often help users who are unsure of the exact query to run. In this paper, we propose the development of a feature that can provide users with additional possible queries by retrieving popular keywords from the current search space. In other words, once users have queried a set of documents, we will provide alternative queries based on information extracted from the reduced document space.

To provide such features, we exploit On-Line Analytical Processing (OLAP) to build data cubes of possible keywords. OLAP is a set of exploratory database techniques that allow the user to efficiently retrieve specific aggregations [3]. OLAP users often try to find interesting results by analyzing subsets of dimensions, which can be anything from medical factors to document keywords. The underlying structure of OLAP is the dimension lattice. While OLAP is generally quite slow when processed within the DBMS, optimizations such as cube precomputation and User-Defined Function (UDF) can help greatly improve performance. For this paper, we apply the exploration power of OLAP to documents in order to retrieve the most popular keywords, which to the best of our knowledge is one of the first attempts to use OLAP as a backbone for query refinement.

The main motivation for our research is to assist users in producing queries that are more likely to return what they are really searching for. We believe that the correlation of an initial query search, can be used to produce a list of alternative popular keywords in the selected documents. Such set of keywords can greatly help the users to define more specific or more efficient search queries. The novelty of our research lies in our utilization of the benefits of OLAP to generate such a set of keywords. The various aggregation levels of OLAP also allows us to return alternative queries that consists of different number of keywords.

In this paper, Section II describes, in detail, the techniques that were employed while Section III provides a description of the demonstration that we would present.

II. Technical Detail

In this section, we will first explain how the documents are searched and ranked within a DBMS. Next, we show how OLAP queries can be used to generate popular keywords. Finally, we provide details on how the two areas are

This research work was supported by NSF grants CCF 0937562 and IIS 0914861.

978-0-7695-4257-7/10 $26.00 © 2010 IEEE
DOI 10.1109/ICDMW.2010.37

integrated into one system.

A. Document Ranking

Let C be a collection of N documents $\{d_1, d_2, ..., d_n\}$. Each document d_i or query q is composed of words to which we apply the Porter Algorithm to obtain a reduced set of stemmed-words, which we are going to identify as t. The Vector Space Model (VSM) computes the similarity of a function based on the cosine of the angle of the query vector and each document ($Sim(q, d_i) = \frac{\vec{q} \cdot \vec{d_i}}{\|\vec{q}\| \|\vec{d_i}\|}$). During the scoring phase, the keywords are stored in a large frequency table, by relating the document id, the keyword, and the frequency count for each of them. Then we compute the weight of a keyword in the collection w_t given by $log(N/cf(t))$ and store them in an idf table. Finally, we compute the norm of each document. For the ranking phase, we have to compute the document frequency of a keyword in a document $f(t, f_i)$ using summarization such as the number of documents containing a keyword in the collection $df(t)$, the total count of keywords in a document $|d|$, the frequency of a keyword in a document $tf(t)$, the frequency of a keyword in the collection $cf(t)$, and the norm of a relation (vector keyword weights) $||d_i||$. The ranking of a document is represented by a similarity function $Sim(q, d_i)$.

B. OLAP Data Cube

We will now provide a more detailed analysis of how OLAP techniques can be performed using UDFs as well as how they can be altered to work with documents. The goal of this portion of our research is to perform OLAP on top of a naive database management system, without the need to alter any of the source code [2]. For the remainder of this section, we will consider a typical OLAP dataset as having n records, d cube dimensions, and e measure attributes. To generate an OLAP data cube, we must have at least one record and one dimension while there does not necessarily need a measure. A dimensional lattice is the structure that represents all the subsets of dimensions and is composed of nodes, which represent a combination of dimensions.

For OLAP, the queries generally involve either a retrieval of information or an aggregation of dimensions. Operations such as slicing, dicing, and pivots [6] can be considered retrievals because they normally involve gathering data from computed portions of the lattice. On the other hand, aggregations include operations such as drill down and roll up, which requires the computation of additional nodes in the lattice. These operations are used to traverse the large dimensional lattice in preparation for the retrieval queries. For small dimensionality, it is often recommended to compute the entire dimensional lattice at once and then only use retrieval operations to obtain information.

For our research, we use OLAP to generate a keyword data cube in which each node of the lattice would represent one specific combination of keywords. The data cube would store both the number of occurrences of the combination in the document space as well as the total number of documents in which the set of keywords appears. For example, suppose the node "Water Pollutant" stores the values 100 and 10, then we know that in our filtered document space, this combination of keywords appears a total of 100 times in 10 different documents. The first stored value represents the minimum frequency of the two keywords. For example, if "Water" appears 5 times in one document while "Pollutant" appears only 3 times in the same document, then we consider the pair "Water Pollutant" as having a frequency of 3 in this document. In general, the rule is: $\min q_1, q_2, ..., q_j$ where q_j represents the frequency of the keyword j in a particular document. The second store value represents the total number of documents that contains the keywords. Thus, the p-th level of the lattice would contain all possible combinations of p keywords. For keywords, the dimensionality is often too high to generate the entire lattice at once. At the same time, presenting all the possible combinations of keywords would give the user too much information to view at once. As a result, we perform a user-driven traversal of the dimensional lattice. In this process, we first provide the user with the set of all pairs of keywords, level 2 of the dimensional lattice, found within the document space. From here, the user can activate further calculations by drilling down the lattice. For example, suppose the user wants to drill into the keywords "New York", then our application would calculate only those nodes involving these two keywords at the next level to form sets of three keywords, such as "New York State". In this manner, we avoid computing large portions of the lattice at once while still providing the user with the flexibility to drill as far into the lattice as needed.

Currently, we have three main ways in which we can execute the OLAP queries. The pure SQL approach involves the use of standard SQL to generate the required nodes of the lattice. In general, this approach requires one aggregation per node of the lattice. For the retrieval operations, the SQL requires a straight Select statement with the needed dimensions while for aggregation operations, a Group-By statement is required to perform the aggregation. In the case that multiple nodes of a lattice needs to be computed, then one aggregation would be performed for each node. Further details on the pure SQL approach can be found in [5].

1) UDF Methods: The UDF approach involves pushing as much of the calculations into main memory as possible. For this method, the main backbone is the storage of a data structure that needs to represent the dimensional lattice and is held in main memory for fast updates. Currently, we have implemented two different data structures for use with the UDF approach. The first is a lattice-based structure [1] that has been altered to work with keywords. In the original design, a two-tiered system was used to store both node information and specific group data. For keywords, the second tier is not required since each node contains only one

possible value. As a result, we reduce the structure to only contain one-tier, each of which represents a specific node within the dimensional lattice. We also added additional links to provide quick navigation between the various sets of keywords. These links not only point from one level of the lattice to another, but also within. For example, the node "New York" would not only connect to next level nodes such as "New York City", but also same level nodes containing one of the keywords, such as "New Jersey". The power of this structure is that only one pass of the dataset is required to produce the required nodes. This is accomplished by updating all appropriate nodes for each record.

In the past, the only common UDF types were scalar and aggregation, both of which only requires a single value. For this paper, we used multi-statement table-valued functions, which allow for the return of result sets. This is extremely useful because it allows us to avoid additional intermediate steps involving the transformation of the data structures into a single string. With the TVF, we can read entire datasets, create and update the data structure, and return entire tables.

2) OLAP Network: We also created a graphical representation of the OLAP data cube in the form of an OLAP network, similar to [4]. In its basic form, each vertex represents a specific keyword while each edge connect two vertices that form a pair. The weight or thickness of the edges depends on the strength of the link between the two keywords. In our case, it can be changed to represent either occurrences or documents. For example, an edge connecting "water" to "arsenic" with a value of 10 can be interpreted as ten documents within the document space contains both of the keywords. For the more complex cases of more than two keywords in a set, then additional vertices are required to distinguish the sets. In such cases, the number of vertices depends on the number of keywords and the level of the lattice that we are observing. We also provided various filtering methods by which the user can reduce the number of edges. These filters include hiding certain vertices, applying a threshold to the smallest number of occurrences or documents to show, and also changing the thickness of the edges by altering the thickness factor.

C. Integration of OLAP with Searching

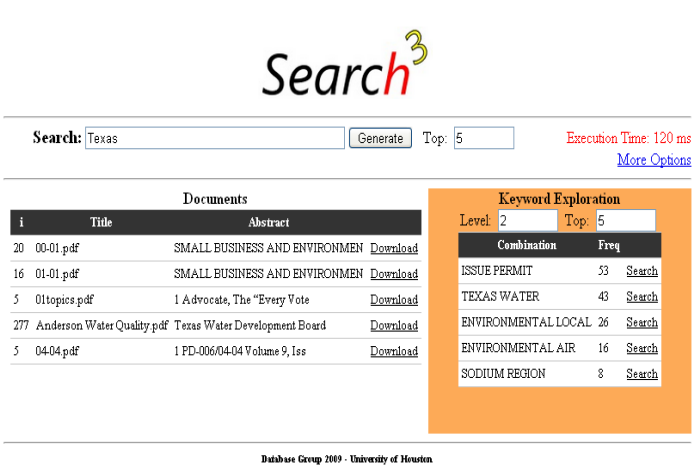
Let us now observe how OLAP queries can be integrated with document searching to provide users with additional information. Within the DBMS, the documents and keywords are stored in a transactional format, table C, with each record containing the document id, the keyword, and the number of occurrences. This large table C is reduced in size through the use of the vector space model to only keep the documents that are similar to the search keyword(s). Once the document space is reduced,a list of the k most popular single keywords can be obtained. This value is chosen by the user with a default of 30 keywords. This can be implemented by grouping on all the keywords in the reduced document table, C_1 and retrieving only the top k keywords.

Once the keyword list has been decided, we use a UDF to simultaneously retrieve the frequency of these keywords in each document and pivot the entire table. The input table for the UDF, C_2, is a further reduced version of C_1 because of the removal of all keywords not in the top k. For this UDF, only one pass of C_2 is needed and the resulting fact table, F, contains k+1 columns. Only one pass is required because we can keep track of each document and the values of the keywords as we are progressing through the document table. Once F is produced, we can run OLAP on this table to initially generate the 2nd level of the lattice. Should the user wish to drill down, the same F is used to create those new nodes. In fact, the fact table only needs to be regenerated once the user chooses to perform another keyword search.

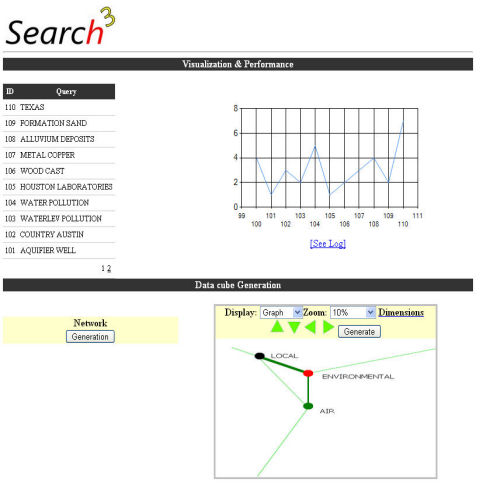
Once OLAP has generated the required lattice for a particular set of documents, the manner in which the results are displayed according to a sorting criteria. There are two measures by which the keyword sets can be ranked: total frequency and total documents. The total frequency measure will rank the keyword sets based on the number of times they appear in all the documents, which the total documents measure will sort based on those results that appear in the most documents. Even though this demonstration only incorporates these two sorting criteria, more complex metrics can be used. The additional data required by these more complex methods can efficiently stored and retrieved through the use of OLAP. For example, additional metrics may take into account the size of the document, which can then be used to normalize the appearance of a keyword set in a single document. An average can then be applied on all documents to obtain the average normalization. In this case, we would need to store the size of the documents in the lattice

III. System Demonstration

In this section, we will present a demonstration of our application (see Figure 1). Our goal for this demonstration is to show the audience that using OLAP techniques on the documents' keywords will allow the user to efficiently refine searches by displaying information that cannot be obtained from a straight search through the dataset. We developed a Web application using ASP .NET and User-Defined Functions, where the user is capable of refining their searches based on the top-k sets of keywords. The digital collection that we will use during this demonstration is a Water Pollution Dataset from the Texas Commission on Environmental Quality, which was pre-processed to have around 10,000 documents. We do not require Internet access to present our demonstration. Our demonstration will be separated into two parts: (1) the enhanced searching process provided by OLAP and (2) a graphical way to explore the keywords within the document space.



Cube Exploration



Lattice Generation & Options

Figure 1. Document Exploration Tool

In the first section, we will demonstrate how our application is able to generate the alternative keywords at runtime by searching for various keywords and observing the execution time. For example, we will search for "water" and observe the speed at which the application will reduce the document space and provide the OLAP results. We will show how a search for "water" will bring too many hits for the user to analyze, but the OLAP suggestions can help narrow the documents. We can also show how the user can sort the OLAP suggestions by number of occurrences or number of documents. The appropriate sort to use would depend on the goal of the user. For now, the suggestions show that "water arsenic" is one of the top keyword pairs. We then initiate a search for "water arsenic" and show how the number of documents returned has been reduced. Because we have performed a new keyword search, the OLAP suggestions have also been recomputed to show the alternative pairs of keywords based on the new document space.

For the second portion of the demonstration, we will ignore the keyword searching and only focus on how a user can navigate through the sets of keywords of the document space. For example, we will again search for "water" and receive the pairs of keywords as sorted by number of occurrences. We can then activate our OLAP network to graphically view these pairs of keywords. Once again, the vertices represent each keyword and the edge between each pair of vertex represents the weight of the connection. This weight can represent either the occurrences or documents, depending on user selection. In addition, we will also show how one can drill down or roll up based on a chosen set of keywords. For example, we can choose the "water" and "arsenic" nodes and have the application drill down on these two nodes. The result is a network showing all the keywords that form triples with these two nodes as well as the appropriate edges. The power of this graphical option is that it allows the user to quickly navigate among the keyword sets to gather more knowledge regarding the appropriate type of keyword search to execute.

For future work, we want to improve the data structure that we are currently using to store the keyword information in the DBMS. We believe that we can improve efficiency by using even smaller data structures. Also, we want to add context to both the keyword search and the OLAP suggestions. Finally, we want to re-use and update materialized cuboids for query refinement by not having to recompute all the cuboids every time a new keyword search is performed.

ACKNOWLEDGMENT

We thank Mario Navas for providing invaluable assistance in the implementation of our application.

REFERENCES

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

[2] C. Garcia-Alvarado, Z. Chen, and C. Ordonez. OLAP-based query recommendation. In *ACM CIKM*, 2010.

[3] 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.

[4] K. Morfonios and G. Koutrika. OLAP cubes for social searches: Standing on the shoulders of giants. WebDB, 2008.

[5] C. Ordonez and Z. Chen. Evaluating statistical tests on OLAP cubes to compare degree of disease. *IEEE Transactions on Information Technology in Biomedicine*, 13(5):756–765, 2009.

[6] C. Ordonez and Z. Chen. Horizontal aggregations in SQL to prepare data sets for data mining analysis. *IEEE Transactions on Knowledge and Data Engineering*, 2011.