ONTOCUBE: Efficient Ontology Extraction using OLAP Cubes

Carlos Garcia-Alvarado
University of Houston
Dept. of Computer Science
Houston, TX 77204, USA

Zhibo Chen
University of Houston
Dept. of Computer Science
Houston, TX 77204, USA

Carlos Ordonez
University of Houston
Dept. of Computer Science
Houston, TX 77204, USA

ABSTRACT

Ontologies are knowledge conceptualizations of a particular domain and are commonly represented with hierarchies. While final ontologies appear deceivingly simple on paper, building ontologies represents a time-consuming task that is normally performed by natural language processing techniques or schema matching. On the other hand, OLAP cubes are most commonly used during decision-making processes via the analysis of data summarizations. In this paper, we present a novel approach based on using OLAP cubes for ontology extraction. The resulting ontology is obtained through an analytical process of the summarized frequencies of keywords within a corpus. The solution was implemented within a relational database system (DBMS). In our experiments, we show how all the proposed discrimination measures (frequency, correlation, lift) affect the resulting classes. We also show a sample ontology result and the accuracy of finding true classes. Finally, we show the performance breakdown of our algorithm.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database applications—*Data mining*

General Terms

Algorithms, Experimentation

Keywords

OLAP, Ontologies, DBMS

1. INTRODUCTION

Ontologies model a view of a continuously evolving world. However, this formal specification of a shared conceptualization, made into a machine readable format, is dependent on a specific context and point of view [1]. Due to this specialization, it is required that multiple ontologies be created for every world's view, giving as a result a huge task that normally undergoes human supervision, creating a knowledge acquisition bottleneck. In this paper, we focus on the problem of extracting the main concepts of a set of documents stored in a DBMS and then using them to build an ontology. The problem of extracting concepts is not trivial and has been tackled through natural language processing, clustering, singular value decomposition (SVD), and statistics, among many other proposed solutions [1]. The main problem with these approaches is that they are time-consuming or require previous knowledge to be provided. Moreover, it has been shown that these proposals are not the ultimate answer for ontology extraction.

Online Analytical Processing (OLAP) can be used to efficiently obtain these frequency summarizations' and keywords' correlation in order to extract and organize the most representative concepts in a corpus. The main characteristic of this approach is that the ontology can be built without the need of a given pattern or a set of rules in an automated form. However, OLAP has never been exploited to build ontologies. The ontology construction process can be summarized as the extraction of concepts (classes) and the relations between them based on an analysis of the keywords of a corpus. The documents within the corpus are initially preprocessed, transformed and analyzed, and finally organized to build an ontology. The result of the analysis phase is a set of concepts built from a group of keywords. The analysis phase determines the concepts and properties that characterize each concept. In a similar manner, basic relations, such as "has a" or "is a" can be inferred with the analysis of the preprocessed data set. The final phase builds the ontology by organizing the extracted relations and classes. In this research, we present how all these steps can be included within the OLAP process in order to generate efficiently, in an automated fashion, an ontology from a given set of documents stored in a DBMS. This paper is organized as follows: Section 2 presents the notation of the main concepts of ontologies and OLAP. Section 3 shows our OLAP process for building ontologies. In Section 4, the quality and performance results are presented. Similar works are discussed in Section 5. Finally, in Section 6, we make our final remarks.

2. PRELIMINARIES

Let us focus on defining the notation that will be used throughout this paper. Let C be a collection, or corpus, of n documents $\{d_1, d_2, \ldots, d_n\}$, where each document d_i is composed of a set of keywords t_{ij}. In addition, let x_i be a frequency vector of all the different keywords in C for each document d_i. We assume that C is stored in a vertical list format. This list is stored in a vtf table that contains the document id, keyword, and position in the i^{th} document. A summarization table tf is computed from vtf as a table with the document id i, the keyword t, and the term frequency f stored. The backbone data structure for OLAP data cubes is the dimensional lattice, which has a size of $2^{\hat{t}-1}$, where $\hat{t}$ is the number of selected keywords. One level (or depth in the lattice) of the cube is denoted by $\binom{|\hat{t}|}{level}$, s.t. $level \leq |\hat{t}|$.

2.1 Ontologies

Regardless of the ontology language, all these knowledge representations rely on the specification of the main concepts in the given context. 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 (instances) that share common characteristics. This class classification is obtained through a taxonomy (hierarchy). In the case of individuals, the relationship between them is given by the specification of properties, which can be represented by an object (e.g. owl:ObjectProperty) and by a datatype (e.g. owl:DatatypeProperty). In this work, we do not focus on the language, but on the extraction of classes and relations.

2.2 OLAP Cubes

While normal applications of OLAP include business reports and financial analysis [2], we believe that the OLAP dimensional lattice can be used to efficiently obtain classes and their relations. In this case, the data is represented by the collection of keywords and documents while the level of aggregation allows us to obtain various combinations of these keywords to generate concepts.

While traditional OLAP accepts inputs that are horizontal, in our system, vtf has a vertical format, with each row containing a single keyword. This presents a challenge to aggregations because we are unable to use the normal techniques, such as slicing, with the vertical format. Despite this, we are only analyzing a small subset (the most frequent keywords), $\widehat{t}$, of all keywords in the collection.

We developed an algorithm that can obtain the necessary aggregations and perform the required auxiliary computations with the associated keyword frequency (correlations and lift) directly from the vertical format. In this case, a data cube is computed from the list of keywords in order to form sets of words. OLAP is the perfect tool because it is able to use data cubes that allow for quick access.

2.3 Correlation

We also utilize correlation, which will represent how related the frequency of a keyword is with another in terms of the whole collection. In addition to the previous definitions, let L be an additional set containing the total sum of all the different terms in the collection ($L = \sum_{i=1}^{n} x_i$). Moreover, let Q be a lower triangular matrix containing the squared measures between the terms $Q = \sum_{i=1}^{n} x_i x_i^T$ (see [7]). In the last step, the correlation of a pair of keywords a and b is obtained by computing equation $\rho_{ab} = \frac{nQ_{ab} - L_a L_b}{\sqrt{nQ_{aa} - L_a^2}\sqrt{nQ_{bb} - L_b^2}}$. An additional measure to evaluate a pair of keywords is obtained through lift, λ. λ represents how often these two keywords appear in the collection. This value is computed as $\lambda = \frac{n_{a,b}}{Max(n_a, n_b)}$, where n_a and n_b are the number of documents containing keywords a and b, respectively.

```
Input: $level$ , $\hat{t}$
Set of Classes $S$
Init $List_{tf}$
foreach<i,$\widehat{t}$,p> $r$ in $vtf$ sorted by $i$
    if $i$ changed
        $C \leftarrow$ GetNextCombo($List_{tf}$, $level$)
        $DoWhile$ ($C$!=null)
            $S_C$.freq++
            $S_C$.pos+=$r.p$
            $C \leftarrow$ GetNextCombo($List_{tf}$, $level$)
        $endwhile$
    else
        $List_{tf} \leftarrow \{r.t, r.p\}$
    $endif$
$endforeach$
return $S$
```

Figure 1: Ontocube.

3. ONTOCUBE

In this section, we detail the ontology extraction process and every optimization performed to obtain classes and build the ontology.

3.1 Concept extraction

A one-time preprocessing step is required for setting the environment for obtaining the frequency of the keywords. The data set was preprocessed by removing the stopwords and assigning a position for every keyword in every document. Our proposed process for extracting concepts is the result of analyzing the most frequent keywords and generating combinations with these keywords. The frequencies of these combinations are also computed, as well as the correlation and lift of these pairs of keywords. The classes are obtained from the combinations by answering these questions: (1) Do the keywords appear in the same documents frequently? (2) Are the set of keywords close to each other? (3) How often do they appear together within the corpus? (4) How often is one keyword found in a document but not the other? The first question addresses the problem of finding keywords that appear in the same documents. In addition to this, we are required to know if they appear in the same proportion within the collection. As such, the correlation of each combination is computed. In the second question, we are interested in finding how close the keywords appear to each other. This distance between keywords is obtained as the difference between the positions of the keywords. The last two questions try to identify those sets of keywords that commonly appear together throughout the collection, or those that do not. The rationale behind this is that the core concepts appear consistently together throughout the collection. On the other hand, there are concepts that may appear strong when present together, but are often found in separate areas. Both of these questions can be answered by observing the lift of the concept.

Efficient summarization is performed in one-pass by creating the keyword combinations "on-the-fly" during the OLAP cube aggregation. The final result for this step is a set of classes from which the ontology will be extracted. The algorithm for obtaining such classes is as follows: (1) obtain all available classes by pairing all keywords within each document, (2) for each class, determine the frequency of occurrence and average position gap within the corpus, (3) filter out all classes whose keywords have a position gap greater than a user-defined threshold, $pDiff$, and (4) filter out all classes whose frequency, correlation, or lift does not meet user-defined thresholds.

The pseudocode for the first two steps is shown in Figure 1. The remaining two steps can be accomplished by traversing S and removing those classes that do not meet the user-defined thresholds. We are only interested in the classes formed by keywords within a specific document. As a result, we found it is easiest to first gather all the keywords within a document into $List_{vtf}$ before processing them once the document id, i, changed. Notice that this algorithm would not generate the combinations of keywords that are not present in the data set.

We adapted an efficient one-pass method to compute the correlation values of unique terms in the collection similar to the one presented in [2, 6]. The correlation and lift values are computed with the resulting aggregations from the summarization step. As a result, the vtf table is not scanned again to compute both values. Instead, we only used a summarization table. The lift values for each set of terms are computed from a separate aggregation step where we stored the number of documents that each term appears in.

3.2 Ontology building

The ontology is built from the pool of classes generated from the previous step. We then proceed to build links between the individual classes. Initially, we pair each class with all other classes within the pool. Let each set of two classes be a relationship. For each of these relationships, we extracted the correlation and lift of each set of two keywords. For example, suppose we have the relationship $\{A, B\}, \{C, D\}$. We would need to obtain the correlation and lift for the following set of keywords: (A,C), (A,D), (B,C), (B,D). These values would allow us to observe how closely related these two classes are to one another. A high correlation and a high lift from any one of these four subsets would confirm that this is a valid relationship. Otherwise, if these four subsets do not show a high connection, we would discard this relationship.

Upon completion of this process, we have a set of valid relationships, each formed from two classes. The final step is identification of relations of the form "has a", which define a hierarchy. In other words, we are only concerned with parent-child relationships to build a vertical ontology. The parent is determined with a heuristic based on the frequency of appearance of these classes. We considered a class (A) to be the parent of another class (B) in a relationship if the following criteria is satisfied: (1) Number of documents containing A - Number of documents containing B $<$ 10% of total documents, (2) Number of classes containing keywords in class A $>$ Number of classes containing keywords in class B. If class A passes both criteria, then we consider it to be a parent of class B.

4. EXPERIMENTS

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 OLAP algorithm was implemented entirely with SQL. We tested our approach with a real data set comprised of a corpus of sports articles centered around the Boston Celtics basketball team. There are a total of 50 documents based on game recaps with a total of nearly 15K individual terms.

Table 1: Num. of Classes varying corr, lift, and freq.

$Corr$	$Freq$	$Lift$	$No. t$	$Corr$	$Freq$	$Lift$	$No. t$
0.6	10	0.0	3765	0.8	10	0.0	2383
0.6	10	0.5	146	0.8	10	0.5	48
0.6	20	0.0	1122	0.8	20	0.0	529
0.6	20	0.5	83	0.8	20	0.5	22
0.6	40	0.0	229	0.8	40	0.0	91
0.6	40	0.5	42	0.8	40	0.5	5

Table 2: Sample Ontology Result.

Parent	Child	Parent	Child
Boston Celtics	Rajon Rondo	Box Score	Ray Allen
Boston Celtics	Ray Allen	Box Score	Kevin Garnett
Boston Celtics	Doc Rivers	Key Moment	Ray Allen
Boston Celtics	Kevin Garnett	Key Moment	Rajon Rondo
Box Score	Rajon Rondo	Key Score	Kevin Garnett

4.1 Summarization

Table 1 displays the total number of classes that are found when the shown thresholds are applied. These are the classes that form the pool from which the ontologies are later extracted. It is important to note that the total number of possible classes is 786382. Thus, even with minimal thresholds, we are still able to filter more than 99% of the possible classes. We can also observe that most influential threshold on the number of classes is lift. An important property that we observed during the experiments is that the larger the collection of documents, the fewer the number of incorrect classes we obtain. The rationale behind this is that true concepts tend to be more consistent throughout the collection.

4.2 Ontologies

The next step is to extract meaningful ontologies from this set of classes. Table 2 shows several examples of meaningful ontologies that were extracted. We were able to extract several key players from the Celtics team and also apply the correct parent-child relationship. Rondo, Allen, and Garnett are all players, while Rivers is the coach of the Celtics. The link of Box Score to the actual players makes perfect sense, since each player would have their own set of scores. The last three relationships are the most interesting because they provide us with descriptive terms. From them, we can surmise that the writers of these documents consider both Allen and Rondo to be vital in Key Moments while Garnett is a Key Scorer. We conducted additional verification on these discoveries and found them to be true.

It is also important to look at how many erroneous links are formed. To showcase this, we studied the accuracy of our resulting ontologies. In this case, accuracy represents whether a certain link makes sense or not. For example, Boston Celtics $\rightarrow$ Rajon Rondo is correct because Rondo is a player on the Celtics. Had the relationship been reversed, then this would be considered incorrect. The final tally of the accuracy is shown in Table 3. We can see that parents that are players consistently had lower accuracies than parents that represented items or stats. This can be explained by the fact that the documents we obtained were all game recap articles. As such, those articles contain a wide variety of players and teams, but are quite consistent in reporting the scores and stats of each game. The classes Key Box and Key Line obtained 0% accuracy because these classes themselves do not represent viable sets. We believe that with a larger and more varied pool of documents, the accuracies would improve.

Table 3: Accuracy of Results

Parent	No. Ch.	Acc. (%)	Parent	No. Ch.	Acc. (%)
Boston Celtics	19	89	Key Moment	11	91
Box Line	11	82	Key Score	12	83
Box Score	12	75	Paul Pierce	6	50
Doc Rivers	3	0	Points Rebounds	9	88
Game Celtics	17	100	Rajon Rondo	6	50
Glen Davis	1	100	Ray Allen	4	50
Kevin Garnett	3	33	Score Line	12	83
Key Box	11	0	Score Points	7	43
Key Line	11	0			
Overall	155	67			

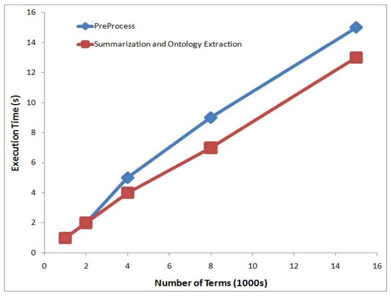

Figure 2: Ontology generation varying no. of terms.

4.3 Performance

Performance is a key factor in determining the usefulness of an algorithm. Thus, we provide two performance indicators. First, in Table 2, we show the general trend for both the preprocessing steps and the summarization/ontology extraction steps as we vary the number of initial terms. We can observe that the trend is linear with respect to the number of terms. This is to be expected since we are mostly using self-joins and table scans, so the number of terms should not exponentially affect the execution time.

We show a breakdown of the times for executing our algorithm with an initial term size of 15K in Table 4. We can see that the most heavily-weighted steps are the preprocessing step and the validating pairs step. We expected these two steps to be the most costly because of the self-join that is present in both steps. The preprocessing step includes the NLQ calculation for pairs of terms while the other step involves the actual pairing of the terms to produce the classes.

5. RELATED WORK

The need to automate the time-consuming generation of ontologies has led to multiple solutions that rely on previous knowledge or time-consuming tasks. OntoLT [1] is a plug-in for the Protégé ontology tool. It allows the user to define a set of rules for extracting an ontology from annotated text collections. Data mining approaches, such as [3], propose using a C4.5 decision tree to extract the main concepts. The non-leaf nodes are classes and the leaf nodes are individuals. In contrast to our approach, we perform this class extraction in one pass through the data. Ontobuilder is a ontology extractor based on a schema matching approach [5]. The ontology extractor is based on heuristic methods. Doddle-OWL [4] reuses given knowledge to extracts classes and relations (mostly WordNet). Then it relies on a refinement (feedback) to build a final ontology.

In [6], we realized that our query recommendation algorithm could also discover the main ontology concepts behind

Table 4: Breakdown of computation time (in secs).

Step	*Time*	*Percentage*
PreProcessing	11.0	40
Corr and Lift	2.3	8
Pairing Terms	11.9	44
Ontology	2.3	8

a particular topic. Our main contribution is that we are able to obtain classes and relations based only on efficient computations of several discrimination values (correlation and lift.) Unlike the proposed solutions, we require minimal human intervention and only a few parameters to be tuned.

6. CONCLUSIONS

We presented an algorithm for ontology extraction with OLAP cubes. The core of the proposal is the obtaining of an efficient keyword frequency summarization with a one-pass algorithm for computing the correlation and lift from pairs of keywords from a corpus. Once this analysis has been performed, a set of heuristics are used to create a hierarchy and find the relation between the classes. Our experiments show that our candidate classes are pruned early with minimal thresholds for lift and correlation to obtain meaningful classes. We observed that most influential threshold on the number of classes is lift, with frequency and correlation having similar selectivity effects. We also found that the most of the classes are quite relevant, and the overall hierarchy accuracy had an overall performance of 67% with a few misclassified concepts decreasing the overall average. However, for most of the concepts, the hierarchy achieved outstanding results of 75% or higher. The algorithm also showed linear scalability with term pairing and preprocessing as the most costly steps, respectively. Future research includes hybrid approaches (NLP, ML, OLAP), trying other heuristics, and finding other types of relationships between classes.

Acknowledgments

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

7. REFERENCES

[1] P. Buitelaar, D. Olejnik, and M. Sintek. OntoLT: A protégé plug-in for ontology extraction from text. In *Proc. of ISWC*, 2003.

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

[3] A. Elsayed, S. El-Beltagy, M. Rafea, and O. Hegazy. Applying Data Mining for Ontology Building. In *Proc. of ISSR*, 2007.

[4] N. Fukuta, T. Yamaguchi, T. Morita, and N. Izumi. DODDLE-OWL: Interactive Domain Ontology Development with Open Source Software in Java. *IEICE TOIS*, 91(4):945–958, 2008.

[5] A. Gal, G. Modica, and H. Jamil. Ontobuilder: Fully automatic extraction and consolidation of ontologies from web sources. In *Proc. of ICDE*, page 853, 2004.

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

[7] C. Ordonez. Statistical Model Computation with UDFs. *IEEE TKDE*, 2010.