Multidimensional Content eXploration

Alkis Simitsis[1] Akanksha Baid[2] Yannis Sismanis[1] Berthold Reinwald[1]

[1]IBM Almaden Research Center
San Jose, CA, USA
{asimits,syannis,reinwald}@us.ibm.com

[2]Univ. Wisconsin
Madison, USA
baid@cs.wisc.edu

ABSTRACT

Content Management Systems (CMS) store enterprise data such as insurance claims, insurance policies, legal documents, patent applications, or archival data like in the case of digital libraries. Search over content allows for information retrieval, but does not provide users with great insight into the data. A more analytical view is needed through analysis, aggregations, groupings, trends, pivot tables or charts, and so on. Multidimensional Content eXploration (`MCX`) is about effectively analyzing and exploring large amounts of content by combining keyword search with OLAP-style aggregation, navigation, and reporting. We focus on unstructured data or generally speaking documents or content with limited metadata, as it is typically encountered in CMS. We formally present how CMS content and metadata should be organized in a well-defined multidimensional structure, so that sophisticated queries can be expressed and evaluated. The CMS metadata provide traditional OLAP static dimensions that are combined with dynamic dimensions discovered from the analyzed keyword search result, as well as measures for document scores based on the link structure between the documents. In addition, we provide means for multidimensional content exploration through traditional OLAP rollup-drilldown operations on the static and dynamic dimensions, solutions for multi-cube analysis and dynamic navigation of the content. We present our prototype, called `DBPubs`, which stores research publications as documents that can be searched and –most importantly– analyzed, and explored. Finally, we present experimental results of the efficiency and effectiveness of our approach.

1. INTRODUCTION

Content Management (CM) and OnLine Analytical Processing (OLAP) are two separate fields in information management. Although both fields study models, concepts, and algorithms for managing large amounts of complex data, they started with very different applications as their major technology drivers. CM focuses on uniform repositories for all types of information, document/record management and archiving, collaboration, integrated middleware, etc., while OLAP is driven by financial reporting, marketing, budgeting, forecasting, and so on. Consequently, the two different fields emphasize very different aspects of information management: information capturing, storing, retention, and collaboration on the CM side [33], while data consistency, clear aggregation semantics, and efficiency on the OLAP side [17].

Various advanced applications recently emerged, impose modern user and business needs that require the benefits of both, CM and OLAP technologies. Digital libraries [21] are becoming very rich content repositories containing documents along with metadata/annotations stored in semistructured data formats. Intranet data, wikis and blogs [4] represent examples of this theme. The above examples are just a subset of modern applications, where the traditional information retrieval techniques –e.g., keyword or faceted search– is not enough, since advanced analysis and understanding of the information stored are required. On the other hand, application areas such as customer support, product and market research or health care applications with both structured and unstructured information, are mission-critical and require both CM and OLAP functionality, as well.

However, the synchronous application of both techniques is not straightforward. At first, the user models for CM and OLAP are dramatically different. In CM (as in information retrieval too), a user is a human with cognitive capabilities. CM queries are best effort formulations of a user's information needs, and support an interactive process of data exploration, query rephrasing, and guidance towards the final query result. In OLAP, a user is more like an application programmer using MDX, SQL, XQuery or some API to access the data. OLAP employs a multidimensional data model allowing for complex analytical queries that are precise, and provide exact query results as fast as possible. The OLAP query result is typically a matrix or pivot table, with OLAP dimensions in rows and columns, and measures in cells. In summary, CM query processing depends on ranking, while OLAP query processing is an aggregation task based on testing and grouping logical predicates.

Additionally, navigating and dynamically analyzing large amounts of content effectively is known to be a difficult problem. Keyword and semantic search help alleviate this problem to some extent by returning only the top-k relevant documents that contain the search keywords. While this maybe a satisfactory result for short hit lists and for information retrieval, it is not acceptable when thousands of documents qualify for the search keywords and *the user is interested in all of them*, i.e., ranking is not effective. The user wants to understand the entire hitlist, look at the result from many different angles and different levels of detail. Traditional OLAP techniques seem to be a desideratum to such a problem, since they have been known to be effective in analyzing and navigating through large amounts of structured data. Unfortunately, unstructured data does not lend itself well to *traditional* OLAP style analysis.

In this paper, we introduce a novel framework for multidimen-

VLDB '08, August 24-30, 2008, Auckland, New Zealand

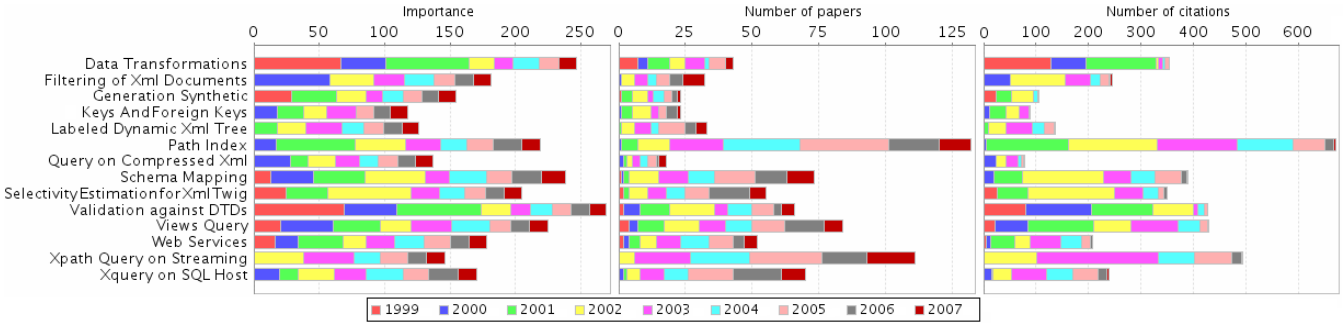


Figure 1: Keyword Query: "xml", evolution of discovered topics over years

sional content exploration, MCX. We propose keyword driven OLAP over both structured and unstructured content to populate static as well as dynamic dimensions, which allow for roll-up and drill-down operations based on the *content* and the *link-structure* of the dynamically selected document subset. Analyzing text content and extracting properties of such dynamic subsets is a challenging problem. Even more so, since in MCX the system has only a few seconds to perform the text analysis and the corresponding OLAP aggregation. Although there is a plethora of work in analyzing text, most of it is not applicable under the extreme time constraints that we face. For that reason, we present a novel technique for performing the *core* operation of extracting frequent phrases. Our approach scales extremely well w.r.t to the size of the selected subsets (hit list.) It is designed so that, as the hit list size increases, the performance increases! We demonstrate that such frequent phrases are the foundations for more elaborate content-based extraction. We show how to further filter and refine such frequent phrases, by applying modified well-known techniques (such as relevance boosting and latent-semantic-indexing) to group documents with similar text together, and even extract representative group descriptions. Finally MCX incorporates the link-structure among the documents to extract elaborate measures (like document centrality/importance) over the dynamically selected subset in order to provide interesting reports to the user. As a proof of concept, we have implemented an end-to-end prototype, called DBPubs (see Section 6), which allows OLAP style analysis –e.g., aggregations, groupings, trend analysis, and excel-like pivot tables– on top of a CMS containing a large set of publications from the field of databases and other adjacent areas. Each publication has metadata such as *title*, *authors*, *venue*, *citation links* and *citation count*, *year*, and so on.

As a motivating example, assume that a user is interested in finding out the various research topics that have been investigated in the XML domain in the past years, locating each area's *seminal* papers, and finding out the corresponding *influential* authors. Regular keyword search using a text index could be used to retrieve all publications containing the keyword 'XML' in the text or metadata. However, the query result comprises thousands of papers and would require extensive browsing and paper reading in order to get a grip on the subject area. Our approach contributes towards facilitating such procedure. It uses the navigational and analytical power of OLAP in synergy with text analysis. Static dimensions like location, time, venues are combined with dynamic dimensions based on the publication content like frequent and relevant phrases, topics, and so on. For the latter, DBPubs analyzes the publications in the query result. It efficiently extracts the frequent phrases, it boosts the most relevant phrases (i.e., frequent phrases that appear proportionally much more often in the result than the full corpus, like 'XQuery'), it groups documents with similar relevant phrases together and it dynamically identifies meaningful group descriptions; e.g., 'Schema Mapping' and 'XPath' (see Figure 1.) Furthermore, we exploit citations as a link structure to rank the publications and extract publication importance, and then, by leveraging the aggregation power of OLAP show, for example, the maximum importance per topic in the course of time and compare against the total number of papers and citations (Figure 1.) Hence, the dynamic analysis of the query result combined with an OLAP model, provides the user with the necessary analytical means to slice and dice the dataspace, discover and understand important trends.

Contributions. In summary, in this paper, we make the following contributions:

- We introduce MCX, a novel extensible framework that allows the non-technical user to get insight (reports, forecasts, trend analysis) by combining content-based, structure-based and link-based explorations of CMSs.
- We describe a formal method for mapping CM metadata to a structure-based multidimensional schema, and, most importantly, present means for enriching such schema *at query-time* with content-based dimensions based on the properties of the dynamically selected subsets and measures based on the link structure of the subsets.
- We propose a novel efficient and accurate algorithm for computing the *core* content-based property *frequent phrases*, on the dynamically selected subsets.
- We present an implementation of the above and a case study with DBPubs that demonstrates that our approach builds highly accurate and useful dynamic OLAP reports, while requiring negligible human effort in maintaining it and expanding it.

Outline. The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 provides a system overview. Section 4 addresses modeling issues and challenges. Section 5 explains the process of extracting content-based and link-based properties of dynamically selected subsets. Section 6 describes our case study and experimental evaluation.

2. RELATED WORK

CMS's store vast amounts of content with limited metadata in document stores. MCX enriches search on the content in CMS's by providing an analytical view over the content through analysis, aggregations, groupings, trends, pivot tables, and so forth. There is a significant amount of literature that addresses different aspects of MCX, e.g., content management, search, OLAP, and data warehousing. Traditionally speaking, from a data perspective, the related

work is about structured or unstructured data, while from a processing perspective, the related work is about search or analytical processing. In the future, these four different perspectives will converge. MCX contributes to this ultimate goal addressing OLAP on unstructured documents. Depending on the particular research perspective, other solutions towards this ultimate goal are proposed.

Multi-Dimensional Modeling. Principled multi-dimensional schema (MD) design has been researched extensively [24, 29, 23, 22].) Our work is orthogonal to this body of literature, since, we are not interested in automated multi-dimensional schema design. Instead, we focus on (a) identifying the appropriate mappings between the CM and MD that can be used as input to such approaches, and, most importantly, (b) enriching the MD with content-based and link-based properties of dynamic subsets.

OLAP. OLAP traditionally focused on large data warehouses of structured data. We refer to [9] for an overview of the topic. Dynamic Data Warehousing is about adding mining and analysis capabilities and deriving new elements to adapt to change (HP Neoview and IBM Dynamic Warehousing.) Keyword driven analytical processing over warehouses of structural data is described in [34]. Our work extends OLAP by combining content-based, structure-based, and link-based reporting over text collections.

Search. Search on unstructured data evolved to multifaceted search [1] that is often deployed at e-commerce web sites. It provides a keyword-driven way to search and further refine query results. For example, Endeca's "MDEX" engine implements guided navigation that is integrated with search. Similarly, "Autonomy" performs automatic categorization, hyperlinking, retrieval and profiling of unstructured data, and therefore enables additional search operations on such data. To the best of our knowledge, multifaceted search applications provide only count-based information, while we support advanced OLAP functionality providing more sophisticated measures, like document importance and link-analysis. Unlike (multifaceted) search, MCX is not limited to navigating in order to locate a single interesting document; it provides the means for reporting, forecasting, trend analysis, and exploration of aggregates.

Content-based Extraction. Automatically extracting content properties (like frequent phrases, relevant phrases, topics or summaries and clusters) is very important and has been the focus of a huge amount of research work. For example, discovering models [7] or applying linear projection techniques, such as latent semantic indexing (LSI) [12] are commonly used for topic extraction by representing high dimensional vectors in a low dimensional space. Local regions in the lower dimensional space can then be associated with specific topics. Several well known document clustering techniques [27] have also been investigated. Most of these techniques suffer from the drawback that one document can only fall in one cluster. Probabilistic alternatives like pLSI [19] have also been explored. While pLSI works well in a number of domains it has not been found very effective with new documents that differ considerably from the training data. In MCX it is extremely important to perform such extraction over dynamically selected subsets efficiently, since the system has only a few seconds to present the results to the user. We experimented with a novel technique for performing the *core* operation of extracting frequent phrases and then for completeness we applied a classic LSI technique to discover topics.

Research systems. Although our approach per se is more generic, still, there are several research projects that are related to our prototype, such as (a) DBLife [13], which is a portal for the database community that focuses on high accuracy extraction and integration operators; (b) DBLP [11], a website that manages only the metadata for publications; and also, (c) Faceted DBLP [14], which provides faceted search on the bibliographic metadata without analyzing any content, according to our understanding by its use; e.g., citation analysis is not taken into consideration. The latter comment also holds for some other systems like Eventseer [15], and Publish or Perish [18]. In a different context, BlogScope [4] is a system for online analysis of streaming text, which focuses on text analytics, but does not consider multidimensional OLAP processing.

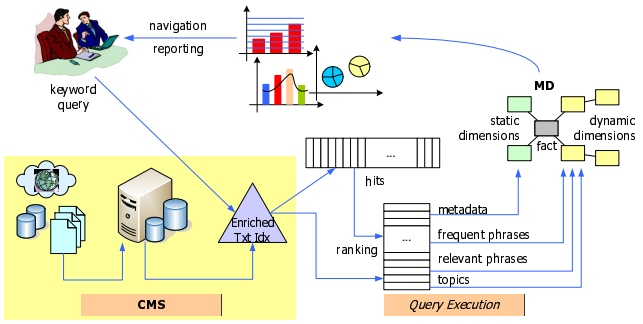


Figure 2: Overview of MCX

3. SYSTEM OVERVIEW

In this section, we present an overview of our system. We focus on the end-to-end description of the dataflow in the system from submitting the keyword query to returning analytical query results. In the rest, without loss of generality, we will frequently refer to examples adapted from the context of DBPubs, in order to clarify better the concepts introduced. Sections 4 and 5 will fill in the algorithmic details on modeling and query execution, respectively.

Figure 2 depicts the system overview. The CMS stores the content in a document store. It also maintains a content model for the metadata, which includes links between the documents. The metadata can be either simple attributes such as document timestamps, authors, title, and so on, or more sophisticated attributes extracted from the content, such as vehicle type in a car insurance policy or a link to the insurance agent who sold the insurance policy. The CMS maintains an enriched text index that allows for keyword search on both the content and the metadata.

For obtaining OLAP style functionality, our goal is to create a multidimensional, MD, schema by using information from the content and metadata of the documents stored in the CMS. This procedure generates some modeling and operational challenges.

Modeling issues. The upper right of Figure 2 shows a generic multidimensional schema. For creating it, we interpret the individual concepts of the CMS (i.e., document, content, metadata, and links) as multidimensional entities (i.e., facts, dimensions, and measures) of this MD. The mapping should be complete –missing elements are not allowed– and correct –the aggregating results in the MD should be correct throughout the navigation into the schema. In doing so, there are several challenges that we have to deal with, such as the cases of multi-valued attributes and multi-cubes. The mapping from CMS to MD is formally presented in Section 4.

Operational issues. The user approaches the system by submitting a keyword query. The query language is generic enough and supports all usual operators (AND, OR, NOT, and so on) used in modern textual indices. In fact, our approach may support, as well, any form of querying such as NLP, taxonomy search, semantic search, and so forth. Our query processing consists of a sequence of operations executed each time a query is issued. First, the text index on the documents in the CMS is probed with the query keywords to generate a potentially large hit list. Then, the

population of the MD takes place. The metadata of the documents in the hit list are used for populating the *static dimensions*, i.e., the dimensions that can be created a priori having only the knowledge emanated from the CMS metadata. Analyzing the content of the documents in the hit list, yields to *dynamic dimensions* in the MD, e.g., dimensions based on the topics of the documents, the frequent or relevant phrases, and so on. Hence, query execution involves the following operations, that are described in detail in Section 5.

Content-based dimension extraction A core property of dynamically selected documents are the frequent phrases in this subset. Such frequent phrases form the foundation for much more elaborate property extraction. Since in MCX the system has only a few seconds to perform any extraction, we provide a novel technique that scales extremely well w.r.t. to size of selected subset and actually, it performs faster as the selected subset size increases. Finally, we demonstrate how frequent phrases can be used to group documents with similar text together and even provide meaningful group descriptions by applying techniques such as relevance boosting and latent-semantic-indexing.

Importance extraction. Ranking of documents is essential in our design. Our framework allows for different kinds of ranking techniques, either for example, based on the link structure between documents or relevance from a text index. Ranks are used as measures in MD, but also for dealing with scalability issues in discovering dimensions, as ranks allow us to focus on the important content and deal with noisy content otherwise.

The keyword query helps users to easily search for content, and the MD organizes the results to enable *OLAP-style navigation* and allows us to use OLAP tools for interactive reporting. We provide analytical reports built on the dynamic dimensions –clearly, one may consider more such dimensions than the above– and highlighting interesting observations for helping users better understand the entire query result, before starting the navigation.

4. MULTIDIMENSIONAL DESIGN FOR CONTENT

In this section, we formally present the mapping of the content and the metadata of a CM system to a multidimensional schema in order to leverage the aggregate power of OLAP. We introduce design guidelines for such a mapping and we discuss modeling challenges. Particularly, we discuss problems and propose solutions for modeling multi-valued dimensions and we provide means for coherent navigation in crossing several document types.

4.1 Preliminaries

A finite set containing n members is denoted as $\Xi = \{\xi_i\} = \{\xi_1, \ldots, \xi_n\}$, where ξ_i is the ith member, $i \in [1, n]$, and its cardinality as $|\Xi| = n$. We denote a *total strict order* and a *partial order* on a set $\{\xi_i\}$ as $\prec\{\xi_i\}\succ$ and $\preccurlyeq\{\xi_i\}\succcurlyeq$, respectively. In both cases, the order is considered from the left (min) to the right (max); i.e., $\prec\{\xi_i, \xi_j\}\succ \Rightarrow \xi_i < \xi_j$ and $\preccurlyeq\{\xi_i, \xi_j\}\succcurlyeq \Rightarrow \xi_i \leq \xi_j$.

Content Management System. A CMS comprises different types of documents –e.g., text documents, web pages, e-mails, and so on– along with their respective sets of metadata. Documents of the same type share the same set of metadata, while, usually, different sets of metadata correspond to multiple types of documents. Each *document type* consists of *metadata* and *content*. The latter contains a set of *terms* and *phrases*, which are, potentially, overlapping sequences of consecutive terms. Documents may connect with each other through *links* that signify a semantic relationship between the documents. Formally, a CMS is denoted as a set $S_{CM} = \prec\{D^T\}\succ$ of document types D^T. Each document d_i is an instance of a document type, consists of metadata, links, and content, and is defined as $d_i = \prec M_{d_i}, L_{d_i}, C_{d_i}\succ$. The metadata of a document follow a structured schema of attributes, both single-valued, $a = \prec name, val\succ$, and multi-valued attributes, $a^* = \prec name, \{val\}\succ$, and they are defined as $M_{d_i} = \prec\{a\}, \{a^*\}\succ$. The links of a document are incoming, L^-, and outgoing, L^+, thus: $L_{d_i} = \prec L^-, L^+\succ$. The content of a document comprises a total ordered set of the terms existed in the document: $c_{d_i} = \prec\{t_i\}\succ$. The content of document type is defined as the content of all the documents of that type, i.e., $C = \{c_{d_i}\}$.

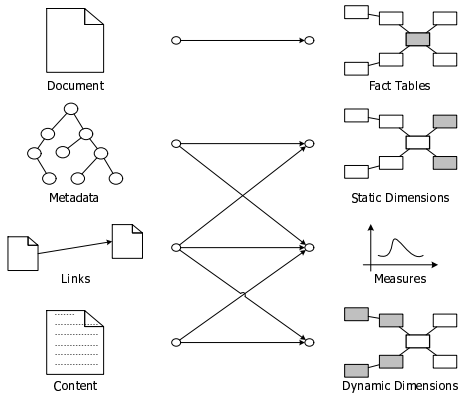


Figure 3: Mapping of CM concepts to MD

Multidimensional Schema. A multidimensional schema (MD) comprises a set of factual schemata, i.e., $S_{MD} = \prec\{F_i\}\succ$. A factual schema or simply, fact, F_i, comprises a set of dimensional schemata and a set of measures: $F_i = \prec\{D_j^{F_i}\}, \{\mu^{F_i}\}\succ$. A dimensional schema or simply, dimension, D_j, is defined as a total ordered set containing a name and a poset of levels, i.e., $D_j = \prec name, \preccurlyeq\{l_i\}\succcurlyeq\succ$. The dimensions are orthogonal to each other or equally, there are no functional dependencies among attributes from distinct dimension schemata. A level is defined as a total ordered set of a name and an attribute, i.e., $l_k = \prec name, a\succ$. The levels in a dimension are functionally dependent to each other. A measure μ is an instance of an attribute, that contains values of a numeric data type. The measures are functionally determined by the levels occurring in the dimension schemata. Also, the measures can be computed from each other and they are useful as they allow to perform roll-ups and drill-downs beyond the hierarchies.

4.2 Creating a multidimensional schema for content

This section formalizes the mapping of a CMS, S_{CM}, to a multidimensional schema, S_{MD}, i.e., $\mathcal{M}$: $S_{CM} \rightarrow S_{MD}$. The following definition ensures the correctness of the mapping in the sense that all the information stored in the CMS should be mapped to a multidimensional schema and that the result of any valid aggregation in the latter schema would be correct.

Definition 1. For each document type $D_l^T \in D^T$ of S_{CM}, a factual schema F_l is created in the multidimensional schema S_{MD} through the mapping $\mathcal{M}_{D_l^T \rightarrow F_l}$ that satisfies the following two properties: (a) completeness w.r.t. coverage, and (b) correctness.

Completeness w.r.t. coverage. This property ensures that all the elements of the CMS, and therefore, all the information stored in it, are represented in the multidimensional schema. Each document d_i of a certain document type D_l^T, its corresponding metadata M_{d_i}, links L_{d_i}, and content C_{d_i} is mapped to the factual schema F_l, namely the dimensions $D_j^{F_l}$ and the measures μ^{F_l}. Figure 3

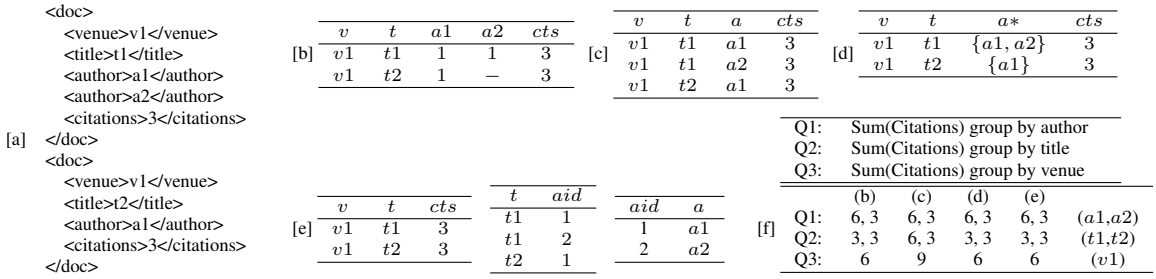


Figure 4: Design cases for multi-valued dimensions

pictorially depicts the aforementioned mappings. Formally, the satisfaction of the first property is ensured by the following definition.

Definition 2. The mapping $\mathcal{M}_{D_l^T \to F_l}$ is complete w.r.t. coverage iff each element of a document $d_i = \prec M_{d_i}, L_{d_i}, C_{d_i} \succ$ belonging to a document type D_l^T is mapped to a fact $F_l = \prec \{D_j^{F_l}\}, \{\mu^{F_l}\} \succ$ through the following mappings:

- mapping of d_i to the dimensions (mapping function f): f_M: $M_{d_i} \to D_j^{F_l}$, f_L: $L_{d_i} \to D_j^{F_l}$, f_C: $C_{d_i} \to D_j^{F_l}$
- mapping of d_i to the measures (mapping function g): g_M: $M_{d_i} \to \mu^{F_l}$, g_L: $L_{d_i} \to \mu^{F_l}$, g_C: $C_{d_i} \to \mu^{F_l}$

A concrete definition of the mapping functions f and g heavily depends on the application considered each time. However, a set of generic properties characterize both functions. (For the formal representation of those properties, we refer the interested reader to the long version of the paper, which is available upon request.)

Correctness. The second property of the Definition 1 requires that the aggregates in the MD should be correct whatever slice of the MD is used to produce them. Traditionally, the multidimensional design deals with single-valued attributes, for which –and for the respective levels and dimensions as well– the correctness is ensured by a *summarizability* condition. However, this does not hold for multi-valued attributes, as we discuss in section 4.3.

The notion of summarizability was introduced in [30] as a property of statistical objects. Later on, it was described in the context of OLAP [23, 24, 29], first in [25], in a rather informal way, and then, formally, in [20]. Summarizability refers to whether a simple aggregate query correctly computes a single-category dimension instance from another precomputed single-category dimension instance in a particular database instance. In our approach, we adopt the definition proposed in [25], and therefore, formally, the satisfaction of the summarizability in our context is ensured by the following definition.

Definition 3. The mapping $\mathcal{M}_{D_l^T \to F_l}$ ensures the summarizability iff the following conditions are satisfied: (a) disjointness of levels and dimensions, (b) completeness of the information stored in the dimensions, i.e., all instances exist in each level and are assigned to an instance of an upper level, and (c) type compatibility.

The second condition is enforced during the population of the multidimensional schema. The third condition is taken into consideration during the selection of the measures, that is, it is considered as a condition in the application of the function g. In the rest, we elaborate more on the first condition, which so far has been dealt with by the research community as an implicit restriction of dimension models: the mappings between members of two levels are many-to-one relations. In the next subsection, we discuss this restriction, which is very common in CMS, and we propose a method for dealing with many-to-many mappings among the levels.

4.3 Multi-valued dimensions

In this section, we deal with the correctness problem occurred by the presence of multi-valued attributes, as they introduce many-to-many relationships between two levels of a dimension or across dimensions. We generalize this issue by referring to it as the multi-valued dimension challenge. Relational-based DW and OLAP systems avoid such many-to-many relationships, because they complicate the semantics of the returns aggregates due to double counting and overlapping problems. However, recent proposals (like [5] for XQuery) already start to depict the complexity of the problem in the semi-structured world. It's becoming increasingly more clear that we need to incorporate cleanly such many-to-many relationships in the MD design.

Definition 4. A dimension that has levels containing multi-valued attributes a^* is called *multi-valued dimension* and is denoted as D_j^*. Similarly, those levels are called *multi-valued levels* and are denoted as l_i^*.

Consider the xml snippet of Figure 4(a), which describes two documents: $\{d_1, d_2\}$, of the same type, which have been published in the venue v_1 and have 3 citations each. Also, the first one has two authors: $\{a_1, a_2\}$, while the second has only one author: $\{a_1\}$. Thus, there is a many-to-many relationship between titles and authors. The unique document type should be mapped to a fact F. However, for the design of the dimensions there are four different candidate design choices that are discussed below.

Case I: Expand the width of the fact table. Create $card(l^*)$ attributes in the fact table F, each one corresponding (through a FK relationship) to a different dimension. This solution is clearly infeasible in the presence of a reasonably high number of unique values in that level or even worst, in the general case, where the contents of D^* are dynamically evolving during the time. For the example of Figure 4(a), this case practically signifies the creation of one dimension per different author (see Figure 4(b).)

Case II: Expand the height of the fact table. Essentially, each F-tuple expands to the number of D^*-tuples that it can join with. This case is pictorially depicted in Figure 4(c).

Case III: Treat corresponding D^*-tuples as a set inside F. In that case, each F-tuple corresponds to exactly one D^*-tuple, but in that case the multi-valued attribute is of the type 'set'; e.g. of the type 'xml' (see Figure 4(d).)

Case IV: Use a bridge table between F and D^*. Finally, another case is to use a bridge table [22] for relaxing the many-to-many relationship. In that case, a bridge table intervenes between the in-

volved entities, containing one tuple per each unique pair of values from both entities (see Figure 4(e).)

Given that the first candidate cannot scale up in the general case, we examine the other three cases w.r.t. the correctness of the aggregating results. To facilitate the discussion, we consider the example aggregate queries listed in Figure 4(f). The second case produces correct results for the multi-valued level, i.e., authors. Still, due to the violation of the summarizability condition –the property disjointness of levels does not hold– the aggregating results for the upper levels are not correct (see 4(f).)

The last two cases produce correct results. The third case has solved the disjointness problems among the levels with the use of a set. The aggregating results for the single-valued levels are obviously correct, while the aggregation for the multi-valued level can be realized at query-time by the appropriate expansion of the sets. (Modern DBMS's support table functions, e.g., XMLTABLE, which return a table from the evaluation of XQuery expressions [5]; after the expansion, the groupings are realized as usual.) In the last case, the use of a bridge table converts the many-to-many relationship between a single-valued and a multi-valued level, e.g., titles and authors, into a many-to-one relationship; in the example of Figure 4(e), between the bridge table and the authors table a many-to-one relationship exists. In our implementation, we followed the last approach, since the expansion of multi-valued sets does not perform well for the amount of data we are dealing with.

4.4 Multi-cubes

According to Definition 1, each document type is mapped to a factual schema. At a later stage, the final multidimensional schema corresponding to the CMS, probably, would contain more than one factual schema. Our modeling framework scales up to the calculation of aggregates through different document types represented as different facts. For example, assume that the CMS contains two different document types representing publications and web homepages of the authors, respectively. Following our modeling approach, in the multidimensional schema two facts would exist: F_{PUBS} and F_{HOMEP}. The two schemas may share a number of common dimensions; e.g., the one about authors, D_{AUTHOR} (see Figure 5b.) In order to create calculated measures on top of those facts and/or to answer queries by combining information from more than one fact, we need a virtual fact defined on top of the existing facts based on the dimensions they share.

For this kind of modeling, we adopt the terminology of [31] and we refer to that virtual fact schema as a 'multi-cube' and to the columns used to join the original fact tables as the 'join dimensions'. The construction of multi-cubes is pictorially depicted in Figure 5(a) and is formally defined by the following definition.

Definition 5. Assume the existence of k factual schemata $F_i = \prec\{D_j^{F_i}\}, \{\mu^{F_i}\}\succ$, $i\in[1,k]$ that share l dimensions. These dimensions are called *join dimensions* and are represented as $D_j^{\bowtie}$, $j\in[1,l]$. A *multi-cube* $\mathcal{F}^m = \prec\{D_j^{\mathcal{F}^m}\}, \{\mu^{\mathcal{F}^m}\}\succ$ is defined on top of the k facts as a view that joins the k facts on their join dimensions s.t. $\{D_j^{\mathcal{F}^m}\} = \preccurlyeq\{D_j^{\mathcal{F}^1}\}, ..., \{D_j^{\mathcal{F}^k}\}\succcurlyeq$ and $\{\mu^{\mathcal{F}^m}\} = \prec\{\mu^{\mathcal{F}^1}\}, ..., \{\mu^{\mathcal{F}^k}\}, \{\mu^{new}\}\succ$, where μ^{new} is a measure calculated from a combination of measures from the k facts constituting the multi-cube.

Notice that a certain record in an instance of a multi-cube may contain values other than 'null' only in the join dimensions. During the navigation of a multi-cube, other dimensions beyond the join dimensions can be used as well. However, in each navigation only the join dimensions and the dimensions of one fact can be used synchronously. In the example of Figure 5(b), when we navigate through the F_{HOMEP}, it is possible to get values from measures

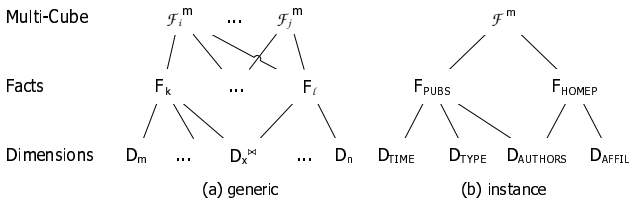


Figure 5: Multi-cubes' design

defined on (a) the dimension D_{AFFIL}, e.g., the number of affiliations related with a certain homepage, and (b) the F_{PUBS} through the join dimension D_{AUTHORS}, e.g., the number of papers (co-)authored by the author having that homepage. However, we cannot get any values w.r.t. the D_{TIME}, e.g., the number of papers broken-down by year. In the latter case, null values are returned instead.

5. DYNAMIC CONTENT-BASED EXTRACTION

Having discussed the mapping of CMS to MD, we elaborate on its population and enrichment by dimensions with content-based extracted properties of the query subset. In particular, `MCX` executes queries by primarily accessing the text index on the documents in the CMS. The text index is probed with the keyword query, and the corresponding documents are returned as "hits". A fundamental extracted property is the set of frequent phrases in the hit list. Such phrases provide the foundations for extracting more elaborate dimensions. With that in mind, we show how to filter the most "relevant" phrase dimension and finally, combine term/phrase synonymy and document importance in order to group together documents with similar text and extract representative labels for the groups. In `MCX` we deal with extreme time constraints since the system has only a few seconds to perform any extraction from potentially very large hit lists. Hence, we propose a novel and efficient algorithm for discovering the *core* frequent phrase dimension.

5.1 Globally Frequent Phrases

The frequent phrases that consist of two to five words that appear in more the $\tau = 5$ documents in the corpus are denoted as *globally frequent phrases*. For example, globally frequent phrases in `DBPubs` are "data mining", "query optimization", "data integration", and so on. Intuitively, when authors write about something, they usually repeat the subject-related keywords to keep the attention of the readers. When the same phrases are repeated a lot in *different* documents, then it is a strong indication that the phrases describe an interesting subject in the entire corpus. We emphasize that we use the latter definition (distinct document-frequency) as the frequency of a phrase. In other words, globally frequent phrases capture interesting information about the corpus and complement the single term information that is typically kept in the text index. The threshold τ is used in order to filter "noise" phrases that just happen to appear in a small number of documents.

We "enrich" the text index with posting lists for globally frequent phrases in a preprocessing step, using a sliding window over the document content, in order to generate the phrases, and lossy-counting [26] to discover the phrases with frequency above τ. These posting lists enable the efficient discovery of dynamic frequent phrases as described below.

5.2 Dynamic Frequent Phrases/Terms

Dynamic frequent phrases are the globally frequent phrases that appear frequently in the documents of the hit list. One way for

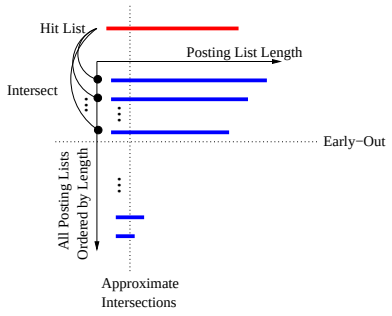


Figure 6: Frequent Phrase/Term Discovery Overview

discovering the top-k dynamic frequent phrases/terms is to intersect the hit list with each individual phrase/term posting list in the text index. We maintain a priority queue of the top-k most frequent phrases/terms in the process. When all the posting lists have been processed, the queue contains the top-k most frequent phrases/terms for the query. Finally, we create a multi-valued dimension for each document in the hit list by associating it with all the discovered frequent phrases that it contains.

5.2.1 Pruning Methods

Although the above idea is simple, it does not scale well to posting and hit lists with millions of documents. In order to minimize the processing required, we deploy two novel pruning methods: *early-out* and *approximate intersections*.

Early-out. We process the posting lists in descending order of their length [32]. As the algorithm progresses, and we find a posting list with a length less than the current minimum intersection size in the queue, then it is certain that the rest of the posting lists cannot make it into the queue, and we can stop the processing.

Approximate intersections. The exact intersection of the full hit list H with a posting list P requires $O(|H|+|P|)$ comparisons. We propose a novel technique (see section 5.2.3, that provides accurate estimates for the intersection size by performing a small and bounded number of comparisons. The technique provides performance improvements by orders of magnitude as hit and posting lists contain millions of documents.

5.2.2 Description of the DFP Algorithm

In Algorithm 1, given the hit list H for a keyword query, the sorted posting lists P for the phrases/terms from the text index and the number k, we efficiently find the top-k frequent phrases/terms for the documents in the hit list.

Algorithm 1 Dynamic Frequent Phrase Discovery (DFP)

```
1:  P: Posting lists sorted by descending order of length
2:  H: Hit list
3:  k: number of frequent phrases/terms
4:
5:  queue: priority queue of ⟨phrase,count⟩
6:  for i = 0 to |P| do
7:    if |P[i]| < minimum count in queue then
8:      break                                   // early-out (section 5.2.1)
9:    end if
10:   count ← P[i] ∩ H                         // approximate intersections (section 5.2.3)
11:   insert ⟨P[i]_phrase, count⟩ in queue
12:   if queue contains more than k items then
13:     pop entry with smallest count from queue
14:   end if
15: end for
16: find the document ids in H for each frequent phrase/term in queue
```

The effects of early-out and approximate intersections are depicted graphically in Figure 6. Notice, that both methods nicely complement each other. On the vertical dimension, early-out bounds the total number of posting list intersections that are performed, while on the horizontal dimension, approximate intersections limit the number of individual comparisons between the hit list and the current posting list.

5.2.3 Approximate Intersections (AI)

We describe a technique to efficiently discover frequent phrases by approximately intersecting posting lists. By *estimating* the intersection size in line 10 of Algorithm 1, we get a huge performance benefit without sacrificing the quality of the returned top-k results.

Algorithm 2 Posting List Randomization

```
1: L: Posting list to randomize
2: h: hash function into (0, 1) domain
3:
4: for i = 0 to |L| do
5:   L[i] ← h(L[i])
6: end for
7: sort L in increasing order
```

The idea behind the estimation is to "randomize" all the posting lists using Algorithm 2, which hashes each document id into a random number in the $(0, 1)$ domain, and then sorts the hashed values in increasing order. We can exploit certain properties of the hashed values to estimate the union and intersection size very efficiently. We apply Algorithm 2 to randomize all the posting lists in the text index as a preprocessing step. At query time, we use the same algorithm to randomize the hit list, and then we use Algorithm 3 to estimate the intersection size in line 10 in Algorithm 1.

The approximate intersection Algorithm 3 takes as input two randomized posting lists P_1, P_2, the maximum number M of comparisons and the maximum number I of common points. The algorithm returns an accurate estimate of the intersection size $|P_1 \cap P_2|$. The operation of the algorithm is illustrated in Figure 7. We perform a "zipper" algorithm to intersect the posting lists (lines 8-35). Initially the indexes c_1 and c_2 point to the beginning of the randomized posting lists. We compare the pointed at values, and we forward the pointer that points to the smaller value. The nI counter maintains the common points found while zipping the two posting lists together. This would be a classic zipper algorithm, but we introduce two novel optimizations:

1. we *skip uniformly* over the randomized posting lists (lines 21 and 28), and
2. we *stop* after M comparisons (line 10) or after we've found I common points (line 18.)

The intuition behind the two optimizations is, that the hashed values in the randomized posting lists resemble closely uniform points in the $(0, 1)$ domain. We call gap_1 (and gap_2) the average distance between two consecutive values in P_1 (and P_2). Notice, that the expected distance between two consecutive points in P_1 is $\text{gap}_1 = P_1[|P_1| - 1]/|P_1|$ due to the uniformity of the points.

A classic zipper algorithm visits *all* the points in P_1 and P_2. However, in our case, we can exploit the uniformity of the points by *skipping* in a principled way over comparisons that can never be true. As an example, let's consider the first comparison that the zipper algorithm performs between values $P_1[0]$ and $P_2[0]$ in Figure 7. We see, that $P_2[0]$ is greater than $P_1[0]$ and therefore we must increase the index on the P_1 list. We can use the distance $(P_2[0] - P_1[0])$ as an indicator for where the corresponding point

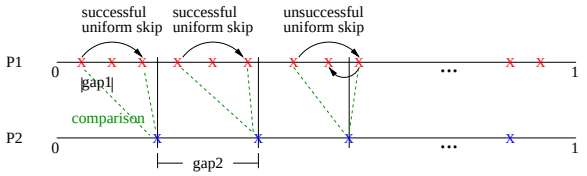


Figure 7: Approximate Intersection

(if any) on P_1 can be. More precisely, since the points resemble uniform random points, we *uniformly skip* $(P_2[0] - P_1[0])/\text{gap}_1$ points on P_1. Similar skipping can occur on P_2. Since the hash function is not perfect and the hash values are not exactly equi-distributed in the $(0, 1)$ domain, it is possible that the skipping may "overshoot". If overshooting happens, then we abort the skip and increase the corresponding index by one. This case is handled in lines 22 and 29 in Algorithm 3. One can observe, that uniform skipping is very effective, when posting lists of very different sizes are intersected as wide skips happen over the large posting list. On the other hand, when posting lists of similar sizes are intersected, more frequent overshooting happens, which can affect in a negative way the performance of the algorithm. Thus, we propose to disable the uniform skipping, when the posting lists have about the same size.

Figure 7 shows that the pattern of comparisons is normal and repetitive (due to the uniformity of the hashed values). We avoid doing comparisons all the way to the end of the posting list, and we stop as soon as we've performed M comparisons in total or we've found I common points. Typically, M is chosen to be much smaller than $|P_1| + |P_2|$. It is important to mention that M is actually independent of the actual posting list sizes (which could be many million points in practice) and that the values of M and I provide a strict confidence interval. M bounds the absolute error, while I bounds the relative error. We leave the details of the exact confidence interval computation as a future work, but most of the results in [6] apply in this domain too. We must emphasize, however, that our technique and our assumptions are very different from KMV-synopses[6]. We are not interested in producing small set-synopses for performing generic set operations. Instead, we maintain the full set and optimize the number of comparisons required to compute only the intersection size. However, both, our approach and the KMV-synopses, share the same randomization idea and the methodology/proofs developed in [6] can still be applied for our technique.

Since we are not performing all the comparisons (but at most M), we apply the formula in line 39 to *estimate* the actual intersection size based on only these observations. We can show, that this formula is a tight estimator for the intersection (for space consideration, the details can be found in [6]). The intuition however is, that (a) the quantity $c_1 + c_2 - nI$ maintains the number of *distinct* points in the union of P_1 and P_2 that we have seen at any given moment, and $\frac{c_1+c_2-nI-1}{max\{P_1[c_1],P_2[c_2]\}}$ is an estimator for $|P_1 \cup P_2|$. (b) The quantity $\frac{nI}{c_1+c_2-nI}$ is an estimator of the Jaccard Distance $J = \frac{|P_1 \cap P_2|}{|P_1 \cup P_2|}$. By multiplying the Jaccard Distance estimator with the union estimator, we get the estimator for the intersection $(|P_1 \cap P_2|)$.

With the *early-out* and the *approximate intersection* optimization, we can intersect huge posting lists by performing a very small number of comparisons. In our implementation, we use a simple linear hash function to randomize the posting lists.

Algorithm 3 Approximate Intersections

```
1:  P1: first randomized posting list
2:  P2: second randomized posting list
3:  M: maximum number of comparisons
4:
5:  nM ← 0; nI ← 0
6:  c1 ← 0; c2 ← 0
7:  gap1 ← P1[|P1| − 1]/|P1|; gap2 ← P2[|P2| − 1]/|P2|
8:  while c1 < |P1| and c2 < |P2| do
9:    nM ← nM + 1
10:   if nM >= M then
11:     break
12:   end if
13:   if P1[c1] = P2[c2] then
14:     c1 ← c1 + 1
15:     c2 ← c2 + 1
16:     nI ← nI + 1
17:     if nI >= I then
18:       break
19:     end if
20:   else if P1[c1] < P2[c2] then
21:     c1' ← c1 + (P2[c2] − P1[c1])/gap1        // attempt uniform skipping
22:     if P1[c1'] <= P2[c2] then
23:       c1 ← c1'                                // successful skipping
24:     else
25:       c1 ← c1 + 1                             // unsuccessful skipping
26:     end if
27:   else
28:     c2' ← c2 + (P1[c1] − P2[c2])/gap2        // attempt uniform skipping
29:     if P2[c2'] <= P1[c1] then
30:       c2 ← c2'                                // successful skipping
31:     else
32:       c2 ← c2 + 1                             // unsuccessful skipping
33:     end if
34:   end if
35: end while
36: if c1 = |P1| or c2 = |P2| then
37:   apIntersect= nI
38: else if nI > 0 then
39:   apIntersect= (nI/(c1+c2−nI))((c1+c2−nI−1)/max{P1[c1],P2[c2]})
40: else
41:   apIntersect= 0
42: end if
43: return apIntersect
```

5.3 Relevant Phrases/Terms

Although dynamic frequent phrases describe sufficiently the content of the hit list, in practice they still contain a large list of stop-word combinations and uninteresting phrases. In order to remove such noise, we define relevant phrases and we propose a technique that attempts to weight appropriately the relevant from the irrelevant phrases.

A *relevant phrase* is defined as a frequent phrase in the hit list that appears disproportionally more often in the hit list w.r.t to the full corpus. For example, consider the query "Jim Gray". Phrases like Gray's email address, affiliation, address, "isolation level" and "transactional consistency" appear much more often in the hit list w.r.t. to the full corpus and identify the content of hit list much better and concisely than the frequent phrases. On the other hand, phrases like "database performance" appear equally often in the hit list as they do in the full corpus and is denoted an *irrelevant* phrase. Relevant phrases describe the hit list in a much more concise and exact way than frequent phrases do. Irrelevant phrases on the other hand offer almost no hit list specific information, since they cannot discriminate among the hit list and the full corpus.

Algorithm 4 summarizes the process of discovering the relevant phrases/terms for each document in our hit list. It's input is the top-k frequent phrases previously discovered, the size $|H|$ of the hit list, the total number $nDocs$ in our corpus and the number l $(< k)$ of most relevant phrases that we want to compute. The algorithm processes all the dynamic frequent phrases that we discovered with

Algorithm 1 and computes a relevance score using the formula in line 10. This formula boosts phrases/terms, that are infrequent in the full corpus but frequent in our hitlist, a similar intuition to TF·IDF. Algorithm 4 is very efficient since it only has to process the top-k (typically a few thousand) frequent phrases discovered in Section 5.2, and is independent of the size of the corpus.

Once the top-l relevant phrases have been discovered, we create another multi-valued dimension by associating each document in the hit list with all the discovered relevant phrases that it contains (line 16).

Algorithm 4 Dynamic Relevant Phrases/Terms

```
1:  Freq: Dynamic Frequent Phrases/Terms from Algorithm 1
2:  H: Hit list
3:  nDocs: total number of documents in the corpus
4:  l: number of relevant phrases/terms
5:
6:  queue: priority queue of ⟨phrase,relevance⟩
7:  for each phrase in Freq do
8:     phrase_f ← frequency of phrase in Freq
9:     phrase_post ← posting list length of phrase
10:    relevance ← (log(phrase_f)(log(|H|)+1)) / ((log(phrase_post)+1) log(nDocs))
11:    insert ⟨phrase,relevance⟩ in queue
12:    if queue has more than l items then
13:       pop item from queue with the smallest relevance
14:    end if
15: end for
16: find the document ids in H for each relevant phrase/term in queue
```

5.4 Importance extraction & Topics Discovery

We further filter and enhance the list of relevant phrases by (a) taking document importance into account and (b) analyzing the occurrences of relevant phrases in the important documents.

Document ranking is extremely important since it helps separate the really important documents from noise documents. An effective way of statically ranking documents that have link information is PageRank [8]. Similarly for dynamically selected documents a better algorithm for ranking is ObjectRank [3]. Such algorithms essentially compute a measure of "centrality" of a document, i.e., how easy it is to reach that document by following a small number of links from another documents. For example, in `DBPubs`, ranking based on citations returns the most cited (seminal) papers. In `MCX` we can incorporate directly such measures as OLAP measures (called importance in the Figures), in order to compute interesting aggregations. In addition to that we can also use the most important papers to further filter the relevant phrases.

Our approach is inspired by Latent Semantic Indexing (LSI [12]), which is a technique in natural language processing for analyzing relationships between a set of documents and the terms that they contain by producing a set of topics related to the documents and terms. LSI uses a sparse term-document matrix, which describes the occurrences of terms in documents. For the weighting of the elements of the matrix we used the typical TF·IDF score.

In our case, we construct a relevant phrase-document matrix using the relevant phrases we discover using Algorithm 4. A full relevant phrase-document matrix however, is presumed too large for our computing resources and our interactive navigation requirements. We only take the top-l (typically 100) relevant phrases and the top-m (typically 400) ranked documents into account while constructing the (sparse) occurrence matrix.

After the construction of the occurrence matrix, we find a low-rank approximation to the relevant phrase-document matrix using Singular Value Decomposition (SVD) [16] in the spirit of LSI.

In our implementation, we use the top-fifteen most important eigenvalues of the SVD, which conceptually return an approximate occurrence matrix for the top-fifteen most important topics in the ranked hit list. Each topic is associated with both documents and relevant phrases/terms. For each such topic, we pick the relevant phrase with the biggest cosine similarity with the documents in that concept as a *representative topic description*.

Once the representative topic descriptions have been discovered, we create another multi-valued dimension by associating each document in the hit list with all the topic descriptions, that the documents contain.

5.5 Hierarchical Dynamic Dimensions

So far, the discovered dimensions have been flat (single-level) dimensions. In this section, we describe how the exact same algorithms are re-used to create *hierarchical* dynamic dimensions. The idea is to create lower levels in the hierarchy for each one level that we've already computed, until the levels correspond to a small number of documents (in our implementation fifteen documents.) For example, consider the documents corresponding to the topic with description "Schema Mapping", which we discovered given the keyword query "xml". We can apply the previously introduced algorithms to discover frequent phrases, relevant phrases, and topics for the documents in this concept, and start creating sub-levels for "Schema Mapping". Although we apply the same algorithms for all discovered topics, we do not compute those levels immediately for performance reasons. Only as the user drills down, we trigger the population of the appropriate dynamic dimensions.

6. EVALUATION

We have conducted a case study by applying the `MCX` framework to build `DBPubs`, a system that allows the dynamic analysis and exploration of research publications. We first present our prototype system `DBPubs`, while in section 6.2, we experimentally evaluate the efficiency and effectiveness of our approach.

6.1 DBPubs - Case Study

`DBPubs` currently stores the content of approximately 30,000 publications from major venues (e.g., sigmod, vldb, tods, vldb j., sigir, pods, stoc) in the field of databases and other adjacent areas. The publications are gathered from the Sigmod Anthology Collection as well as downloaded from the Web. `DBPubs` also contains the associated metadata from the DBLP entries [11], matching citations from CiteSeer [10], and geographical information from the Mondial database [28] (see Figure 8). We completed the citations taken from Citeseer by extracting citations from the publication content [2]. This way, we collected approximately 210,000 citations. We converted the PDF files to text, and created a Lucene text index on both the text and some metadata fields such as *title*, *authors*, *venue*, *year*, *location* of the conference, and so forth.

As described in Section 5.4, we used the citations as a link structure and defined the *importance* of a publication as its computed *static page rank* [8]. (Self-citations were not considered.) Also, we consider *importance* of a publications w.r.t. the query issued as its computed *dynamic rank* using ObjectRank [3].

The system uses the open-source OLAP server Mondrian[1] running in Tomcat as a webapp. Mondrian uses the JPivot[1] framework that renders OLAP tables and charts for interactive reporting. In Mondrian, the MD schema is described as an XML file and comprises multi-cubes, facts, and dimensions. In our case, this file –i.e., the MD– is created on-the-fly once a new keyword query issued.

[1] http://mondrian.pentaho.org/ and http://sourceforge.net/projects/jpivot

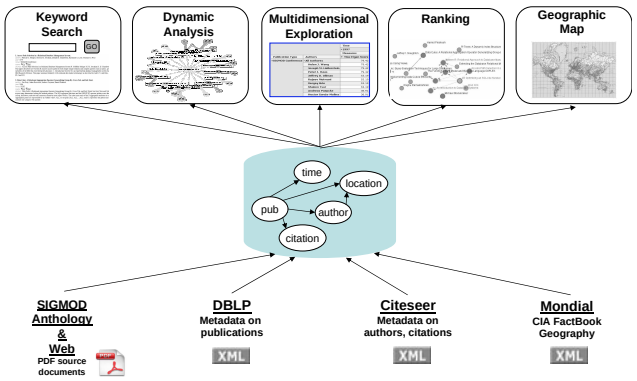


Figure 8: DBPubs

Figure 8 shows the query interface for the end user. The user can submit a keyword query having several options; e.g., fielded search on the metadata, range queries, and so on. As described in Section 4, the metadata of the publications populate static dimensions in the MD, and also measures such as count of citations and importance based on static page rank. As described in Section 5, the publications in the hit list are dynamically analyzed for frequent phrases, relevant phrases, and topics. The results are populated as dynamic dimensions in the MD. Apart from the traditional, frequently-used, count-based measures, we also offer analytical opportunities based on dynamically created measures such as importance –based on dynamic ranking– of authors and papers per keyword query or several citation measures like H-index, impact factor, and so on. Finally, our prototype, offers visualization of the results in a graphical view and exploits location information –e.g., conference venues, affiliations– to annotate appropriately a geographic map.

As example behavior of DBPubs for the query: *+venue:"sigmod" +year:1997*, the system pinpoints that the SIGMOD 1997 paper by J. Hellerstein, P. Haas, and H. Wang on Online Aggregation, which received the 2007 SIGMOD test of time award, was the highest ranked paper. As another example, the system shows, besides many other features, that "Web Browsers", "Document Stores", "OLAP Data Cube", "Data Mining", and "Index and Retrieval" where the top 5 most important topics in 1997.

More information about DBPubs can be found in [2].

6.2 Experiments

We present our extensive performance and qualitative evaluation of the MCX framework using DBPubs as a testbed.

6.2.1 *Efficiency*

This section provides a comprehensive performance evaluation of our system. We used a set of twenty keyword queries depicted below, with hit lists varying from 3% to 75% of the corpus.

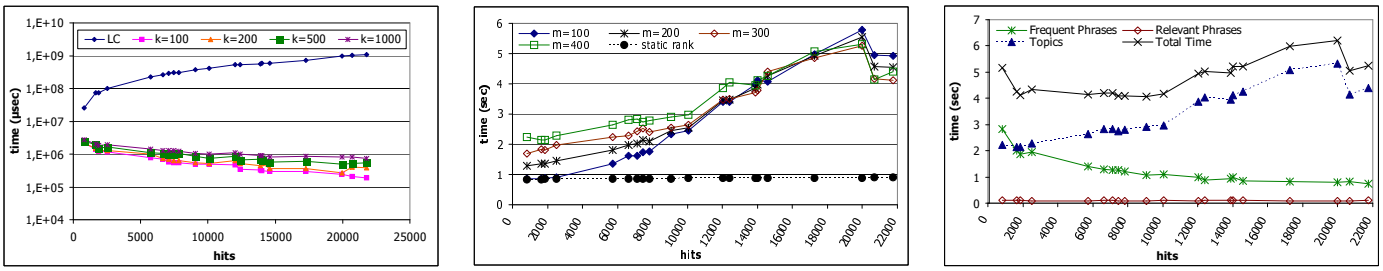
keyword query	hits	keyword query	hits	keyword query	hits
OLAP	810	optimization	7459	management	13987
Garcia Molina	1650	architecture	7824	design	14592
Stonebraker	1867	framework	9073	model	17279
XML	2523	network	10073	research	19964
semantic	5730	query	12025	system	20683
schema mapping	6649	implementation	12388	data	21759
service	7107	database	13882		

Figure 9 presents how the hit list size affects the response time in the various system components. More specifically, Figure 9(a) depicts the Dynamic Frequent Phrase Discovery (DFP) algorithm, that we described in Section 5.2 for discovering top-k frequent phrases for various k values. We compare it to the Lossy Counting (LC) algorithm [26], which has been shown to perform very well in streaming environments for discovering frequent itemsets accurately. The x-axis shows the hit list size, and the y-axis depicts μs response time in log scale. We see that both approaches scale well with respect to the #hits, however DFP is many orders of magnitude faster. We also observe that DFP is affected very slightly by the parameter k. For that reason, we pick $k = 1000$, for the study of the behavior of DFP in the next subsection. LC is not affected at all by k, since it has to keep all frequent phrases above a minimum support always in memory, regardless of the requested top-k. For this experiment, we used $M = 64$ and uniform skipping for DFP.

Figure 9(b) shows how the performance of the topic discovery algorithm is affected by the #hits and for various values of the parameter m (since we use the top-m important publications based for topic discovery.) We experimented with two methods for evaluating the importance (see section 6.1): the static and the dynamic ranking. In the first case, the topic discovery takes almost constant time and it is independent of m, since all publications are ordered a priori by page rank. In the latter case, the topic discovery scales almost linearly with the #hits, although we are using only the top-m documents. The reason is that we have to order on-the-fly the full hit list by the ObjectRank value, and then apply the topic discovery. The small discrepancy in the performance for large hit lists has to do with the query plan our OLAP engine chose for the query.

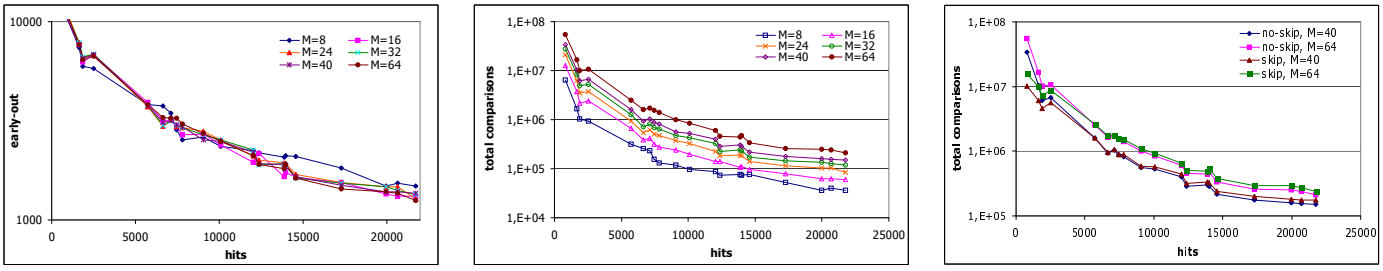
In Figure 9(c), we demonstrate how the system response time is broken down into the various components. The x-axis shows the #hits, and the y-axis shows the total response time in secs. We denote as 'Frequent Phrases', the time the DFP algorithm requires to extract the top-1000 frequent phrases (with M=64 and uniform skipping), 'Relevant Phrases', the time we require to boost the scores for the frequent phrases that appear a lot in the hit list but not in the full corpus, 'Topics', the time for the topic-discovery (here we used the top-$m = 100$ most important publications based on dynamic ranking), and finally 'Total Time' for the total time spent. 'Frequent Phrases' scales extremely well w.r.t to the #hits (as explained below, the early-out kicks in sooner for larger hit lists.) The time for 'Relevant phrases' is constant and very small, since it takes one pass over the top-1000 frequent phrases. 'Topics' scales as explained in Figure 9(b) and becomes the dominant factor for the 'Total Time' due to the ranking technique used. However, this delay is not an inherent problem of our technique, rather it is a bottleneck imposed by ObjectRank. The execution time is significantly improved when the static rank is considered instead. But even in the worst case of dynamic ranking, the 'Total Time' seems to be relative stable, due to the fact that as #hits increases the time needed for 'Frequent Phrases' decreases and balances the delay of 'Topics'. Experimentation with an improved version of ObjectRank or other suitable algorithm is an interesting future challenge for dynamic ranking. Finally, 'Total Time' does not consider the time needed by the text index, since it is negligible.

In Figure 10, we analyze in detail the behavior of the DFP algorithm for discovering the top-1000 most frequent phrases. More precisely, in Figure 10(a) we depict on the y-axis in log scale, the depth until the early-out optimization kicks in w.r.t the #hits on the x-axis in linear scale. We point out, that for all values of M, the early-out optimization kicks in sooner as the #hits increases. The reason is, that as the hit list becomes larger, the distribution of the frequencies of the phrases in the hit list, matches closer the distribution of the global frequent phrases. The latter is already precomputed, and therefore it takes very few posting lists intersections, before the top-1000 list is computed. We observe, that the M parameter affects only slightly the early-out depth, because of ap-



(a) DFP vs. LC for various top-k (b) Topic Discovery Performance (c) Total Query Time Distribution

Figure 9: System Response Time Analysis



(a)Early-out depth vs #hits ($M = 64$) (b) #comparisons vs #hits and M (c) #comparisons vs #hits and skip ($M = 64$)

Figure 10: DFP Performance Analysis for top-1000

proximation errors on the intersection size estimates might trigger the early-out either sooner or later. In Section 6.2.2, we discuss in detail the quality of the frequent phrases that we discover.

In Figure 10(b), we show on the y-axis (log scale) the total #comparisons that it takes for DFP with uniform skipping to return the top-1000 most frequent phrases for varying #hits (on the x-axis, log scale) and M values. It is surprising, at first glance, that the performance of DFP gets better as the hit list gets bigger, and that's true for all M values depicted. The reason has to do with the early-out optimization that is being triggered much sooner for large hit lists, as depicted in Figure 10(a).

Finally, Figure 10(c) depicts the effect of uniform skipping in the performance of DFP. The x-axis shows the #hits, and the y-axis (log scale) the total #comparisons required for discovering the top-1000 most frequent phrases. We observe, that uniform skipping helps a lot (requiring about half the total #iterations without skipping) for relatively small hit lists, but actually hurts for large hit lists. As explained in Section 5, uniform skipping helps, when a small posting list is intersected with a large one, because it jumps over large gaps on the large posting lists, avoiding comparisons with points, that are most probably useless. However, when the posting lists to be intersected have about the same size, then the uniform skipping overshoots quite often, resulting in extra comparisons. This observation leads us to check the sizes of the postings lists in the DFP and avoid uniform skipping, if those sizes are about the same.

6.2.2 *Effectiveness*

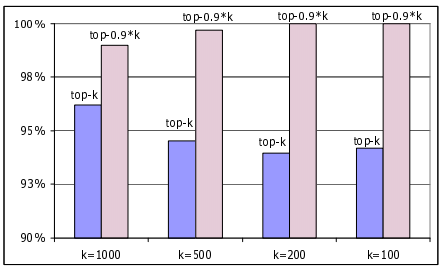
In this section, we describe the quality of the returned results. It is challenging to provide a qualitative evaluation of the end results of a system that incorporates an inherently imprecise keyword-based system. It is even more difficult in `MCX`, since we are dealing with the content of the full hit list. For the dynamic dimensions, we depict how the approximate nature of DFP affects the quality of the recall of the top-k returned frequent phrases. For the topic descriptions, we show actual results from `DBPubs` for keyword queries and publications that the database community is familiar with.

In Figure 11(a), we depict the average recall of the DFP algorithm with $M = 64$ and uniform skipping for various top-k most frequent phrases for the testset of twenty queries. More precisely, we use as a baseline the *exact* top-k frequent phrases, and then we depict the recall on the bar titled "top-k", i.e. how many of those phrases, DFP managed to find. We notice for example, that for $k = 100$, DFP found 95% of all the correct top-k values. Further analysis of the missing phrases revealed, that the tail of the discovered frequent phrases had pretty similar frequencies. Since DFP is an approximate algorithm, the estimated frequencies trigger some frequent phrases in the tail to make it into the top-k, and others not. However, the higher ranked frequent phrases were always there. We show this effect in Figure 11(a) with the bar called "top-$0.9k$". This bar demonstrates what happens, if we ignore a small part of the tail in the top-k returned phrases. For example, if we use DFP with $k = 100$, we see all the top-90 phrases in the result.

Figure 11(b) shows representative sample results from the operation of our system. We picked two representative keyword queries: *xml* and *OLAP*. We depicted the discovered topics in the corresponding "Topics for" column. In the "Hits" columns, we show the number of documents with that topic description. We remind, that a document may be assigned many different topic descriptions. The "Imp" column depicts an *importance measure* for the corresponding group of documents. In this example, we computed the importance of a topic as the *maximum* importance –based on static page rank– among the papers belonging to that group. So, if for example we roll-up in the topic dimension, the "Imp" for 'xml' will be 69.09 and for 'OLAP' 87.18. If we drill-down to a certain topic, e.g., 'Path Index', one or more papers will have "Imp" equals to 60.9, while the others will have lower values. Different aggregate functions can be applied as well; e.g., sum, avg, and min.

7. SUMMARY AND FUTURE WORK

In this paper, we introduce our general framework MCX for effectively analyzing and exploring large amounts of content with



(a) DFP recall

Topics for 'XML'	Hits	Imp	Topics for 'OLAP'	Hits	Imp
Validation against DTDs	66	69.09	Summary Tables and Index	39	87.18
Data Transformations	578	66.53	Materialized Views	25	60.68
Selectivity Estimation for Xml Twig	55	63.33	Attributes Hierarchies	26	51.89
Path Index	132	60.9	Extended Wavelets	8	48.89
Filtering of Xml Documents	32	58.42	Approximate Aggregate Queries	22	48.89
Schema Mapping	73	45.6	XML Queries	8	30.13
Views Query	84	40.25	CPU Caching	7	28.23
Xpath Query on Streaming	111	38.31	Warehouse Updates	17	27.17
Web Services	53	33.68	Mining Task	16	26.13
Query on Compressed Xml	18	28.34	Data Cube	5	14.93
Xquery on SQL Host	71	27.92	Hierarchical Clustering	14	24.01
Labeled Dynamic Xml Tree	33	26.96	Compression Measures	8	21.5

(b) Sample Discovered Topics

Figure 11: Quality Evaluation

limited metadata by combining keyword search with OLAP-style aggregation, navigation, and reporting. We formally describe the organization of the CMS content and metadata in a multidimensional structure, and present our novel algorithms for efficiently discovering and populating the dynamic dimensions. We introduce our case study `DBPubs`, that implements our approach to dynamically analyze and explore database publications.

Currently, `DBPubs` [2] is a working system available to the CS group at IBM's Almaden Research Center. During its operation, we have gathered several fruitful comments and feedback from our colleagues and due to them our system has been improved a lot. Our on-going research involves different ranking schemes for different application domains such as patent data and email archives. For future work, we plan to investigate techniques for visually exploring large amount of data and sharing interesting highlights in slices and dices of the multidimensional dataspace.

8. REFERENCES

[1] A. Anagnostopoulos, A. Z. Broder, and D. Carmel. Sampling search-engine results. *World Wide Web*, 9(4):397–429, 2006.

[2] A. Baid, A. Balmin, H. Hwang, E. Nijkamp, J. Rao, B. Reinwald, A. Simitsis, Y. Sismanis, and F. V. Ham. DBPubs: Multidimensional Exploration of Database Publications. Accepted as a demonstration proposal at VLDB'08.

[3] A. Balmin, V. Hristidis, and Y. Papakonstantinou. Objectrank: Authority-based keyword search in databases. In *VLDB*, pages 564–575, 2004.

[4] N. Bansal and N. Koudas. Blogscope: A system for online analysis of high volume text streams. In *VLDB*, pages 1410–1413, 2007.

[5] K. S. Beyer, D. D. Chamberlin, L. S. Colby, F. Özcan, H. Pirahesh, and Y. Xu. Extending xquery for analytics. In *SIGMOD Conference*, pages 503–514, 2005.

[6] K. S. Beyer, P. J. Haas, B. Reinwald, Y. Sismanis, and R. Gemulla. On synopses for distinct-value estimation under multiset operations. In *SIGMOD Conference*, pages 199–210, 2007.

[7] D. M. Blei and J. D. Lafferty. Dynamic topic models. In *ICML*, pages 113–120, 2006.

[8] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30(1-7):107–117, 1998.

[9] S. Chaudhuri and U. Dayal. An overview of data warehousing and olap technology. *SIGMOD Record*, 26(1):65–74, 1997.

[10] CiteSeer. http://citeseer.ist.psu.edu.

[11] DBLP. http://www.informatik.uni-trier.de/ ley/db.

[12] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by latent semantic analysis. *JASIS*, 41(6):391–407, 1990.

[13] P. DeRose, W. Shen, F. Chen, A. Doan, and R. Ramakrishnan. Building structured web community portals: A top-down, compositional, and incremental approach. In *VLDB*, page 399, 2007.

[14] J. Diederich. Faceted DBLP, http://dblp.l3s.de.

[15] Eventseer. http://eventseer.net.

[16] G. H. Golub and C. Reinsch. Singular value decomposition and least squares solutions. *Numerische Mathematik*, 14(5):403–420, 1970.

[17] 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.

[18] Harzing. *Publish or Perish, http://www.harzing.com/pop.htm*.

[19] T. Hofmann. Probabilistic latent semantic indexing. In *SIGIR*, pages 50–57, 1999.

[20] C. A. Hurtado, C. Gutiérrez, and A. O. Mendelzon. Capturing summarizability with integrity constraints in olap. *ACM Trans. Database Syst.*, 30(3):854–886, 2005.

[21] Y. E. Ioannidis, D. Maier, S. Abiteboul, P. Buneman, S. B. Davidson, E. A. Fox, A. Y. Halevy, C. A. Knoblock, F. Rabitti, H.-J. Schek, and G. Weikum. Digital library information-technology infrastructures. *Int. J. on Digital Libraries*, 5(4):266–274, 2005.

[22] R. Kimball, L. Reeves, M. Ross, and W. Thornthwaite. *The Data Warehouse Lifecycle Toolkit*. Wiley, 1998.

[23] J. Lechtenbörger and G. Vossen. Multidimensional normal forms for data warehouse design. *Inf. Syst.*, 28(5):415–434, 2003.

[24] W. Lehner, J. Albrecht, and H. Wedekind. Normal forms for multidimensional databases. In *SSDBM*, pages 63–72, 1998.

[25] H.-J. Lenz and A. Shoshani. Summarizability in olap and statistical data bases. In *SSDBM*, pages 132–143, 1997.

[26] G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *VLDB*, pages 346–357, 2002.

[27] A. McCallum, K. Nigam, and L. H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *KDD*, pages 169–178, 2000.

[28] Mondial. http://www.dbis.informatik.uni-goettingen.de/mondial.

[29] T. B. Pedersen, C. S. Jensen, and C. E. Dyreson. A foundation for capturing and querying complex multidimensional data. *Inf. Syst.*, 26(5):383–423, 2001.

[30] M. Rafanelli and A. Shoshani. Storm: A statistical object representation model. In *SSDBM*, pages 14–29, 1990.

[31] A. Shukla, P. Deshpande, and J. F. Naughton. Materialized view selection for multi-cube data models. In *EDBT*, page 269, 2000.

[32] D. Takuma and I. Yoshida. Top-n keyword calculation on dynamically selected documents. IBM Research Report, RT-0760, October 2007.

[33] D. S. Whelan. Filenet integrated document management database usage and issues. In *SIGMOD Conference*, page 533, 1998.

[34] P. Wu, Y. Sismanis, and B. Reinwald. Towards keyword-driven analytical processing. In *SIGMOD Conference*, page 617, 2007.