Beyond Basic Faceted Search

Ori Ben-Yitzhak[1] Nadav Golbandi[1] Nadav Har'El[1] Ronny Lempel[2]
Andreas Neumann[3] Shila Ofek-Koifman[1] Dafna Sheinwald[1] Eugene Shekita[4]
Benjamin Sznajder[1] Sivan Yogev[1]

IBM Silicon Valley Lab[3]
555 Bailey Ave.
San Jose, CA 95141

IBM Haifa Research Lab[1]
Haifa 31905, Israel

Yahoo! Research[2*]
Haifa, Israel

IBM Almaden Research Center[4]
650 Harry Road
San Jose, CA 95120

[orib, nadavg, nyh, shila, dafna, benjams, sivany]@il.ibm.com[1]
rlempel@yahoo-inc.com[2] aneuman@us.ibm.com[3] shekita@almaden.ibm.com[4]

ABSTRACT

This paper extends traditional faceted search to support richer information discovery tasks over more complex data models. Our first extension adds flexible, dynamic business intelligence aggregations to the faceted application, enabling users to gain insight into their data that is far richer than just knowing the quantities of documents belonging to each facet. We see this capability as a step toward bringing OLAP capabilities, traditionally supported by databases over relational data, to the domain of free-text queries over metadata-rich content. Our second extension shows how one can efficiently extend a faceted search engine to support correlated facets - a more complex information model in which the values associated with a document across multiple facets are not independent. We show that by reducing the problem to a recently solved tree-indexing scenario, data with correlated facets can be efficiently indexed and retrieved.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

General Terms

Algorithms

Keywords

Search engines, multifaceted search, business intelligence

1. INTRODUCTION

The Web has long since stopped being just a resource of information. More and more transactions are happening online, with the decision process driving users to make these transactions often involving interaction with complex and high-dimensional data. Accordingly, information discovery and e-commerce systems must feature intuitive and easy interaction modes to allow non-experts to explore such data.

Multifaceted search, also known as guided navigation, is a popular and intuitive interaction paradigm for discovery and mining applications that allows users to digest, analyze and navigate through multidimensional data. Faceted search applications are implemented in many Web sites - especially e-commerce sites - and are sold by several software vendors (e.g. Endeca[1], Mercado[2] and IBM[3]). A typical user's interaction with a faceted search interface involves multiple steps in which the user may (1) type or refine a search query, or (2) navigate through multiple, independent facet hierarchies that describe the data by drill-down (refinement) or roll-up (generalization) operations. When certain values across several facets are chosen as the current search context, faceted applications show possible refinements of those facets (categories) to sub-categories, typically along with the number of search results (satisfying both the free-text query and the current facet constraints) present in each sub-category. These counts provide guidance to the user by presenting a quantitative overview on the variety of data available, thereby hinting at the refinement operations that seem most promising for zooming in on the target information need.

Nevertheless, guided navigation interfaces can be greatly improved by providing richer insight into the data. The ability to view flexible and dynamic aggregations over faceted data - as typically found in business intelligence applications over structured data - would allow users to make more informed drill-down and roll-up choices, which in turn will support them in making better decisions.

Another shortcoming of faceted search is that its basic data model, where documents are associated with sets of values across several *independent* facet hierarchies, is too restrictive to model some real-life data. For example, documents that describe products may have non-independent, or *correlated*, facet values associated with them. One example is an article of clothing that comes in both red and blue, and in both large and small sizes - however, the small instance

*Work done while author was at IBM Haifa Research Lab

[1] http://endeca.com/
[2] http://www.mercado.com/
[3] http://www-306.ibm.com/software/data/discovery/content/

WSDM'08, February 11–12, 2008, Palo Alto, California, USA.

comes only in red whereas the large instance comes in both colors. Thus, the "color" and "size" facets are correlated rather than independent.

This paper addresses both of the above shortcomings. Our first contribution extends faceted search applications to return not only counts of result documents across several facets, but also richer aggregations that support better decision making. We proceed to compare and contrast the setting of faceted search over textual data with business intelligence applications in the relational setting. Our second contribution shows how to efficiently index and search documents with correlated facet values by reducing the problem to a recently solved instance of indexing shared content in an ordered tree of documents [5].

While not the focus of this paper, our third contribution is in detailing some of the engineering aspects involved in supporting faceted search. Although many sophisticated implementations of faceted search exist, none seems to have been described in depth in the literature.

The rest of this paper is organized as follows. We survey related work in Section 2. Section 3 describes our reference implementation of basic faceted search, on which we build in the following sections. Section 4 presents our first contribution - an extension of faceted search with business-intelligence aggregations. Section 5 presents our second contribution, of enabling faceted search over a more complex data model in which the values associated with documents across different facets are correlated rather than independent. We conclude in Section 6.

2. RELATED WORK

This section briefly surveys related work in two areas - multifaceted search and on-line analytical processing (OLAP). More on the connection between faceted search and OLAP can be found in Section 4.3.

2.1 Multifaceted Search

The pillars of any faceted collection are (1) the facet hierarchies and (2) the mapping of the documents onto those hierarchies. In certain metadata-rich semi-structured corpora, both may come as an integral part of the corpus. However, in less structured corpora, mapping documents to facet hierarchies (which themselves might need to be constructed) is challenging. In two related papers, Dakka et al. [8, 7] describe algorithms for extraction of facet hierarchies from a corpus based on lexical subsumption, and assignment of the documents to those facets. Stoica et al. [24] use synsets and hypernym relations to accomplish a similar task. Feinstein and Smadja [11] describe the RawSugar social tagging system, which supports tag hierarchies. Those hierarchies, while typically shallow, enable faceted search on the space of tagged documents. Kohlschütter et al. [15] use personalized PageRank values for multiple ODP[4] categories to (1) infer dominant facets in Web search results, and (2) support drill-down operations on the result set.

At the other extreme of the flow, opposite data preparation, lies the faceted user interface. Its purpose is to ease the presentation of a complex, multidimensional information space and to enable intuitive discovery-oriented navigation within the space. Hearst [14] provides observations based on many years of experiments on interface design, most recently as part of UC Berkeley's Flamenco Search Interface project[5]. Visualization of multidimensional information is also the main focus of [3]. There, the described system also calculates scalar statistics of numeric fields over the documents returned by search queries. Schneiderman et al. [23] plot two-dimensional tables with *hieraxes* - axes of hierarchical categories. Each cell of the table contains several counts of documents (corresponding to a few types) using color-coding, thus essentially supplying a 3D table. Meredith and Pieper's inverted index based BETA system [17] also displays two-dimensional tables for correlating pairs of facets.

Typical faceted search applications aggregate counts for all documents that match the query. However, both Anick and Tipirneni [2] and Krellenstein [16] describe implementations where facets are extracted only from the top-ranking result documents.

Ross and Janevski [20] argued that relational algebra is ill-suited to treat faceted hierarchies, and proceeded to define a query language and algebra for querying hierarchically classified data. They showed that their query algebra requires lower time and space complexity than relational algebra, and discussed the differences between the expressive power of the two algebras.

2.2 On Line Analytical Processing

A robust review of OLAP is beyond the scope of this paper. The following brings a brief introduction to OLAP in general and the CUBE operator in particular. For a more comprehensive view of these topics, please refer to the vast data warehousing and data mining literature (e.g. [4]).

The term OLAP - On Line Analytical Processing - was coined by Codd et al. in [6]. The motivation for their work was the inability of relational databases to efficiently and intuitively support analysis of multi-dimensional data at multiple aggregation levels, which are crucial in decision-support systems. They defined 12 criteria for evaluating the fit of OLAP products. The authors envisioned OLAP (almost exclusively) as analysis of historical data rather than incremental, up-to-date data.

The *cube* operator was introduced in [13] for supporting the computation of aggregates needed in OLAP databases. It can logically be thought of as an n-dimensional generalization of the *group-by* operator. In other words, cubing involves computing aggregations that are grouped by all possible combinations of values for a list of dimensions or attributes [1]. Often, the number of tuples of the actually occurring combinations is much less than the cardinality of the cross-product of the attribute domains, resulting in *sparse* cubes [21].

Naturally, query response time is a crucial criterion for interactive OLAP systems. With the number of records in many warehouses measured in billions, extensive preprocessing and clever data structures are needed to ensure that accurate results for complex aggregations are returned in a timely manner. However, one implication of extensive preprocessing is that such systems struggle to keep up with dynamic data, when new records arrive and/or are deleted at high rates. Roussopoulos et al. [22] propose a method for updating cubes in bulks, as they argue that document-at-a-time updates or frequent complete recomputations are impractical. Palpanas et al. [19] present a method for up-

[4] `http://www.dmoz.org/`

[5] See the Flamenco site `http://flamenco.berkeley.edu/index.html` for references to many additional publications.

dating cubes in the presence of non-distributive aggregate functions, and contain many references to additional works on updating cubes. Fagin et al. [9, 10] introduced a theoretical framework for solving optimization problems in a data model they called a *multi-structural database* (MSDB). Roughly speaking, whereas traditional OLAP queries ask for certain aggregations to be computed on specified regions of a cube, Fagin et al. defined problems whose solutions require the identification of regions in the MSDB that optimize certain objective functions.

3. IMPLEMENTATION OF BASIC FACETED SEARCH

While faceted search applications have become very popular over the last few years, we are not aware of papers describing the indexing and runtime internals of faceted search engines. Furthermore, the implementation in the *Solr*[6] open source project deals with "flat" and non-hierarchical facets. We thus describe our reference implementation, commenting along the way on requirements we had, tradeoffs we faced, and design choices we made. As this is not the main focus of the paper, the discussion is kept at a high level, leaving out some low-level details.

3.1 Lucene Background

Our implementation is written in Java, on top of Apache Lucene[7], a popular open-source search library. Lucene indexes *Document*s, each being a collection of *Field*s. A field has a name and some associated text.

Lucene maintains a logical *inverted index* per field. The index holds for each term (word) a *postings list* — a list of document identifiers and word offsets within those documents in which this term occurs[8]. Furthermore, for each occurrence, an arbitrary *payload* can be stored - a byte array that encodes extra information and that is accessible at runtime. Each triplet of ⟨docID, offset, payload⟩ is called a *postings element*. During search, Lucene uses these posting lists to quickly iterate over all documents matching the search criterion (*hits*) and to evaluate the *relevance* of each hit to the query. Finally, the most relevant documents are returned.

Faceted search enablement requires some additional processing for each matching document - namely, adding its contribution to its associated facets. Lucene makes it easy to plug such functionality into its iteration over the hits, since it can call a *hit collector* of our choice for each matching document it encounters.

3.2 Data Model and Document Ingestion

The basic requirement when ingesting documents in a faceted corpus is the ability to associate facets with each document in the collection, and to have a coherent view of the *taxonomy* - the hierarchical relationships among those facets. The *taxonomy* is generally a DAG (directed acyclic graph) whose nodes represent facets and whose directed edges denote the refinement relations between them.

There are two approaches to ingesting documents in faceted collections: the search engine can either be given the full taxonomy before indexing, or learn it, while indexing, from the ingested documents. In the first approach, documents must specify the taxonomy nodes with which they are associated, while in the second approach documents specify the taxonomy *paths* to which they correspond. With either approach, the application that owns and indexes the data must add taxonomy nodes or *facet paths* to each document prior to adding it to the index. Note that there are many works that describe the mapping of documents into facet hierarchies, e.g. [8, 24, 7] - however, this aspect of preparing the information to be indexed is upstream of the indexing process we describe and is beyond the scope of this paper.

Our implementation uses the second ingestion approach. We implicitly infer the facet hierarchy from the plurality of paths encountered when indexing the individual documents. We essentially collect all encountered facet-paths to a forest-like graph, where each tree corresponds to a dimension, or top-level facet. This approach allows for easy and natural evolution of the facet hierarchies - as new documents are ingested, new facet paths may be seamlessly introduced without the need for any administrative action. Rather, our inferred taxonomy will automatically expand to accommodate the new data.

Technically, applications associate facet paths with documents by adding to each Lucene Document object, prior to indexing it, a specially-named field F. We will denote by F_d the field F associated with document d. Assume now that a document d is added to the index, with field F_d containing the set of k facet paths $P_1, P_2, \ldots, P_k$ associated with document d. Our processing of field F performs the following operations for each path $P_i = v_1^i/v_2^i/\ldots/v_{\ell_i}^i$ of length ℓ_i:

1. Creates postings elements for document d for each prefix of P_i. This is equivalent to adding the following ℓ_i tokens (terms) to document d:

   $$v_1^i \ , \ v_1^i/v_2^i \ , \ v_1^i/v_2^i/v_3^i \ , \ \ldots \ , \ v_1^i/v_2^i/\ldots/v_{\ell_i}^i.$$

   This, in turn, will cause document d to appear in the postings lists associated with these path prefixes. These postings lists are used when evaluating queries that are restricted to certain facet (path prefix) values.

2. Adds each path to a *taxonomy index* - a structure that maintains an internal representation of the taxonomy as perceived from the documents ingested so far, and in particular (1) assigns an ordinal number n to each distinct path prefix, and (2) maintains a function f from the ordinal of each path to that of its *father path*[9].

3. Encodes the k paths associated with the document and stores the encoding as the payload of a special *FacetInfo* term associated with d. Specifically, each path P is encoded by the integer $n(P)$, i.e. by the ordinal number of the path. The integers corresponding to the k paths of document d are then written in the payload of d's *FacetInfo* term.

At the end of this process for all documents, we have:

- Postings lists for each path prefix P, listing all documents that are associated with some path Q such that P is a prefix of Q.
- A taxonomy index that contains the facet hierarchy seen so far and that maintains the function f.

[6] `http://incubator.apache.org/solr/`

[7] `http://lucene.apache.org/java/`

[8] Offsets are needed for phrase matching, for example.

[9] The father path of $v_1/v_2/\ldots/v_{\ell-1}/v_\ell$ is $v_1/v_2/\ldots/v_{\ell-1}$.

- A special postings list for the term *FacetInfo* that contains, for each document d, a posting element whose payload contains a list of integers corresponding to the facet paths associated with d.

Note that since all our operations are performed on path prefixes rather than on individual node labels, the resulting taxonomy is a forest of trees rather than a general DAG. Figure 1 exemplifies the output of the indexing process after ingesting two documents *doc*1 and *doc*2, each associated with the facet paths shown in Table 1. It shows the facet forest maintained by the taxonomy index, the ordinal number n of each path, the function f mapping paths to father paths, the inverted index and the payloads of the *FacetInfo* postings list.

3.3 Runtime

Before describing our implementation of query evaluation in a multifaceted setting, we first describe the input and output - the notions of *faceted query* and *faceted result set*.

The *faceted query* is a query string coupled with a set of subtrees in the taxonomy, for which facet counting is required. The *faceted result set* for this query is a ranked list of documents matching the query, along with a set of counters for all the nodes in the facet subtrees specified by the query. Formally, we define a faceted query as a tuple $FQ = (qc, TF)$ as follows:

1. A free-text query qc, containing *constraints* that define the set of documents to be returned in ranked order by the search engine. This portion may (and usually does) contain category/facet constraints that limit the documents to those residing in certain subtrees of the facet hierarchy.

2. A set of *target facets* $TF = \{tf_1, tf_2, \ldots, tf_k\}$, where each target facet $tf_i = (P_i, n_i)$ is composed of a target path P and a depth level $n \geq 1$.

The expected output, a faceted result set is also a tuple $FRS = (D_{qc}, \{FS_1, \ldots, FS_k\})$ where D_{qc} is a ranked list of documents satisfying the constraints qc, and where FS_i is the *facet set* corresponding to tf_i. In particular,

$$FS_i = \{(Q_1^i, c_1^i), (Q_2^i, c_2^i), \ldots, (Q_{m_i}^i, c_{m_i}^i)\}$$

where:

- Each taxonomy path Q_j^i satisfies the following:

  $$P_i \text{ is a prefix of } Q_j^i \text{ and } |P_i| + n_i \geq |Q_j^i|$$

  In other words, the taxonomy node reached by the path Q_j^i is in the subtree of depth n_i rooted at the taxonomy node reached by the path P_i.

- c_j^i is the count of the number of documents in D_{qc} that are associated with the path Q_j^i. With a slight abuse of notation, $c_j^i = |D_{qc} \cap Q_j^i|$.

Table 1: Facet paths associated with *doc*1 and *doc*2

*doc*1 facets	*doc*2 facets
clothing/children's/coats	clothing/women's/accessories
clothing/winter/coats	clothing/all seasons
price/30-40/30-35	price/30-40/36-40
color/blue	color/red

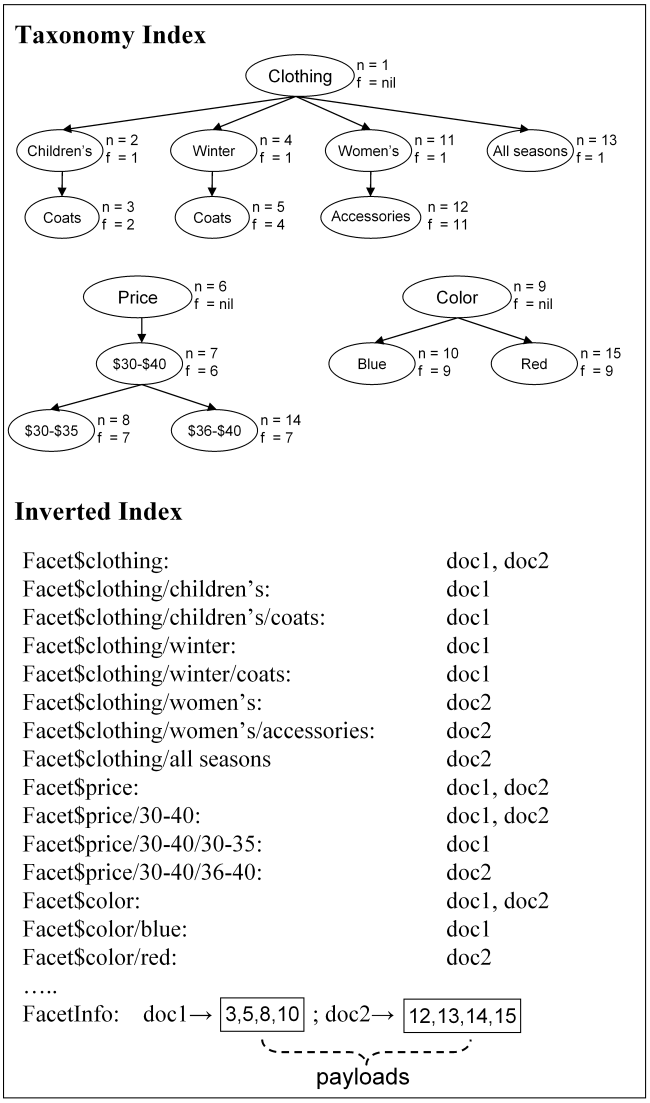

Figure 1: Indexing process byproducts

Note that often, only the paths associated with the highest counts in each facet set are actually returned and displayed to the user. This is similar to returning only the most relevant documents in D_{qc}.

Given a faceted query (qc, TF), the standard Lucene query engine will determine (and rank) the set of documents D_{qc} corresponding to qc. For each document $d \in D_{qc}$, Lucene then calls our modified hits collector which matches the target facets specified in TF against the facets associated with d as contained in the payload of the *FacetInfo* postings list. Since the payload contains the ordinal numbers corresponding to the full facet paths associated with d, we use the Taxonomy Index' f-function to determine the ordinal number of each path prefix, and then check which of d's facet path prefixes fall within the scope of any target facet. We create and increment counters for each such qualifying path, taking into account also the depth constraints. The set of qualifying paths that are encountered per each target facet tf_i constitutes the corresponding Facet Set FS_i that will be part of the faceted result set.

An alternative approach to facet counting is to (1) use the Taxonomy Index to enumerate the set of facet paths that fall within the scope of each target facet, and (2) intersect the postings list corresponding to each such path with D_{qc}. We preferred our design of consulting the single (large) *FacetInfo* postings list that encodes all facet information pertaining to all documents since it is more suited for broad facet queries, in which the target facets define large subtrees. Note that the alternative must consult a separate postings list for each taxonomy node within the subtrees defined by the target facets, implying more random I/O. A full comparison of the two approaches is beyond the scope of this paper.

4. EXTENDING MULTIFACETED SEARCH TO BUSINESS INTELLIGENCE

Business Intelligence is defined in Wikipedia[10] as "...a broad category of applications and technologies for gathering, providing access to, and analyzing data for the purpose of helping enterprise users make better business decisions."

As discussed in the Introduction, traditional faceted search applications provide an attractive interface for exploring multidimensional data. In particular, such applications provide the number of documents found in all (or some) refinements of a set of target facets, and the combination of the refinements and counts may guide the user toward satisfying the information need. However, guiding a search session based on document counts alone is simplistic. In many scenarios, other hints rising from the data - and in particular, other views into the data per sub-category of each target facet - would provide users with much better hints for refining their searches. In other words, qualitative information may ease search and discovery tasks more than quantitative information. For example, when shopping for books and looking to refine the search by author, it might make more sense to drill down to the author who wrote the most best-sellers or who sold the most overall copies, instead of focusing on the author who wrote the most (perhaps mediocre) books. Likewise, when analyzing a corpus of contract documents and trying to comprehend the strength of the business across European countries, it might make more sense to drill down to the country that has the largest average contract value, instead of drilling down to the country in which more contracts exist, but are perhaps of lower value.

One such source of qualitative information are aggregations (per sub category) of arithmetic and Boolean functions over numeric meta-data attributes that are associated with the documents. Figure 2 shows an example of such aggregations when searching for "world wide web" over a subset of Amazon's book catalog[11], as crawled and indexed by us. For example, instead of displaying the authors that wrote the most query-matching books, we populated the "Author" facet with the authors that wrote the most best selling books on the topic. We also display the average price of each author's books. The "Rating" facet is sorted by the score of the most relevant book (as determined by Lucene's ranking function) for each rating level; this allows us to examine the correlation between relevance scores and the public's rating of the books. The "Binding" facet is sorted by the traditional document counts; however, we display the average weight per page for the books of each binding, discovering and confirming the (rather intuitive) fact that paperbacks are generally lighter than hardcover books.

4.1 Implementation Details

We extend the multifaceted search model presented in Section 3 by allowing a faceted query to specify any number of aggregation expressions that are to be calculated per target facet, returning the values of these aggregations for each path of the corresponding facet set in the result set. Formally, a target facet becomes now a triplet $tf_i = (P_i, n_i, E_i)$ where $E_i = \{e_1^i, e_2^i, \ldots, e_{x_i}^i\}$ is a set of x_i expressions. The facet set corresponding to tf_i becomes

$$FS_i = \{(Q_1^i, c_1^i, AV_1^i), (Q_2^i, c_2^i, AV_2^i), \ldots, (Q_{m_i}^i, c_{m_i}^i, AV_{m_i}^i)\}$$

where for all $j = 1, \ldots, m_i$, AV_j^i is an ordered set of x_i aggregated values, and for all $k = 1, \ldots, x_i$, $AV_j^i[k]$ is the aggregated value of expression e_k^i, as calculated over all the documents in $D_{qc} \cap Q_j^i$. Essentially, each path within the scope of target facet tf_i now maintains x_i expression accumulators in addition to the counter.

The expressions we support involve arithmetic, relational and Boolean operators over numeric constants and numeric attributes and fields associated with each document (e.g. price, weight etc. in the Amazon book catalog). As in the C/C++ programming languages, each Boolean expression resolves to 1 if the predicate is *true* and to 0 if it is *false*. The set of expression values for each matching document are aggregated by one of four functions - *min, max, average* and *sum*, which are all *algebraic* as defined in [13] and require $\mathcal{O}(1)$ memory at each accumulator (and $\mathcal{O}(1)$ update time as well). These four aggregation functions, along with the *count* aggregation, are exactly the five functions for aggregating table values defined by the SQL standard from 1992 [12]. We currently do not support *holistic* aggregators [13] (e.g. median) that would require us to store $\Omega(1)$ values in memory.

Furthermore, we consider Lucene's relevance score for document d as a dynamic (query dependent) pseudo numeric field of d, and allow it to take part in the expressions. Thus, expressions may calculate the average relevance of documents in a certain facet path to the query, or the number of documents per path whose relevance is above a certain threshold. Some example expressions, and in particular those used in Figure 2, are shown in Table 2.

Since a single query may involve several target facets, each with their own list of expressions, and thus may require access to numerous numeric fields, we decided to mirror the design choice we made for the set of facet paths associated with each document and added all numeric fields associated with a document to the payload of its *FacetInfo* token. Thus, the *FacetInfo* entry corresponding to document d now contains two parts - the list of facet paths associated with d as described in Section 3, and a list of name-value pairs for all numeric fields and attributes associated with d. Each name-value pair is encoded in 12 bytes - 4 bytes encode the 32-bit hash value of the numeric field's name, and 8 bytes encode the field's value (in double precision).

During query evaluation, when our hits collector is called per document d with its *FacetInfo* payload, we identify the facet paths associated with d that are contained within any

[10] `http://en.wikipedia.org/wiki/Business_Intelligence`

[11] `http://aws.amazon.com`

[12] IS 9075 International Standard for Database Language SQL 1992

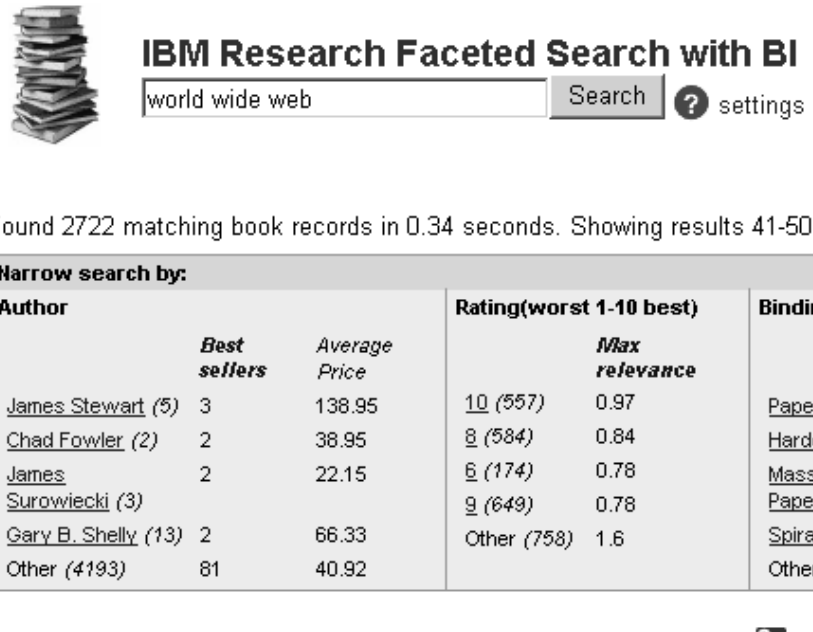

Figure 2: Faceted search with Business Intelligence aggregations

Table 2: Examples of supported business intelligence aggregation expressions

Count	$sum\{1\}$	Count the number of matching documents
Best sellers	$sum\{salesrank < 5000\}$	Count the number of results among the 5000 top-selling books in Amazon
Average price	$avg\{price\}$	Calculate the average price of the matching books
Max relevance	$max\{relevance\}$	Return the highest relevance score assigned by Lucene to a matching book
Grams per page	$avg\{grams/pages\}$	Calculate the average weight (in grams) per book page

taxonomy subtree defined by a target facet. For each such pair of path P and containing target facet tf, we compute the values of the expressions associated with tf using d's numeric fields, and aggregate the values in the set of accumulators (one per expression) associated with path P.

Note that aggregations similar in spirit to those described above are performed in FAST Data Search [3]. There, however, a single set of aggregations are computed for the entire result set and not per each refinement of each facet. Thus, while summary information is made available to the user, hints for guiding the navigation process further are not.

4.2 Dynamic Facets

Typical faceted search applications operate over a set of (predetermined) indexed facets, i.e. the facets and attributes associated with each document must be known at indexing time. One such attribute might be the date of a document, which then trivially supports facets such as years, months and days. However, applications that want to present information such as "documents dated in the last day/week" as facets require more complex machinery, as the mapping of documents to these facets is not known at indexing time and can only be determined at the time when the query is submitted. We refer to such facets as *dynamic facets*.

The ability to sum Boolean expressions as described above allows an application to easily support dynamic time-based facets as follows: assume each document's date (denoted by *doc.time*) is indexed as a numeric field encoding the number of seconds since January 1, 1970. Also assume that the submission time of the query, denoted by *qtime*, is given in the same units. Then, the following expression will count the number of matching documents that are dated within the last week, as the Boolean condition will evaluate to 1 only for those recent documents:

$$\text{Sum } \{\text{qtime-doc.time} \leq 60 * 60 * 24 * 7\}$$

In a similar fashion, one can support the categorization of search results into spatial dynamic facets, e.g. count the

number of results in certain radii around a location that is specified by the query. This requires the indexing of lat/long coordinates per document, and might present an easier option than integrating the search application with a separate GIS system.

4.3 Contrasting our approach with OLAP

Supporting dynamic aggregation expressions in faceted search applications naturally enhances the analyzability of the indexed multi-dimensional data, and provides richer insight into the data than is provided by document counts alone. In particular, our implementation provides the ability to view the distribution of expression values across the possible refinements of a facet, which may sometimes be the information task itself. In this sense, the functionality we provide is a form of hierarchical OLAP (see Section 2.2) over free-text queries on semi-structured data, with the aggregation expressions corresponding to OLAP *calculated measures*. Furthermore, OLAP cubes can be thought of as providing aggregations on Cartesian products of facets.

From the description of our implementation in Section 4.1 it is clear that the complexity of computing k aggregation expressions for a query whose result set is of size n is $O(nk)$, as each expression requires $O(1)$ work per matching document. Thus, high-recall queries would run slower than low-recall queries. OLAP cubes, on the other hand, typically require a constant time per operation, thus computing k measures per query would typically require $O(k)$ time regardless of the query's recall. This brings up the question of whether our approach is valid, or are we using the wrong tool - an enhanced free-text search library - to solve a problem that seems to be efficiently addressed with data warehousing technology [4]. In the following paragraphs, we discuss several features of our implementation that are not easily realized with traditional cubes.

First and foremost, we support aggregations on results of free-text search queries. There is practically an infinite space of such queries - even single term queries define way too many values for any cube to consider.

Second, our approach supports dynamic corpora, relying on the incremental indexing capabilities of the underlying search engine. Lucene, like most engines, supports instantaneous document deletions, and has some small latency when indexing new documents (several minutes may pass until new documents become *searchable*). The Taxonomy Index is incremental as well, and dynamically grows the facet hierarchy as new facet paths are discovered in new documents. In contrast, updating OLAP cubes is expensive. For example, precomputed aggregate functions such as *min* and *max* are difficult to maintain when data is deleted [19]. In a sense, dynamic corpora represent a similar challenge to cubes that free-text queries do - the set of records on which to compute the cube cannot be determined prior to receiving a specific query at a specific point in time.

Third, we support dynamic expressions that need not be predetermined at indexing time. Only the numeric fields over which the expressions are formulated must be indexed. The expressions themselves, and their association with the target facets, are entirely dynamic. In contrast, the measures computed in OLAP cubes per dimension must be known in advance, at cube computation time.

Fourth, faceted applications naturally allow for a document to be associated with multiple subcategories of the same top-level facet, i.e. documents may have multiple values in the same dimension. For example, a film might belong to both genres *"romance"* and *"comedy"*. OLAP does not easily support such multi-value flexibility.

Fifth, by combining the search engine's relevance score into the calculated expressions, we can present the refinements of each facet in relevance order, as well as show only the most relevant subset of those options. This is particularly useful when the number of possible refinements is larger than what can be easily visualized by users, i.e. when the fan-out of a node in the hierarchy of facets is large. No similar notion of relevance exists in traditional OLAP.

Ultimately, we predict that future BI applications requiring the support of free-text queries will involve hybrid strategies, where the nature of both the application and each specific query will determine the mix of cubing operations and runtime computations that is most suitable. For example, in a case where an initial free-text query is performed, and then many drill-down/roll-up operations are done on its result set, one might compute an ad-hoc cube for that result set and use that cube for the subsequent operations. This was proposed (in a different context) in [18], where cubes are built on-demand, over small subsets of data that are defined by users' MDX[13] queries. On the other hand, when for most result sets, the number of expected navigation operations is relatively small, or the measures are dynamic and change often, or the data is highly incremental - cubing may not be economical since each cube can only be reused a small number of times.

5. CORRELATED FACETS

A key characteristic of the standard faceted search data model is that each document has a certain set of facet values, e.g. a product (represented by a document) will have a certain color, size, and price. It could be the case that a product comes in a variety of colors (e.g. both in red and in blue) and sizes (e.g. small, medium and large). However, the standard model then implies that the product is essentially available in all combinations of these colors and sizes, i.e. in the cross-product {color/red, color/blue} × {size/small, size/medium, size/large}. One can view the attributes of the document as being independent across the facets in the sense that any value of one facet can co-exist with any value of another facet.

In many e-commerce situations, the above independence of facets does not model the data correctly. Returning to our example of colors, sizes and prices, it may be the case that a product only comes in red for the small size, only in blue for the large size, and in both colors for the medium size. Also, it could be that the large size is more expensive than the small and medium sizes. To complicate things further, it could be that the price of the product also depends on the particular store where the product is sold (e.g. buying a certain instance of the product in New York may be cheaper than buying the same product in San Jose). With this behavior, if a document has sets of values V_1, V_2, V_3 across three facets, only certain tuples of the cross product $V_1 \times V_2 \times V_3$ will co-exist, implying that the product should only be returned (and counted) for queries that select those correlated tuples of values across the multiple facets.

[13] `http://msdn2.microsoft.com/en-us/library/ms145514.aspx`

Table 3: Correlated Facets e-Commerce Example

Product	Instance	Manufacturer	Type	Model	Color	Size	Store	Price
1	1	Arthur's Sports	Running Shorts	Excalibur	red, blue	small	New York	$20
1	2	Arthur's Sports	Running Shorts	Excalibur	red, blue	small	San Jose	$15
1	3	Arthur's Sports	Running Shorts	Excalibur	red, black	medium	New York	$20
1	4	Arthur's Sports	Running Shorts	Excalibur	red, black	medium	San Jose	$15
1	5	Arthur's Sports	Running Shorts	Excalibur	red, green	large	New York	$22
1	6	Arthur's Sports	Running Shorts	Excalibur	red, green	large	San Jose	$20
2	1	Arthur's Sports	Running Shorts	Lancelot	red, blue	small, medium	New York	$17
2	2	Arthur's Sports	Running Shorts	Lancelot	black, green	large	San Jose	$17
3	1	Arthur's Sports	Running Shorts	Galahad	blue	small	San Jose	$10
3	2	Arthur's Sports	Running Shorts	Galahad	black, white	medium, large	San Jose	$12

Table 4: Facet counts for the query "running shorts"

Color	Size	Price	Location
red(2)	Small(3)	Below $15 (1)	San Jose (3)
blue(3)	Medium(3)	$15-$20 (2)	New York (2)
black(3)	Large(3)	Over $20 (1)	
green(2)			
white(1)			

Table 5: Counts across facets for the query "running shorts Size:large"

Color	Price	Location
red(1)	Below $15 (1)	San Jose (3)
black(2)	$15-$20 (2)	New York (1)
green(2)	Over $20 (1)	
white(1)		

Table 6: Counts across facets for the query "running shorts Size:large Color:black"

Price	Location
Below $15 (1)	San Jose (2)
$15-$20 (1)	

Table 3 contains an example of three products (fictitious running shorts), which can be found in various sizes, colors and prices in two different stores. As Table 3 shows, different products in the same corpus might behave differently in terms of what instantiations they may have. For example, while the Excalibur model has different sets of colors per size and different prices in each store, the Lancelot product has a fixed price - but different sizes are sold in different stores. The Galahad model is sold in a single store, with the small articles costing less than the medium and large ones.

E-commerce sites with a faceted search interface typically display and count results at the product level. Hence, for the data above, the query "running shorts" should return three results corresponding to the three models, with the histogram of colors, sizes, prices and locations as shown in Table 4. For example, all three models have at least one instance that is sold in San Jose, while only two of the models (Excalibur and Lancelot) can be found in New York.

Once drilling down into "Size:large" items, there are still three returned results (each model comes also in large) but the counts across the other facets change as indicated in Table 5. For example, although there are two instances of red large shorts, both are of the Excalibur model and so the count for "red" at the product level is 1. We see that although the size of the result set doesn't change in terms of documents (products), it does change in terms of the facet counts. Proceeding with the example, drilling further down into "Color:black" will yield only two matching products (large Excalibur shorts do not ship in black), with the counts for price and store location as shown in Table 6.

The straightforward way to model data with correlated facets as in Table 3 is by creating one document per row, indexing the facet values across all columns. In the example above, that would mean indexing 10 documents, each with a product number and values across the 7 facets "Manufacturer", "Type", "Model", "Color", "Size", "Store" and "Price". Such documents respect the independence of facet values, namely if a document has facet values $V_1, V_2, \ldots, V_k$, all tuples of the cross product $V_1 \times V_2 \times \ldots \times V_k$ are valid[14]. However, this representation suffers from two shortcomings:

1. Large index size - attributes that are common to many product instances will need to be repeatedly indexed for each instance. Note that in general, each product will also have some marketing text associated with it, and enabling free-text search on such text requires it to be reindexed for each instance of the product.

2. Difficulty in aggregating counts at the product level. Returning to the example above, the three products are representable by 10 documents (product instances). The query "color:red" is satisfied by 7 of those instances - 6 stemming from the Excalibur model and one stemming from the Lancelot model. Some "group by" mechanism is needed to ensure that facet values found in multiple instances of a single product are tallied up correctly, i.e. counted only once.

These two issues can be resolved by adopting a tree view of the data - we model each product as a tree, in which the leaves represent specific instantiations, and where the attributes corresponding to each leaf are the union of attributes on the unique path from the root of the tree to the leaf. In other words, each node of the tree shares its attributes (text and associated metadata) with all its descendants. When we factor out common attributes of leaf nodes to intermediate nodes, this representation avoids significant duplication of text and metadata that are common

[14]Note that this representation resembles an OLAP *fact table*, with the difference being that we allow columns to hold multiple values (e.g. two colors or sizes) per row.

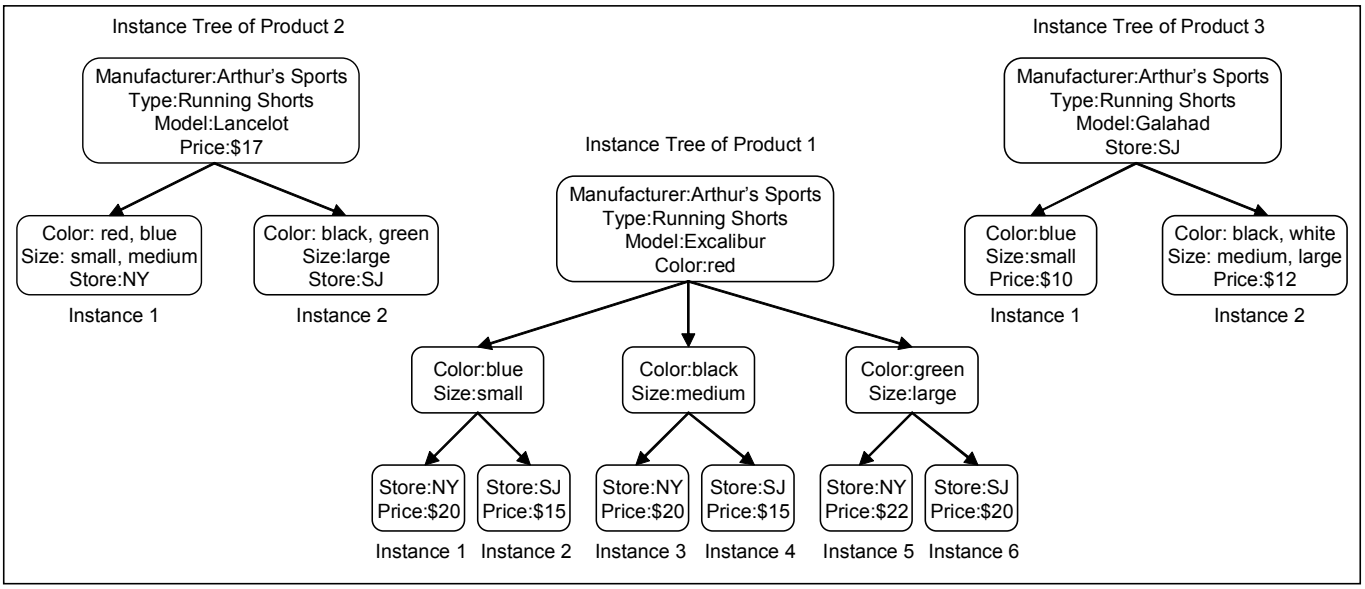

Figure 3: Tree representation of the three products and 10 instances

to many variations of each product. An example of the tree representation of the data in Table 3 is given in Figure 3.

A method for efficiently indexing such document trees using inverted indices was recently presented in [5]. Given a tree, the method indexes each node as a document in the inverted index. The nodes are indexed such that all content that is associated with a node is logically shared with all its descendants in the tree, e.g. tokens that are associated with the root of a tree are indexed once but are attributed to all the nodes of the tree. Using this method, the three trees in Figure 3 are indexed using 16 "virtual" documents, with the 10 documents in the tree leaves representing actual product instances. The text and set of attributes logically associated with each product instance are the union of the leaf's text and attributes with the text and attributes of all of its ancestors. In the running example of Table 3 and Figure 3, the number of indexed facet values is reduced from 81 in the straight-forward approach (the number of values in columns "Manufacturer" through "Price" of Table 3) to 47 in the tree approach as seen in Figure 3. When each product also contains some general textual description (e.g. marketing content that pertains to all instances), modeling each product as a tree yields even more dramatic space savings since that text is indexed only for the root of the tree rather than being reindexed for each instance of the product.

In addition to greatly reducing the space required to index the various instantiations of a set of products as compared with the straightforward approach, the search algorithm of [5] ensures that result leaves corresponding to the same tree will be enumerated contiguously. This enables, with relative ease, to correctly count facets at the product (tree) level without performing "group by" or other post-processing, as the search algorithm natively groups its result documents by tree.

We note that there are many possible tree representations for a set of documents, depending on the order by which facet values are factored out along the various paths from the root to the leaves. The study of the associated optimization problem that receives as input a set of documents with associated facet values and computes their most compact tree representation is beyond the scope of this paper.

6. CONCLUSIONS AND FUTURE WORK

This paper extended traditional faceted search to support richer information discovery tasks over more complex data models. We started out by describing a reference implementation of a middleware for faceted search, built upon a standard search library (Apache Lucene). Although not the focus of the paper, this constitutes our first contribution since while faceted search has been around for several years, the literature is lacking descriptions of the engineering details required to efficiently support it.

We then described two extensions to the basic faceted search paradigm. Our first extension adds flexible, dynamic business intelligence aggregations to the faceted application, enabling users to gain insight into their data that is far richer than just knowing the quantities of documents belonging to each facet. We see this capability as a step toward bringing some OLAP capabilities, traditionally supported by databases over structured data, to the world of free-text queries over semi-structured or metadata-rich data. Our second extension shows how one can efficiently extend a faceted search engine to support correlated facets - a more complex information model in which, for a document associated with value sets $V_1, V_2, \ldots, V_k$ across k facets, only a subset of the tuples in $V_1 \times V_2 \times \ldots \times V_k$ actually co-exist. After presenting the difficulty and inefficiency of handling correlated facets within the standard framework of faceted search, we reduced this problem to a tree-indexing scenario, which was recently shown to be efficiently indexable and searchable.

The following directions are left for future work. First, one basic and attractive feature of OLAP cubes is the ability to perform aggregations for cross products of dimensions. Current faceted search applications still show refinements of

single facets, i.e. return several count arrays that each detail the distribution of documents across the possible refinements of one dimension. Essentially, today's faceted search applications return a set of "one-dimensional cubes". The next step is to extend faceted search engines to return counts and general aggregations over any cross product of facets, i.e. a set of multi-dimensional cubes, built from documents satisfying a combination of free-text and hierarchical constraints. This would greatly advance the reach of business intelligence applications over semi-structured data. In this context, note that [17, 23] already return two-dimensional count tables over searched documents.

Second, we discussed in Section 4 how we can aggregate general and dynamic arithmetic expressions per facet. A next related step is adding the ability to restrict and sort search results by the value of such expressions. While restricting results and sorting them based on numeric fields of documents is already supported by many search engines [12], we are not aware of engines that retrieve or sort results based on arithmetic expressions involving multiple fields.

Third, we intend to study additional complex data models that exist in e-commerce, such as those where shoppers may search for a product that cannot be purchased alone but rather only in combination with other products (e.g. a certain tie may be available only in combination with a suit). The many-to-many mapping between individual products and the merchandising combinations that contain them presents many algorithmic challenges and time-space tradeoffs to inverted-index based search engines.

Fourth, we intend to study efficient means for incrementally updating facet values and numeric attributes of documents. Typically, updating a document in an inverted index involves deleting its former version and reindexing it in its current form. However, metadata such as facet values and numeric attributes might be amenable to indexing and storage models that can be efficiently updated "in situ".

Finally, many interesting research questions arise when considering faceted search across a distributed index. For example, at indexing time, we can either maintain a unified and global view of the taxonomy implied by the documents indexed across all nodes, or have each node maintain a local view of the taxonomy and reconcile those views at query evaluation time. Another issue is adapting distributed query evaluation schemes to not only identify the top-k documents matching a query, but to also efficiently compute and return the facet paths of highest count for each target facet, where counts of paths are accumulated across all nodes.

7. ACKNOWLEDGEMENTS

We thank fellow IBMers John Bosma, Andrew Chasin, Cliff Leung, Michael McCandeles, John McPherson and Mike Moran for many useful discussions over the course of this work. We also thank former IBMers Andrei Broder, Nadav Eiron, Marcus Fontoura and Runping Qi for early discussions on this topic.

8. REFERENCES

[1] Sameet Agarwal, Rakesh Agrawal, Prasad M. Deshpande, Ashish Gupta, Jeffrey F. Naughton, Raghu Ramakrishnan, and Sunita Sarawagi. On the computation of multidimensional aggregates. In *Proc. 22nd Int. Conf. Very Large Databases, VLDB*, pages 506–521, 1996.

[2] Peter Anick and Suresh Tipirneni. Method and apparatus for automatic construction of faceted terminological feedback for document retrieval, 2003. US Patent 6519586.

[3] Will Archer Arentz and Aleksander Øhrn. Multidimensional visualization and navigation in search results. In *Proc. 8th International Conference on Knowledge Based Intelligent Information and Engineering Systems (KES'2004)*, pages 620–627, September 2004.

[4] Ramon Barquin and Herb Edelstein (editors). *Building, Using and Managing the Data Warehouse*. Prentice-Hall, Inc, 1997.

[5] Andrei Z. Broder, Nadav Eiron, Marcus Fontoura, Michael Herscovici, Ronny Lempel, John McPherson, Runping Qi, and Eugene J. Shekita. Indexing of shared content in information retrieval systems. In *Proc. 10th International Conference on Extending Database Technology (EDBT 2006)*, pages 313–330, March 2006.

[6] E.F. Codd, S.B. Codd, and C.T. Salley. Providing olap (on-line analytical processing) to user-analysts: An IT mandate. Technical Report Technical Report, E.F. Codd & Associates, 1993.

[7] Wisam Dakka, Rishabh Dayal, and Panagiotis G. Ipeirotis. Automatic discovery of useful facet terms. In Andrei Z. Broder and Yoelle S. Maarek, editors, *Proc. SIGIR 2006 Workshop on Faceted Search*, pages 18–22, August 2006.

[8] Wisam Dakka, Panagiotis G. Ipeirotis, and Kenneth R. Wood. Automatic construction of multifaceted browsing interfaces. In *Proc. 14th ACM Int. Conf. on Information and knowledge management (CIKM'2005)*, pages 768–775, November 2005.

[9] Ron Fagin, R. Guha, Ravi Kumar, Jasmin Novak, D. Sivakumar, and Andrew Tomkins. Multi-structural databases. In *Proc. 24th ACM Symposium on Principles of Database Systems (PODS'2005)*, pages 184–195, June 2005.

[10] Ron Fagin, Ph. Kolaitis, Ravi Kumar, Jasmin Novak, D. Sivakumar, and Andrew Tomkins. Efficient implementation of large-scale multi-structural databases. In *Proc. 31th Int. Conf. Very Large Databases, VLDB'2005*, pages 958–969, August 2005.

[11] Daniel Feinstein and Frank Smadja. Hierarchical tags and faceted search. the rawsugar approach. In Andrei Z. Broder and Yoelle S. Maarek, editors, *Proc. SIGIR 2006 Workshop on Faceted Search*, pages 23–25, August 2006.

[12] Marcus Fontoura, Ronny Lempel, Runping Qi, and Jason Zien. Inverted index support for numeric search. *Internet Mathematics*, 3(2):153–185, May 2007.

[13] Jim Gray, Adam Bosworth, Andrew Layman, and Hamid Pirahesh. Data cube: A relational operator generalizing group-by, cross-tab and sub-totals. In *Proc. 12th International Conference on Data Engineering*, pages 152–159, 1996.

[14] Marti A. Hearst. Design recommendations for hierarchical faceted search interfaces. In Andrei Z. Broder and Yoelle S. Maarek, editors, *Proc. SIGIR 2006 Workshop on Faceted Search*, pages 26–30, August 2006.

[15] Christian Kohlschütter, Paul-Alexandru Chirita, and Wolfgang Nejdl. Using link analysis to identify aspects in faceted web search. In Andrei Z. Broder and Yoelle S. Maarek, editors, *SIGIR'2006 Workshop on Faceted Search*, pages 55–59, August 2006.

[16] Marc F. Krellenstein. Method and apparatus for searching a database of records, 1999. US Patent 5924090.

[17] Daniel N. Meredith and Jan H. Pieper. Beta: Better extraction through aggregation. In Andrei Z. Broder and Yoelle S. Maarek, editors, *SIGIR'2006 Workshop on Faceted Search*, pages 8–12, August 2006.

[18] Tapio Niemi, Marko Niinimaki, Jyrki Nummenmaa, and Peter Thanisch. Constructing an olap cube from distributed xml data. In *Proc. 5th Int. Workshop on Data Warehousing and OLAP (DOLAP)*, pages 22–27, 2002.

[19] Themistoklis Palpanas, Richard Sidle, Roberta Cochrane, and Hamid Pirahesh. Incremental maintenance for non-distributive aggregate functions. In *Proc. 28th Int. Conf. Very Large Databases, VLDB*, pages 177–188, 2002.

[20] Kenneth A. Ross and Angel Janevski. Querying faceted databases. In *Proc. 2004 Semantic Web and Databases Workshop (SWDB'2004)*, pages 199–218, August 2004.

[21] Kenneth A. Ross and Divesh Srivastrava. Fast computation of sparse datacubes. In *Proc. 23nd Int. Conf. Very Large Databases (VLDB)*, pages 116–125, 1997.

[22] Nick Roussopoulos, Yannis Kotidis, and Mema Roussopoulos. Cubetree: organization of and bulk incremental updates on the data cube. In *Proc. 1997 ACM SIGMOD Int. Conf. on Management of Data*, pages 89–99, 1997.

[23] Ben Schneiderman, David Feldman, Anna Rose, and Xavier Ferré Grau. Visualizing digital library results with categorical and hierarchical axes. In *Proc. fifth ACM Conference on Digital Libraries (DL'2000)*, pages 57–66, 2000.

[24] Emilia Stoica, Marti A. Hearst, and Megan Richardson. Automating creation of hierarchical faceted metadata structures. In *Proc. NAACL-HLT 2007, Rochester, NY*, pages 244–251, April 2007.