Knowl Inf Syst (2016) 47:697–732
DOI 10.1007/s10115-015-0871-2

REGULAR PAPER

Partial materialization for online analytical processing over multi-tagged document collections

Grzegorz Drzadzewski[1] · **Frank Wm. Tompa**[1]

Received: 12 November 2014 / Revised: 28 April 2015 / Accepted: 6 July 2015 /
Published online: 7 September 2015

Abstract The New York Times Annotated Corpus, the ACM Digital Library, and PubMed are three prototypical examples of document collections in which each document is tagged with keywords or phrases. Such collections can be viewed as high-dimensional document cubes against which browsers and search systems can be applied in a manner similar to online analytical processing against data cubes. After examining the tagging patterns in these collections, a partial materialization strategy is developed to provide efficient storage and access to centroids for document subsets that are defined through queries over tags. By adopting this strategy, summary measures dependent on centroids (including measures involving medoids, sets of representative documents, or sets of representative terms) can be efficiently computed. The proposed design is evaluated on the three collections and on several synthetically generated collections to validate that it outperforms alternative storage strategies.

Keywords Document warehouse · Document tags · Text analytics · Selective materialization · OLAP · Faceted browsing

1 Introduction

Large document collections such as the New York Times Annotated Corpus and the ACM Digital Library cover many diverse topics. It can be a daunting task to decide what to read on a new topic or to find which combinations of topics deserve more attention. As an aid to readers, various tags (usually key words and phrases) are assigned to each document in these

✉ Grzegorz Drzadzewski
gdrzadze@uwaterloo.ca

Frank Wm. Tompa
fwtompa@uwaterloo.ca

1 David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON N2L 3G1, Canada

collections, reflecting the topics covered by that document. In a large collection, each tag can be assigned to hundreds or thousands of documents.

Following standard practice in information retrieval, we model a document as a *bag of terms* represented by a document term vector (DTV), which is a vector of values where each entry corresponds to a term together with the term's (normalized) frequency in that document. A set of DTVs can be aggregated together to obtain a *set centroid* for the corresponding documents, which can be used to summarize the document set. Given the centroid, a system can produce other summary measures, such as a representative set of "bursty" terms [19,37], the medoid [9,18], or a diverse set of representative documents [12].

In addition to the DTV, each document in our collections is assigned a *set of tags* that are external to the document. Each metadata tag is a value chosen from a facet (such as location, time, organization, person, cost, or event) that corresponds to a conceptual dimension used to describe the data [49]. For simplicity, we assume that the facets are unstructured (i.e., that the value space within a facet is not hierarchically organized) and that each document is assigned zero or more values from each of the facets. Following standard practice, we assume that the assigned tags have been selected with care: They are typically of high quality and identify topics or important concepts found in the document.

A facet-based browsing environment supplements a traditional search engine by adding facilities that allow users to benefit from the metadata tags. With the help of faceted search a user may start to explore the ACM tagged document collection by issuing a traditional search request, say "databases cloud." As in other systems, the user is presented with the top *k* matching documents, but in addition the user is also informed by the system of the tags associated with those documents. In response to "databases cloud," the user might learn that all the corresponding documents are tagged "database," 90 % are tagged "cloud computing," 55 % of the top responses are also tagged "service-oriented architecture," 35 % are tagged "security and privacy," and 10 % are tagged "genome informatics." (Instead of precise percentages, similar information might instead be provided in the form of tag clouds.) The user could then select tags of interest to formulate a refined query by issuing a Boolean query over tags (i.e., "slicing and dicing" the collection).

Faceted search helps to narrow a large result set down to a more manageable size for browsing, and a study at the University of North Carolina library showed that it was preferred by users over traditional search interfaces based on text content alone [40]. In a variety of other settings, user studies have found that systems supporting faceted search and browsing are superior to traditional search engines and to systems based on clustering by document content [17]. For example, Yee et al. [49] found that "Despite the fact that the interface was often an order of magnitude slower than a standard baseline, it was strongly preferred by most study participants. These results indicate that a category-based approach is a successful way to provide access to image collections." Kipp and Campbell [29] found that "Users would find direct access to the thesaurus or list of subject headings showing articles indexed with these terms to be a distinct asset in search." Hearst [23] concluded that "Usability results show that users do not like disorderly groupings like [those produced by clustering systems], preferring understandable hierarchies in which categories are presented at uniform levels of granularity." Pratt et al. [38] found that "a tool that dynamically categorizes search results into a hierarchical organization by using knowledge of important kinds of queries and a model of the domain terminology . . . helps users find answers to those important types of questions more quickly and easily than when they use a relevance-ranking system or a clustering system." Zhang and Marchionini [52] found that "[A faceted search and browsing] interface will bring users added values beyond simple searching and browsing by in fact combining these search strategies seamlessly." Faceted search has thus emerged as a valuable technique for information access in many e-commerce sites, including Wal-Mart, Home Depot, eBay, and Amazon [45].

We envision an enhanced interface that, in addition to a traditional faceted search interface, provides summary information about the resulting document set. For example, a summary may consist of the *k* most representative articles in the sub-collection that satisfies the query, the most common terms used within articles in that sub-collection, and the distribution of tags that are assigned to articles in the sub-collection. If the summary matches the user's information need, individual articles in that set can be retrieved; otherwise the user can reformulate the query (often "drilling down" by specifying additional tags or "rolling up" by removing some tags from the query) to arrive at a more appropriate set of articles. We have recently described a prototype of the enhanced interface in further detail [14]. For an analyst (or even a casual reader) armed with the New York Times, this approach might uncover sets of articles that provide a comprehensive summary of news reports on a specific subtopic of interest. For a computer scientist investigating an unfamiliar research area through the ACM Digital Library, providing summaries based on tag-based queries can identify the most relevant articles to read in the area and how those articles relate to topics identified by other tags.

We rely on the previous studies to validate the utility of faceted search and browsing: It is prevalent, effective, and satisfies users' needs. In this paper, we concentrate on making such systems more efficient. If a document collection is already provided with meaningful metadata tags so that faceted search and browsing is feasible, the main problem that needs to be solved is to find a fast way of calculating centroids, which are required to provide summaries of document sets that match users' tag-based queries. Because some sets can be very large, aggregating large amounts of data in order to calculate summary measures may be too time-consuming to be performed online. For conventional data warehouses, *On-Line Analytical Processing* (*OLAP*) systems have been developed in order to speed up the aggregation of multidimensional data through full or partial materialization of summaries. Similarly, we show that partial materialization is required in order to provide summaries at each step of a faceted search when space is limited.

Unfortunately, current OLAP systems are designed for data collections that have tens of dimensions and will not work for document collections that have hundreds of facets with millions of tags. To handle such a large number of dimensions, we propose to materialize centroids for sub-collections that correspond to all documents sharing small subsets of tags. Thus, centroids are stored for predetermined subsets of the data, and calculating centroids for arbitrary subsets corresponding to users' queries requires aggregating data from several overlapping subsets (because documents with multiple tags will contribute to multiple materialized centroids). The techniques used in current OLAP systems, however, do not accommodate such overlap.

This paper includes the following contributions:

- detailed analyses of tagging patterns in two representative multi-tagged document collections: the New York Times Annotated Corpus and the ACM Digital Library;
- a storage design that performs well for calculating centroids of document sets that result from both short and long conjunctive queries over tags and enables aggregation of cells with overlapping data;
- the development and evaluation of several partial materialization strategies for high-dimensional, sparse data.

The paper is organized as follows. Related work is described in Sect. 2, and requirements for a browsing system are proposed in Sect. 3. Next, the properties of prototypical multi-tagged document collections are introduced in Sect. 4. A new storage architecture for multi-tagged document collections that supports efficient computation of topic centroids is described in Sect. 5, and the partial materialization techniques that take advantage of it are described in Sect. 6. Then, in Sect. 7, the performance of the storage architecture and the materialization strategies are evaluated on three real and several synthetic collections. Conclusions and further work are summarized in Sect. 8.

2 Background and related work

2.1 Folksonomies and tag recommendation systems

The New York Times and the ACM Digital Library rely on tags being chosen by users with care so as to maximize the reuse of tags where applicable and to distinguish between concepts through the use of disjoint tag sets where possible. To achieve these ends, they employ controlled vocabulary for some facets and allow only limited use of uncontrolled vocabulary.

In contrast, many social Web sites, such as Delicious and Flickr, allow users to attach arbitrary tags to documents to organize content on the Web. Tags can be chosen by users at will, and different users may assign different tags to the same object. This results in so-called folksonomies [24] that include many tags per document, large tag vocabularies, and significant noise. Faceted browsing has been implemented over folksonomies in systems such as dogear [34], and the complex tagging patterns involved can benefit from more efficient exploration, which is the aim of our work.

User studies that examine users' perceptions of the role and value of tags [28] showed that one common view of tags is as keywords (tags describe key aspects of the document) and another common view is for categorization. This is supported by another study [1], a taxonomy of tagging motivations in ZoneTag/Flickr, which concluded that one of the purposes of tags is to show context and provide content description.

To help reduce noise, there is much research on tag recommender systems, which are designed to help users assign tags to documents. Content-based tag recommender systems assume that tags for a specific resource can be extracted by processing the textual information about the resource to be annotated. These approaches adopt techniques from information retrieval [2] in order to extract relevant terms to label a resource. More specifically, term frequency and inverse document frequency have been shown to yield good keyword identification results [6,16,48], and their use has been adopted by tag recommendation systems [5]. Content authors and editors do not explicitly compute inverse document frequency when tagging an article, but their intuition regarding which words are informative replaces the use of this measure by human judgment.

In related work, a tag recommender system that relies on topic modeling was developed to provide an annotator with a set of diverse tags that represent the various topics covered in the document [3]. The generative model in the system simulated the users' tagging process in a social tagging system. It assumed that for any resource there are a multitude of topics, and that when users tag a resource they first identify topics of interest from the resource, after which they express the chosen topics via a set of words (tags). Each topic accordingly corresponds to a probability distribution over tags, which gives the probability of picking out a tag with respect to a certain topic. The user studies performed as part of the evaluation of the system suggested that users preferred the tags suggested by this new system.

From these previous studies, we conclude that a carefully annotated document has tags representing all the topics that have sufficiently high presence in that document.

2.2 Browsing document collections

In place of faceted search, which has been described in the introductory section, browsing systems might rely on document clustering. For example, traditional search engines are designed to provide a user with a ranked list of documents that satisfy a query, and clustering may be performed on top of the result set in order to organize similar documents into groups [50]. Because clustering can be fully automated, it can be applied to text collections that have not been assigned metadata tags. However, if clustering is applied online and if it were to be applied to result sets consisting of thousands of documents, it would impose unacceptably long delays. To avoid this bottleneck, systems such as Clusty.com perform clustering on the top k results only. On the other hand, if clustering is applied offline, cluster labels can be interpreted to be metadata tags and the techniques proposed here can be similarly applied.

Scatter/Gather is a well-known document search interface based on clustering [9]. Users explore a document collection by dynamically clustering a set of documents (scattering), selecting clusters of interest based on their summaries, and then treating all documents in the selected clusters as one set (gathering). These steps are then repeated to further investigate the contents of the sub-collection. The summaries used to characterize clusters take the form of a set of representative terms, chosen on the basis of frequency alone, together with the headlines of the documents closest to the centroids.

Like Scatter/Gather, our proposal allows users to repeatedly select subsets of the document collection, examining summaries for each grouping of documents to determine whether or not to include specific groupings in the refinements. However, unlike Scatter/Gather, the system we envision is based on a multi-valued, faceted labeling for each document rather than on hard clusterings; thus even if cluster labels at each step were to be treated as if they were metadata tags for externally-specified classes, Scatter/Gather would correspond to a single-valued labeling of documents. Furthermore, in Scatter/Gather it is difficult for users to predict what clusters will be generated since the grouping criterion is hidden, unlike when aggregation is specified through tags visible to user. Finally, we envision a search system in which a user is free to broaden the search at any step, rather than being expected to restrain themselves to drilling down alone.

To make Scatter/Gather usable in an interactive manner, offline hierarchical clustering can be performed on the document collection [10]. In this approach, meta-documents corresponding to a union of documents are created offline, and during the scatter phase the meta-documents are clustered instead of the actual documents, thus reducing the number of items to be clustered and thereby reducing execution time. Document clustering is therefore only approximated. In addition, inter-document distances are computed based on selected features instead of the full text of the meta-documents, thus again reducing execution time. Interestingly, the third variant of our proposal stores centroids for meta-classes, somewhat akin to Scatter/Gather's meta-documents, but those centroids are corrected to exact centroids for the associated classes before they are used in browsing.

An even faster implementation of Scatter/Gather (LAIR2) was developed by Ke et al. [27], based on precomputing a complete (binary) hierarchical clustering of the documents. Thus, for a collection with N documents, LAIR2 materializes $N - 1$ clusters. Then, instead of clustering documents during the browsing stage, it retrieves prematerialized nodes from the cluster hierarchy. Like the previous approach, however, the authors are only concerned with improving the execution time and do not consider the storage cost required to store every sub-cluster of a full hierarchical clustering. In contrast, the amount of storage required by a browsing system, as well as execution time, is central to our work.

2.2.1 Tag exploration

One difficulty in browsing via tags is to determine which tags are present in the collection and how tags are related to each other. A query and browsing interface can display the distribution of tags that are assigned to articles in each result set, thereby suggesting tags that can be used for further refinement. For broadening a search, the system could display tags that are associated with carefully chosen supersets of the result set. Alternatively, the system could provide a mechanism to browse the tags themselves (as opposed to the documents associated with those tags) through an interface to a thesaurus or ontology [8,36,43]. We make no assumptions about the structure of the tag space for our work, and the incorporation of tag-browsing facilities is orthogonal to our work.

2.2.2 Multi-document summarization

The set of documents that result after each browsing or search step must be presented to the user in some form. Search engines, for example, display the top-k matches after ranking, and browsing systems can similarly present the k most representative documents of a result set, as is done in Scatter/Gather. As a special case, the medoid document, i.e., the one closest to the centroid, can be displayed. Another form of summarization is to display the most representative terms that appear in the result set, for which Scatter/Gather chooses the most frequently occurring terms, but representativeness might be defined using inverse document frequency as well or using other statistical measures, such as information gain.

Alternatively, a more informative summary of a result set may be a précis generated from the documents. There are many different approaches to perform such multi-document summarization, based on abstraction and information fusion, topic-driven summarization, clustering, graphs, and ranking [11,21]. Of particular relevance here are multi-document summarization methods that rely on using the centroids of document sets [39], which is the measure for which we are designing an efficient infrastructure.

2.3 OLAP for data warehouses

Data cubes serve as the model for describing OLAP operations [20]. A cubes' dimensions reflect the attributes that characterize the facts stored in the cube's cells; for example, a set of sales records might have dimensions for date of sale, location of sale, type of product sold, customer demographics, etc. Because multidimensional analysis often requires aggregated measures over some of the dimensions (e.g., average sale prices for each product per day, regardless of location and customer), OLAP systems provide the materialization of selected *cuboids* defined over a subset of dimensions, storing precomputed aggregates in each resulting cell. The dimensionality of a cuboid is equal to the number of unaggregated dimensions, and the space is proportional to the number of cells (the product of the number of possible values in each unaggregated dimension). Thus a d-dimensional cuboid stores aggregated values in cells indexed by the possible values for each of the d unaggregated dimensions, and if each dimension is binary requires $O(2^d)$ space.

2.3.1 Full materialization

OLAP systems that materialize all possible cuboids offer the best response time to user queries. However, full materialization requires $O(2^n)$ space for cubes with n dimensions. Compression can be applied to achieve full materialization while reducing the storage cost;

this can save space in situations where there is significant repetition in cell measures, as is the case with sparse cubes. Compression techniques for data cubes include condensed cubes [46], dwarf cubes [44], and quotient cubes [30]. However, these techniques do not scale to a high number of dimensions [32].

2.3.2 Partial materialization

Partial materialization techniques are used to materialize a subset of cuboids (also referred to as views) from the lattice of cuboids [22]. When answering a query, instead of fetching the data from the base cuboid and performing aggregation on it, the cuboid corresponding to the query can be calculated from the closest materialized superset cuboid. Therefore, the subset of cuboids to materialize is picked so as to minimize the time needed for the expected query workload, while requiring no more than a given amount of storage.

Thin cube shell materialization is a partial materialization where only the base cuboid and certain low-dimensional (most highly aggregated) cuboids are stored [32]. More specifically, in addition to the base cuboid, the strategy stores all cuboids having exactly d dimensions, where $d \ll n$, n is the total number of dimensions, and there are $\binom{n}{d}$ d-dimensional cuboids. Alternatively, we could materialize all cuboids having d or fewer dimensions, which would further reduce the execution time of short queries at the expense of additional storage space. However, d-dimensional cuboids can be used to answer queries that involve at most d dimensions only; this involves choosing a materialized cuboid and aggregating the data for the dimensions omitted in the query. On the other hand, queries involving more than d dimensions are answered by aggregating over the base cuboid. Picking a larger d for materialization results in increased storage cost and increases the time required to calculate queries with few dimensions, but picking a small d results in much longer computation time for queries with more than d dimensions. If the expected workload has a wide range of queries, there may not be a fixed d that is appropriate.

As an improvement over a thin cube shell, Li et al. [32] proposed a shell fragment approach for dealing with high-dimensional cubes. The technique relies on the assumption that high-dimensional data have limited interactions among dimensions (tags). It assumes that on average any one tag interacts with at most K other tags, where K is at most five and these tag interactions can usually be well clustered. Under such circumstances when a collection has T unique tags, it can be partitioned into T/K nonoverlapping fragments. Depending on the properties of the data and the query workload, it may be necessary to choose fragments of various sizes. However, larger fragments require more storage space. If the tag interactions cannot be clustered well, it may be necessary to store overlapping fragments to provide satisfactory query response time, in which case more fragments need to be stored. This, in turn, leads to greater storage requirements. For each of these fragments a full cube materialization is stored; thus, all the cuboids of dimensions ranging from 1 to K are materialized. This results in $2^K - 1$ cuboids materialized per fragment, where a cuboid with d dimensions has $2^d - 1$ cells, which therefore implies $\sum_{i=1}^{K} \binom{K}{i}(2^i - 1)$ cells for a fragment. For a fragment of size $K = 3$, 19 cells per fragment are needed. For scenarios in which the prematerialized fragments do not enclose the user's query, again the view needs to be calculated from the base cuboid, which can be time-consuming.

2.4 Document warehouses

A document warehouse is like a data warehouse, except that instead of performing analyses over tabular data, it supports analyses over documents. OLAP in document warehouses has been used to provide users with summaries of related documents through the use of centroids and medoids of the clusters found in cells of a cube [26,51]. Efficient storage strategies for OLAP over nonoverlapping sets of documents have been proposed [51], and a fully materialized approach that deals with overlapping sets has also been proposed [26], but efficient storage strategies that can handle overlapping document sets—the focus of this paper—have not been explored. In tagged document collections, tags are treated as dimension values. Two different forms of schema can be used for determining how tags are assigned to dimensions: multidimensional schemas and single-dimensional schemas.

2.4.1 Multidimensional schema

A multidimensional schema (MDS) stores each tag in a separate binary dimension, where 0 signifies that the corresponding tag is not assigned and 1 signifies that it is. For example, if a document d_1 has tags (*Finances, Stocks*) and d_2 is tagged with (*Stocks*) only, then d_1 is stored in cell (1, 1) and d_2 is stored in cell (0, 1) of the 2D cuboid with those two dimensions. This cuboid can answer the query $Finances \vee Stocks$ by aggregating cells (1, 1), (0, 1), and (1, 0) together, where, for this small example, the cell (1, 0) is empty. By having a separate dimension for each tag, we can ensure that aggregations performed on a cuboid do not double count any documents. Storing a data cube for MDS is a challenge when there are many tags.

2.4.2 Single-dimensional schema

A single-dimensional schema (SDS) stores all tags in one dimension. The dimension can take on a value ranging from 1 to T, where T is the number of unique tags in the collection. This approach works well in situations where each document is assigned only a single tag. Zhang et al. [51] used this approach for organizing a collection of documents into nonoverlapping cells and developed a partial materialization scheme on top of it.

In contrast, Jin et al. [26] used SDS for storing documents with multiple tags. Unfortunately, this can result in the same document being assigned to multiple cells, which is problematic when the cells in a cuboid are aggregated. Continuing with the example above, because there is only one “tags” dimension, cell(*Finances*) stores d_1 and cell(*Stocks*) stores both d_1 and d_2. In this situation, simply adding the counts for cell(*Finances*) and cell(*Stocks*) to count the number of results for the query $Finances \vee Stocks$ will result in double counting d_1. We adopt the solution to this problem developed by Jin et al., namely storing document membership information for each cell, so that when multiple cells are aggregated, cell overlaps can be detected and compensations applied. Jin et al. use a full materialization on a small data set and focus on the union operation only; optimizing conjunctive queries involving overlapping cells has not been considered.

3 System requirements

In this section we describe requirements and associated challenges for a system that will support online analytical processing for a large document collection. The requirements are derived in part by examining characteristics of the PubMed interface to biomedical literature [15].

PubMed includes more than 24 million abstracts and corresponding citations to articles, which are annotated with a variety of tags[1] chosen from Medical Subject Headings (MeSH), the National Library of Medicine controlled vocabulary thesaurus used for indexing articles for PubMed; the EC/RN Number, assigned by the Food and Drug Administration (FDA) Substance Registration System for Unique Ingredient Identifiers; and Supplementary Concept tags, which include chemical, protocol or disease terms. The PubMed interface supports searching for documents using a standard text search, matching query terms against the abstract, the citation, and all assigned metadata tags, as well as by specifying that some of the query terms should be restricted to matching MeSH terms (or some other facet) only.

A corpus of MeSH terms assigned to PubMed documents[2] includes 244,553,378 tags assigned to 20,997,401 documents, or 11.65 tags per document on average. The corpus identifies 71,690,729 assigned tags as "major," that is, the topics play a major part in the associated paper, yielding on average 3.28 major tags per document.

PubMed users looking for relevant articles can benefit immensely from searching with the aid of metadata tags [35]. Since PubMed is a very large collection and the sizes of sets of search results are often large, it can certainly benefit from more efficient calculation of aggregate measures that summarize the contents of query results.

3.1 Supported measures

As explained in Sect. 1, a document is considered to be a bag of terms, represented by its document term vector (DTV). All but the top m terms, based on mutual information, can be ignored so as to avoid storing stop words and other uninformative terms, and for every document, the frequency of each remaining term is stored as a normalized DTV.

In order to provide meaningful summaries about a document set (e.g., its medoid, any set of representative documents, or a set of representative terms), we need to compute the set centroid C, which can be represented by a vector of term frequencies equal to the mean of all the DTVs for documents that belong to that set. However, instead of storing the means directly, for a set of documents S, we store its centroid C_S as a dictionary that maps terms to $(sum, count)$ pairs, which is then easily updated when documents are added to or removed from the set:

$$C_S[term].sum = \sum_{d \in S} d[term] \tag{1}$$

$$C_S[term].count = |\{d \in S \mid d[term] > 0\}| \tag{2}$$

where d is a normalized DTV of length m. Thus, a set centroid vector has length m regardless of how many documents are in the set.

3.2 Supported queries

Associated with each document is a set of "metadata" tags, each of which is assumed to represent some aspects of the document's content. We allow the user to pose queries as Boolean formulas over tags, such as $Election \wedge President \wedge (Stocks \vee Stock\_Market)$. Conjunctions of terms narrow down the scope of documents to those that involve all the concepts represented by the conjuncts. The use of negation is allowed, but only in the form

[1] For consistency within this paper, we continue to use "tag" to refer to a metadata term assigned to an article from a controlled vocabulary, even though PubMed's use of "tag" refers to the attachment of a facet label to a query term.

[2] Available at http://mbr.nlm.nih.gov/Download/2014/Data/Full_MH_SH_items.gz.

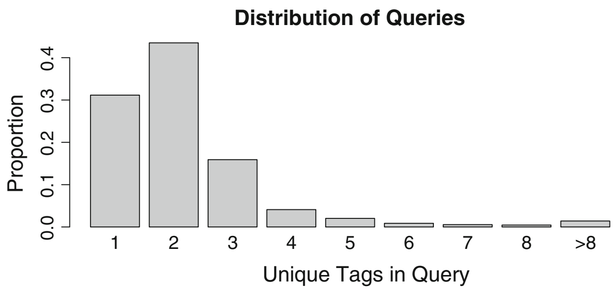

Fig. 1 Analysis of unique tags per query

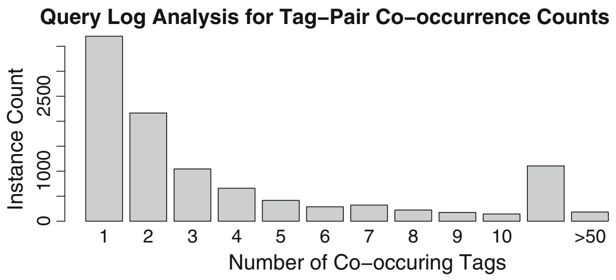

Fig. 2 Number of distinct tags co-occurring with tags in a query

"and not" to allow a conjunction with the complement of the documents having a given term, as in the example $President \wedge \neg Election$. Disjunction provides a means of query expansion, allowing synonyms and related tags to be included in a query [33].

3.3 Expected workload

Users explore a multi-tagged document collection through a browser front end that enables them to invoke Boolean queries. As part of their exploration, they may pose queries and read summaries (in the form of data derived from set centroids, such as representative documents or representative terms). After users examine summaries of document sets, they may choose to drill down to smaller subsets of documents by issuing more specific queries. The browsing system we are developing[3] is required to provide quick responses to the generated queries.

We rely on data from a PubMed query log[4] [35] to help characterize a feasible query workload. Among 2,996,301 queries collected over a single day, 16,928 queries include only terms chosen from facets that have a controlled vocabulary (specifically, MeSH terms [MH], MeSH major topics [MAJR], MeSH subheadings [SH], filters [FILTER], EC/RN Numbers [RN], and supplementary concepts [NM]), with the possible addition of one or more pure text terms. Treating each text conjunct or disjunct as if it were a single tag, these queries involve anywhere from 1 to 46 tags, with the majority of the queries using between 1 and 3 tags (Fig. 1). Figure 2 shows the number of distinct tags that co-occur in queries having a given query tag, from which we observe that tags are used repeatedly in queries in a variety of contexts specified by other tags. The usage patterns of the three

[3] http://dsg.uwaterloo.ca/TagBrowser2015/.

[4] Available at ftp://ftp.ncbi.nlm.nih.gov/pub/wilbur/DAYSLOG.

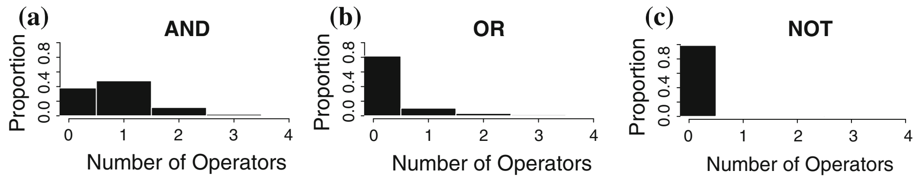

Fig. 3 Distribution of operator counts in PubMed query log: **a** AND, **b** OR, **c** NOT

Boolean operators are summarized in Fig. 3, which shows that the "NOT" operator occurs in 1 % of queries, the "OR" operator occurs in 18 % of the queries, and the "AND" operator occurs in 62 % of the queries; 31 % of queries are of length 1 and so use no operators. These observations suggest that a realistic query workload will likely include primarily short queries that are predominantly conjunctive, as has been observed for other Web search systems [25].

In summary, our expectation is that (after some preliminary traditional searches of the text) users explore a collection by starting with a small set of tags of interest and then iteratively refining their queries to be more focused by including additional tags. For some queries, query expansion will be applied to incorporate some alternative tags. Thus, most queries will be conjunctions of tags (i.e., no negations and only occasionally disjunctions to accommodate alternative tags), most queries will be short, and most queries will match sets that include a large number of documents.

3.4 Design objective

We wish to provide a fast response to user queries by having an upper bound on the number of documents or centroids of materialized sets that need to be retrieved from secondary storage. At the same time we wish to minimize the number of set centroids that need to be precomputed and materialized to accomplish this. We focus on providing upper bound guarantees on execution costs for (positive) conjunctive queries, since they are expected to be most frequent: For each such query, no more than k DTVs or set centroids need to be accessed, for some fixed k. Queries that involve disjunction and negation will be answered using multiple conjunctive subqueries, and they may therefore require more than k DTVs or set centroids in total.

A document cube provides an excellent mechanism for structuring the collections of documents so as to answer Boolean queries on tags. Each cell in the cube represents the set of documents that have a specific tag assignment, and each cuboid represents document sets that are aggregated ("rolled up") by grouping on specific tags and ignoring others. As a result, conjunctive tag queries can be answered by selecting specific cells from appropriate cuboids, and centroids of document sets that correspond to other Boolean queries can be computed by combining the centroids from selected cuboid cells. The problem to be addressed is to determine which cells or cuboids to materialize to balance space and time.

4 Document collections

To motivate the design of our proposed index, we evaluate document collections from two different domains: the New York Times Annotated Corpus (NYT) [41] and the ACM Digital Library (ACM) [47].

Table 1 An article from (a) NYT with corresponding general online descriptors assigned to it, and (b) ACM with corresponding category and keyword tags assigned to it

(a) NYT		
Article headline		
Stocks drop in Tokyo		
General online descriptors		
Stock prices and trading volume		
Stocks and bonds		
Finances		
Prices (fares, fees and rates)		
(b) ACM		
Title of article		
The complex dynamics of collaborative tagging		
Categories and subject descriptors		
H.5.3 [Group organizational interfaces]: collaborative computing		
I.2.4 [Artificial intelligence]: knowledge representation		
Keywords		
Tagging	Del.icio.us	Power laws
Complex systems	Emergent semantics	Collaborative filtering

4.1 New York Times Annotated Corpus

The NYT collection includes 1.8 million articles spanning 20 years. The collection has 1 million tags that cover many different facets, such as people, places, companies, and descriptors, and multiple tags can be assigned to each article. Out of the various types of tags contained in the collection, we consider only the tags found in the general online descriptors, which are the ones that correspond to the text found in the articles. Table 1a shows a tag assignment for a single document found in the NYT collection. In our analysis we consider only tags that have been assigned to at least 200 documents, yielding 1015 such tags that are applied to 1.5 million documents.

The tagging patterns exhibited by a document collection affects system design choices and determines whether any of the previously developed OLAP materialization strategies can be applied. We analyzed the tagging pattern for NYT using measures adopted from analyzing tagging patterns in folksonomies such as Delicious [7]. These measures capture the frequency of tags appearing in the collection, the distribution of tag counts per document, and the amount of co-occurrence between the 10 most frequent tags and other tags.

The plot of tag frequencies is shown in Fig. 4a. The frequencies are normalized by the count of the most popular tag, which happens to be *Finances*, with a count of 142 thousand documents. At the other end of the spectrum, the least frequent tag (with our cutoff) has 201 documents. When the tag frequencies are sorted in descending order, the distribution resembles Zipf's law.

The number of tags assigned per document is shown in Fig. 5a. This ranges from 34 % of the documents being assigned just one tag to a few documents having 43 tags, with 2.7 tags per document on average. Figure 6a shows the amount of document overlap between each of the 10 most popular tags and all the other tags, with the other tags shown in descending

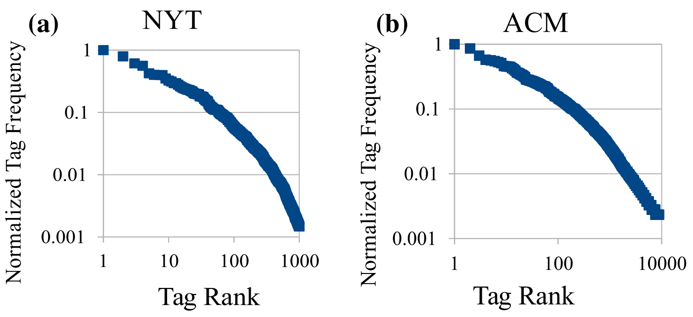

Fig. 4 Tag assignment frequency in **a** NYT and **b** ACM

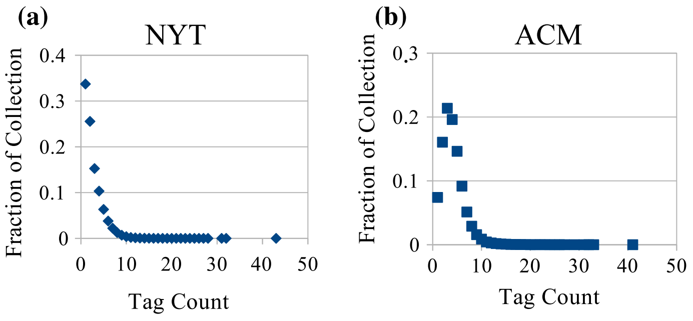

Fig. 5 Distribution of the number of tags per document for **a** NYT and **b** ACM

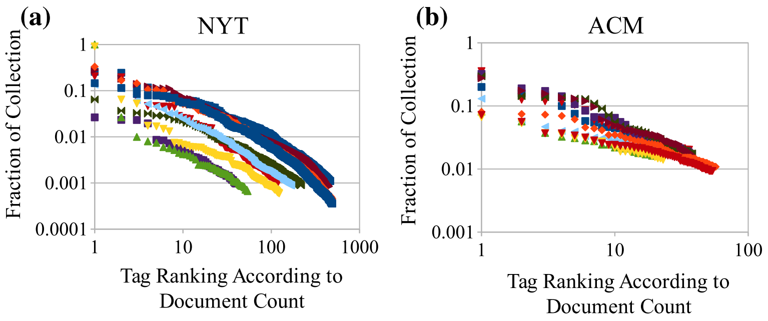

Fig. 6 Tag co-occurrence frequencies for the 10 most frequent tags in **a** NYT and **b** ACM

order by their frequency of co-occurrence. The presence of multiple tags per document and the high co-occurrence among the tags produce many nonempty document sets that match the conjunction of multiple tags.

4.2 ACM digital library

The much smaller ACM collection contains 66 thousand articles organized with categories, general terms, and keywords. In our analysis we consider the categories and keywords tags only, since there are only 16 general terms available. Table 1b shows an instance of a tag assignment for a single article found in the ACM collection. Since this collection is so much

Table 2 Number of conjunctions of n tags that contribute to high multi-way co-occurrence for NYT and ACM, with threshold limits of 50 for NYT and 5 for ACM

n	1	2	3	4	5	6	7	8	9	10	11	12	Total
NYT	1015	16,448	20,905	12,217	5289	2401	1152	493	151	27	2		60,100
ACM	9098	14,262	5280	3860	3700	3199	2390	1520	776	297	79	13	44,474

smaller than NYT, we include all the tags with at least five occurrences in our analysis, resulting in 9098 tags that satisfy this criterion.

The plot of tag frequencies is shown in Fig. 4b. Again, the tags are normalized by the count of the most popular tag, which has been used to tag 2144 documents, and the least frequent tag with our cutoff has been assigned to five documents. Just as for NYT, the distribution of tag frequencies resembles Zipf's law.

The number of tags assigned per document ranges between 1 and 41, as shown in Fig. 5b. The mean number of tags per document is 4.1 (when both keywords and categories are combined). As was true for the NYT collection, the distribution has a very wide range, with the majority of documents having fewer than 10 tags. Since there is a higher mean number of tags per document in the ACM, we expect a larger number of tag conjunctions to produce nonempty document sets.

Figure 6b shows the proportion of documents that have one of the 10 most popular tags and some other tags. The shape of the ACM graph is somewhat similar to that for NYT, but its magnitude is significantly higher, showing that the ACM tags are more inter-correlated.

4.3 Deeper analysis of tagging patterns

For additional insight into the tagging patterns exhibited by the NYT and ACM collections, two more properties are analyzed. The first property refers to the order of tag co-occurrences, while the second property refers to the similarity between sets in the collections.

4.3.1 Higher order tag co-occurrence

We define a collection to have a *high n-way co-occurrence* among its tags if the number of documents having n tags in common is greater than k for many different combinations of tags. Such document sets are of interest because their document count may be too high to have the set centroid calculated online, and this measure indicates the dimensionality of cuboids that need to be materialized in order to answer queries on tags efficiently.

The tag co-occurrence measures we have described for the NYT and ACM collections show that there is significant correlation among various pairs of tags, but it does not tell us if there is also high n-way co-occurrence for $n > 2$. The threshold used for determining whether an n-way co-occurrence is high should be set to be the size of a document set for which it would be efficient to calculate a summary online. For our experiments, we have chosen $k = 50$, which is quite appropriate for NYT. However, ACM is significantly smaller, and we wish to test how well our approaches scale up; therefore, we have chosen to use $k = 5$ for ACM in order to expose its tagging structure in more detail.

Table 2 shows that there are many surprisingly large tag sets where the corresponding document count is above the threshold. If tags were assigned to each document independently of other tags, then the chance that the intersection of more than four tags would result in a

Table 3 Percent overlap of cells for given query lengths

NYT

	1	2	3	4	5	6	7	8	9	10	11
1		3	1	0	0	0	0	0	0	0	0
2			55	38	34	40	53	63	72	82	100
3				94	90	90	95	99	100	100	100
4					100	100	100	100	100	100	100

ACM

	1	2	3	4	5	6	7	8	9	10	11	12
1		5	9	15	16	14	11	9	6	3	1	0
2			71	87	96	98	100	100	100	100	100	100
3				98	100	100	100	100	100	100	100	100
4					100	100	100	100	100	100	100	100

document set of size greater than the threshold would be low. The multi-way correlation that exists between tags has a big impact on the number of cells that need to be materialized.

4.3.2 Overlap between document sets

The similarity between two sets of documents S_a and S_b can be quantified by evaluating the size of their symmetric difference $|S_a \triangle S_b|$. We consider two sets as similar if $|S_a \triangle S_b|$ is less than or equal to the predefined threshold k, which is again set to 50 for NYT and 5 for ACM.

Let $\mathcal{D}$ be a document collection and let $\mathbb{A} \subseteq 2^{\mathcal{D}}$ and $\mathbb{B} \subseteq 2^{\mathcal{D}}$ be sets of document sets. We define the *k-overlap* $O_k(\mathbb{A}, \mathbb{B})$ between $\mathbb{A}$ and $\mathbb{B}$ as follows:

$$O_k(\mathbb{A}, \mathbb{B}) = \frac{|\{S_a \in \mathbb{A} \mid \exists S_b \in \mathbb{B} \wedge |S_a \triangle S_b| < k\}|}{|\mathbb{A}|} \tag{3}$$

By applying $O_k(\mathbb{A}, \mathbb{B})$ to pairs of sets reflected in Table 2, we can compute the percentage of overlap between document sets corresponding to the conjunction of i tags and document sets corresponding to the conjunction of j tags. This is summarized in Table 3, where the i^{th} column shows the percentage of overlap with sets of size j ($j < i$) and all entries are 100 for $i > j \geq 4$. Thus, many documents share many tags, but whenever significantly many documents share tags $G = \{t_1, t_2, \ldots, t_i\}$ for $i > 4$, not many *additional* documents share any subset $G' \subset G$ of those tags if $|G'| \geq 4$.

4.4 Significance of tagging patterns

The analyses of tagging patterns found in NYT and ACM reveal important challenges associated with such document collections. First, since there are many popular tags, some of which are assigned to as many as 140,000 documents, it is infeasible to compute set centroids for corresponding document sets online. Instead, it is necessary to precompute the data in order to guarantee satisfactory response times. Second, since many documents are assigned multiple tags and there is high correlation among the tags, it is reasonable to expect users to issue queries using tag conjunction. The more tag combinations that can be meaningfully conjoined, the more possible sets of documents exist about which a user can enquire. Third, surprisingly many tags can appear in a conjunctive query that yields a large document set. This leads to the existence of many sets for which centroids need to be computed; far more than one would expect if tags were assigned randomly. Materializing centroids for all these combinations is infeasible.

Fortunately, there is a lot of overlap between sets of documents of different query lengths. Because of this overlap, we can achieve considerable savings in storage cost by developing a suitable partial materialization strategy that can scale to large document collections with large numbers of tags. We explore this further in Sect. 6.

5 Storage architecture

5.1 Basic infrastructure

We assume that documents are stored as files and that the collection is indexed by a mapping from document IDs to the corresponding files. In order to support queries over tags, we further assume the existence of an inverted index that stores a postings list of document IDs for each of the tags. Finally, we assume that normalized document term vectors have been precomputed and that an index from document IDs to DTVs (as might be produced by a standard search engine) is also available.

With this minimalistic storage structure, tag queries may be answered using the following steps:

1. Use the inverted index over tags to return set S of document IDs that satisfy the query.
2. Initialize the document set centroid C_S to be empty.
3. For each $s \in S$:
    (a) Retrieve DTV for s.
    (b) Add DTV to document set centroid C_S using Eqs. 1 and 2.

This algorithm requires $|S|$ DTVs to be read from secondary storage, which will be quite slow for large sets. Even for systems with sufficient main memory to store the whole collection, it will be beneficial to avoid online aggregation of $|S|$ documents, especially when supporting many concurrent users. Therefore, it is desirable to bound the number of documents that must be retrieved for each query.

One way to reduce the cost of answering a query is to store precomputed centroids for well-chosen sets of documents $\mathbb{P}$ and to use these at query time to reduce the number of documents that must be retrieved. Given a document set S, we find a highly overlapping set $P \in \mathbb{P}$ and retrieve the precomputed set centroid C_P as well as the DTVs for all documents in $S \triangle P = (S - P) \cup (P - S)$, the symmetric difference between sets S and P. To calculate C_S, the DTVs of the retrieved documents are added to or subtracted from the centroid of P (Eqs. 1 and 2) in order to compensate for the difference between S and P. As a result, instead of retrieving $|S|$ DTVs, we need to retrieve $|S \cup P| - |S \cap P|$ DTVs as well as C_P. As a special case, $S \in \mathbb{P}$, in which case we merely need to retrieve C_S and avoid accessing any DTVs.

When choosing $\mathbb{P}$, the sets for which we precompute and store centroids, its size is constrained by the amount of available storage space. To benefit from $\mathbb{P}$, we wish to select sets that closely match the expected query load. To this end, we adopt the practice of using partial materialization of document cubes [22].

5.2 Storing document member sets

To support intersection and union operations on cells with overlapping sets of documents, the IDs of member documents need to be accessible for any given cell. Rather than storing document membership information for cells, even for those that are materialized, we store the list of defining tags with each materialized cell and rely on the postings list of documents for each tag to find the corresponding set of document IDs (Sect. 5.1). By storing each tags's postings list as a compressed bitmap with Word-Aligned Hybrid (WAH) encoding [31], we require 17.8MB for the NYT collection, which has 1.5 million articles, and 1.7MB for ACM collection that has 66 thousand articles. With such a low memory footprint, it is feasible to have the *tag* postings lists stored in memory. Thus, this approach conserves storage space and efficiently supports finding the set of documents associated with a cell.

5.3 Granularity of materialization

Materialization decisions can be made at two levels of granularity: whole cuboids or individual cells. We show here that it is preferable to decide whether or not to materialize individual cells.

5.3.1 Full cuboid materialization

A d-D cuboid, which can answer all queries that involve the subset of dimensions found in it, has $\prod_{i=1}^{d} n_i$ cells, where n_i is the number of unique values in dimension i. Since in a multidimensional schema each tag dimension has only two values (0 and 1), a d-D cuboid has 2^d cells. However, the cell that has all dimensions set to zero (i.e., none of the tags present) is not required when evaluating queries with at least one positive term (Sect. 3.2), and so only $2^d - 1$ cells need be stored.

An example of a 3D cuboid is shown in the first two columns of Table 4a. The cuboid consists of seven cells, one for each assignment of three tags (except for the cell that has all dimensions set to zero). Although a d-D cuboid can answer all queries involving any or all of the d tags defining the cuboid, it is optimized to answer queries that include all d dimensions; if fewer are specified, several cell measures must be aggregated. For example, C_{t_1} (the centroid for all documents having tag t_1, regardless of whether or not they also have tags t_2 and t_3) can be computed as $C_{t_1 \wedge t_2 \wedge t_3} + C_{t_1 \wedge t_2 \wedge \neg t_3} + C_{t_1 \wedge \neg t_2 \wedge t_3} + C_{t_1 \wedge \neg t_2 \wedge \neg t_3}$, which requires accessing four of the 3D cuboid's cells. In general, to answer a conjunctive query involving t tags by using a d-D cuboid defined over d tags with $d \geq t$, the set of tags defining the cuboid must include the set of tags used in the query and 2^{d-t} cells must be aggregated.

5.3.2 Individual cell materialization

Given a tag set $T = \{t_1, \ldots, t_d\}$, instead of materializing a whole d-D cuboid, this strategy materializes $I(T)$, the set of cells corresponding to all conjunctive queries without negation, which can be defined as follows:

$$I(T) = \{X_t.alltags \mid t \in (2^T - \emptyset)\} \quad (4)$$

where X_t is a cuboid for tag set t and *alltags* refers to the cell corresponding to all tags present. Table 4b shows the set of cells that will be materialized for $T = \{t_1, t_2, t_3\}$. The source column of the table identifies the cuboid from which the cell is taken.

Table 4 (a) 3D cuboid for tag set $\{t_1, t_2, t_3\}$, (b) $I(T)$ for tag set $\{t_1, t_2, t_3\}$

Dim			Cell	Computation
t_1	t_2	t_3	Centroid	From $I(T)$ cells
1	0	0	$C_{t_1 \wedge \neg t_2 \wedge \neg t_3}$	$C_{t_1} - C_{t_1 \wedge t_2} - C_{t_1 \wedge t_3} + C_{t_1 \wedge t_2 \wedge t_3}$
0	1	0	$C_{t_2 \wedge \neg t_1 \wedge \neg t_3}$	$C_{t_2} - C_{t_1 \wedge t_2} - C_{t_2 \wedge t_3} + C_{t_1 \wedge t_2 \wedge t_3}$
0	0	1	$C_{t_3 \wedge \neg t_1 \wedge \neg t_2}$	$C_{t_3} - C_{t_2 \wedge t_3} - C_{t_1 \wedge t_3} + C_{t_1 \wedge t_2 \wedge t_3}$
1	1	0	$C_{t_1 \wedge t_2 \wedge \neg t_3}$	$C_{t_1 \wedge t_2} - C_{t_1 \wedge t_2 \wedge t_3}$
1	0	1	$C_{t_1 \wedge t_3 \wedge \neg t_2}$	$C_{t_1 \wedge t_3} - C_{t_1 \wedge t_2 \wedge t_3}$
0	1	1	$C_{t_2 \wedge t_3 \wedge \neg t_1}$	$C_{t_2 \wedge t_3} - C_{t_1 \wedge t_2 \wedge t_3}$
1	1	1	$C_{t_1 \wedge t_2 \wedge t_3}$	$C_{t_1 \wedge t_2 \wedge t_3}$

Dim			Cell	
t_1	t_2	t_3	Centroid	Source
1	*	*	C_{t_1}	1D cuboid
*	1	*	C_{t_2}	1D cuboid
*	*	1	C_{t_3}	1D cuboid
1	1	*	$C_{t_1 \wedge t_2}$	2D cuboid
1	*	1	$C_{t_1 \wedge t_3}$	2D cuboid
*	1	1	$C_{t_2 \wedge t_3}$	2D cuboid
1	1	1	$C_{t_1 \wedge t_2 \wedge t_3}$	3D cuboid

For $|T| = d$, this approach materializes $2^d - 1$ cells, which is equal to the number of cells in the d-D cuboid. As shown in the last column of Table 4a, the set centroids for any cell in the 3D cuboid for tag set $\{t_1, t_2, t_3\}$ can be derived using the set of cells in $I(\{t_1, t_2, t_3\})$. In general, similar conversions can be derived by taking advantage of the inclusion–exclusion principle.

5.4 Query performance evaluation

For tag set T, full cuboid materialization and $I(T)$ both require the same number of cells to be stored, and both can answer all Boolean queries over T. However, the cost to answer a query depends on which set of materialized cells the query engine stores. Table 5 shows the number of cells that need to be aggregated to compute the answers to all queries involving one or two of t tags using a t-D cuboid vs. using individual cells included in the corresponding $I(T)$. In this table, the column labeled "count" shows how many distinct queries have the format shown in the column labeled "pattern"; for example, given four possible tags, there are 12 distinct queries that involve the conjunction of one (positive) tag and one negated tag; to answer any one of these queries, we need to access four cells in the cuboid (for every combination of tag presence and absence for the remaining tags), but only the two cells from $I(T)$ that correspond to the sets of documents having each tag.

Table 6 compares the cost of computing set centroids for all possible 3-tag queries when relying on a materialized 3D cuboid against the cost when relying on cells materialized using $I(T)$. For three tags, there are seven nonoverlapping sets of documents (corresponding to the seven cells in the 3D cuboid) and thus $2^7 - 1$ equivalence classes of queries. For each class, we found the minimum-length query (one with fewest literals) and, using these, tabulated the number of queries of each possible length against the cost (number of cells) needed to

Table 5 Cost of answering simple queries using a materialized t-D cuboid versus the individual cell materialization strategy

Query			Cost	
Length	Pattern	Count	t-D cuboid	$I(T)$
1	t_1	t	2^{t-1}	1
2	$t_1 \wedge t_2$	$\binom{t}{2}$	2^{t-2}	1
2	$t_1 \wedge \neg t_2$	$t(t-1)$	2^{t-2}	2
2	$t_1 \vee t_2$	$\binom{t}{2}$	$3 \times 2^{t-2}$	3

answer them under each materialization strategy. For very short queries, the query cost is lower when using $I(T)$ than when using the cuboid, and, importantly, the difference—as well as the length of query for which $I(T)$ outperforms the cuboid—becomes more pronounced as the number of dimensions in the materialized cuboids increases.

To compare query runtime when adopting the full cuboid materialization model against using the $I(T)$ model, we design a neutral query workload that has no bias toward any type of queries: each query includes at most three tags (with repetitions allowed), and the frequency of occurrence for each query depends on its length only and is independent of how many times it requires negation, union, intersection, or a combination of these operations. Using Table 6, Table 7 shows the average cost when using the $I(T)$ and complete cuboid (C) architectures to answer queries when the query length probability distribution is uniform, zero truncated Poisson, and geometric. For all but the uniform distribution (where performance differs by only 10 %), the $I(T)$ architecture outperforms the full cuboid materialization approach, even when all queries involve fewer than four distinct tags.

Additionally we evaluated the performance of the two storage architectures with a query workload model derived from the analysis of the PubMed query log. Since it was observed that the ‘NOT’ operator occurred in only 1 % of queries, the derived model will not generate queries that have that operator. This leaves us with 18 queries that it can generate, which are characterized by the number of ‘AND’ and ‘OR’ operators that they use. The probability of seeing queries with a ‘AND’ operators and o ‘OR’ operators is calculated from the probability distribution observed in Fig. 3 under the assumption of independence of the two distributions. Table 7 shows that the $I(T)$ architecture outperforms the complete cuboid (C) architecture on the resulting generated workload.

Since we expect short queries to be more frequent than long ones, it is advantageous to use the individual cell materialization strategy.

5.5 Storage performance evaluation

The choice of materialization strategy affects the storage efficiency of the system, which can be evaluated by looking either at the amount of storage space necessary to support a fixed set of queries or at the number of queries that can be answered when a fixed amount of storage space is used. In this section, the storage efficiency of the $I(T)$ storage architecture is compared to two partial materialization strategies that rely on full cuboid materialization: thin cube shells and shell fragments.

5.5.1 Thin cube shell

The thin cube shell approach, described in Sect. 2.3.2, relies on materializing all cells in all cuboids of a prescribed depth. As a result, the number of cuboids, and in turn the number

Table 6 Number of minimal queries having given costs and query lengths when using a 3D cuboid (cub) or $I(T)$ strategy

	Query length																			
	1		2		3		4		5		6		7		8		9		10	
Cost	Cub	$I(T)$	Cub	$I(T)$	Cub	$I(T)$	Cub	$I(T)$	Cub	$I(T)$	Cub	$I(T)$	Cub	$I(T)$	Cub	$I(T)$	Cub	$I(T)$	Cub	$I(T)$
1		3		3	7	1														
2			9	6		6			9		3									
3				3	15	12		9	15	3		3			5					
4	3					3	15	6	4	16	12	3				1			1	
5					9	6			9	9		6			3					
6			3			3			3	6	1	1				3				
7					1	1				6		3				4				1

Table 7 Average cost of answering query

Tag count	Average cost	
Prob distribution	C	I(T)
Uniform	3.7	4.1
Zero truncated Poisson $m = 1$	3.6	1.6
Zero truncated Poisson $m = 2$	3.5	2.3
Zero truncated Poisson $m = 3$	3.4	2.9
Geometric	3.6	1.9
Based on PubMed Query Log	3.0	1.3

of cells, that would need to be materialized grows rapidly as the number of dimensions increases. For a collection with T tags, thin cube shell materialization with d-dimensional cuboids requires $\binom{|T|}{d}(2^d - 1)$ cells to be materialized. In contrast, the $I(T)$ approach that supports all queries up to d tags requires $\sum_{i=1}^{d} \binom{|T|}{d}$ cells to be materialized.

For example, since NYT uses 1015 tags, there are $\binom{1015}{3}$ three-dimensional cuboids, which corresponds to 1.2×10^9 cells that would need to be materialized by the thin cube shell strategy. On the other hand, the $I(T)$ approach requires 1.7×10^8 cells to be materialized, which is 7 times less. However, for most of those tag combinations there are at most 50 corresponding documents, and often there are none at all. Therefore, there is no need to materialize all cells with the $I(T)$ approach or all cuboids with the thin cube shell approach. With the $I(T)$ architecture unnecessary cells can be easily pruned, and only 38,368 cells are required to answer all queries that involve up to three distinct tags and contain more than 50 documents (which corresponds to the sum of the first three columns in Table 2) or 60,100 cells are required to ensure all the conjunctive queries that produce a result set above the threshold size can be answered efficiently. On the other hand, the thin cube shell approach requires 25,010 3D *cuboids* to be stored, which corresponds to 175,070 cells (at 7 cells per cuboid). That is, the thin cube shell approach requires almost three times as many cells as the $I(T)$ approach if the cuboid size is chosen to be 3, and the multiplier gets larger as the prescribed cuboid size increases.

5.5.2 Shell fragments

The shell fragment approach, proposed by Li et al. [32] and described in Sect. 2.3.2, relies on materializing all cuboids for each fragment, where the fragments form a partitioning of the tags. For NYT's 1015 tags, we would need to store 338 fragments corresponding to tag triples and one fragment corresponding to the remaining tag, which would result in 6423 cells being stored (at 19 cells per fragment for the triples). This is 11 % of the size needed by the $I(T)$ storage approach that stores all 60,100 cells (and can answer all conjunction queries that produce result sets of size greater than 50), but that design is efficient only for queries including tags that are *all* found in the same fragment—at most 338 out of 20,905 tag triples that correspond to document sets larger than our threshold. When the tags specified in a query are not all found in the same fragment, the set of fragments that contain all the involved tags need to be intersected to identify which documents need to be aggregated online, and the response time might be unacceptable if the number of documents that need to be aggregated is above the threshold. If we choose to partition by six tags per fragment instead, we will

Table 8 Analysis of the number of conjunctive queries producing result sets above the threshold size that can be answered when 6D or 3D shell fragment materialization is used on (a) NYT (b) ACM

Query	Unique	6D shell fragment		3D shell fragment	
Length	Conjunctions	Count	%	Count	%
NYT					
1	1015	1015	100.00	1015	100.00
2	16,448	1002	6.09	564	3.43
3	20,905	585	2.80	119	0.57
4	12,217	280	2.29	0	0.00
5	5289	90	1.70	0	0.00
6	2401	13	0.54	0	0.00
7	1152	0	0.00	0	0.00
8	493	0	0.00	0	0.00
9	151	0	0.00	0	0.00
10	27	0	0.00	0	0.00
11	2	0	0.00	0	0.00
Total	60,100	2985	4.97	1698	2.83
ACM					
1	9098	9098	100.00	9098	100.00
2	14,262	2006	14.07	1199	8.41
3	5280	797	15.09	186	3.52
4	3860	350	9.07	0	0.00
5	3700	114	3.08	0	0.00
6	3199	17	0.53	0	0.00
7	2390	0	0.00	0	0.00
8	1520	0	0.00	0	0.00
9	776	0	0.00	0	0.00
10	297	0	0.00	0	0.00
11	79	0	0.00	0	0.00
12	13	0	0.00	0	0.00
Total	44,474	12,382	27.84	10,483	23.57

require 111,496 cells (86 % more than what is needed by $I(T)$), and still at most 170 out of 2401 important sextets of tags will appear within a single fragment.

The effectiveness of answering conjunctive queries of various length on NYT and ACM collections, when relying on shell fragment materialization, with fragment sizes of 6 and 3 are analyzed in Table 8. The fragments used are nonoverlapping and chosen using a greedy heuristic that builds fragments that can answer the longest conjunctions. In both the NYT and ACM collections only a very small percentage of conjunctions can be answered using the nonoverlapping shell fragments. Thus, the partial materialization generated by shell fragments cannot guarantee acceptable performance when tags co-occur with many other tags and there is a high order of tag co-occurrences, as is true in both the NYT and ACM collections.

6 Partial materialization strategies

Because each centroid term vector includes a (sum,count) pair for each of the m most significant terms found in the document collection (Sect. 3.1), and for our collections $m = 500$, the space for storing a single centroid can be as much as 4KB (if uncompressed). Even if centroids were compressed, they will still require considerable space. Therefore, it is worthwhile to avoid materializing cells as much as feasible.

To this end, three partial materialization strategies are proposed: threshold materialization (TM), threshold materialization with ancestors (TMA), and materialization of cluster centroids (MCC). For each materialization strategy, we give algorithms for choosing the centroids to materialize and for answering queries using those centroids with appropriate compensations when a requested cell centroid is not materialized.

6.1 Threshold materialization

Assuming that we can afford to access and aggregate at most k documents when computing a centroid (Sect. 5.1), we start by precomputing and materializing the centroids for all conjunctive queries for which the result contains at least k documents. We therefore need to identify which combinations of tags produce "cells of significant size" after intersection, as enumerated for Table 2. Algorithm 1 returns a list M of intersection cells that have more than k member documents.

The algorithm is based on the simple observation that including additional tags in a conjunctive query cannot increase the number of documents in the resulting intersection. It starts with all possible single tags, which correspond to 1D cells, and the set of documents associated with each tag. Using the method *augmentSet*(), it then repeatedly includes one more tag in the conjunction. (The method returns a list of sets, each augmenting the base set with one tag not already included in that base. To avoid repeated consideration, only tags that have a higher index than the maximum tag index in set u are included in the list of sets returned by $u.augmentSet(T)$.) The tag sets (together with their corresponding document sets) that have more than k documents and therefore require further exploration are kept in a queue L. Each time the number of documents in a cell exceeds the threshold k, it is included in the result set, the intersections with each remaining tag is computed, and the resulting augmented tags sets are enqueued on L for further consideration. The algorithm continues to examine cells with more and more intersecting tags until no further candidates have more than k documents.

Algorithm 1 TM: Find cells exceeding threshold filter

```
Input: Threshold $k$; Tags $T = \{(t_i,\ t_i \to S_{\{t_i\}})\}$
Returns: Set of (candidate) cells with their centroids
$M \leftarrow \emptyset, u \leftarrow \emptyset$
$L \leftarrow u.augmentSet(T)$                        ▷ start with a list of tag sets of size 1
while $|L| > 0$ do
    $u \leftarrow L.dequeue()$
    if $|S_u| > k$ then          ▷ include augmented tag sets that represent more than $k$ documents
        $M \leftarrow M \cup \{(u, C_u)\}$
        $L.enqueue(u.augmentSet(T))$                  ▷ ... and continue to check supersets of $u$
    end if
end while
return $M$
```

Given this partial materialization, the following steps are performed to evaluate a query:

1. Transform the query into an equivalent representation R using the inclusion–exclusion principle.
2. For each resulting conjunction, check if the corresponding cell has been materialized and, if so, retrieve the centroid measure.
3. For each of the nonmaterialized conjunctions:
   (a) determine the set of documents in the intersection (merge the tags' postings lists);
   (b) retrieve and aggregate the document term vectors to generate the corresponding centroid measure.
4. Combine the centroid measures in accordance with R.

6.2 Threshold materialization with ancestors

Table 3 shows that the set of documents found by intersecting a set of tags G is often equal to or very similar to the set found by intersecting tags in $G' \subset G$. With this insight, we extend Algorithm 1 to include the additional materialization constraint that the size of the symmetric difference between each cell and its closest materialized ancestor must be greater than k. This materialization approach, described by Algorithm 2, is designed to take advantage of the similarity between cells that involve similar tags. (The method $M.getClosestAncestor(u)$ retrieves the closest materialized ancestor for u as measured by symmetric difference.) Notice, however, that even if a cell is not materialized because it is similar to a materialized ancestor, some of its descendant cells might still require materialization; in this respect, the approach uses a greedy algorithm rather than finding the optimal set of cells to materialize. Nevertheless, for collections with a large amount of co-occurrence between tags, this approach will provide significant storage saving, and for collections with very little similarity, it will perform like the TM algorithm.

An additional lookup table is stored for this strategy, where for each nonmaterialized cell c having more than k documents, we store a pointer to the closest materialized ancestor a. Although this requires a small amount of space, it is far less than what is required to store a centroid.

Algorithm 2 TMA: Find cells exceeding threshold filter given materialized ancestors

```
Input: Threshold $k$; Tags $T = \{(t_i,\ t_i \rightarrow S_{\{t_i\}})\}$
Returns: Set of cells with their centroids
$M \leftarrow \emptyset, u \leftarrow \emptyset$
$L \leftarrow u.augmentSet(T)$
while $|L| > 0$ do
    $u \leftarrow L.dequeue()$
    if $|S_u| > k$ then                                  ▷ centroid might need to be materialized
        $a \leftarrow M.getClosestAncestor(u)$
        if $|S_a \triangle S_u| > k$ then       ▷ ... but not if the document set is sufficiently close to an ancestor
            $M \leftarrow M \cup \{(u, C_u)\}$
        else
            $M \leftarrow M \cup \{(u, *a)\}$                ▷ ... (in which case, just point at that ancestor)
        end if
        $L.enqueue(u.augmentSet(T))$
    end if
end while
return $M$
```

The following steps are now required to evaluate a query:

1. Transform the query into an equivalent representation R using the inclusion–exclusion principle.
2. For each resulting conjunction, check if it has been materialized and, if so, retrieve the centroid measure.
3. For each of the nonmaterialized conjunctions:
    (a) determine S_c, the set of documents in the intersection (merge postings lists for the given tags);
    (b) if $|S_c| \leq k$, retrieve and aggregate the document term vectors to generate the corresponding centroid measure.
    (c) if $|S_c| > k$:
        i. retrieve document member set S_{*a};
        ii. retrieve and aggregate the document term vectors in set $S_{*a} - S_c$ and call the result δC;
        iii. compute the centroid measure to be the value $C_{*a} - \delta C$.
4. Combine the centroid measures in accordance with R.

6.3 Materialization of cluster centroids

A third approach is to compute centroids for carefully selected document sets that do not necessarily correspond to cells in the data cube. Instead of depending on the closest materialized ancestor to provide an approximate centroid, it stores centroid measures of sets that do not correspond to any specific query but from which a result to a query can be derived. For document collections with little similarity among cells, this algorithm will materialize at most as many cells as Algorithm 1.

Algorithm 3 starts by calling the *candidateCells*() function (returning the sets of documents for the cells chosen to be materialized by Algorithm 1) to obtain the set M of document sets representing cells whose centroids cannot be computed by merely combining at most k document term vectors. Next, the method *closePairs*() initializes a priority queue Q that will contain (c_i, c_j, δ) triples, ordered by ascending δ, where $c_i, c_j \in M \wedge c_i \neq c_j$ and $\delta = |S_{c_i} \triangle S_{c_j}|$. In this method, all child and sibling relationships among pairs of cells are examined to identify those pairs (c_i, c_j) that have a low δ. This corresponds to initializing Q with the following candidate pairs: $c_i \subset c_j \wedge |c_j| - |c_i| = 1$ (parent–child relationship) or $|c_i| = |c_j| \wedge |c_i \triangle c_j| = 2$ (sibling relationship).

The algorithm then applies complete-link clustering [33] to find collections of highly overlapping document sets. Unlike traditional hierarchical clustering, however, we do not care about the order in which clusters are merged, as long as all the members of each cluster satisfy the complete linkage requirement that they are within a symmetric distance of $2k$ from the furthest member in the cluster (which ensures that all the sets' centroids can be computed from the cluster centroid by considering at most k documents). This relaxation in preserving the cluster hierarchy allows an efficient implementation for large numbers of cells by using a standard union-find algorithm [42]. To accomplish its clustering, the algorithm first invokes the method *initializeClusters*() to generate a disjoint set data structure G, where $G[M_i] = M_i$, that is used to track which cells (document sets) are assigned to which clusters (partitions). The method *getCluster*() retrieves the partition to which a specified cell belongs, *setCluster*() assigns a specified cell to a partition, and *p.maxDistance* () returns $\max_{c_i, c_j \in C}(|S_{c_i} \triangle S_{c_j}|)$.

Algorithm 3 returns a set of partitions $p_1, \ldots, p_n$, where each partition p_i represents a cluster of cells $S_{(i,1)}, \ldots, S_{(i,i_n)}$ and each $S_{(i,j)}$ corresponds to a conjunction of tags. For

Algorithm 3 MCC: Find clusters for materialization

```
Input: Threshold $k$; Tags $T = \{(t_i,\ t_i \to S_{\{t_i\}})\}$
Returns: Set of clusters with their centroids
$M \leftarrow candidateCells(k, T)$        ▷ use Algorithm 1 to identify all sets requiring materialization
$Q \leftarrow M.closePairs()$        ▷ collect pairs of sets with small symmetric distance
$G.initializeClusters(M)$        ▷ every candidate cell is initially in its own cluster
while $|Q| > 0$ do
    $q \leftarrow Q.dequeue()$        ▷ greedily merge clusters using union-find
    $p_1 \leftarrow G.getCluster(q.c1)$
    $p_2 \leftarrow G.getCluster(q.c2)$
    if $p_1 \neq p_2$ then
        $p_u \leftarrow p_1 \cup p_2$
        if $p_u.maxDistance() \leq 2k$ then        ▷ if all pairs are within $2k$, merge clusters
            $G[q.c1].setCluster(p_u)$
            $G[q.c2].setCluster(p_u)$
            $M \leftarrow M - \{p_1\} - \{p_2\}$
            $M \leftarrow M \cup \{p_u\}$
        end if
    end if
end while
return $M$
```

each partition p_i, we determine a set of documents S_{p_i}, that is, within distance k of each $S_{(i,j)}$ (which must exist since no two documents in the partition are further than $2k$ apart). The term centroid C_{p_i} for this "artificial cell" is then calculated by aggregating together all the document term vectors found in S_{p_i}.

This strategy requires all the C_{p_i} measures to be materialized and the required cell centroids to be computed based on the closest materialized artificial cell. To accomplish this, we store a table with four attributes: *cell, cluster, docsToAdd, docsToRemove*; where *cell* corresponds to a cell representing a conjunction of tags, *cluster* is the centroid for the partition to which that cell is assigned, *docsToAdd* is the set of document IDs in the cell but missing when computing the partition centroid, and *docsToRemove* is the set of document IDs included when computing the partition's centroid but missing from the cell. The documents in *docsToAdd* and *docsToRemove* need to be retrieved and aggregated to the partition's centroid measure to determine the centroid for the cell. By construction, $|docsToAdd| + |docsToRemove| \leq k$.

Thus, at query time the following steps are performed to evaluate a query:

1. Transform the query into the equivalent representation R using the inclusion–exclusion principle.
2. For each resulting conjunction j, find the cluster centroid measure C_{p_j} from the cluster reference table.
3. If there is no match, retrieve and aggregate the document term vectors to generate the corresponding centroid measure.
4. Otherwise, retrieve the documents included in the *docsToAdd* and *docsToRemove* attributes and aggregate them with C_{p_j}.
5. Combine the centroid measures in accordance with R.

6.4 Comparative example of materialization strategies

In this section we demonstrate how the use of different materialization strategies affects which sets of documents have their centroids materialized, and how these materialized centroids are used to answer queries over tags. For simplicity we assume a document collection that consists

Table 9 Tag assignment to documents

	d_1	d_2	d_3	d_4	d_5	d_6
t_1	✓	✓	✓	✓	✓	✓
t_2	✓	✓	✓	✓	✓	
t_3	✓	✓	✓			✓

Table 10 Materialized cells for TM, TMA, and MCC materialization strategies

	TM	TMA	MCC
$S_1 = \{t_1\}$	$\{d_1, d_2, d_3, d_4, d_5, d_6\}$	$\{d_1, d_2, d_3, d_4, d_5, d_6\}$	
$S_2 = \{t_2\}$	$\{d_1, d_2, d_3, d_4, d_5\}$	$\{d_1, d_2, d_3, d_4, d_5\}$	
$S_3 = \{t_3\}$	$\{d_1, d_2, d_3, d_6\}$	$\{d_1, d_2, d_3, d_6\}$	
$S_4 = \{t_1, t_2\}$	$\{d_1, d_2, d_3, d_4, d_5\}$		
$S_5 = \{t_1, t_3\}$	$\{d_1, d_2, d_3, d_6\}$		
$S_6 = \{t_2, t_3\}$	$\{d_1, d_2, d_3\}$		
$S_7 = \{t_1, t_2, t_3\}$	$\{d_1, d_2, d_3\}$		
S_a			$\{d_1, d_2, d_3, d_4\}$

of six documents: $d_1, \ldots, d_6$; a set of 3 tags: t_1, t_2, t_3, whose assignment to documents is specified in Table 9; and materialization threshold $k = 2$.

Table 10 shows the document sets for which centroids are stored for each of the three materialization strategies. The TM approach materializes centroids for all documents sets of size greater than 2 after considering every conjunction of tags, which results in materializing centroids for seven documents sets. With TMA, only three document set centroids are materialized (corresponding to document sets for single tag queries: t_1, t_2, and t_3), since the other conjunctions are descendants of them and the size of the symmetric difference between them and the materialized sets is no more than the threshold 2. With MCC only a single centroid is materialized (for a set of documents corresponding to no tag conjunction), from which the centroids of all the conjunctive queries can be derived with no more than 2 DTVs involved.

Table 11 demonstrates how the materialized centroids produced by each of the three materialization strategies are used to derive a centroid for various conjunctive queries over tags. Since with the TM strategy the materialized centroids match the document sets requested by the queries, no further work needs to be performed to produce the required results. When relying on the TMA and MCC strategies, the centroid for a query answer is derived by aggregating the centroid of the closest materialized document set with DTVs from individual documents.

7 Performance of partial materialization

Our approach to partial materialization was designed to perform well for the tagging patterns we observed in NYT and ACM. In this section, we show that it indeed does well for those collections, as well as for PubMed, where PubMed's average tag count per document is between the NYT and ACM; the number of documents and total number of tags are larger than NYT, even when considering only the major tags; and the depth of tag co-occurrence is shallower than for either of the other two collections (compare Table 12 to Table 2).

Table 11 Answering conjunctive queries with TM, TMA, and MCC materialization strategies

Query	TM	TMA	MCC
t_1	C_1	C_1	$C_a + d_5 + d_6$
t_2	C_2	C_2	$C_a + d_5$
t_3	C_3	C_3	$C_a - d_4 + d_6$
$t_1 \wedge t_2$	C_4	C_2	$C_a + d_5$
$t_1 \wedge t_3$	C_5	C_3	$C_a - d_4 + d_6$
$t_2 \wedge t_3$	C_6	$C_3 - d_6$	$C_a - d_4$
$t_1 \wedge t_2 \wedge t_3$	C_7	$C_3 - d_6$	$C_a - d_4$

Table 12 Number of conjunctions of n tags that contribute to high multi-way co-occurrence for PubMed, with threshold limit of 50

n	1	2	3	4	5	6	7	8	9	10	11	12	Total
PubMed	114,946	218,596	38,392	7410	1305	221	43	5					380,918

In order to demonstrate the approach's wider applicability, we also developed a generative model for tagged collections [13], which we have used to synthesize a variety of realistic collections. In particular, we adapted earlier work on topic modeling [4] to generate tags instead of document content and then fit the model to NYT and ACM, resulting in NYT model and ACM model, respectively. By substituting other values for some of the parameters of the model (including the topic distribution vector, topic correlations, tag distribution for each topic, distribution of tag counts for each topic, and minimum topic presence per document to warrant a tag), we generated six additional synthetic collections (see Table 13). For example, reducing the minimum number of mentions of a topic to warrant using a tag induces more tags per document; modifying the entries in the topic covariance matrix affects the number of times certain topics co-occur in documents; increasing the mean of the tag counts in the topic distribution induces more tags per topic and thus more tags per document; and lowering the γ parameter that defines the Dirichlet distribution from which tag distribution for topics is drawn increases the dominance of the most popular tags. We display the resulting effects on tag distributions at the bottom of Table 13.

In the remainder of this section, the performance of the three partial materialization strategies (TM, TMA, and MCC, which are based on the proposed $I(T)$ storage architecture defined in Sect. 5.3.2) are compared against six other materialization strategies. Specifically we compare them against the shell fragment materialization (SF) with both 3D and 6D cuboids,[5] thin cube shell materialization (CS) with both 3D and 6D cuboids, materialization of hierarchical clusters used by Scatter/Gather (SG), and a standard IR approach, which is equivalent to the strategy with no materialization (NM).

Shell fragments are nonoverlapping, and for the evaluation they are chosen using a greedy heuristic that builds fragments that can answer the longest conjunctions. Similarly, thin cube shells were generated by a greedy algorithm tuned to minimize the number of stored cuboids while ensuring that all conjunctions of less than or equal to a fixed length can be answered from materialized cuboids. For both the shell fragment and thin cube approaches,

[5] For the ACM model collection the longest conjunctive queries are of length 5 and so instead of materializing shell fragments and shell cuboids of size 6, we materialize cuboids of size 5 so that they are not penalized by storing unnecessary data.

Table 13 Input parameters and resulting tagging patterns for synthetic collections

	NYT Model	ACM Model	S1	S2	S3	S4	S5	S6
Topic count (K)	25	70	25	70				
Unique tag count	1015	6109	1015	6109				
Document count	145,701	66,041	145,701	66,041				
Topics' tag distrib.	Inferred from NYT	Inferred from ACM	*Dirichlet* ($\gamma = 0.005$)	*Dirichlet* ($\gamma = 0.001$)				*Dirichlet* ($\gamma = 0.0005$)
Topic means distrib.	Inferred from NYT	Inferred from ACM	*Dirichlet* ($\varphi = 20$)	*Dirichlet* ($\varphi = 20$)				
Topic covariance matrix	Inferred from NYT	Inferred from ACM	No correlation	No correlation			Triple tag correl.	No correlation
Tag count per ropic	Inferred from NYT	Inferred from ACM	*zero trunc. Poisson*, $\lambda \in$ [0.08,2.07]	*zero trunc. Poisson*, $\lambda \in$ [0.06,2.86]		*zero trunc. Poisson*, $\lambda \in$ [1.06,3.86]	*zero trunc. Poisson*, $\lambda \in$ [0.06,2.86]	
Min. presence req'd for tag	11 %	9 %	11 %	9 %	7 %	9 %		

Materialization threshold k=5								
Query length	NYT Model	ACM Model	S1	S2	S3	S4	S5	S6
1	812	4890	560	1532	1583	1758	1669	1040
2	15,875	15,675	22,022	14,259	21,776	54,204	28,692	16,258
3	18,633	2836	26,775	6266	10,759	103,073	49,951	9606
4	8118	213	5516	1455	2962	95,189	38,030	2849
5	1872	7	448	133	348	51,913	14,127	605
6	234		16	6	12	15,784	2201	76
7	18		1			2895	95	7
8						449		
9						70		
10						5		
Total	45,562	23,621	55,338	23,651	37,440	325,340	134,765	30,441

each d-dimensional cuboid records its d dimensions and a centroid measure. In addition, both approaches also require a mechanism to map tags to the corresponding dimension column of the cuboid. If there is no cuboid that contains all the tags found in a query, that query is answered by accessing the postings list of documents with each tag (as is true when no appropriate centroid is materialized in using any of the strategies).

The LAIR2 [27] implementation of Scatter/Gather was designed to support a browsing interface in which exploration of the collection starts from a root set of document clusters and then follows various paths offered by the stored cluster hierarchy. To support this interface, it is necessary to store the hierarchy, as well as the centroid, representative documents, and most frequent terms for each cluster that corresponds to a node in the hierarchy. This infrastructure, however, does not provide support for efficient computation of a centroid for a set of documents that results from a conjunctive tag query, since it is extremely unlikely that the query result set matches any of the stored document sets exactly. As a result, Scatter/Gather must revert to online aggregation from the base documents like standard IR approaches.

The performance of each of the nine strategies is evaluated in terms of the amount of storage space it consumes and the execution cost of answering queries. Precision is not a consideration, since all matches are exact, as required by our problem definition.

7.1 Storage cost

Table 14 shows the storage space consumed by each of the materialization strategies. Every strategy, including NM (no materialization), requires postings lists to map from tags to sets of documents associated with those tags, in order to answer queries that must rely on the base data (or to compensate for document sets that do not match the stored centroids exactly when using the $I(T)$ strategies). Thus the first column reflects the space for the postings lists alone. The remaining columns show the additional space used by each algorithm to store the materialized cells and other required supporting data when a centroid is represented by 500 terms. Figure 7 depicts the storage requirement as a multiplier with respect to the space used by TM.

Recall that, in addition to the cluster centroids, MCC must store a cluster reference table with as many as k document IDs for every cell that would be materialized by TM. In practice, many fewer IDs need to be stored: The mean number of documents in the difference between an unmaterialized cell and the corresponding cluster centroid for NYT is 8.1, it is 0.75 for ACM, and 4.6 for PubMed; for NYT model it is 0.9, and it is 0.9 for ACM model; it is similarly low for the other six synthetic collections. As a result, even with the additional cost of storing the cluster reference table, MCC requires less space than either TM or TMA.

The shell fragment strategy using 3D cuboids consumes less space than other partial materialization strategies for all collections other than ACM, ACM model, and PubMed, but it exhibits very poor query time performance, as shown in the next section.

In order to save clustering time, Scatter/Gather requires space that is proportional to the collection size but independent of the tag patterns, so it performs relatively well on small collections with rich tag structures, such as S4. To provide a fair comparison to our approaches, we applied a similar partial materialization strategy for storing the clusters in the hierarchy: Only nodes exceeding the predefined size threshold k for each collection are materialized. As a result, the number of nodes that need to be materialized for Scatter/Gather depends on

Table 14 Storage cost comparison for materialization strategies (megabytes)

	NM	TM	TMA	MCC	CS3	CS6	SF3	SF6	SG (range)
NYT	18	248	129	104	686	2252	42	447	[130, 5603]
ACM	2	172	97	81	401	2017	222	3849	[52, 255]
PubMed	462	1918	1812	1734	6143	20,644	3,246	49,106	[2051, 79,888]
NYT model	2	176	140	119	687	2055	21	344	[114, 560]
ACM model	2	92	90	77	422	*860	120	*790	[53, 255]
S1	2	213	198	173	982	2501	16	238	[114, 561]
S2	1	91	80	68	400	1047	38	649	[52, 255]
S3	2	145	123	101	631	1611	40	672	[52, 255]
S4	2	1245	775	525	3206	18,475	45	746	[53, 256]
S5	2	517	390	301	1655	8813	42	708	[53, 255]
S6	1	117	90	71	464	1115	26	440	[52, 254]

*5D cuboids used for ACM model for both CS6 and SF6

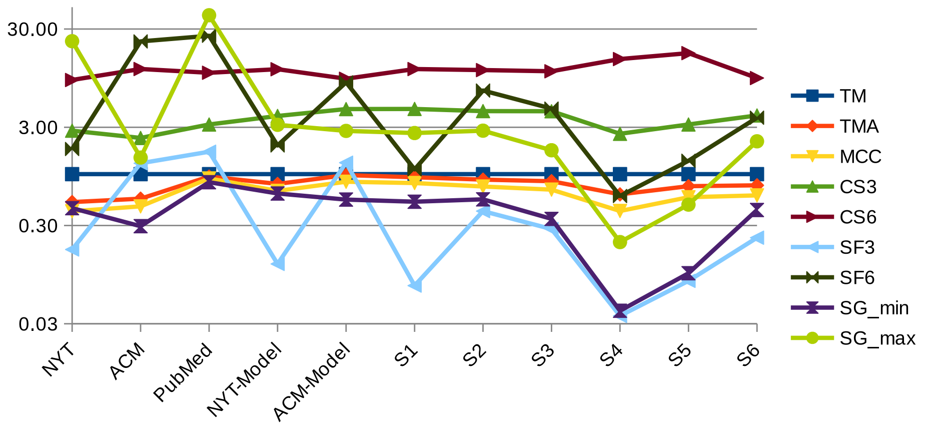

Fig. 7 Space multiplier relative to TM

the form of the dendogram representing the cluster hierarchy: If k individual documents are always clustered before any two clusters are merged, $\frac{N}{k}$ clusters would need to be stored, but in the worst case, the dendogram is essentially linear and $N - k$ clusters might need to be materialized. The corresponding ranges of space that bracket the actual space that would be required by Scatter/Gather are shown in the final column of Table 14. Scatter/Gather does not scale well to very large collections such as the NYT and PubMed, where for NYT (*at least* 25 %) more space is required and for PubMed (*at least* 18 %) more space is required than when using MCC . For the ACM collection the partially materialized cluster hierarchy for LAIR2 might be competitive in space with MCC, but when we performed hierarchical clustering on the documents in the ACM collection using single linkage and cosine similarity, we found that 208MB would be required; this is closer to the upper bound of the range than to the lower bound and 2.6 times as much space as required by MCC. We hypothesize that for the other collections the actual storage space used for LAIR2 would be similarly much higher than the theoretical lower bound.

In summary, the TM, TMA, and MCC approaches, which are based on the individual cell materialization architecture $I(T)$, typically require fewer cells to be stored, and thus less storage space, than alternative approaches, and among these three, the MCC approach is the most storage efficient.

7.2 Query execution cost

Table 15 and Fig. 8 show the number of cell accesses required to answer all possible conjunctive queries having a result that contains more than k documents. The TM and Scatter/Gather strategies are not included in the table. TM precalculates and materializes all answers to such queries, and hence its cost is universally 1 (with standard deviation 0). The Scatter/Gather approach, on the other hand, has similar query time performance for retrieving centroids of nodes stored in its cluster hierarchy, as would be expected from a fully materialized solution; however, it was not designed to calculate centroids for document sets that do not map to any of the nodes found in the hierarchy. Therefore, for almost all conjunctive queries Scatter/Gather does not have the necessary infrastructure to compute centroids from its materialized data, and therefore, it must resort to the approach with no materialization. As a result the mean number of cells accessed per query across all queries for SG is just marginally less than the figures shown in the NM column of the table.

Table 15 Mean number of cells aggregated per query (with standard deviations σ)

Collection	# Queries	TMA	MCC	CS3	CS6	SF3	SF6	NM
NYT	60,100	$4.9_{\sigma=9}$	$9.1_{\sigma=14}$	$38_{\sigma=88}$	$11_{\sigma=11}$	$162_{\sigma=533}$	$152_{\sigma=432}$	$238_{\sigma=1461}$
ACM	44,474	$1.4_{\sigma=1}$	$1.8_{\sigma=1}$	$3.6_{\sigma=2}$	$14_{\sigma=11}$	$5.9_{\sigma=6}$	$5.5_{\sigma=6}$	$12_{\sigma=37}$
PubMed	380,918	$2.3_{\sigma=6}$	$5.6_{\sigma=12}$	$4.5_{\sigma=17}$	$20_{\sigma=9}$	$73_{\sigma=123}$	$67_{\sigma=114}$	$265_{\sigma=964}$
NYT model	45,562	$1.4_{\sigma=1}$	$1.9_{\sigma=1}$	$3.5_{\sigma=9}$	$10_{\sigma=6}$	$18_{\sigma=51}$	$16_{\sigma=49}$	$28_{\sigma=233}$
ACM model	23,621	$1.1_{\sigma=0}$	$1.9_{\sigma=2}$	$2.3_{\sigma=1}$	$*9.1_{\sigma=4}$	$7.4_{\sigma=8}$	$*6.9_{\sigma=7}$	$19_{\sigma=49}$
S1	55,338	$1.2_{\sigma=1}$	$1.8_{\sigma=1}$	$2.3_{\sigma=4}$	$11_{\sigma=5}$	$18_{\sigma=41}$	$16_{\sigma=30}$	$28_{\sigma=181}$
S2	23,651	$1.3_{\sigma=1}$	$2.0_{\sigma=2}$	$2.3_{\sigma=2}$	$14_{\sigma=6}$	$11_{\sigma=16}$	$8.7_{\sigma=12}$	$23_{\sigma=75}$
S3	37,440	$1.3_{\sigma=1}$	$2.1_{\sigma=2}$	$2.4_{\sigma=3}$	$13_{\sigma=6}$	$11_{\sigma=17}$	$9.2_{\sigma=13}$	$20_{\sigma=69}$
S4	325,340	$1.8_{\sigma=1}$	$2.5_{\sigma=1}$	$5.5_{\sigma=7}$	$7.0_{\sigma=5}$	$13_{\sigma=23}$	$12_{\sigma=19}$	$14_{\sigma=44}$
S5	134,765	$1.5_{\sigma=1}$	$2.3_{\sigma=2}$	$4.3_{\sigma=5}$	$8.1_{\sigma=5}$	$13_{\sigma=21}$	$12_{\sigma=19}$	$16_{\sigma=51}$
S6	30,441	$1.4_{\sigma=1}$	$2.1_{\sigma=1}$	$2.7_{\sigma=4}$	$13_{\sigma=6}$	$12_{\sigma=19}$	$8.7_{\sigma=12}$	$22_{\sigma=83}$

*5D cuboids used for ACM model for CS6 and SF9

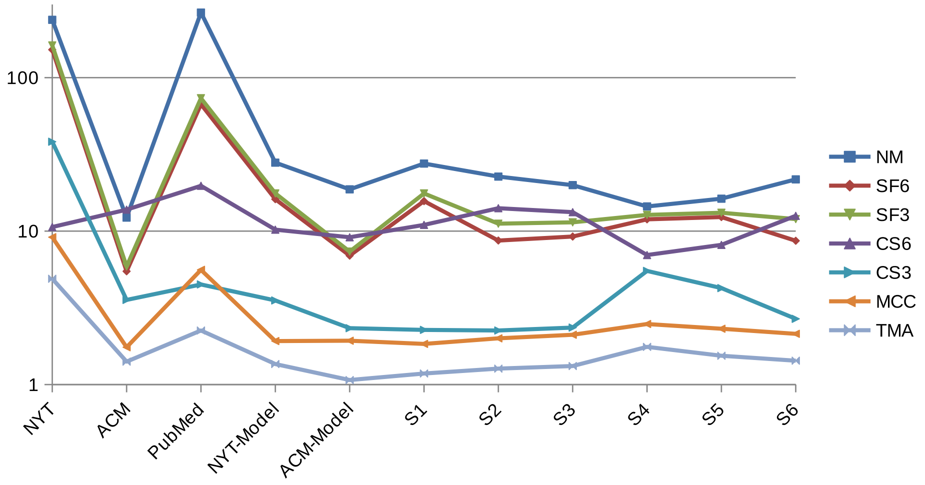

Fig. 8 Mean number of cells accessed per query

Since the query time is proportional to the average number of cells that need to be aggregated, these results represent comparative run times when each centroid is equally likely to be requested. By design, all three materialization strategies based on individual cell materialization (TM, TMA, MCC) have a computation cost that is less than the collection-specific computation threshold k and less than when using thin cube shells or shell fragments. On the other hand, when no cells are materialized (NM)—when IR is used alone—or when using Scatter/Gather with or without prematerialization, the computational costs far exceed acceptable response times.

Thus, summarizing space and time, all three $I(T)$ approaches have excellent query time performance (low mean and small standard deviation) while consuming generally the least amount of storage compared to the other materialization techniques. Furthermore, the three options provide a good space–time tradeoff, with TM being the largest and fastest and MCC being the smallest but slowest for all 11 data collections (see Fig. 9, where TM resides at the

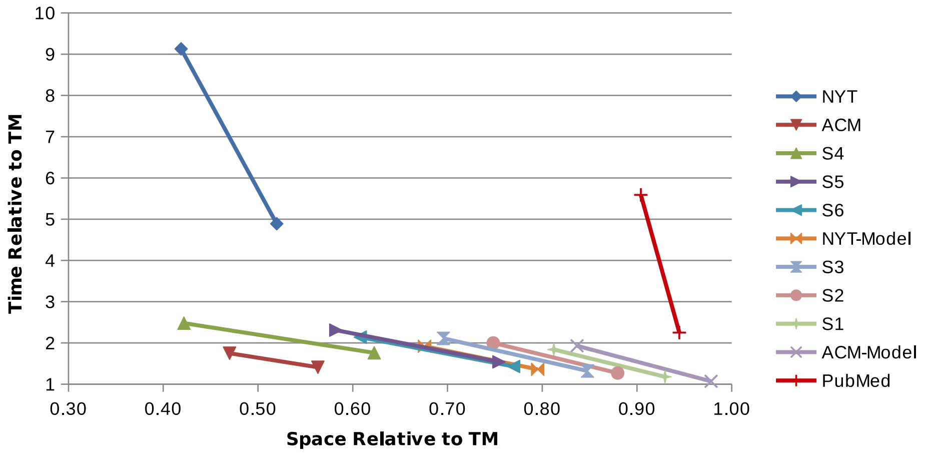

Fig. 9 Space–time tradeoff for $I(T)$ strategies, where all points are relative to TM (for each collection, the *upper left point* represents MCC and the *lower right point* represents TMA)

lower right-hand corner of the graph—at (1.00, 1)—for each collection). Scatter/Gather can be fast, but only if it consumes an unacceptable amount of space for large collections and only if the query workload is severely restricted to correspond to the sets of documents that happen to be chosen by the clustering algorithm. Shell fragments with 3D cuboids can be space-efficient, but only at the expense of unacceptably slow execution.

8 Conclusions and future work

The tagging patterns found in two multi-tagged document collections were analyzed and found to have properties that are not common in data for which standard OLAP systems have been designed. In particular, multi-tagged document collections include many distinct tags and significant co-occurrence among tags. Based on the observed properties of the collections, we propose a storage architecture based on materializing individual cells from aggregated data cubes and taking into account cell similarities. The proposed partial materialization strategies were evaluated on three real collections and eight synthetic collections, comparing them against previously proposed partial materialization strategies. We showed that the new strategies can provide significant savings in storage space and query time over the existing strategies and that some space–time tradeoff remains to choose between TM, TMA, and MCC.

In this paper we focused on efficient storage and calculation of the set centroid measure, which is an essential input for deriving a medoid, a larger set of representative documents, or a set of representative terms. We will continue to explore how to use this infrastructure to build a prototype browser that supports exploration of multi-tagged document collections through efficient navigation based on tags [14].

Acknowledgments Financial assistance from NSERC's PGS program, NSERC's Business Intelligence Network, Mitacs, and the University of Waterloo is greatly appreciated. We thank Lukasz Golab, Evangelos Milios, Wayne Oldford, and the anonymous reviewers for their suggestions, encouragement, guidance, and feedback.

References

1. Ames M, Naaman M (2007) Why we tag: motivations for annotation in mobile and online media. In: Proceedings of ACM SIGCHI conference on human factors in computing systems, ACM, CHI '07, pp 971–980
2. Baeza-Yates RA, Ribeiro-Neto B (1999) Modern information retrieval. Addison-Wesley, Reading
3. Bi B, Cho J (2013) Automatically generating descriptions for resources by tag modeling. In: Proceedings of 22nd ACM international conference on information and knowledge management, ACM, CIKM '13, pp 2387–2392
4. Blei DM, Lafferty JD (2007) A correlated topic model of science. Ann Appl Stat 1:17–35
5. Brooks CH, Montanez N (2006) Improved annotation of the blogosphere via autotagging and hierarchical clustering. In: Proceedings of 15th international conference on World Wide Web, ACM, WWW '06, pp 625–632
6. Budzik J, Hammond K (1999) Watson: anticipating and contextualizing information needs. In: 62nd annual meeting of the American Society for Information Science, pp 727–740
7. Cattuto C, Barrat A, Baldassarri A, Schehr G, Loreto V (2009) Collective dynamics of social annotation. Proc Natl Acad Sci USA 106(26):10,511–10,515
8. Côté R, Reisinger F, Martens L, Barsnes H, Vizcaino JA, Hermjakob H (2010) The ontology lookup service: bigger and better. Nucleic Acids Res 38:W155–W160
9. Cutting DR, Pedersen JO, Karger DR, Tukey JW (1992) Scatter/Gather: a cluster-based approach to browsing large document collections. In: Proceedings of 15th annual international ACM SIGIR conference on research and development in information retrieval, pp 318–329
10. Cutting DR, Karger DR, Pedersen JO (1993) Constant interaction-time Scatter/Gather browsing of very large document collections. In: Proceedings of 16th annual international ACM SIGIR conference on research and development in information retrieval, pp 126–134
11. Das D, Martins AFT (2007) A survey on automatic text summarization. Technical report, literature survey for the language and statistics II course at Carnegie Mellon University
12. Deolalikar V (2014) Distance or coverage? Retrieving knowledge-rich documents from enterprise text collections. In: Proceedings of 23rd ACM international conference information and knowledge management, pp 1771–1774
13. Drzadzewski G (2015) Online analytical systems for multi-tagged document collections. PhD thesis, Cheriton Sch. Comp. Sci., Univ. Waterloo (in preparation)
14. Drzadzewski G, Tompa FW (2015) Enhancing exploration with a faceted browser through summarization. In: Proceedings of 15th ACM SIGWEB international symposium on document engineering, ACM, DocEng '15
15. Ebbert JO, Dupras DM, Erwin PJ (2003) Searching the medical literature using PubMed: a tutorial. Mayo Clin Proc 78(1):87–91
16. Efthimiadis EN (1995) User choices: a new yardstick for the evaluation of ranking algorithms for interactive query expansion. Inf Process Manag 31(4):605–620
17. Fagan JC (2013) Usability studies of faceted browsing: a literature review. Inf Technol Libr 29(2):58–66
18. Gelbukh AF, Alexandrov M, Bourek A, Makagonov P (2003) Selection of representative documents for clusters in a document collection. In: Proceedings of 8th international conference on applied natural language information systems, pp 120–126
19. Goorha S, Ungar L (2010) Discovery of significant emerging trends. In: Proceedings of 16th ACM SIGKDD international conference on knowledge discovery and data mining, ACM, KDD '10, pp 57–64
20. Gray J, Chaudhuri S, Bosworth A, Layman A, Reichart D, Venkatrao M, Pellow F, Pirahesh H (1997) Data cube: a relational aggregation operator generalizing group-by, cross-tab, and sub-totals. Data Min Knowl Discov 1(1):29–53
21. Gupta V, Lehal GS (2010) A survey of text summarization extractive techniques. J Emerg Technol Web Intell 2(3):258–268
22. Harinarayan V, Rajaraman A, Ullman JD (1996) Implementing data cubes efficiently. In: Proceedings of 1996 ACM SIGMOD international conference management data, ACM, SIGMOD '96, pp 205–216

23. Hearst MA (2006) Clustering versus faceted categories for information exploration. Commun ACM 49(4):59–61
24. Hearst MA (2009) Search user interfaces, 1st edn. Cambridge University Press, Cambridge
25. Jansen BJ, Spink A, Saracevic T (2000) Real life, real users, and real needs: a study and analysis of user queries on the web. Inf Process Manag 36(2):207–227
26. Jin X, Han J, Cao L, Luo J, Ding B, Lin CX (2010) Visual cube and on-line analytical processing of images. In: Proceedings of 19th ACM international conference on information and knowledge management, ACM, CIKM '10, pp 849–858
27. Ke W, Sugimoto CR, Mostafa J (2009) Dynamicity vs. effectiveness: studying online clustering for Scatter/Gather. In: Proceedings of 32nd annual international ACM SIGIR conference on research and development in information retrieval , pp 19–26
28. Kim YM, Rieh SY (2011) User perceptions of the role and value of tags. In: Proceedings of ACM SIGCHI conference on human factors in computing systems, ACM, CHI '11, pp 671–674
29. Kipp ME, Campbell DG (2010) Searching with tags: do tags help users find things? Knowl Organ 37(4):239–255
30. Lakshmanan LVS, Pei J, Han J (2002) Quotient cube: how to summarize the semantics of a data cube. In: Proceedings of 28th international conference on very large data bases, VLDB endowment, VLDB '02, pp 778–789
31. Lemire D, Kaser O, Aouiche K (2010) Sorting improves word-aligned bitmap indexes. Data Knowl Eng 69(1):3–28
32. Li X, Han J, Gonzalez H (2004) High-dimensional OLAP: a minimal cubing approach. In: Proceedings of 30th international conference on very large data bases, VLDB endowment, VLDB '04, pp 528–539
33. Manning CD, Raghavan P, Schütze H (2008) Introduction to information retrieval. Cambridge University Press, Cambridge
34. Millen DR, Feinberg J, Kerr B (2006) Dogear: social bookmarking in the enterprise. In: Proceedings of ACM SIGCHI conference on human factors in computing systems, ACM, CHI '06, pp 111–120
35. Mosa ASM, Yoo I (2013) A study on PubMed search tag usage pattern: association rule mining of a full-day PubMed query log. BMC Med Inf Decis Mak 13(1):8
36. Mu X, Lu K, Ryu H (2014) Explicitly integrating MeSH thesaurus help into health information retrieval systems: an empirical user study. Inf Process Manag 50(1):24–40
37. Popescul A, Ungar LH (2000) Automatic labeling of document clusters. Available at: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.33.141
38. Pratt W, Hearst MA, Fagan LM (1999) A knowledge-based approach to organizing retrieved documents. In: Proceedings of 16th national conference on artificial intelligence and 11th conference on innovation and application of artificial intelligence, pp 80–85
39. Radev DR, Jing H, Styś M, Tam D (2004) Centroid-based summarization of multiple documents. Inf Process Manag 40(6):919–938
40. Ramdeen S, Hemminger BM (2012) A tale of two interfaces: How facets affect the library catalog search. J Am Soc Inf Sci Technol 63(4):702–715
41. Sandhaus E (2008) The New York Times Annotated Corpus. http://www.ldc.upenn.edu/Catalog/catalogEntry.jsp?catalogId=LDC2008T19
42. Sedgewick R, Wayne K (2011) Algorithms, 4th edn. Addison-Wesley, Reading
43. Shiri A, Ruecker S, Doll L, Bouchard M, Fiorentino C (2011) An evaluation of thesaurus-enhanced visual interfaces for multilingual digital libraries. In: Gradmann S, Borri F, Meghini C, Schuldt H (eds) Proceedings of international conference on theory and practice ofdigital libraries, TPDL 2011, Berlin, Germany, September 26–28.Lecture notes on computer science,vol 6966. Springer, pp 236–243
44. Sismanis Y, Deligiannakis A, Roussopoulos N, Kotidis Y (2002) Dwarf: shrinking the petacube. In: Proceedings of 2002 ACM SIGMOD international conference on managing data, ACM, SIGMOD '02, pp 464–475
45. Tunkelang D (2009) Faceted search. Synthesis lectures on information concepts, retrieval, and services. Morgan & Claypool, San Rafael
46. Wang W, Feng J, Lu H, Yu J (2002) Condensed cube: an effective approach to reducing data cube size. In: Proceedings of 18th international conference on data engineering, ICDE '02, pp 155–165
47. White J (2001) ACM opens portal to computing literature. Commun ACM 44(7):14–16, 28
48. Witten IH, Paynter GW, Frank E, Gutwin C, Nevill-Manning CG (1999) Kea: practical automatic keyphrase extraction. In: Proceedings of 4th ACM conference on digital library, ACM, DL '99, pp 254–255
49. Yee K, Swearingen K, Li K, Hearst MA (2003) Faceted metadata for image search and browsing. In: Proceedings of ACM SIGCHI conference: human factors in computing systems, pp 401–408

50. Zamir O, Etzioni O (1999) Grouper: A dynamic clustering interface to web search results. In: Proceedings of 8th international conference on World Wide Web, Elsevier, WWW '99, pp 1361–1374
51. Zhang D, Zhai C, Han J (2011) Mitexcube: microtextcluster cube for online analysis of text cells. In: Proceedings of 2011 conference on intelligent data understanding, CIDU'11, pp 204–218
52. Zhang J, Marchionini G (2005) Evaluation and evolution of a browse and search interface: relation browser. In: Proceedings of 2005 national conference on digital governance and research, pp 179–188

Grzegorz Drzadzewski is a Ph.D. candidate in the David R. Cheriton School of Computer Science at the University of Waterloo and a member of Waterloo's Data Systems Group. His research is focused on topics that include partial materialization, OLAP on text, text mining, text summarization, and faceted browsing. He received his M.Sc. in Computer Science from the University of Guelph and B.A.Sc in Computer Engineering from the University of Toronto.

Frank Wm. Tompa is a Distinguished Professor Emeritus in the David R. Cheriton School of Computer Science at the University of Waterloo and a member of Waterloo's Data Systems Group. His research interests include the design of flexible and efficient text management systems that are effective for maintaining large reference texts and large, heterogeneous text collections. He is a co-founder of Open Text Corporation, a Fellow of the ACM, and a recipient of Canada's Queen Elizabeth II Diamond Jubilee Medal, and he holds an honorary degree from Dalhousie University.