Context-sensitive Ranking for Document Retrieval*

Liang Jeff Chen
UC San Diego
La Jolla, CA, US
jeffchen@cs.ucsd.edu

Yannis Papakonstantinou
UC San Diego
yannis@cs.ucsd.edu

ABSTRACT

We study the problem of context-sensitive ranking for document retrieval, where a context is defined as a sub-collection of documents, and is specified by queries provided by domain-interested users. The motivation of context-sensitive search is that the ranking of the same keyword query generally depends on the context. The reason is that the underlying keyword statistics differ significantly from one context to another. The query evaluation challenge is the computation of keyword statistics at runtime, which involves expensive online aggregations. We appropriately leverage and extend materialized view research in order to deliver algorithms and data structures that evaluate context-sensitive queries efficiently. Specifically, a number of views are selected and materialized, each corresponding to one or more large contexts. Materialized views are used at query time to compute statistics which are used to compute ranking scores. Experimental results show that the context-sensitive ranking generally improves the ranking quality, while our materialized view-based technique improves the query efficiency.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Retrieval models*; H.2.4 [**Database Management**]: Systems—*Query processing*

General Terms

Algorithms, Performance

Keywords

Context-sensitive ranking, materialized views, view selection

1. INTRODUCTION

While research in information retrieval (IR) has generated many effective ranking models for general-purpose search, existing ranking models may not deliver satisfactory rankings for domain experts. In this paper, we propose a new query model that allows expert users to specify contexts. The ranking of the query result is computed based on keyword statistics collected from the specialized contexts. This is motivated by the observation that keyword statistics usually vary dramatically from one domain to another and therefore the ranking of the query result will vary accordingly. For example, while "leukemia" is rare over the Web, it is a fairly common term in biomedical science (as captured in the PubMed[1] database of 18 million articles), and is extremely common among articles of PubMed that are annotated to be cancer-related. Conventional ranking models will consider "leukemia" as a discriminative term, which is not true from the perspective of a cancer researcher or doctor, who typically narrows his/her interest in the cancer-related articles. Given that all ranking models and functions use keyword statistics to compute ranking scores, specialized keyword statistics naturally lead to specialized rankings for users interested in narrow domains.

1.1 A Motivating Example

PubMed contains 18 million biomedical citations. All the citations include title, abstract, and authors' information. Citations are often linked to full-text articles. Additionally, every citation is annotated with one or more MeSH (Medical Subject Headings) terms from a controlled vocabulary, which specifies a variety of concepts in biomedical science, e.g., "anatomy", "diseases", "diagnosis". MeSH terms in the vocabulary are organized in a hierarchy, as shown in Figure 1. A MeSH term may appear in several places in the hierarchy tree.

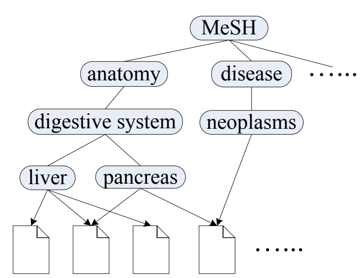


Figure 1: MeSH terms and the hierarchy

The vocabulary and the hierarchy of MeSH terms represent an ontology of biomedical science. Each MeSH term represents a biomedical concept and indexes a list of related citations. A combination of MeSH terms represents a context that spans the corresponding concepts. For example, "neoplasms" and "digestive system" represent two concepts under "diseases" and "anatomy" respectively. The combination of the two terms identify a set of

*Work supported by NSF IIS 0713672 and NSF DC 0910820

[1] http://www.ncbi.nlm.nih.gov/pubmed/

SIGMOD'11, June12–16, 2011, Athens, Greece.

citations, which form a search context for researchers and doctors concentrating on gastrointestinal (GI) cancer.

A researcher or a doctor can specify such a context by utilizing tools that visualize the MeSH term ontology and enable the user to navigate in the ontology and select terms of interest for the context. For example, the tool of Figure 2 mimics the widely used ontology navigator provided by PubMed[2] and extends it with the ability to select terms during the navigation. In Figure 2, we see two snapshots of the ontology navigator, at the point where the user has just selected the context terms. Note that the use of such tools for specifying the context removes the risk of mistyping the context trems, which would otherwise be an important risk since in context-based search only documents that contain the context terms will be retrieved.

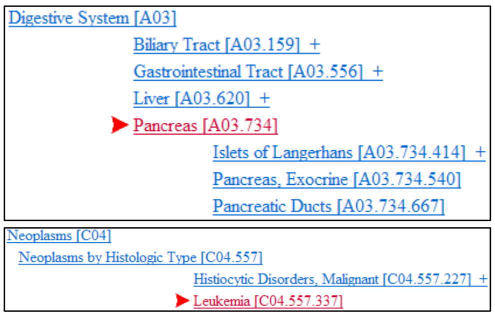


Figure 2: Choosing context terms using hierarchical ontology navigation in PubMed

Keyword distributions and statistics often vary dramatically from one specialized context to another. For example, research on cancer and research on digestive system have very different terminology. Using keyword statistics from a specialized context in ranking functions will deliver a specialized ranking order for the documents in that context. For example, the classical TF-IDF model uses *document frequency* as term weights to boost the ranking of documents that contain query terms that are rare in the collection. The rationale is that rare query terms are more discriminative, and therefore are more important in identifying relevant documents than frequent query terms. In the above example, a query term that is frequent for the citations on “neoplasms” may be rare for the citations on “digestive system”. The ranking order of two documents may be reversed when users are interested in different contexts.

Consider the query {pancreas, leukemia}, and two citations C_1: “Complications following pancreas transplant” and C_2: “Organ failure in patients with acute leukemia”, both annotated with the MeSH term “digestive system”. Assume we rank the two citations’ titles by $tf \times idf$. Since both citations match precisely one single query term, the ranking order of the two citations is only determined by idf. Without a context specification, the frequency of leukemia is higher than pancreas in PubMed. Hence, C_1 is ranked higher than C_2. However, if the query is issued by a GI doctor or researcher, whose focus is on digestive system, the frequency of leukemia is much less than pancreas in the corresponding context, and therefore C_2 should be ranked higher than C_1. Intuitively, pancreas transplant is a common topic among GI researchers. Leukemia in the query is more discriminative in the context. Given that C_2 is annotated with “digestive system”, it is very likely that the organs mentioned in the C_2’s title refer to digestive organs, which include pancreas.

[2] http://www.nlm.nih.gov/mesh/MBrowser.html

1.2 Contribution

We propose a new query model that extends conventional keyword queries and allows domain-interested users to specify search contexts. A search context is defined as a sub-collection of documents that users are interested in. The goal of context-sensitive ranking is to use keyword statistics based on user-defined contexts to rank documents in the contexts.

The query processing of context-sensitive queries presents novel performance challenges that are not met in the processing of conventional keyword queries. In conventional keyword query evaluation, the context is always fixed; it is the entire document collection. So the statistics are precomputed at indexing time. For context-sensitive ranking, however, contexts are specified by users at query time and can be arbitrary subsets of the document collection. Therefore, the collection-specific statistics (such as document frequency) also have to be computed at query time.

A straightforward solution to compute keyword statistics is using standard text search techniques to materialize contexts at query time and gathering required statistics accordingly. Unfortunately, this solution is not always cheap. The challenges are two. First, a common approach to materialize contexts is to intersect inverted lists of keywords, e.g., MeSH terms in PubMed. While intersection operations are efficient for most keyword combinations, intersecting very long inverted lists is still expensive [9]. Second, computing keyword statistics not only requires intersections, but also aggregations. As we will show later, some statistics in conventional ranking models demand aggregations of the documents in the contexts, which can be very expensive when the contexts are large.

In this paper, we propose a materialized view technique to overcome the above challenges. We reduce the problem of computing keyword statistics to evaluating aggregation queries, and use materialized views to improve query performance. Given that there is a huge number of possible context specifications, the technical challenge is how to choose a reasonable number of views to materialize. We present two algorithms for view selection. The goal is to guarantee good system performance for worst-case queries.

Key contributions of the paper include:

1. We propose a novel query model that allows expert users to specify search contexts. The ranking model uses keyword statistics collected from the specified contexts to rank documents in the contexts.

2. We reduce the problem of computing keyword statistics to evaluating aggregation queries, and leverage materialized views to improve query efficiency. Two algorithms are proposed to select a number of views to materialize.

3. We perform thorough experiments on the PubMed data set. Results show that context-sensitive ranking improves the ranking quality remarkably, compared with the conventional ranking models. The materialized view technique improves the efficiency of worst-case queries significantly. The overall performance of the system is guaranteed.

The paper is organized as follows: Section 2 defines the query model and the ranking model for context-sensitive ranking. Query evaluation is discussed in Section 3. Section 4 reduces the problem of computing keyword statistics to evaluating aggregation queries, and presents a view-based technique to compute statistics. Two view selection algorithms are presented in Section 5 to choose a reasonable number of views to materialize. Experiments and results are elaborated in Section 6. Related work is discussed in Section 7. Section 8 is the conclusion.

2. DATA MODEL & QUERY SEMANTICS

In this section, we formally define the query model (Section 2.1) and the ranking model (Section 2.2) for context-sensitive ranking.

2.1 Query Model

A document, denoted by d, is modeled as a tuple of *fields*, each consisting of a bag of words. A field may refer to abstract, category (e.g., MeSH annotations in PubMed), keywords or full content. Let $\mathcal{D}$ denote a collection of documents.

A query for context-sensitive search, $Q_c = Q_k|P$, consists of two parts: a context specification (P) that defines a set of documents as the search context, and a set of keywords (Q_k) as the conventional keyword query. The unranked result of Q_c is a set of documents in the search context that contain all the keywords.

Definition 1. A predicate field of $\mathcal{D}$ contains literals, each of which is a single keyword, called *context predicate*.

Definition 2. A *context specification* $P = p_1 \wedge p_2 \ldots \wedge p_c$ is a conjunction of context predicates that specify a sub-collection $\mathcal{D}_P \subseteq \mathcal{D}$ such that $\forall d \in \mathcal{D}_P$, d satisfies P. In other words, $\mathcal{D}_P = \sigma_P(\mathcal{D}) = \sigma_{p_1 \wedge p_2 \ldots \wedge p_c}(\mathcal{D})$.

Consider a conventional n-keyword query $Q_k = w_1 \wedge \ldots \wedge w_n$ in context P. The unranked result of $Q_c = Q_k|P$ is a set of documents in $\mathcal{D}_p$ that contain all the keywords:

$$Q_c(\mathcal{D}) = Q_k(\mathcal{D}_P) = \sigma_P(\mathcal{D}) \cap \sigma_{w_1}(\mathcal{D}) \cap \ldots \cap \sigma_{w_n}(\mathcal{D})$$

where $\sigma_{w_i}(\mathcal{D}) = \{d | d \in \mathcal{D} \wedge d \text{ contains } w_i\}$, $i = 1, \ldots, n$.

Without loss of generality, we assume the predicate field in context predicates is fixed, and simply use a conjunction of keywords $m_1 \wedge m_2 \wedge \ldots \wedge m_c$ to denote the context specification. For example, the query $Q_c = w_1 \wedge w_2 | m_1 \wedge m_2$ over PubMed defines a set of documents $\mathcal{D}_P$ that contain MeSH terms m_1 and m_2 as the context. The unranked result of Q_c is a set of documents in $\mathcal{D}_P$ that contain w_1 and w_2.

2.2 Ranking Model

Next we define the ranking of the result of Q_c. We start by presenting a generic representation of ranking functions for conventional keyword queries. We then evolve it to the context-sensitive ranking function. In the following, we use Q_t to denote conventional keyword queries.

Various ranking functions were developed in the literature to rank documents with respect to keyword queries. In general, they combine keyword statistics to a single score to evaluate the relevance between a query and a document. The statistics used in the ranking models can be classified into three categories: *query-specific*, *document-specific* and *collection-specific*:

- A *query-specific* statistic, denoted by $S_q(Q_t)$, is a statistic computed from the input query Q_t, e.g., query length.
- A *document-specific* statistic, denoted by $S_d(d)$, is a statistic computed from document d, e.g., term count of w_i in d. Every document has its unique document-specific statistics.
- A *collection-specific* statistic, denoted by $S_c(\mathcal{D})$, is a statistic computed from the collection $\mathcal{D}$. Conceptually, a collection-specific statistic is calculated by aggregating parameters of individual documents in the collection to a single value. For example, term count of w_i in the collection is calculated by summing up the number of occurrences of w_i in every document in the collection.

Table 1 summarizes atomic statistics used in a variety of ranking models, including vector space models (TF-IDF), language models and probabilistic relevance models. Note that some compound statistics used in the models can be computed by combinations of atomic statistics. For example, average document length ($avgdl$) is calculated by collection length divided by collection cardinality: $avgdl = \frac{len(\mathcal{D})}{|\mathcal{D}|}$.

Let $\mathcal{S}_q(Q_t)$ be a set of query-specific statistics for Q_t, $\mathcal{S}_d(d)$ be a set of document-specific statistics for d, and $\mathcal{S}_c(\mathcal{D})$ be a set of collection-specific statistics for $\mathcal{D}$.

Given a query Q_t and a document $d \in \mathcal{D}$, a conventional ranking function $f(\cdot)$ takes as arguments statistics from $\mathcal{S}_q(Q_t)$, $\mathcal{S}_d(d)$, $\mathcal{S}_c(\mathcal{D})$, and computes a score of d with respect to Q_t:

$$score(Q_t, d) = f(\mathcal{S}_q(Q_t), \mathcal{S}_d(d), \mathcal{S}_c(\mathcal{D})) \tag{1}$$

In context-sensitive ranking, a context specification P defines a set of documents $\mathcal{D}_P \subseteq \mathcal{D}$ of interest. Accordingly, the statistics used in the ranking function should be based on $\mathcal{D}_P$, rather than $\mathcal{D}$. Specifically, given a context-sensitive query $Q = Q_k|P$ and a document $d \in \mathcal{D}_P$, the ranking score is computed as:

$$score(Q_k|P, d) = f(\mathcal{S}_q(Q_k), \mathcal{S}_d(d), \mathcal{S}_c(\mathcal{D}_P)) \tag{2}$$

The ranking model for context-sensitive ranking uses the same computation function f as the conventional IR model, but different input statistics. In particular, Q_k in Q_c is a n-keyword conventional keyword query, and $\mathcal{S}_q(Q_k)$ in Formula 2 is equivalent to $\mathcal{S}_q(Q_t)$ in Formula 1. $\mathcal{S}_d(d)$ is the same in both formulas. The only difference is $\mathcal{S}_c$: $\mathcal{S}_c(\mathcal{D}_P)$ in Formula 2 collects statistics from $\mathcal{D}_P$, whereas $\mathcal{S}_c(\mathcal{D})$ in Formula 1 collects statistics from $\mathcal{D}$.

EXAMPLE 2.1. *TF-IDF weighting is a well-known ranking model. Among its variants, the pivoted normalization formula [30] is considered to be one of the best performing vector space models and is widely used in many text search systems. Its mathematical representation is shown in Formula 3, where s is a constant and is usually set to 0.2. The other variables' meanings can be found in Table 1.*

$$\begin{aligned} score(Q_t, d) &= \sum_{w \in Q_t} \frac{1 + \ln(1 + \ln(tf(w, d)))}{(1 - s) + s \cdot \frac{len(d)}{avgdl}} \cdot \\ &\qquad tq(w, Q_t) \cdot \ln \frac{|\mathcal{D}| + 1}{df(w, \mathcal{D})} \end{aligned} \tag{3}$$

where $avgdl = \frac{len(\mathcal{D})}{|\mathcal{D}|}$.

The statistics used in the pivoted normalization formula are classified as follows:

- $tq(w, Q_t)$ *is a query-specific statistic.*
- $tf(w, d)$, $len(d)$ *are document-specific statistics.*
- $df(w, \mathcal{D})$, $|\mathcal{D}|$, $len(\mathcal{D})$ *are collection-specific statistics.*

The context-sensitive version of the pivoted normalization formula for $Q_c = Q_k|P$ *replaces every* $S_c(\mathcal{D})$ *with* $S_c(\mathcal{D}_P)$, *i.e.,* $\forall d \in \mathcal{D}_P$,

$$\begin{aligned} score(Q_k|P, d) &= \sum_{w \in Q_k} \frac{1 + \ln(1 + \ln(tf(w, d)))}{(1 - s) + s \cdot \frac{len(d)}{avgdl_P}} \cdot \\ &\qquad tq(w, Q_k) \cdot \ln \frac{|\mathcal{D}_P| + 1}{df(w, \mathcal{D}_P)} \end{aligned} \tag{4}$$

where $avgdl_P = \frac{len(\mathcal{D}_P)}{|\mathcal{D}_P|}$.

Table 1: Statistics used in ranking functions

Scope	*Statistics*	*Notation*
collection-specific	term count in the collection (for keyword w)	$tc(w, \mathcal{D})$
	collection length	$len(\mathcal{D})$
	collection cardinality	$\lvert\mathcal{D}\rvert$
	document count (for keyword w)	$df(w, \mathcal{D})$
	unique term count in the collection	$utc(\mathcal{D})$
document-specific	term count in document	$tf(w, d)$
	document length	$len(d)$
	unique term count in document	$utc(d)$
query-specific	term count in query (for w)	$tq(w, Q_t)$
	query length	$len(Q_t)$
	unique term count in query	$utc(Q_t)$

3. QUERY EVALUATION

We discuss query evaluation of context-sensitive queries in this section. We first describe a straightforward evaluation (Section 3.1), and then analyze its performance bottlenecks (Section 3.2), which will be tackled in later sections.

3.1 Straightforward Evaluation

Query evaluation of a context-sensitive query evaluates the unranked result set, and computes keyword statistics to further compute ranking scores. Unlike conventional keyword query evaluation where all statistics are precomputed at indexing time, collection-specific statistics $\mathcal{S}_c(\mathcal{D}_P)$ for context-sensitive ranking must be computed at query time, because contexts are specified by queries and can be arbitrary subsets of the document collection.

A straightforward evaluation of a context-sensitive query is to materialize the context collection and compute required statistics accordingly. Thereafter the query evaluation is the same as conventional keyword queries. Let $L_w = \sigma_w(\mathcal{D})$ be the inverted list of w. Consider the query $Q_c = w_1 \wedge w_2 | m_1 \wedge m_2$ and the TF-IDF ranking function in Formula 4. By query semantics, the unranked result of Q_c is evaluated as $L_{w_1} \cap L_{w_2} \cap L_{m_1} \cap L_{m_2}$. In other words, intersecting the inverted lists of the four keywords returns the complete result set.

To compute collection-specific statistics, the query plan must satisfy the following constraints:

1. Document count for w_i, $df(w_i, \mathcal{D}_P)$, is the number of documents in the context that contain w_i, which is evaluated as $|\sigma_{w_i}(\mathcal{D}) \cap \sigma_P(\mathcal{D})|$. Therefore, the query plan must include $L_{w_1} \cap L_{m_1} \cap L_{m_2}$ and $L_{w_2} \cap L_{m_1} \cap L_{m_2}$, where the first expression computes document count for w_1, and the second expression computes document count for w_2.

2. Collection cardinality $|\mathcal{D}_P|$ is evaluated as $|\sigma_{m_1}(\mathcal{D}) \cap \sigma_{m_2}(\mathcal{D})|$. Hence, the query plan must include $L_{m_1} \cap L_{m_2}$.

3. Collection length $len(\mathcal{D}_P)$ requires a SUM aggregation on the lengths of the documents in the context, i.e., $\gamma_{\mathsf{sum}}(\sigma_{m_1}(\mathcal{D}) \cap \sigma_{m_2}(\mathcal{D}))$ where γ denotes an aggregation operator.

Putting the above constraints together, the execution plan of Q_c is shown in Figure 3, where $\cap_\gamma$ means "intersection with aggregation". At the bottom level, L_{m_1} and L_{m_2} are intersected to return documents in the context. Two aggregations, denoted by γ_{count} and γ_{sum}, are performed upon $L_{m_1} \cap L_{m_2}$ to compute collection cardinality and collection length. The result of $L_{m_1} \cap L_{m_2}$ is further intersected with L_{w_1} and L_{w_2} respectively to obtain document count for w_1 and w_2. The final result is computed by the highest intersection operator.

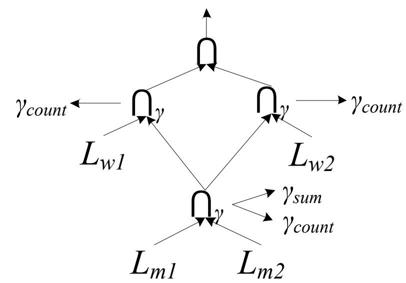


Figure 3: The execution plan of $Q_c = w_1 \wedge w_2 | m_1 \wedge m_2$

3.2 Performance Analysis

We introduce a simple cost model to quantify the cost of the straightforward evaluation. The purpose of the model is not to estimate the cost as accurate as possible, but to analytically demonstrate the bottlenecks of the straightforward evaluation.

3.2.1 Cost Models for Inverted List Intersection and Aggregation

The core operation of the query plan in Figure 3 is the intersection of inverted lists. In standard text search systems, a simple representation of an entry in an inverted list is a pair of document ID and term count, i.e., $\langle docid, tf \rangle$. Inverted lists are ordered by document ID so that two lists can be merged efficiently. A simple cost model for the merge join is $|L_i| + |L_j|$, where L_i and L_j are two inverted lists.

In addition to the standard merge join, inverted lists are partitioned into segments and skip pointers are maintained to jump between consecutive segments [24]. When two inverted lists are scanned, if the current document ID of the first inverted list does not fall in the segment of the second inverted list, the whole segment of the second inverted list can be skipped. Let M_0 be the number of entries in one segment, N_i^o be the number of segments in L_i whose ranges overlap with some segment(s) in L_j, and N_j^o be the number of segments in L_j whose ranges overlap with some segment(s) in L_i. Then the cost of the intersection with skip pointers is $M_0 \cdot (N_i^o + N_j^o)$. Since $N_i^o \leq \frac{|L_i|}{M_0}$ and $N_j^0 \leq \frac{|L_j|}{M_0}$, we have $M_0 \cdot (N_i^o + N_j^o) \leq |L_i| + |L_j|$. Therefore, the cost model of the intersection is:

$$cost(L_i \cap L_j) = M_0 \cdot (N_i^o + N_j^o)$$

An aggregation over a list requires a full scan of the elements in the list. Hence, the cost model of the aggregation is:

$$cost(\gamma(P)) = |\bigcap_{m_i \in P} L_{m_i}|$$

3.2.2 Analysis

Intersecting inverted lists is generally considered to be efficient. The skip pointer optimization improves the efficiency significantly when the join cardinality is small, as many segments can be skipped. In particular, when $|L_i|$ is orders of magnitude smaller than $|L_j|$, L_i's entries span at most $|L_i|$ segments of L_j, i.e., each entry in L_i falls in a separate segment of L_j. In such a case, the cost for the intersection of L_i and L_j is $|L_i| + |L_i| \cdot M_0$, which can be much cheaper than $|L_i| + |L_j|$.

However, intersecting very long inverted lists is not cheap [9]. In particular, when the join cardinality is not small, the intersection cannot take advantages of skip pointers and all segments must be scanned. The cost of the context materialization is bounded by $\sum_{m_i \in P} |L_{m_i}|$.

While inverted-list intersections in conventional keyword query evaluation can start from the most selective keyword, the evaluation of context-sensitive ranking must fully materialize the context. Intuitively, the context size tends to be fairly large, because the purpose of the context specification is to define a general search scope, rather than to filter out specific information as the keywords in conventional queries.

In standard text search systems, when the keywords in the query are not selective and the result size (i.e., the join cardinality) is very large, top-K processing techniques have been developed to reorder inverted lists so that only a small fraction of the lists are processed to generate top K results. This strategy, however, is not applicable for context-sensitive ranking before all collection-specific statistics are computed. The performance of the query will still be bounded by the complexity of the context materialization.

In addition to the cost of intersections, the cost of aggregations is proportional to the context size which is less than $\sum_{m_i \in P} |L_{m_i}|$.

PROPOSITION 3.1. *The cost of a context-sensitive query $Q_c = Q_k|P$ is bounded by $O(\sum_{m_i \in P} |L_{m_i}|)$ in the worst case.*

The above analysis shows that context-sensitive ranking can be fairly expensive when the context is not selective. The performance of $Q_c = Q_k|P$ can be orders of magnitudes slower than the conventional query $Q_t = Q_k \cup P$ (by query semantics, the unranked result of $Q_t = Q_k \cup P$ is the same as the unranked result of Q_c). This makes context-sensitive ranking unacceptable in these scenarios, as it sacrifices efficiency too much.

4. COMPUTING STATISTICS USING MATERIALIZED VIEWS

While context-sensitive ranking leverages customized statistics to provide specialized rankings, current text search systems cannot efficiently evaluate queries specifying very large contexts. The technical challenge is to maintain query efficiency as close as possible to conventional keyword queries. In the section, we reduce the problem to evaluating aggregation queries, and leverage materialized views to achieve this goal.

Computing collection-specific statistics essentially involves online aggregations. Similar problems were encountered in OLAP [12], an approach to quickly analyze multi-dimensional data. A large body of OLAP queries involve expensive aggregations which must be answered in a short time. The most important mechanism in OLAP that allows such performance is the use of the data cube, a materialized view that aggregates data along some dimensions. Aggregation queries then can be answered from the materialized views which are typically much smaller than the raw table. We incorporate a similar idea for the context-sensitive ranking problem.

4.1 Formalization

A document collection $\mathcal{D}$ is modeled as a wide sparse table T, as shown in Table 2. In addition to document ID, columns are classified into two categories: *keyword columns* (e.g., m_1, m_2), each corresponding to a keyword m_i that can be used in context specifications, and *parameter columns* (e.g., $len(d)$ and $tf(d, w_i)$), each corresponding to a parameter on which a collection-specific statistic aggregates. Every row corresponds to a document d_i. An entry in row d_i and column m_j is 1 if d_i contains m_j; otherwise, the entry is 0.

Table 2: A relational representation of the document collection

docid	$len(d)$	$tf(d, w_1)$	...	m_1	m_2	...	m_n
d_1	156	15	...	1	0	...	0
d_2	98	7	...	0	1	...	1
...	...	...	...	...	...	...	...

Given the wide sparse table T, computing a collection-specific statistic $S_c(\mathcal{D}_P)$ of context $P = m_{j_1} \wedge m_{j_2} \ldots \wedge m_{j_c}$ is equivalent to evaluating an aggregation query:

$S_c(\mathcal{D}_P)$: SELECT Stats-Aggre$(para(d))$
FROM T
WHERE $m_{j_1} = 1$ AND ... AND $m_{j_c} = 1$

where Stats-Aggre is the aggregation function for S_c and $para(d)$ is the document parameter upon which S_c aggregates.

Let $K = \{m_{i_1}, m_{i_2}, \ldots, m_{i_k}\}$ be a subset of $\{m_1, \ldots, m_n\}$. V_K is a materialized view that groups by K and aggregates the documents' parameters of every group:

V_K: SELECT $m_{i_1}, m_{i_2}, \ldots, m_{i_k}$,
Stats-Aggre$(para(d))$ AS ContxPara
FROM T
GROUP BY K

We refer to $m_{i_1}, m_{i_2}, \ldots m_{i_k}$ as *keyword columns*, and ContxPara as *parameter columns* in the materialized view.

Given the view definition, if $P \subseteq K$, the aggregation query that computes $S_c(\mathcal{D}_P)$ can be rewritten as follows:

$S_c(\mathcal{D}_P)$: SELECT Stats-Aggre(ContxPara)
FROM V_K
WHERE $m_{j_1} = 1$ AND ... AND $m_{j_c} = 1$

The GROUP BY clause in the view definition essentially partitions the document collection. Every tuple in the view is an aggregation on one partition. The evaluation of the rewritten query aggregates partial aggregation results and avoids scanning the raw table.

EXAMPLE 4.1. *Consider a two-keyword context specification $P = m_1 \wedge m_3$. Collection length $len(\mathcal{D}_P)$ and collection cardinality $|\mathcal{D}_P|$ can be translated to a SUM and a COUNT aggregation on the wide sparse table respectively:*

$len(\mathcal{D}_P)$: *SELECT SUM($len(d)$)*
FROM T
WHERE $m_1 = 1$ AND $m_3 = 1$

$|\mathcal{D}_P|$: *SELECT COUNT(*)*
FROM T
WHERE $m_1 = 1$ AND $m_3 = 1$

Let $K = \{m_1, m_2, m_3\}$. The view

$$\begin{array}{lll} V_K\text{: SELECT} & m_1, m_2, m_3, \\ & \textit{SUM}(len(d)) \text{ AS } \textit{ContxtLen}, \\ & \textit{COUNT}(*) \text{ AS } \textit{ContxCount} \\ \text{FROM} & T \\ \text{GROUP BY} & m_1, m_2, m_3 \end{array}$$

partitions $\mathcal{D}$ into 2^3 partitions. The tuple $V(m_1 = 0, m_2 = 1, m_3 = 1)$ aggregates the statistics of the documents that contain m_2 and m_3, but do not contain m_1. Similarly, the tuple $V(m_1 = 0, m_2 = 0, m_3 = 0)$ aggregates the statistics of the documents that do not contain m_1, m_2 or m_3.

Having the view V_K, collection length and collection cardinality for $P = m_1 \wedge m_3$ can be computed as follows:

$$len(\mathcal{D}_P) = V_K(m_1 = 1, m_2 = 0, m_3 = 1).\textit{ContxLen} + V_K(m_1 = 1, m_2 = 1, m_3 = 1).\textit{ContxLen}$$

$$|\mathcal{D}_P| = V_K(m_1 = 1, m_2 = 0, m_3 = 1).\textit{ContxCount} + V_K(m_1 = 1, m_2 = 1, m_3 = 1).\textit{ContxCount}$$

4.2 View Usability

A view is *usable* for a query if it can be used to compute complete or partial results of the query.

THEOREM 4.1. *View V_K is usable for computing the collection-specific statistic $S_c(\mathcal{D}_P)$ for context P if*

1. *V_K includes a parameter column that aggregates the documents' parameters of S_c;*
2. *$P \subseteq K$.*

V_K groups by K and projects out all the other keyword columns. For a context specification that contains $m_j \notin K$, the aggregation query of a collection-specific statistic requires m_j in the WHERE clause. Therefore, V_K cannot be used to answer the aggregation query if $P \nsubseteq K$.

4.3 Complexity

If a materialized view is usable, collection-specific statistics can be computed by aggregating the materialized view, whose complexity is only determined by the view size, regardless of the context size. In other words, by choosing appropriate view sizes, query performance of context-sensitive ranking can be guaranteed.

THEOREM 4.2. *If view V_K is usable for $S_c(\mathcal{D}_P)$ in context P, the complexity of computing $S_c(\mathcal{D}_P)$ is $O\big(ViewSize(V_K)\big)$, which is bounded by $O\big(2^{|K|}\big)$.*

Without additional indexes built on the view, computing collection-specific statistics using a view requires a full scan of the view. Theoretically, the number of tuples in the view is exponential to the number of keywords columns. However, the actual number of non-empty tuples can be much smaller. Consider two keywords m_1, m_2 that always appear in the same documents. The tuples $V_K(m_1 = 1, m_2 = 0)$ and $V_K(m_1 = 0, m_2 = 1)$ are always empty. Similarly, if m_1 and m_2 never appear in the same document, the tuple $V_K(m_1 = 1, m_2 = 1)$ is always empty.

While computing accurate view size needs a full scan of the entire document collection, a simple approach to estimate the view size is sampling: a small number of documents are sampled and mapped to V_K. The number of non-empty tuples after the mapping is estimated as the view size. In the following, we use $ViewSize(\cdot)$ to denote a function that returns the size of a given view, either by sampling or by scanning.

5. VIEW SELECTION

Materialized views improve query performance, in particular the computation of collection-specific statistics, significantly. Ideally, if we can materialize views that cover all possible context specifications, query performance of context-sensitive ranking is guaranteed. However, this would cost exponential disk storage, which is not feasible for any system. The challenge is how to choose a small number of views to materialize to guarantee the system's overall performance.

Cost analysis in Section 3.2 shows that the straightforward approach relying on standard text search systems can still achieve acceptable performance for small contexts. Query performance would be orders of magnitudes slower when user-specified contexts are very large. Hence, context specifications (i.e., keyword combinations) corresponding to large contexts should be covered by at least one view, so that performance of worst-case queries is bounded. Queries whose context specifications are not covered by any views are evaluated by the straightforward approach.

In addition to the context size, view size is the second parameter that needs to be constrained. The cost of computing collection-specific statistics using views is proportional to the view size. Therefore, the sizes of materialized views should be as small as possible.

Given a context specification P, let $ContextSize(P)$ be the size of context P. We formalize the view selection problem as follows:

PROBLEM STATEMENT 5.1. *Given a threshold of context size T_C and a threshold of view size T_V, find a set of views $\mathcal{V} = \{V_{K_1}, V_{K_2}, \ldots\}$ such that*

1. *$\forall V_{K_i} \in \mathcal{V}$, $ViewSize(V_{K_i}) \leq T_V$.*
2. *For every possible context specification P, if $ContextSize(P) \geq T_C$, then $\exists V_{K_j} \in \mathcal{V}$ such that $P \subseteq K_j$.*

Finding keyword combinations that specify large contexts is equivalent to mining association rules of keywords such that their supports, in terms of the number of documents that contain the keywords, are greater than T_C. Based on this reduction, we propose two view selection algorithms in the following sections.

5.1 Data-Mining-based Selection

A number of algorithms for mining association rules have been proposed in the data mining literature, e.g., Apriori [2], FP-growth [13], Eclat [36]. Given a set of items and a set of transactions, the algorithms scan the transaction set one or more times and return combinations of items whose occurrences in transactions (called support) are greater than a pre-specified threshold (called minimum support). In our problem setting, an item is mapped to a keyword, and a transaction is mapped to a document. Association rule mining algorithms return a set of keyword combinations, whose supports are greater than T_C.

Given a set of high-support keyword combinations, a naive approach for view selection is to create one view for each combination. However, this would result in a very large number of views. While aggregations on individual views are efficient enough, matching a view for the given query at query time would be prohibitively expensive. Therefore, under the data-mining setting, we reformulate the view selection problem as follows:

PROBLEM STATEMENT 5.2. *Given a set of high-support keyword combinations $\mathcal{P} = \{P_1, P_2, \ldots\}$, find the minimal number of views $\mathcal{V} = \{V_{K_1}, V_{K_2}, \ldots\}$ such that*

1. $\forall V_{K_i} \in \mathcal{V}$, $ViewSize(V_{K_i}) \leq T_V$.

2. $\forall P \in \mathcal{P}$, $\exists V_{K_j} \in \mathcal{V}$ *such that* $P \subseteq K_j$.

THEOREM 5.1. *Given a set of high-support keyword combinations, the view selection problem is NP-hard.*

Algorithm 1 presents a greedy algorithm that takes as an input a set of keyword combinations generated by association rule mining algorithms, and returns a set of views to materialize. Two heuristics are used for algorithm design. First, for two keyword combinations P_1, P_2, if $P_1 \subset P_2$, a view covering P_2 is usable for P_1. In other words, we only need to consider P_2 for the view selection purpose. Second, in order to reduce the total number of views, the overlap of the keyword combinations that are covered by a view is expected to be maximized.

The algorithm first removes keyword combinations that are subsets of other combinations (Line 1 in Algorithm 1), according to the first heuristic. For each newly created view V_{K_i}, the algorithm iteratively scans uncovered keyword combinations and adds the one that has the maximal overlap with K_i (Line 6-9 in Algorithm 1), until the size of V_{K_i} reaches T_V.

Algorithm 1: Data-mining-based View Selection

input : A set of keyword combinations $\mathcal{P} = \{P_1, P_2, \ldots\}$ generated by association rule mining algorithms
output: A set of views $\mathcal{V} = \{V_{K_1}, V_{K_2}, \ldots\}$

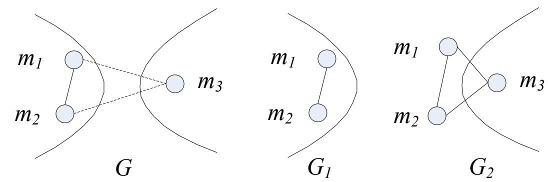
1 Scan $\mathcal{P}$ and remove P_i such that $\exists P_j \in \mathcal{P}, P_i \subset P_j$;
2 $i \leftarrow 0$;
3 **while** *$\mathcal{P}$ is not empty* **do**
4   Create a new view V_{K_i}, $K_i = \emptyset$;
5   Remove P_j with the largest size from $\mathcal{P}$, and add it to V_{K_i}, i.e., $K_i = P_j$;
6   **while** $ViewSize(V_{K_i}) < T_V$ **do**
7   Remove P_m from $\mathcal{P}$ such that (1) $|K_i \cap P_m|$ is maximized, and (2) $ViewSize(V_{K_i \cup P_m}) < T_V$;
8         $K_i \leftarrow K_i \cup P_m$;
9   **end**
10     $\mathcal{V} = \mathcal{V} \cup \{V_{K_i}\}$;
11     $i \leftarrow i + 1$;
12 **end**
13 **return** $\mathcal{K}$

An implicit assumption of Algorithm 1 is that for any input keyword combination P, $ViewSize(V_P) < T_V$. This assumption can be guaranteed by setting an upper bound on the number of keywords when applying association rule mining algorithms. The upper bound on $|P|$ is reasonable in practice. Statistics from standard text search systems have shown that most user queries have no more than 5 keywords [3]. The number of keywords in context specifications is expected to be even smaller.

5.2 Graph-Decomposition-based Selection

Many existing algorithms for mining association rules achieve good efficiency. But mining association rules is still an expensive operation. In particular, to discover a combination of size k, $P_k = \{m_1, m_2, \ldots, m_k\}$, $k-1$ combinations must be visited, i.e., $P_1 = \{m_1\}, P_2 = \{m_1, m_2\}, \ldots, P_{k-1} = \{m_1, m_2, \ldots, m_{k-1}\}$, and their supports must be computed accurately, even though we are only interested in P_k for the view selection purpose.

The selection algorithm in Section 5.1 essentially presents a bottom-up approach to select views: keyword combinations whose



(a) The original graph (b) Subgraphs after decomposition

Figure 4: The first graph decomposition scheme

supports are greater than T_C are generated first. Then a set of views are selected to cover all of them.

In this section, we present a top-down approach to select views. The idea is based on decomposing the keyword set to smaller subsets, until each keyword subset is small enough to be covered by one view whose size is less than T_V. The key of this approach is that the decomposition process does not violate the principle of view selection: keyword combinations with high supports should be covered by at least one view. Under this principle, the algorithm skips many combinations and only computes accurate supports when necessary.

5.2.1 Graph decomposition Schemes

Definition 3. A *Keyword Association Graph (KAG)* is a graph of keywords, where vertex m_i represents a keyword, and the weight of the edge $e_{m_i - m_j}$ represents the number of documents m_i and m_j co-occur. Edges with zero weight do not appear in the graph.

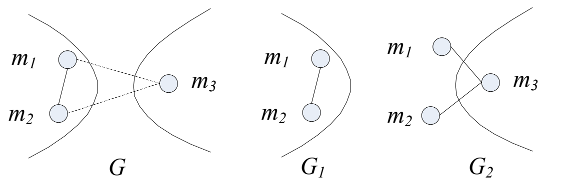
A KAG constructs pair-wise relationships between keywords, and implicitly captures k-ary ($k \geq 3$) keyword relationships: $m_1, m_2, \ldots, m_k$ co-occur in the same document only if $m_1, m_2, \ldots, m_k$ form a clique in the KAG. Initially, edges whose weights are less than T_C can be removed from the graph, because cliques containing such edges do not have high supports and therefore are not considered for view selection.

A *connected component* is a subgraph of KAG in which any two vertexes are connected to each other. As the first step, the KAG is decomposed to a set of connected components. We only need to consider views covering individual components. Without loss of generality, we assume the KAG is fully connected, and has only one connected component.

For a view that covers a subgraph, the view size is determined by the number of vertexes in the subgraph. Initially, the KAG has one component, which contains all vertexes. It is too large to be covered by one view. We need to decompose the KAG into subgraphs so that views covering individual subgraphs are smaller than T_V.

A cut divides the KAG $G = (V, E)$ into two parts, as shown in Figure 4a. Since the graph is fully connected, some edges' endpoints are in different parts. In Figure 4a, m_1, m_2, m_3 form a clique and some of its edges cross the two parts. The goal of the decomposition is to completely separate the graph. The question is: how to deal with the crossing edges?

The principle of the decomposition is that if the support of a clique (i.e., a keyword combination) is greater than T_C, the clique must be kept holistically in one subgraph after the decomposition, so that at least one view will cover it. In Figure 4a, if the support of $\{m_1, m_2, m_3\}$ is greater than T_C, after the decomposition, at least one subgraph needs to contain the clique. To this end, m_1, m_2 and the edge between them are replicated in G_2 after the decomposition, as shown in Figure 4b. Notice that m_1, m_2 and the edge $e_{m_1 - m_2}$ are kept in G_1 as well. The reason is that other vertexes in G_1 may form cliques with them. Removing m_1, m_2 and the edge



(a) The original graph (b) Subgraphs after decomposition

Figure 5: The second graph decomposition scheme

$e_{m_1-m_2}$ from G_1 may lose keyword combinations that should be covered by views.

If the support of $\{m_1, m_2, m_3\}$ is less than T_C, the corresponding clique is decomposable, because we do not need any view to cover it. This is the second decomposition scheme, as shown in Figures 5. Compared with the first decomposition scheme, the edge $e_{m_1-m_2}$ is not replicated in G_2. Hence, G_2 in Figure 5b is sparser than G_2 in Figure 4b.

A formal representation of the decomposition schemes is described as follows.

Definition 4. A *vertex separator* is a set of vertexes whose removal separates a graph into two distinct connected components.

Let S_0 be a vertex separator whose removal separates the vertexes in the KAG $G = (V, E)$ into S_1 and S_2, i.e., $V = S_1 \cup S_2 \cup S_0$. Given S_0, $G = (V, E)$ is decomposed into $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$ such that:

- $V_1 = S_1 \cup S_0, V_2 = S_2 \cup S_0$.
- $\forall m_i \in S_1, m_j \in S_1$, if $e_{m_i-m_j} \in E$, $e_{m_i-m_j} \in E_1$.
- $\forall m_i \in S_2, m_j \in S_2$, if $e_{m_i-m_j} \in E$, $e_{m_i-m_j} \in E_2$.
- $\forall m_0 \in S_0$, if $\exists m_i \in S_1, e_{m_0-m_i} \in E$, then $e_{m_0-m_i} \in E_1$; if $\exists m_j \in S_2, e_{m_0-m_j} \in E$, then $e_{m_0-m_j} \in E_2$.
- $\forall m_i \in S_0, m_j \in S_0$, if $e_{m_i-m_j} \in E$, $e_{m_i-m_j} \in E_1$.
- $\forall m_i \in S_0, m_j \in S_0$, if (1) there exists a clique containing m_i, m_j and vertex(es) in S_2, and (2) the support of the clique is greater than T_C, $e_{m_i-m_j}$ is replicated in E_2.

In the example in Figure 4 and 5, $S_0 = \{m_1, m_2\}$. Theoretically, whether to replicate the edge $e_{m_1-m_2}$ in G_2 or not depends on whether the support of the clique containing $e_{m_1-m_2}$ is greater than T_C. Since the support of the clique cannot be derived from the graph, we still need to compute support, which is similar to mining association rules. However, recall that as long as the view selection principle is satisfied, either decomposition scheme is correct. If the support of the clique is unknown, we may implicitly assume that the support is greater than T_C, and all the edges in the clique are replicated in G_2. In other words, using the first decomposition scheme always leads to a correct decomposition.

The above analysis indicates that computing support is not always necessary for the view selection purpose, especially when the subgraphs are large and sparse. The first decomposition scheme becomes less effective when the graphs are smaller and denser, and eventually is invalid for the subgraphs that are cliques.

5.2.2 *Graph Decomposition Algorithm*

Having the decomposition schemes, the remaining question is how to choose the vertex separator S_0 so that the graph can be decomposed efficiently. Two factors are considered: first, S_1 and S_2 should be about the same size, so that the sizes of all subgraphs decreases fast as the decomposition proceeds. Second, the number of vertexes in S_0 should be minimized. Since vertexes in S_0 are replicated in G_1 and G_2, and the view size is directly related to the number of vertexes in a subgraph, we want to minimize the number of replicated vertexes.

The optimization function for the graph decomposition is defined as follows:

$$\min \frac{|S_0|}{\min\{|S_1|, |S_2|\} + |S_0|} \tag{5}$$

The numerator minimizes the number of vertexes to be replicated. The denominator ensures that neither of the subgraphs is too small.

Given the optimization function in Formula 5, the graph decomposition problem is NP-hard [6]. A number of approximation algorithms have been developed. Most recently, paper [11] exhibits an $O(\sqrt{\log n})$ approximation algorithm for finding balanced vertex separators in general graph, with approximation ratio of $O(\sqrt{\log \mathsf{opt}})$ where opt is the size of an optimal separator.

The pseudo code of the algorithm that decomposes the KAG is shown in Algorithm 2.

Algorithm 2: Graph decomposition

input : A KAG $G = (V, E)$
output: A vertex separator (S_1, S_2, S_0)
Let $V = \{v_1, v_2, \ldots, v_n\}$;
foreach $1 \le i \le n$ **do**
  Create the augmented graph by adding a source s and a sink t to G;
  Connect s to $v_j, 1 \le j \le i$, and connect t to v_k, $i < k \le n$;
  Find the minimum capacity $s - t$ separator S_0^i ;
  Let $S_1^i = (V \cup \{s,t\}) - S_0^i, S_2^i = V - (S_1^i \cup S_0^i)$;
end
return (S_1^i, S_2^i, S_0^i) such that $\frac{|S_0^i|}{|E_{12}^i|}$ is minimal, where $|E_{12}^i|$ is the number of edges $e_{u-v}, u \in S_1^i \cup S_0^i, v \in S_2^i \cup S_0^i$;

5.3 Hybrid Approach

The data-mining-based selection and the decomposition-based selection have strengths in different directions. The data-mining-based approach is strict, and only covers keyword combinations that must be covered. Therefore, it is space efficient. However, it has to enumerate a very large number of keyword combinations. The decomposition-based selection, on the other hand, usually covers more keyword combinations than required. While it has high efficiency when the graph is large and sparse, its capability is limited when the graph is small and dense.

In implementation, we uses a hybrid approach to select views. Initially, the graph decomposition algorithm quickly decomposes the KAG into subgraphs, most of which can be covered by individual views. The data-mining-based approach is used thereafter to further decompose the remaining subgraphs, each of which is a clique and is still too large to be covered by one view.

6. EXPERIMENTS

We use the PubMed data set to evaluate the effectiveness of context-sensitive ranking and the efficiency of the materialized view technique. PubMed maintains 18 million citations, each annotated with one or more MeSH terms. We use combinations of MeSH terms to specify contexts and conventional keywords to search the citations' titles and abstracts. To deal with MeSH term inheritance, if a citation is annotated with the term t, all the ancestors of t in the hierarchy are attached to the citation. The average number of MeSH terms in a citation after the inheritance is 44.

In the experiments, we use the Lucene library[3] as the standard text search system. Lucene is a general-purpose text search system and reflects the state-of-the-art of keyword query evaluation. We only use Lucene for performance evaluation, but not for ranking. The reason is that Lucene's ranking module provides limited interfaces for customized ranking, which is not suitable for our context-sensitive ranking model.

The algorithms and the framework are implemented under Java 6. All the experiments are performed on an Intel i7 860 PC, with 8G memory.

6.1 Ranking Quality

We evaluate the effectiveness of context-sensitive ranking using the TREC Genomics benchmark of 2007 [16], which consists of 162,048 full-text documents, a small fraction of the PubMed data set. The TREC Genomics also contains 34 topics in the form of biological questions, which were collected from bench biologists and represent actual information needs in biomedical research. For each query, relevant documents were tagged manually by biologists based on pooled results from team submissions as the gold standard.

Given the TREC Genomics questions, conventional keyword queries are constructed by extracting one or more noun keywords from the questions. For example, for the question "*What symptoms are caused by human parvovirus infection*", a possible keyword query is $Q_k = \mathsf{symptoms} \wedge \mathsf{human} \wedge \mathsf{parvovirus} \wedge \mathsf{infection}$.

Then we rely on PubMed's Automatic Term Mapping (ATM) to construct appropriate contexts. Given a set of keywords, PubMed's ATM maps them to one or more MeSH terms. For the previous example, ATM maps the keywords to two MeSH terms: Humans and Parvovirus. Then P = $\mathsf{Humans} \wedge \mathsf{Parvovirus}$ specifies the context that studies Humans and Parvovirus.

For the constructed context-sensitive queries, we exclude those queries whose result sets are too small (less than 20), or the corresponding relevant document sets in the gold standard are too small (less than 5), since ranking thereof is not so important. Altogether 30 queries qualify for the experiment.

The main concern of ranking quality in practice is the number of relevant results in top few returned results, which are most likely to be examined by users. To study this aspect, we measure the rank precision among top ranked results, i.e., the number of relevant results in top K results. For the TREC Genomics benchmark, the relevance of a document to the query is based on whether the TREC Genomics gold standard includes the document. In the experiments, K is set to 20, as statistics from PubMed has shown that most users do not go beyond looking top 20 [22].

In additional to the precision, the reciprocal rank [33] is another popular measure for evaluating top ranked results. The reciprocal rank is the inverse of the position of the first relevant document in the ranked results. The higher the reciprocal rank of the query, the better the ranking is. In particular, if the first result is relevant, the reciprocal rank is $\frac{1}{1} = 1$.

In the experiments, we use the TF-IDF model as shown in Formula 4. While more sophisticated ranking functions are in use nowadays, TF-IDF still remains at the core and provides a clean way to measure the effect of context sensitivity.

[3]http://lucene.apache.org/

Given a context-sensitive query $Q_c = Q_k|P$, we compare the context-sensitive ranking and the conventional ranking. The conventional ranking of Q_c is equivalent to the ranking of the conventional query $Q_t = Q_k \cup P$, where P is treated as a boolean filter in Q_t and does not contribute to ranking scores. The measures of the precision and the reciprocal rank are shown in Figures 6, where the x-axis denotes the query ID. In Figure 6a and 6b, the y-axis denotes the number of relevant results in top 20 results. In Figure 6c and 6d, the y-axis denotes the reciprocal rank, whose maximum value is 1.

Figure 6a and 6b show that context-sensitive ranking delivers better ranking in 21 out of 30 queries, with occasional large improvements over conventional ranking (e.g., Q8 and Q9), while in the few occasions conventional ranking is superior (Q15, Q16, Q30) and the gap is not large. Statistically, the mean precisions of conventional ranking and context-sensitive ranking over 30 queries are 7.9 and 10.2 respectively; the mean reciprocal ranks over 30 queries are 0.62 and 0.78 respectively.

It is worth pointing out that some queries shown in Figure 6 do not benefit from context-sensitive ranking. Our observation is that ranking effectiveness depends on how well a context specification fits the original TREC query. In the experiments, the contexts are mechanically generated by PubMed's ATM mapping. We expect that context-sensitive ranking can deliver more improvements over conventional rankings for real-life queries, as their contexts are constructed by domain expects.

6.2 View Selection

To select views for materialization, we set T_C to 1% of the PubMed data set. PubMed has 18 million citations, so the absolute value of T_C is $180,000$. In other words, only contexts whose sizes are greater than $180,000$ are covered by views. Query performance under this setting will be shown in Section 6.3. The maximum view size T_V is set to $2^{12} = 4096$ tuples. Note that this is the number of non-empty tuples. The actual number of keyword columns in a view can be much higher than 12.

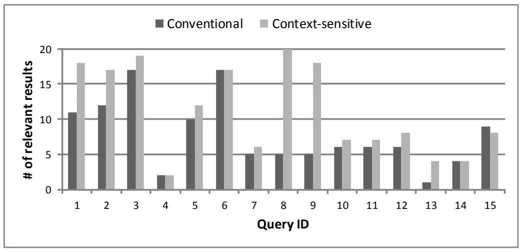
Efficiency of View Selection. We first apply two mining algorithms, Apriori [2], FP-growth [13], on the complete PubMed data set. Unfortunately, both algorithms fail. Specifically, by setting the minimal support to 1% of the number of the documents, the implementation of FP-growth runs out of memory when building the FP-tree, which invalidates the algorithm. The Apriori algorithm can swap intermediate results to disk, but requires multiples scans of the data set. Even if we limit the maximal size of keyword combinations to 8, it would take weeks to generate all valid combinations.

In general, the algorithms for mining association rules have difficulties for the PubMed's scale and our threshold. Although increasing the threshold can improve the efficiency of the mining process, as we will see in Section 6.3, the 1% threshold guarantees that all queries can be evaluated in a reasonable amount of time.

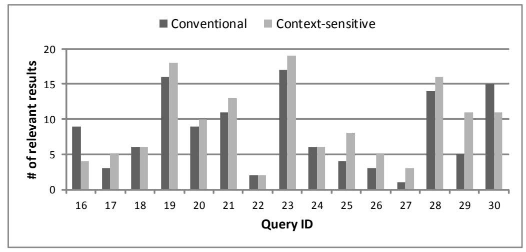
We then test the hybrid approach: the graph decomposition algorithm is first applied. 684 MeSH terms whose frequencies are greater than T_C are selected to form the initial KAG. It takes 24 hours to decompose the original graph to subgraphs, each of which is either (1) small enough to be covered by one view or (2) large and very dense (i.e., a clique).

When a subgraph is a clique and is still too large to be covered by one view, the data-mining-based approach is used for further decomposition. Since individual cliques are much smaller than the original graph, the data-mining-based approach can achieve good efficiency. Altogether, the hybrid approach takes 40 hours, and selects 3,523 views.

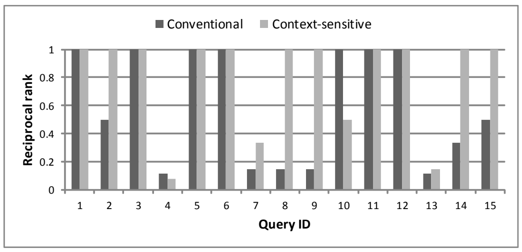
In our problem setting, context specifications are comprised of



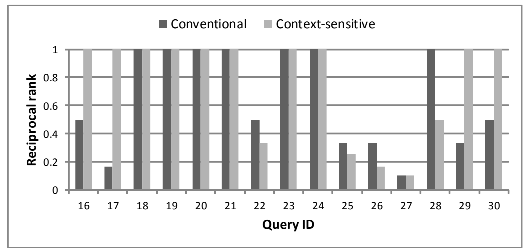
(a) Precision



(b) Precision



(c) Reciprocal rank



(d) Reciprocal rank

Figure 6: Ranking quality of top 20 results

MeSH terms, which are from a well-controlled vocabulary and are fairly stable. Given that the threshold of the context size (T_C) is set to a fixed percentage of the size of the document set , the number of views to materialize is stable, and does not change much as the the document set scales.

The size of the document set ($|\mathcal{D}|$) only has limited impact on the algorithms' complexities. Specifically, since the graph-decomposition-based algorithm is based on the KAG, which is comprised of MeSH terms, its complexity is independent of $|\mathcal{D}|$. The data-mining-based algorithm is based on mining association rules, which need to scan the document set one or more times. Hence, the complexity of the mining process is proportional to $|\mathcal{D}|$. Overall, the complexity of the view selection increases linearly with $|\mathcal{D}|$.

Storage usage. For a materialized view V_K, while keyword columns (i.e., K) determines the number of tuples in V_K, the storage of V_K is also dependent on parameter columns, e.g., $len(\mathcal{D}), tf(d, w_i)$, which are specified by a specific ranking function. In the experiments, we use the TF-IDF formula which demands document count $df(w_i, \mathcal{D}_P)$ of every query term which can be any keyword in the document set. Storing $df(w_i, \mathcal{D}_P)$ for all the keywords in the document set would result in tens of thousands of parameter columns in V_K.

In our system, V_K only stores the $df(w_i, \mathcal{D}_P)$ column if $|L_{w_i}| \geq T_C$. In other words, document counts of keywords with low frequencies are computed at query time. Consider the query $Q_c = w_1 \wedge w_2 | m_1 \wedge m_2$ and the materialized view $V_K, K = \{m_1, m_2, m_3\}$. Assume $|L_{w_2}| < T_C$. Then document count of w_2, which is evaluated as $|L_{w_2} \cap L_{m_1} \cap L_{m_2}|$, cannot be computed from V_K. However, since $|L_{w_2}| < T_C$, the support of $\{w_2, m_1, m_2\}$ must be less than T_C, and $L_{w_2} \cap L_{m_1} \cap L_{m_2}$ can be evaluated efficiently at query time. Notice that the evaluation of $L_{w_2} \cap L_{m_1} \cap L_{m_2}$ can start from the most selective keyword and leverage the optimization of skip pointers. The intersection $L_{m_1} \cap L_{m_2}$ is not enforced in the query plan, because collection cardinality $|L_{m_1} \cap L_{m_2}|$ and other statistics can be evaluated from V_K directly.

There are 910 keywords in the document set whose frequencies are greater than T_C. Therefore, every materialized view contains 912 parameter columns (the other two columns are context length and context cardinality). Given that the maximal number of the tuples in a materialized view is 4096, the maximal storage of a single view is 14.3 MB.

The total storage of the materialized views is 12.77 GB. For comparison, the original data set of PubMed takes 70 GB, and the Lucene index takes 5.72 GB. The average storage of a single view is 3.71 MB, which means that most views have fewer tuples than 4096. The cost of using a materialized view to compute statistics is very small.

6.3 Query Performance

Next we evaluate the performance of context-sensitive queries. The complete PubMed data set is used in the experiments. The straightforward evaluation, which was described in Section 3.1, is implemented as follows: for each collection-specific statistic, a conventional keyword query that materializes the corresponding document set is constructed and sent to Lucene. After Lucene returns the document set, an aggregation is performed upon it. Consider the example query $Q_c = w_1 \wedge w_2 | m_1 \wedge m_2$ in Figure 3. Four collection-specific statistics are required for the TF-IDF function: document count for w_1, w_2, collection cardinality and collection length. Hence, three conventional queries are evaluated by Lucene: $Q_t^1 = m_1 \wedge m_2$, $Q_t^2 = w_1 \wedge m_1 \wedge m_2$ and $Q_t^3 = w_2 \wedge m_1 \wedge m_2$, upon which the required statistics can be computed. Basically, we simulate the execution plan of a context-sensitive query in Lucene by issuing multiple conventional keyword queries.

With the materialized view technique, before sending keyword queries to Lucene, collection-specific statistics are matched over the views first. If a view is usable for a collection-specific statistic, no Lucene evaluation is needed. It is possible that there are more than one views that are usable for a collection-specific statistic. In such cases, the view with the minimal size is picked.

Two categories of queries are tested in the experiments:

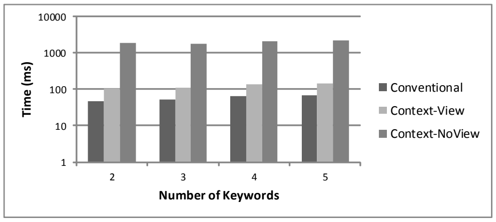


Figure 7: Execution time for the large-context queries

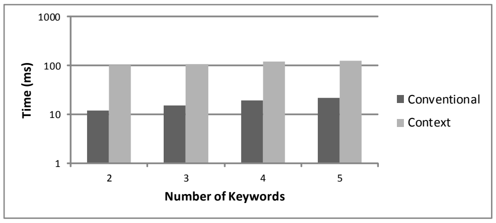


Figure 8: Execution time for the small-context queries

- **Large contexts**: queries whose context sizes are greater than T_C. They are evaluated using some materialized view(s).
- **Small contexts**: queries whose context sizes are smaller than T_C. They are evaluated without any views.

The large-context queries demonstrate the effectiveness of the materialized view technique. The small-context queries show how bad the query performance could be when the context size is below T_C and the evaluation uses the straightforward approach. For each category, context-sensitive queries are randomly generated in the following way: keywords in Q_k are randomly selected from keywords in the citations' titles. Given the generated keywords, PubMed's ATM is used to map them to MeSH terms. We vary the number of keywords from 2 to 5. For each experiment, fifty queries are generated. The values shown in the following figures are the average of the fifty queries.

For a large-context query $Q_c = Q_k|P$, three numbers are compared: (1) the execution time of the conventional query $Q_t = Q_k \cup P$, which returns the same result set as Q_c, but different ranking orders. (2) the execution time of Q_c with materialized views. (3) the execution time of Q_c without materialized views. The numbers are reported in Figure 7.

Figure 7 shows that the materialized view technique improves query efficiency significantly. Query performance of context-sensitive ranking with materialized views is about 2 times slower than the conventional queries, which is much better than the straightforward approach. The performance drop is mainly attributed to the partial coverage of document counts for keywords in the materialized views: for a keyword w_i whose frequency is less than T_C, $df(w_i, \mathcal{D}_P)$ is computed at query time. Overall, the absolute execution time stays around 100 ms.

For a small-context query $Q_c = Q_k|P$, only two numbers are compared: (1) the execution time of the conventional query $Q_t = Q_k \cup P$, and (2) the execution time of Q_c. Note that since the Q_c's context size is smaller than T_C, no views can be used. The results are shown in Figure 8.

As expected, the performance decreases are much larger than the large-context queries, as every collection-specific statistic must be computed at query time. However, the absolute execution time of context-sensitive queries stays around 100 ms. Figure 7 and 8 validate our original goal for query performance: while context-sensitive ranking may sacrifice query performance to some degree, the execution time of worst-case queries should be bounded.

The experiments in Section 6.1 has shown that ranking quality is directly related to the contexts. As a special case, when the context size is too small, the statistics are much less unreliable. For example, one of the most important problems for language models is smoothing, a technique to estimate the keywords' probabilities. When the context size is too small, smoothing becomes harder. The derived language models may not achieve satisfactory ranking performance. This means that the materialized view technique is even more important in practice: real-life queries that can benefit greatly from context-sensitive ranking are most likely to be answered by materialized views.

7. RELATED WORK

Personalized ranking is a problem whose motivation is similar to ours. It aims to bring personalizations to ranking so that users issuing the same query get different rankings for their own preferences, interests or search contexts. Personalized ranking has been extensively studied in IR community, especially in web search. A wide spectrum of personalization models were proposed, including users' profile information [29, 23, 35], and prior search behavior (e.g., query history, click logs) [31, 34, 25, 27]. They either re-rank top K results returned by standard search systems, or reformulate queries before sending to the search systems. The fundamental difference between these models and our ranking model is that they still rely on conventional ranking models and do not "personalize" underlying statistics.

Personalized PageRank [15, 18, 7, 19] is a personalization model specifically studied in web search. Instead of using the uniform distribution for all nodes at the initial state, personalized PageRank uses a set of query or user-specific nodes as the random walk starting points. Since the initial state is query or user-specific, PageRank scores must be computed at runtime. The main challenge is how to compute personalized scores efficiently, as online computations usually involve expensive fixpoint iterations over a very large graph. Among the proposed solutions, the algorithms in [18, 19] share the same spirit of the materialized view technique: some small subgraphs are precomputed in advance. Online computations use the materialized subgraphs to improve efficiency.

Personalized ranking has also been studied in database community. Paper [20] defines a preference model, where preferences are expressed as predicates associated with interest scores. Users' preferences are stored in their profiles and are used to rewrite SQL queries. Paper [1] defines a preference as an order of two tuples when their attributes satisfy some conditions. Preferences are not commutative, and may conflict with each other. Given a selection SQL query, the goal is to compute an order of the retrieved tuples that is consistent with the predefined preferences as much as possible. Unlike the model in [20], this model does not personalize rankings for individual users.

We are not the first one to utilize domain-specific statistics to improve ranking effectiveness. For example, the clustered-based retrieval [21] clusters documents that are semantically related and uses statistics within individual clusters to improve the smoothing of language models. In machine learning, topic model was also studied [26, 5]. However, none of the existing work ever considers dynamic context/domain/topic specifications. Given that static contexts/domains/topics are chosen in advance, they cannot satisfy diverse users' needs. Our query model provides much more power to domain experts.

A lot of work has been done in OLAP query processing, e.g., [12, 14, 17, 8, 32]. Given that we formalize the document collection as a wide sparse table, more OLAP techniques can be leveraged in our problem, e.g., wavelet. Furthermore, while our current definition of context specifications only involves keywords, context specifications can be extended with other variables. For example, with $time$ variable, users are able to specify the context as a set of documents published after 1998. Existing work on range aggregation queries can be used for such queries.

View selection is an important problem in RDBMS [14, 4, 28, 10]. Its problem setting, however, is quite different from ours. Specifically, view selection in RDBMS is formalized as a combinatorial problem: given a query workload and a space constraint, find a set of views to materialize so that the performance improvement for the workload is maximized. Our goal of view selection, on the other hand, is to improve the performance of worst-case queries. This is based on the following considerations. First, no query workload is available for this new query model. Second, even if the query workload is available, it is still dangerous to only rely on it. Unlike RDBMS queries which are fairly stable, queries of keyword search systems are typically unpredictable and may evolve as time passes.

8. CONCLUSION

Many document sets are organized structurally. For example, citations in PubMed are annotated with MeSH terms and are organized by a biomedical ontology. Conventional ranking models for document retrieval ignore such structures and fail to provide specialized rankings to domain experts. In this paper, we proposed a novel query model that allows domain-interested users to specify expected contexts, on which keyword statistics are computed to rank documents in the contexts. We addressed the problem of inefficient query evaluation for context-sensitive ranking. The key of the solution is to reduce the computation of collection-specific statistics to aggregation query evaluation. Materialized views are used to compute collection-specific statistics. The technical difficulty is how to select a small number of views to improve the system overall performance. We presented two algorithms, the data-mining-based algorithm and the graph-decomposition-based algorithm, for the view selection problem. Experiments on the PubMed data set demonstrated that context-sensitive ranking delivers remarkable improvements of the ranking quality. The materialized view-based technique improves the system overall performance significantly.

9. ACKNOWLEDGMENTS

We thank Zhiyong Lu and Chen Li for shedding the light on the benchmark of ranking evaluation. We thank Alin Deutsch and Nathan Bales for many inspiring discussions. We also thank anonymous reviewers for their comments and suggestions.

10. REFERENCES

[1] R. Agrawal, R. Rantzau, and E. Terzi. Context-sensitive ranking. In *SIGMOD*, 2006.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *VLDB*, 1994.

[3] A. Arampatzis and J. Kamps. A study of query length. In *SIGIR*, 2008.

[4] E. Baralis, S. Paraboschi, and E. Teniente. Materialized views selection in a multidimensional database. In *VLDB*, 1997.

[5] D. M. Blei and J. D. Lafferty. Correlated topic models. In *In Proceedings of the 23rd International Conference on Machine Learning*, 2006.

[6] T. N. Bui and C. Jones. Finding good approximate vertex and edge partitions is np-hard. *Inf. Process. Lett.*, 42(3), 1992.

[7] S. Chakrabarti. Dynamic personalized pagerank in entity-relation graphs. In *WWW*, 2007.

[8] C. Y. Chan and Y. E. Ioannidis. Hierarchical prefix cubes for range-sum queries. In *VLDB*, 1999.

[9] S. Chaudhuri, K. W. Church, A. C. König, and L. Sui. Heavy-tailed distributions and multi-keyword queries. In *SIGIR*, 2007.

[10] R. Chirkova, A. Y. Halevy, and D. Suciu. A formal perspective on the view selection problem. *VLDB J.*, 11(3), 2002.

[11] U. Feige, M. T. Hajiaghayi, and J. R. Lee. Improved approximation algorithms for minimum-weight vertex separators. In *STOC*, 2005.

[12] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-total. In *ICDE*, 1996.

[13] J. Han, J. Pei, Y. Yin, and R. Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Min. Knowl. Discov.*, 8(1), 2004.

[14] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *SIGMOD*, 1996.

[15] T. H. Haveliwala. Topic-sensitive pagerank. In *WWW*, 2002.

[16] W. R. Hersh and E. M. Voorhees. Trec genomics special issue overview. *Inf. Retr.*, 12(1), 2009.

[17] C.-T. Ho, R. Agrawal, N. Megiddo, and R. Srikant. Range queries in olap data cubes. In *SIGMOD*, 1997.

[18] H. Hwang, A. Balmin, B. Reinwald, and E. Nijkamp. Binrank: Scaling dynamic authority-based search using materialized subgraphs. In *ICDE*, 2009.

[19] G. Jeh and J. Widom. Scaling personalized web search. In *WWW*, 2003.

[20] G. Koutrika and Y. E. Ioannidis. Personalization of queries in database systems. In *ICDE*, 2004.

[21] X. Liu and W. B. Croft. Cluster-based retrieval using language models. In *SIGIR*, 2004.

[22] Z. Lu, W. Kim, and W. J. Wilbur. Evaluation of query expansion using mesh in pubmed. *Inf. Retr.*, 12(1), 2009.

[23] Z. Ma, G. Pant, and O. R. L. Sheng. Interest-based personalized search. *ACM Trans. Inf. Syst.*, 25(1).

[24] C. D. Manning, P. Raghavan, and H. SchÃijtze. *Introduction to Information Retrieval*. Cambridge University Press, New York, 2008.

[25] F. Qiu and J. Cho. Automatic identification of user interest for personalized search. In *WWW*, 2006.

[26] M. Rosen-Zvi, T. Griffiths, M. Steyvers, and P. Smyth. The author-topic model for authors and documents. In *UAI*, 2004.

[27] X. Shen, B. Tan, and C. Zhai. Context-sensitive information retrieval using implicit feedback. In *SIGIR*, 2005.

[28] A. Shukla, P. Deshpande, and J. F. Naughton. Materialized view selection for multidimensional datasets. In *VLDB*, 1998.

[29] A. Sieg, B. Mobasher, and R. D. Burke. Web search personalization with ontological user profiles. In *CIKM*, 2007.

[30] A. Singhal. Modern information retrieval: A brief overview. *IEEE Data Eng. Bull.*, 24(4), 2001.

[31] J. Teevan, S. T. Dumais, and E. Horvitz. Personalizing search via automated analysis of interests and activities. In *SIGIR*, 2005.

[32] J. S. Vitter and M. Wang. Approximate computation of multidimensional aggregates of sparse data using wavelets. In *SIGMOD Conference*, 1999.

[33] E. M. Voorhees. Overview of the trec-9 question answering track. In *TREC*, 2000.

[34] R. W. White and D. Morris. Investigating the querying and browsing behavior of advanced search engine users. In *SIGIR*, 2007.

[35] S. Xu, S. Bao, B. Fei, Z. Su, and Y. Yu. Exploring folksonomy for personalized search. In *SIGIR*, 2008.

[36] M. J. Zaki. Scalable algorithms for association mining. *IEEE Trans. Knowl. Data Eng.*, 12(3), 2000.