

Tree Information Retrieval

Sreenivas Gollapudi
Ebrary, Inc.

Sreenivas.Gollapudi@ebrary.com

D. Sivakumar*
Google, Inc.

siva@google.com

ABSTRACT

In the context of digital libraries, documents (e.g., books) can be naturally modeled as hierarchies (e.g., chapters and sections); additionally, large collections of documents are often organized as hierarchies (e.g., by topic). We develop concepts and algorithms for fundamental informational retrieval problems in this framework. Specifically, we formulate the problem of summarizing search results for single- and multi-term queries as combinatorial problems on trees, and present efficient algorithms for computing such summaries. Within the context of hierarchies, our formulations are intended to enhance, rather than replace, the traditional vector space and probabilistic models employed in information retrieval. We validate the efficiency and quality of our algorithms on a corpus of nearly 4.5 million documents organized into a tree of 650,000 nodes, derived from the DMOZ Open Directory Project.

Categories and Subject Descriptors

H.3.3 [Information Search and Retrieval]: Summaries

General Terms

Algorithms, experimentation

Keywords

Hierarchy, taxonomy, digital libraries

1. INTRODUCTION

Consider a digital library comprising millions of books, and the associated IR problem of searching the library by keywords. Traditional wisdom as well as existing commercial systems (e.g., Amazon “Search Inside the Book”) treat books as basic units of search results, and within books, pages as atomic units of information.

We take the view that both of these assumptions limit the potential effectiveness of information retrieval in digital libraries. Consider, for example, searching for information via the query “stem cell research.” Books on this subject appear within multiple disciplines (e.g., biology, biotechnology, ethics, politics, religion, etc.); in addition, there are several books that address this subject from many of these perspectives. In such a scenario, it is valuable to (1)

*This work was done while the author was at IBM Almaden Research Center

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CIKM '06, November 6–11, Arlington, Virginia USA
Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...\$5.00.

inform the user that there are books in various disciplines that match the given keywords; (2) provide sample listings of books under various disciplines; (3) provide finer information about the occurrence of the query terms within individual books, not in the form of a snippet extracted from a page, but in the form of chapter and section headings. The first two desiderata outlined above are fairly intuitive. The value of the third can be seen by considering a multidisciplinary book (e.g., an overview book) on stem cell research that both outlines the technological issues and discusses the ethical and political angles on the subject. Here, it is clear that a summary of the relevant chapter and/or section titles is more useful than a snippet from any particular page. Indeed, one may wonder if providing such rich information about the occurrence of a term in a book could be useful even in traditional (print) books. For example, when we looked up the phrase “pseudopolynomial-time algorithms” in a computer science textbook, the listing consisted of “113–14, 140, 143, 136, 317–37, 554–56.” This appears less useful than, say, listing the section headings for the relevant portions of the book. Printed books alleviate this problem to some extent by grouping the index entries along with associated phrases (e.g., the listing for “Scaling algorithms” is grouped by the phrases “basic ideas,” “for convex cost flows,” etc.). While the printed medium suffers from the limitation on real estate, the digital version of the index is not so severely limited (e.g., it is very common to find additional details on a search result by a simple mouse-over operation).

Information retrieval systems are traditionally built around the notion of a “document,” which, in the context of digital libraries, often translates into a book or an article (from a journal). Once the notion of a document is precisely identified, the field of IR has well-developed models, methods, and metrics to build search systems. As the foregoing discussion illustrates, in the context of digital libraries, units of information at various granularities can be logically considered as documents. The work reported in this paper is aimed at developing IR paradigms that expressly take into account this notion of a document.

Main contributions. We formulate the problem of how to organize and summarize search results in the scenario where a large collection of documents is viewed as a tree, and *any* node in the tree is a potential candidate to be returned as a search result. Naturally, the lower the node, the more specific the search result will be (e.g., sections or chapters, as opposed to books or topics or fields). In our abstraction, when a query is issued, a subset of the leaves of the tree are identified as responses to the query. The algorithmic questions that arise include the following: (1) how best can we summarize this (potentially very large) set of responses, taking advantage of the extant hierarchy implied by the tree? (2) when two or more query terms are issued and the corresponding sets of leaves identified, what are good notions of “intersection” and “union” of these sets, and how could one

compute it efficiently?

We model these questions as combinatorial problems on trees, and develop efficient algorithms to solve them. Specifically, we offer two formulations of the summarization problem, one that is aimed at presenting a high-level gist of (nearly all of) the result set, and another that is aimed at presenting a small set of the most important highlights (highly relevant nodes of the hierarchy). In addition, we present optimal efficient algorithms for our formulations. For the case of high-level summarization of nearly all results, we employ dynamic programming techniques; for the case of picking a small number of high-quality nodes of the hierarchy, we present a greedy algorithm. We further present a probabilistic sampling step that significantly improves the efficiency of the dynamic programming algorithm while preserving the quality of the solution.

We demonstrate the quality of our algorithms by presenting sample results for various queries executed on a corpus of nearly 4.5 million document snippets in the DMOZ Open Directory Project.

1.1 Types of summaries.

Our goal is to enrich the search results by exploiting the tree structure within documents and across the document collection. When such additional structure is available, it is no longer clear that a simple ranked list of leaves is the most appropriate response to a query. For example, for a given query, it is conceivable that a single section within a chapter in a book contains as much significant information relevant to the query as is present in an entire topic sub-tree elsewhere in the hierarchy. As an example, for the query “ACL injury,” a section from a medical textbook is equally significant as the topic node `Top/Sports/Injuries/Leg`. In such cases, while it is clear that the textbook section deserves to be placed highly in the list, it is equally clear that no one leaf from the topic sub-tree deserves to be placed alongside the section from the medical textbook. On the other hand, the root of the topic sub-tree, suitably annotated, could be mentioned as a candidate result for the query.

Given a set L of leaves of a tree T , one may summarize L to achieve various goals. As mentioned before, we will be concerned with two types of summaries; we provide brief overviews of each.

Gists. Given a set L of leaves, the goal is to find a few nodes of the tree such that: (1) the nodes do not subsume each other; (2) the number of leaves in L in the sub-tree under each of these nodes is more or less equal; and (3) together, the sub-trees under the nodes cover almost all of L .

This is addressed in [5] and also in [4]; our formulation builds on the latter work. More precisely, given a set of N leaves and integer parameters k and e , the goal is to find a set of up to k nodes in the tree that form an *anti-chain*, and such that the sub-trees under these k nodes contain all but e of the N leaves; this operation is referred to as $\text{DIVIDE}(N, k, e)$. (The exact optimization criteria will be specified in Section 2.) The case of $e = 0$ was formulated in [4], where a simple $O(Nk^2)$ time dynamic programming algorithm is presented for this problem for binary trees. We extend the dynamic programming algorithm to handle up to e omissions, and obtain an $O(Nk^2e^2)$ time algorithm. If $e = \epsilon N$, even for modest values of k , the time complexity renders the algorithm unrealistic for practice. To remedy this, we present a probabilistic algorithm, which, by sam-

pling roughly $n = O(k^2)$ leaves, produces results comparable to the optimal algorithm in time $O(\epsilon^2 k^8)$ via the optimal dynamic program on n leaves with ϵn omissions. In practice, this is blazingly fast. We explain the success of this probabilistic approximation by a simple probabilistic lemma about trees.

Highlights. Given a set L of leaves, the goal is to find a few nodes of the tree such that: (1) the nodes do not subsume each other; (2) focus — a good fraction (e.g., 10% or one-third) of the leaves in sub-trees under each of these nodes consist of leaves from L ; and (3) coverage — together, the sub-trees under the nodes cover a non-trivial fraction (e.g., 10% or one-third) of all of L .

Notice that the main difference from the previous formulation is that instead of asking for a summary that explains almost all of the result set, here we are interested in finding a small set of highlights from the result set. Furthermore, the notion of highlight will be well-defined (in terms of focus and/or coverage). This is a significant qualitative difference, and the resulting algorithms are quite different and more efficient; the running times are dominated by $O(Nd \log(Nd))$, where d is the depth of the tree. Notice that this is more efficient for all values of the analogous “ k ” parameter (the number of nodes desired in the output). This enables us to use these algorithms in lieu of the probabilistic sampling step in obtaining a high-level summary: find a few hundreds of good internal nodes via this step, and summarize them via suitable modifications of the algorithms for `DIVIDE`.

Summarizing multiple sets of leaves. In addition to the two formulations for summarizing a set of leaves returned as results to a query, we also consider the case of multi-term queries. Given two or more sets of leaves and a summarization parameter k , what constitutes a good set of k internal nodes that best summarize the given sets of leaves? For this problem, we present a novel notion of “intersection” relevant in this context. Traditionally, multi-term queries are handled via connectives, with “AND” being the most popular; to compute the set of documents that match a query “ q_1 AND q_2 ”, it is customary to perform an intersection of the set of documents that contain q_1 with the set of documents that contain q_2 . In the context where documents are hierarchically organized and any node of the tree is an acceptable search result, interesting alternatives emerge. For example, for the query “`dog` AND `cat`”, a chapter of a book that contains one section on dogs and one section on cats is a viable solution. Naturally, the closer the document on dogs and the document on cats are in the tree, the more relevant their least common ancestor is. Similarly, it is uninteresting to return a document (at whatever granularity) that is largely about cats but also happens to mention the word “dog” in it once.

For this problem, we show how the formulations of the summarization problem could be suitably enhanced to provide novel solutions that also admit efficient algorithms. Our algorithm attempts to find nodes as low in the tree as possible, where each query term has comparable significance, and such that as many occurrences of each of the terms within the corpus are covered as possible. Analogous formulations and algorithms exist for the disjunctive connective `OR`; we omit details in this preliminary draft.

1.2 Discussion

Our algorithms start with initial sets of leaves that are obtained as responses to query terms. We treat the problem of identifying the most suitable sets of leaves for given queries as a black-box procedure. Indeed, this step can take advantage of the wealth of IR techniques based, for example, on vector-space or probabilistic models of information retrieval, treating the unit of information at each leaf as a document. This implies that our view of documents and document collections as trees does not exclude, but instead complements, well-studied IR models. Also, in general, we may consider weighted sets of leaves (where the weights reflect the relevance of a leaf to a query); all our algorithms naturally extend to the case of weighted sets of leaves.

1.3 Related Work

There is a plethora of work on generating taxonomies, especially from large collections of web pages (similar to YAHOO! or the DMOZ Open Directory Project). Our work is orthogonal to the concerns here, since we are employing hierarchies as meta-data, similar to [5] and [4]. The field of information retrieval for data collections in XML (see [2] addresses some concerns similar to ours, but not from the viewpoint of algorithmic efficiency; also, traditional XML search via XPath or XQuery operations are studied from the largely database-centric viewpoint of producing all results that satisfy a query, whereas we formulate our summarization algorithms as optimization procedures. The work of [7] attempts to provide hierarchically organized summaries of search results; the focus there is on building the hierarchy from the search results, while we focus on employing a hierarchy to offer various granularities for search, and also to explain a number of search results succinctly.

The work of [8] has some similarities to ours in the sense of enriching the set of “returnable results”; they define the notion of a compound document in the context of a hyperlinked collection of documents, and offer them as basic units of search results. Their notion of compound document is not built using a hierarchical organization of the underlying corpus, but the latent hierarchies extant in the form of web pages placed together in file system directories. Also, while they try to group several web pages into a compound document, they do not deal with documents that are finer-grained than web pages. The work in [3, 1] employ fine-grained models of web pages based on parse trees of the underlying HTML content, but make no attempt to compare the significance of documents of different granularities. Finally, in the database community, researchers have considered summarizing “interesting” or “surprising” regions of the attribute space [9, 6]; in the context of hierarchies, these bear some resemblance to our methodology, but the algorithmic formulations and techniques are rather different.

2. CONCEPTS AND FORMULATION

2.1 Preliminaries

The information model underlying our work is that the universe of documents is a rooted tree \mathcal{T} whose nodes are labeled from a set \mathcal{C} of concepts. The set of leaves of \mathcal{T} will be denoted by \mathcal{L} and the set of internal nodes of \mathcal{T} will be denoted by \mathcal{I} . The union of \mathcal{L} and \mathcal{I} will be denoted by \mathcal{N} , the set of nodes of \mathcal{T} . For a node $v \in \mathcal{N}$, $parent(v)$ will denote the unique parent of v in \mathcal{T} , and $children(v)$ will denote the set of children of v in \mathcal{T} . By convention,

we will take $parent(v) = v$ for the root node of \mathcal{T} , and $children(v) = \emptyset$ for leaves v of \mathcal{T} . For an internal node v of \mathcal{T} , we denote by $subtree(v)$ the set of nodes under v in \mathcal{T} (including v and all leaves), and by $leaves(v)$ the set of leaves in $subtree(v)$. For two nodes u and v of \mathcal{T} , we denote by $path(u, v)$ the path from u to v in \mathcal{T} . For two nodes u and v of \mathcal{T} , we denote by $lca(u, v)$ the least common ancestor of u and v in \mathcal{T} , that is, the node w such that $u, v \in subtree(w)$ and at least one of u and v is not in $subtree(w')$ for any w' satisfying $level(w') < level(w)$.

Any node (internal node or leaf) of \mathcal{T} is considered a *document*. We define the level function $level : \mathcal{N} \rightarrow \mathbf{Z}$ inductively by $level(v) = 0$ if $v \in \mathcal{L}$, and for $v \in \mathcal{I}$, $level(v) = 1 + \min\{level(w) \mid w \in children(v)\}$. Intuitively, lower level nodes are fine-grained documents and tend to be more specific in nature, while higher level nodes are coarse-grained documents and tend to be more general in nature.

We will denote by \mathcal{W} the set of “words” or “terms” of discourse. Queries to the search system will be drawn from this set.

Fix a tree \mathcal{T} with node set \mathcal{N} , and let L denote a subset of leaves of \mathcal{T} (think of L as the set of results returned for a query term). We develop a few important notions that will help us state our formulations and algorithms in precise terms.

Definition 1. The *focus* for a node $v \in \mathcal{N}$, the *focus of v with respect to L* , denoted by $focus(v; L)$, is defined as the fraction of the leaves in $subtree(v)$ that are in L , that is, $focus(v; L) = |leaves(v) \cap L| / |leaves(v)|$. For a set $S \subseteq \mathcal{N}$ of nodes, the *average focus of S with respect to L* is defined as $|S|^{-1} \sum_{v \in S} focus(v; L)$.

Definition 2. The *coverage* for a node $v \in \mathcal{N}$, the *coverage of v with respect to L* , denoted by $coverage(v; L)$, is defined as the fraction of the leaves in L that are in $subtree(v)$, that is, $coverage(v; L) = |leaves(v) \cap L| / |L|$. For a set $S \subseteq \mathcal{N}$ of nodes, the *total coverage of S with respect to L* is defined as $\sum_{v \in S} coverage(v; L)$.

The rationale for using average for the focus and total for the coverage is mainly cosmetic. When we add the coverages of the individual nodes in a set S , we obtain a number in $[0, 1]$ that tells us how much of L the set S covers. On the other hand, “total focus” of a set S doesn’t have any intuitive meaning, especially when $|S|$ is not explicit, and averaging the focus over all of S gives a normalized value in $[0, 1]$.

Definition 3. The *count* for a node $v \in \mathcal{N}$, the *count of v with respect to L* , denoted by $count(v; L)$, is defined as the number of the leaves in L that are in $subtree(v)$, that is, $count(v; L) = |leaves(v) \cap L|$.

2.2 The DIVIDE operation for gists

Given a term $t \in \mathcal{W}$, let $L(t)$ denote the set of leaves relevant for the term t (as determined by a black-box IR system). For given integers k and e , the goal of the DIVIDE operation is to partition all but e leaves in $L(t)$ into k subsets, where the partitions are restricted to be induced by subtrees of nodes of \mathcal{T} , and such that each partition contains more or less the same number of nodes from $L(t)$. Formally:

DIVIDE(t, k, e)

Given: $t \in \mathcal{W}$, $k, e \in \mathbf{Z}$, compute a set of $\leq k$ nodes

$v_1, \dots, v_k \in \mathcal{N}$ such that

- (D1) the nodes v_1, \dots, v_k form an anti-chain in \mathcal{T} , namely $leaves(v_i) \cap leaves(v_j) = \emptyset$ for $i \neq j$; (D2) $\sum_i |leaves(v_i) \cap L(t)| \geq N - e$, where $N = |L(t)|$;
(D3) $\max_i count(v_i; L(t))$ is minimized.

A dual formulation arises by asking for at least k nodes such that (D1) and (D2) are satisfied, while (D3) asks to maximize $\min_i count(v_i; L(t))$. The DIVIDE operation without the e parameter was introduced in [4] in the context of databases with highly structured meta-data for each attribute. By minimizing the maximum relevance in each of the k subtrees, the DIVIDE operation makes sure that the leaves in $L(t)$ are more or less evenly divided among the k parts. Thus it allows for the possibility that a small subtree (e.g., one a low level of \mathcal{T}) is as important as a large sub-tree at a much higher level. The rationale for allowing the parameter e is to handle the fact that among the leaves in $L(t)$, there might be a small fraction of outlier leaves that do not admit a clean partition into k regions. Namely, if we try to partition all leaves in $L(t)$ into k partitions, the existence of a very small number ($O(k)$) of outliers in, say, random locations, will result in the roots of the k subtrees to be pushed higher up in the tree so as to cover these outliers. For example, for the query “baseball”, the node `Top/Sports/Baseball` is the perfect result for $k = 1$, but the existence of a document about a football player who also plays baseball in `Top/Sports/Football/NFL/Players/ABC` will result in their least common ancestor, namely `Top/Sports`, to be returned instead. This seriously undermines the specificity of the result computed. On the other hand, if we set $k = 2$, rather than split `Top/Sports/Baseball` into, say `Top/Sports/Baseball/Professional` and `Top/Sports/Baseball/Colleges_and_Universities`, we will have two regions, `Top/Sports/Baseball` and `Top/Sports/Football/NFL/Players/ABC`, placing all of the former subtree as equally significant to the latter irrelevant node.

2.3 Bi-criteria formulations for highlights

Given a term $t \in \mathcal{W}$, as before, let $L(t)$ denote the set of leaves relevant for the term t . The DIVIDE operation attempted to pick k disjoint sub-trees such that the total *count* measure of the k sub-trees is at least $N - e$, while ensuring that no sub-tree has too high a *count* measure. The next two formulations are based on the *focus* and *coverage* measures, and attempt to pick nodes that offer a balance between these two measures.

Given an integer k and a real number $p \in [0, 1]$, the goal of the MINFOCUSOPTCOVERAGE operation is to find at most k nodes of \mathcal{T} such that the subtrees under these nodes are disjoint, at least p fraction of the subtree under each of these nodes belongs to $L(t)$, and the subset of $L(t)$ covered by the subtrees under these nodes is as large as possible. Formally:

MINFOCUSOPTCOVERAGE(t, k, p)

- Given: $t \in \mathcal{W}$, $k \in \mathbf{Z}$, $p \in [0, 1]$, compute a set of at most k nodes $v_1, \dots, v_k \in \mathcal{N}$ such that
(FC1) the nodes v_1, \dots, v_k form an anti-chain in \mathcal{T} , namely $leaves(v_i) \cap leaves(v_j) = \emptyset$ for $i \neq j$;
(FC2) for each i , $focus(v_i; L(t)) \geq p$;
(FC3) $\sum_i coverage(v_i; L(t))$ is maximized.

The dual operation is the MINCOVERAGEOPTFOCUS op-

eration, where we have:

MINCOVERAGEOPTFOCUS(t, k, r)

- Given: $t \in \mathcal{W}$, $k \in \mathbf{Z}$, $r \in [0, 1]$, compute a set of at least k nodes $v_1, \dots, v_k \in \mathcal{N}$ such that
(CF1) the nodes v_1, \dots, v_k form an anti-chain in \mathcal{T} , namely $leaves(v_i) \cap leaves(v_j) = \emptyset$ for $i \neq j$;
(CF2) for each i , $coverage(v_i; L(t)) \geq r$;
(CF3) $Avg_i focus(v_i; L(t))$ is maximized.

2.4 Multi-term queries

To support the summarization of multi-term queries (e.g., with conjunction as the semantics), a straightforward approach is to summarize (as a gist or as a set of highlights) the set of leaves that have all terms in the query. This has the drawback that in a finely-partitioned hierarchy (e.g., a digital library into sections or pages of books), none of the leaf nodes might have all query terms. On the other hand, when internal nodes are allowed to be returned as results of the summarization, there is no reason why we should take this approach. We offer a novel approach for this case; for simplicity, we describe the formulation for two-term queries.

To obtain an analog of DIVIDE for computing gists of leaf sets for the case of two-term queries with AND semantics, we start with the dual formulation, and modify conditions (D2) and (D3) further:

Given: $t_1, t_2 \in \mathcal{W}$, $k, e \in \mathbf{Z}$, compute a set of $\geq k$ nodes $v_1, \dots, v_k \in \mathcal{N}$ such that

- (2D1) the nodes v_1, \dots, v_k form an anti-chain in \mathcal{T} , namely $leaves(v_i) \cap leaves(v_j) = \emptyset$ for $i \neq j$; (2D2a) $\sum_i |leaves(v_i) \cap L(t_1)| \geq N_1 - e$, where $N_1 = |L(t_1)|$;
(2D2b) $\sum_i |leaves(v_i) \cap L(t_2)| \geq N_2 - e$, where $N_2 = |L(t_2)|$;
(2D3) $\min_i \min\{count(v_i; L(t_1)), count(v_i; L(t_2))\}$ is maximized.

Thus we seek at least k disjoint sub-trees that cover all but e leaves in each set $L(t_1)$ and $L(t_2)$, while ensuring that no sub-tree has too few leaves from $L(t_1)$ or from $L(t_2)$.

To obtain analogs of the bi-criteria formulations for computing highlights of leaf sets, we first obtain equivalent definitions for *focus* and *coverage* via a probabilistic viewpoint, and then obtain natural extensions of these definitions for the case of two-term queries.

Recall that for a leaf set L and a node v of \mathcal{T} , we defined $focus(v; L)$ to be $|leaves(v) \cap L| / |leaves(v)|$ and $coverage(v; L)$ to be $|leaves(v) \cap L| / |L|$. Equivalently, we may define *focus* by the following probabilistic process. Repeatedly pick a leaf from $leaves(v)$ uniformly at random (with replacement) until a leaf from L has been chosen; if F denote the expected number of trials in this process, then define $focus(v; L)$ to be $1/F$. Similarly, we may define *coverage* by the following probabilistic process. Repeatedly pick a leaf from L uniformly at random (with replacement) until a leaf from $leaves(v)$ has been chosen; if C denote the expected number of trials in this process, then define $coverage(v; L)$ to be $1/C$. It is easy to see that these definitions are equivalent to the ones given in Section 2.3.

With the probabilistic viewpoint, we define the analogs of *focus* and *coverage* for two-term queries. Repeatedly pick a leaf from $leaves(v)$ uniformly at random (with replacement) until at least one leaf has been chosen from each of $L(t_1)$ and $L(t_2)$; if F denote the expected number of trials in this process, then define $focus(v; L(t_1, t_2))$ to be $1/F$. Similarly, we define $coverage(v; L(t_1, t_2))$ as follows. Repeatedly pick a leaf from $L(t_1)$ uniformly at random (with replacement)

until a leaf from $leaves(v)$ has been chosen, then repeatedly pick a leaf from $L(t_2)$ uniformly at random (with replacement) until a leaf from $leaves(v)$ has been chosen; if C denote the expected number of trials in this process, then define $coverage(v; L)$ to be $1/C$.

Once $focus$ and $coverage$ are defined, the bi-criteria formulations of $MINFOCUSOPTCOVERAGE$ and $MINCOVERAGEOPTFOCUS$ apply in a straightforward way with respect to these definitions.

3. ALGORITHM FOR $DIVIDE$

We now turn to the presentation of our algorithms for the $DIVIDE$ operation.

Let L be a given set of leaves of \mathcal{T} , possibly corresponding to $L(t)$ for some query term t . Let T' denote the portion of \mathcal{T} induced by the set L of leaves. Let $N = |L|$ denote the number of leaves, and let k and e be integer parameters.

Recall that we wish to produce an anti-chain A of k nodes of \mathcal{T} that cover all but e of the N leaves in L , while minimizing the number of leaves of L covered by any one node of A . This leads to a natural cost function which we will recursively compute; the resulting dynamic programming algorithm is analogous to the one in [4] for the case $e = 0$. Namely, for nodes v of \mathcal{T} and integers l and f such that $0 \leq l \leq k$ and $0 \leq f \leq e$, let $cost(v, l, f)$ denote the cost of the optimal solution to the sub-problem of solving $DIVIDE$ on the set $L \cap leaves(v)$ with parameters l and f , that is, the number of leaves in any of the l parts in the optimal solution that covers all but f of the leaves in $L \cap leaves(v)$. This cost function admits the following recursive decomposition. Let d denote the number of children of v , and let v_1, \dots, v_d denote its children.

$$cost(v, 0, 0) = \infty \text{ if } v \in L;$$

$$cost(v, 0, 1) = 0 \text{ if } v \in L;$$

$$cost(v, 1, 0) = 1 \text{ if } v \in L;$$

For $v \notin L$, if $\neg(l = 1 \wedge f = 0)$, then

$$cost(v, l, f) = \min \max\{cost(v_i, l_i, f_i)\},$$

where the minimum is taken over values l_1, \dots, l_d such that $0 \leq l_1, \dots, l_d \leq l, \sum l_j = l$, and f_1, \dots, f_d such that $0 \leq f_1, \dots, f_d \leq f, \sum f_j = f$.

For $v \notin L$, $cost(v, 1, 0) = |leaves(v) \cap L|$,

The above recursive formulation leads to a natural dynamic programming algorithm, where we start the computation of $cost$ at the leaves of T' and proceed toward the root. Along with the entry $cost(v, l, f)$, we also keep track of the corresponding values of l_1, \dots, l_d and f_1, \dots, f_d to enable the computation of the optimal decomposition. If Δ denotes the maximum number of children of any node in T' , then the dynamic programming algorithm runs in time $O(Nk^\Delta e^\Delta)$ and uses space $O(Nke)$.

Clearly, for even moderately large values of k and e , if the tree has non-trivial branching factor, this algorithm is impractical. There is a natural modification of this algorithm that improves the running time to $O(Nk^2 e^2)$ by converting the tree into a binary tree by adding extra internal nodes. The resulting algorithm is modestly practical, but still suffers from some drawbacks. Besides still being inefficient, the addition of extra internal nodes leads to the possibility that the resulting set of nodes might include some of the newly added *un-annotated* nodes whose semantics are not clear.

To address these difficulties, we devise a probabilistic algorithm that offers significant speed-up, besides usually elim-

inating the need to binarize the tree. Specifically, we show that if we pick a random subset of $n = O(k^2 \log k)$ leaves and apply the optimal algorithm on these n leaves, we obtain an $O(nk^2 e^2) = \tilde{O}(k^4 e^2)$ time algorithm whose quality is essentially as good as that of the optimal algorithm run on all the leaves (provided that each of the k sub-trees output by the optimal algorithm on the original data consists of $\Theta(n/k)$ fraction of the leaf set we wish to summarize).

We outline the two key steps in the analysis of this probabilistic algorithm. Let A denote an optimal solution to the $DIVIDE$ problem, where a set L of leaves is divided into k parts rooted at some nodes v_1, \dots, v_k . The first lemma says that if we pick $n = O(k^2 \log k)$ samples from the set of N leaves, with high probability, we will pick at least $n/2k$ samples from each of the k parts in the result set A .

Lemma 1. Let L_1, \dots, L_k denote subsets of $\{1, \dots, N\}$, where each L_i satisfies $N/2k \leq |L_i| \leq 2N/k$. Then there is a constant c such that if we pick a set S of $n \geq ck^2 \log k$ samples uniformly at random from $\{1, \dots, N\}$, then with probability at least 0.99, $n/4k \leq |S \cap L_i| \leq 4n/k$ for $1 \leq i \leq k$.

The proof of this lemma follows from standard Chernoff bounds. It follows from the lemma that a sample of size n is adequate to intersect each region of interest in the correct proportion. The next concern is to argue that the same nodes v_1, \dots, v_k will emerge in the solution obtained by running the optimal algorithm on the set of n samples. This will be based on the following lemma.

Lemma 2. Let Λ be a set of leaves of a tree rooted at a node v . Let Δ denote the number of children of v . Let $0 < \alpha \leq 1$. Assuming that no child u of v contains more than $3/4$ of the nodes in Λ in its subtree, if we pick $\eta \geq 3 \log \frac{\Delta}{\alpha}$ samples from Λ uniformly at random, then with probability at least $1 - \alpha$, the least common ancestor of the chosen samples will be v .

PROOF. Note that if we pick at least one leaf each from the subtrees of two or more distinct children of v , the least common ancestor of the samples must be v . This fails to happen iff all samples are contained in the subtree below a single child of v . For any child u of v , the probability that all samples fall in $subtree(u)$ is at most $(|leaves(u) \cap \Lambda|/|\Lambda|)^\eta \leq (3/4)^\eta$. The probability that this event happens for some child u of v is at most $\Delta(3/4)^\eta \leq \Delta(3/4)^{3 \log(\Delta/\alpha)} \leq \Delta(1/2)^{\log(\Delta/\alpha)} \leq \alpha$, since $(3/4)^3 \leq (1/2)$. \square

Taking $\alpha = 1/(8k)$, we need at least $3 \log(k\Delta)$ of our n samples to fall inside L_i for $1 \leq i \leq k$ (recall that the L_i 's are taken to be the set of leaves in an optimal partition). The considerations for Lemma 1 suggested $n \geq ck^2 \log k$ samples. If each L_i satisfies $N/(2k) \leq |L_i| \leq 2N/k$, then the expected number of samples in each L_i is $cn \log k$; for a suitable choice of c , this bound also implies that whp. at least $3 \log(k\Delta)$ samples will fall inside each of the L_i 's, and by Lemma 2, it will follow that the lca of the set of samples in each L_i is exactly the node v_i in the optimal solution. We summarize this in the following theorem.

THEOREM 1. *Let A denote an optimal solution to the $DIVIDE$ problem, where a set L of leaves is divided into k*

parts L_1, \dots, L_k ; let v_1, \dots, v_k denote the anti-chain corresponding to the partition. Let Δ denote a bound on the number of children of any of the v_i 's. Assume that each L_i satisfies $N/(2k) \leq |L_i| \leq 2N/k$ and that there is no child u of any v_i such that the subtree under u contains more than $3/4$ of all nodes in L_i . Then picking $n = O(k^2 \log k)$ samples and computing the optimal solution for the n samples chosen will yield a result whose quality is no more than a factor of 2 from the optimum.

By suitably choosing a constant factor more samples, we could improve the above factor to $1 + \epsilon$ for any $\epsilon > 0$. Also, the constants $1/2$ and 2 in the first assumption can be replaced by any constants α, β such that $0 < \alpha < 1$ and $\beta > 1$, respectively. The skew factor $3/4$ can be replaced by any constant less than 1. All of these only require a constant factor increase in the number of samples. Finally, we note that after the sampling step, the tree \mathcal{T}' induced by the union of the paths from the samples to the root of \mathcal{T} is, whp. a tree with low average branching factor, hence computing the dynamic programming solution is efficient.

Notice that picking n samples out of N leaves also implies that we will suitably scale down the e parameter to $e' = en/N$. This results in a final running time of $O(nk^2e'^2) = O(k^8 \log^3 ke^2/N^2)$; if $e = \epsilon N$, then this running time is $O(\epsilon^2 k^8 \log^3 k)$. Despite the large power of k , our experiments show that in fact, for very small number of samples (typically 5% of the set of leaves), the performance of the algorithm is excellent.

4. ALGORITHMS FOR THE BI-CRITERIA FORMULATIONS

To obtain optimal algorithms for $\text{MINFOCUSOPTCOVERAGE}$ and $\text{MINCOVERAGEOPTFOCUS}$, we first pre-compute some quantities that will be used by the algorithms. We assume that via suitable pre-processing and data structures, given any node u , we can generate the parent of u in $O(1)$ time, as well as the size of $\text{subtree}(u)$ in $O(1)$ time.

Given a term t and the set $L(t)$ of leaves that are relevant to t , first compute, for each node u of the tree, the quantity $C(u) \doteq \text{count}(u; L(t))$ (which, recall, is defined as $|\text{leaves}(u) \cap L(t)|$). Note that $L(t)$ is usually much smaller than the number of nodes in the tree, so $C(u)$ is zero for most u 's; therefore, for efficiency, it is important to store only the pairs $(u, C(u))$ when $C(u) > 0$. These quantities can be computed by performing a walk starting from each leaf in $L(t)$ to the root of the tree \mathcal{T} , in $O(Nd)$ steps, where $N = |L(t)|$ and d is the depth of \mathcal{T} .

Once $C(u)$ is available for each u , we proceed to solve $\text{MINFOCUSOPTCOVERAGE}$ as follows: produce the subset $\mathcal{N}_p = \{u \mid C(u) \geq p|\text{leaves}(u)|\}$, and sort the elements of \mathcal{N}_p in descending order of $\text{coverage}(u, L(t))$, that is, by $C(u)$. Finally, process the elements of \mathcal{N}_p in this sorted order: if v is the next available element, include v in the output if and only if no ancestor of v has already been picked. The running time is clearly bounded by $O(Nd \log(Nd) + Nkd^2)$, where the first term is for sorting up to $O(Nd)$ nodes, and the second term is to check, for each candidate v , if any ancestor of v has already been picked. Note that, in practice, we are unlikely to explore $\Omega(Nd)$ nodes from the sorted list, and the algorithm usually terminates much faster.

The proof of correctness follows from the following two observations. Let U denote the output of our algorithm,

and let $U^* \neq U$ denote any optimal solution to the problem. Sort the elements of U and U^* in descending order of their coverage with respect to $L(t)$. Let u denote the first vertex in U^* that is not in U . The reason why we did not include u in the output is one of two possibilities: (1) either we have already included in the output some u' that is an ancestor of u , in which case we know that $\text{coverage}(u', L(t)) \geq \text{coverage}(u, L(t))$; (2) or we have already produced k nodes in the output, in which case, since we processed the elements in decreasing order of coverage, every element output has coverage at least as good as that of u ; since u had the highest coverage among members of $U^* \setminus U$, every element we output has coverage at least as good as every element of U^* that we did not include.

The algorithm for $\text{MINCOVERAGEOPTFOCUS}$ is similar. Produce the subset $\mathcal{N}_r = \{u \mid C(u) \geq rN\}$, and sort its elements in descending order of $\text{focus}(u, L(t))$, that is, by $C(u)/|\text{leaves}(u)|$. Finally, pick elements from this sorted list, in order of $C(u)/|\text{leaves}(u)|$: if v is the next available element, include v in the output if and only if no descendant of v has already been picked. The latter tests can be easily done in $O(k)$ time, for a total running time of $O(Nd \log(Nd) + Nkd^2)$, again a conservative estimate.

THEOREM 2. *Given a set L of N leaves of a tree \mathcal{T} of depth d , optimal solutions to $\text{MINFOCUSOPTCOVERAGE}$ and $\text{MINCOVERAGEOPTFOCUS}$ can be computed in worst-case time $O(Nd \log(Nd) + Nkd^2)$.*

Finally, we briefly describe how we can efficiently compute focus and coverage values for multi-term queries, as formulated in Section 2.4. Again, we assume that with linear-time pre-processing, we have, for each u , the quantities $|\text{leaves}(u)|$, $\text{count}(u; L(t_1))$, $\text{count}(u; L(t_2))$, and $\text{count}(u; L(t_1 \wedge t_2))$ available in $O(1)$ time. Here $\text{count}(u; L(t_1 \wedge t_2))$ denotes the number of leaves in $\text{subtree}(u)$ that are in $L(t_1) \cap L(t_2)$. Equivalently, we have $\text{focus}(u; L(t_1))$ and $\text{focus}(u; L(t_2))$, and $\text{focus}(u; L(t_1 \wedge t_2))$.

To compute $\text{focus}(u; L(t_1, t_2))$, we proceed as follows. Let $f_1 = \text{focus}(u; L(t_1))$, $f_2 = \text{focus}(u; L(t_2))$; also define $f = \text{focus}(u; L(t_1 \wedge t_2))$. Thus a fraction f_1 of leaves under u have t_1 , a fraction f_2 of leaves under u have t_2 , and a fraction f of leaves under u have both t_1 and t_2 . Define $f_0 = 1 - (f_1 + f_2 + f)$, the fraction of leaves under u that have neither t_1 nor t_2 . The probability that j trials of picking leaves from the subtree of u fail to produce at least one leaf in $L(t_1)$ and at least one leaf in $L(t_2)$ can now be evaluated as follows. The probability that each of j trials misses $L(t_1)$ is precisely $(1 - f_1)^j$; the probability that each of j trials misses $L(t_2)$ is precisely $(1 - f_2)^j$; and the probability that each of j trials misses both $L(t_1)$ and $L(t_2)$ is precisely f_0^j . Thus, the probability that all j trials miss either $L(t_1)$ or $L(t_2)$ completely is precisely $\text{miss}(j) \doteq (1 - f_1)^j + (1 - f_2)^j - f_0^j$, and the probability that we successfully hit both $L(t_1)$ and $L(t_2)$ in j trials is exactly $1 - \text{miss}(j)$. It follows that the expected number of trials to hit both $L(t_1)$ and $L(t_2)$ is precisely $\sum_{j=1}^{\infty} j \text{miss}(j - 1) (1 - \text{miss}(j))$, where we take $\text{miss}(0) = 1$. In practice, if we truncate this computation after t steps, the error incurred is exponentially small in t ; an excellent practical alternative for the case when one of f_1 and f_2 is much larger than the other is to approximate $\text{focus}(u; L(t_1, t_2))$ by $\min\{\text{focus}(u; L(t_1)), \text{focus}(u; L(t_2))\}$.

The values of $\text{coverage}(u; L(t_1, t_2))$ can be computed by a similar linear-time computation; the details are omitted.

5. EXPERIMENTS

In this section, we will show the efficacy of our algorithms and present results to demonstrate their efficiency. We ran our algorithms on a corpus of nearly 4.5 million documents organized in a tree of 650K+ nodes. Our corpus was drawn from the Open Directory Project (ODP — <http://dmoz.org>). The leaves consist of URLs and the internal nodes represented topics at various granularities in the ODP hierarchy.

The content for each leaf was drawn from the title and the ODP description of the web page. The index consisted of 1.7 million unique terms and nearly 80 million postings. By eliminating the most and least frequent terms, we arrived at an index with nearly 50 million postings for nearly 41K unique terms, where the frequency of each term varied between 100 and 100,000.

Our experiments were conducted with the following settings: $k = 10, 20$ (the number of results desired) for gists or highlights; for gists, we set the number of samples to 100, 200, 300 and 400, and the error parameter $e = 0$; for highlights, we set minimum focus = 3% to 15% (in steps of 3%), minimum coverage = 1% to 5% (in steps of 1%).

In terms of speed, we will merely note that for all values of parameters, the algorithms were blazingly fast, and ran in the order of tens of milliseconds. In the rest of this section, we will present two types of results: the first set of results present the values of average focus achieved for various values of minimum coverage, and the values of total coverage achieved for various values of minimum focus; the second set of results will include the actual results for four randomly chosen (interesting) queries. Let us note that the queries chosen to demonstrate the quality of results were drawn from a pool of 6100 random terms from the corpus. For lack of space, we illustrate the results produced by our algorithms on two single term queries **parmesan** and **acrylic**.

Table 1 illustrates the trade-off between focus and coverage, averaged over the query set. The first three columns of the table present the values of average focus (per result node) achieved when the minimum coverage parameter was set to values from 1–5%. Naturally, the lower the minimum coverage, higher the average focus, since these nodes tend to be lower in the tree (and hence more sharply focused on leaves in the result sets). Also, note that going from $k = 10$ to $k = 20$ nearly halves the focus for low values of coverage, since requiring more nodes with a minimum coverage guarantee forces the algorithm to look for such nodes higher in the tree. The analysis is similar for columns 4–6 of the table, and is omitted due to lack of space.

Min (%) coverage	Focus		Min (%) focus	Coverage	
	$k = 10$	$k = 20$		$k = 10$	$k = 20$
1	22.3	13	3	34	40.2
2	11.2	7.5	6	26.8	32.3
3	7.8	7	9	22.5	27.5
4	6.9	6.8	12	20.3	24.7
5	6.5	6.5	15	18.4	2.1

Table 1: Trade-offs in Focus vs. coverage

Table 2 presents an illustration of our algorithms for the query **parmesan**. The first part of the table shows the results for the DIVIDE operation with 200 samples and $k = 15$. As anticipated, this returns a high-level gist of the result set,

and comprises nodes at various levels of the tree. In the case of the node **Home/Cooking**, numerous sub-trees contain several leaves in the result set, so the higher-level node **Home/Cooking** is picked by the algorithm. On the other hand, most of the **Business** sub-tree is not related to cheese or even to food products, we find three nodes that are much more focused on the topic of parmesan cheese. The node **World** appears since the query is relevant to various geographic regions of the world, from Canada to Italy.

DIVIDE with 200 samples and $k = 15$
Home/Cooking
Regional
Business/Food_and_Related_Products/Dairy/Cheese
Business/Food_and_Related_Products/Import_and_Export
Business/Food_and_Related_Products/Baked_Goods/Pizza
World
Recreation/Food/Cheese/Types
Shopping/Food/Ethnic_and_Regional/European/Italian

MINFOCUSOPTCOVERAGE for focus = 15%
Home/Cooking/Fish_and_Seafood/Tilapia
Home/Cooking/Fish_and_Seafood/Flounder
Home/Cooking/Fruits_and_Vegetables/Sea_Vegetables/Samphire
Recreation/Food/Cheese/Types
World/Franais/Rgional/Amrique/Canada/Qubec/ Municipalits/S/Saint-Bruno
Regional/Europe/Italy/Regions/Emilia-Romagna/Localities /Parma/Travel_and_Tourism

MINCOVERAGEOPTFOCUS for coverage = 5%
Home/Cooking/Sandwiches/Hot
Home/Cooking/Fish_and_Seafood/Shellfish
Home/Cooking/Fruits_and_Vegetables
Home/Cooking/Soups_and_Stews/Fruit_and_Vegetable
Regional

MINCOVERAGEOPTFOCUS for coverage = 1%
Recreation/Food/Cheese/Types
Home/Cooking/Fish_and_Seafood/Tilapia
Home/Cooking/Fish_and_Seafood/Flounder
Home/Cooking/Soups_and_Stews/Fruit_and_Vegetable/Turnip
Home/Cooking/Soups_and_Stews/Fruit_and_Vegetable/Ratatouille
Home/Cooking/Fish_and_Seafood/Shellfish/Mussel
Home/Cooking/Fruits_and_Vegetables/Asparagus
Home/Cooking/Fruits_and_Vegetables/Brussels_Sprouts
Home/Cooking/Fish_and_Seafood/Shellfish/Scallop
Home/Cooking/Baking_and_Confections/Crackers

MINFOCUSOPTCOVERAGE for focus = 3%
Home/Cooking/Fish_and_Seafood/Shellfish
Home/Cooking/Sandwiches/Hot
Home/Cooking/Soups_and_Stews/Fruit_and_Vegetable/Ratatouille
Home/Cooking/Fruits_and_Vegetables/Brussels_Sprouts
Home/Cooking/Fish_and_Seafood/Tilapia
Home/Cooking/Baking_and_Confections/Crackers
Home/Cooking/Fish_and_Seafood/Flounder
Home/Cooking/Sandwiches/Spreads_and_Fillings
Business/Food_and_Related_Products/Dairy/Cheese
Regional/Europe/Italy/Business_and_Economy/Shopping/Food

Table 2: Results for the query **parmesan**

The second part of Table 2 presents results for the bi-criteria algorithm MINFOCUSOPTCOVERAGE with minimum focus set at 15%, that is, each node in the result set has at least 15% of its leaves refer to the term **parmesan**. Interestingly, we find three very specific nodes in the topic tree, one corresponding to the Parma region of Italy, the home of parmesan cheese, one about the municipality of Saint-

Bruno in Quebec, and one about cheese types. The third part of the table presents results for the bi-criteria algorithm MINCOVERAGEOPTFOCUS with minimum coverage set at 5%; thus we are looking for up to ten nodes each of which covers a significant (5%) fraction of the result set. Here, four of the five results produced are under **Home/Cooking**, offering a finer picture of where the results in this sub-tree come from. The last two parts of the table illustrate the ability of the algorithm to automatically drill down further into the tree to find interesting highlights of results. Finally, we note that the algorithm for gists produces nodes across the category tree, since it is required to cover all (or almost all) results, whereas the bi-criteria algorithms pick interesting highlights from lower in the tree (min. focus) or higher in the tree (min. coverage), and these tend to be picked from sub-trees what have large number of leaves in the result set (e.g., **Home/Cooking** in this case).

Table 3 presents the results of the algorithms for the query **acrylic**, which has obvious presence in various categories in the ODP hierarchy. This time, we offer a different presentation of the result that might be valuable in navigating through search results in the context of hierarchies. Namely, we took the results of the gist algorithm and interwove them with the results of the highlights algorithms (MINCOVERAGEOPTFOCUS with minimum coverage of 1% and MINFOCUSOPTCOVERAGE with minimum focus of 3%); this yields natural “annotations” of the gist produced by highlighting more specific regions of the tree. In the table, the highlights are presented with a leading “—” character.

6. CONCLUSIONS

In this paper, we addressed some fundamental considerations of information retrieval in the context of large digital libraries, where logical notions of documents exist at various granularities within a hierarchy. We presented efficient algorithms for two basic problems in this context. A particular highlight of our work is the combination of probabilistic sampling and dynamic programming to obtain near-optimal solutions to the problem of summarizing large result sets, exploiting the hierarchical structure of the corpus. In addition, we presented natural notions of precision and recall applicable to hierarchically organized corpora. Our experiments demonstrate that our algorithms are highly efficient, and also yield qualitatively excellent results. A possible avenue for future work is to validate the tree IR paradigm on a large corpus of books decomposed into fine-grained documents that capture chapters and sections.

7. REFERENCES

- [1] Z. Bar-Yossef and S. Rajagopalan. Template detection via data mining and its applications. In *WWW*, 2002.
- [2] D. Carmel, Y. Maarek, M. Mandelbrot, Y. Mass, and A. Soffer. Searching XML documents via XML fragments. In *SIGIR*, pages 151–158, 2003.
- [3] S. Chakrabarti. Integrating the document object model with hyperlinks for enhanced topic distillation and information extraction. In *WWW*, 2001.
- [4] R. Fagin, R. Guha, R. Kumar, J. Novak, D. Sivakumar, and A. Tomkins. Multi-structural databases. In *PODS*, pages 184–195, 2005.
- [5] S. Gollapudi and D. Sivakumar. Framework and algorithms for trend analysis in massive temporal data

Business
— /Textiles_and_Nonwovens/Fibers
— /Textiles_and_Nonwovens/Textiles/Carpets
— /Textiles_and_Nonwovens/Textiles/Bedding_and_Furnishings
— /Textiles_and_Nonwovens/Textiles/Fabrics
— /Textiles_and_Nonwovens/Industrial_Yarns_and_Sewing_Threads
— /Textiles_and_Nonwovens/Industrial_Yarns_&_Sewing_Threads/Mixed_Blends/Europe
— /Textiles_and_Nonwovens/Industrial_Yarns_&_Sewing_Threads/Mixed_Blends/Asia
— /Consumer_Goods_and_Services/Awards
— /Retail_Trade/Store_Fixtures
— /Retail_Trade/Store_Fixtures/Point_of_Purchase_Displays
— /Chemicals/Coatings_and_Adhesives
Shopping
— /Visual_Arts
— /Visual_Arts/Painting/Artists/Acrylics
Arts/Illustration
Arts/Visual_Arts/Painting/Painters
— /Contemporary
— /Acrylics
— /Abstractism
— /Abstract_Expressionism
— /Landscapes
Arts/Visual_Arts/Multiple_Media_Artists
Arts/Visual_Arts/Personal_Pages
Arts/Visual_Arts/Native_and_Tribal/North_America/Artists_and_Artisans
Arts/Visual_Arts/Sculpture/Installations
Arts/Visual_Arts/Galleries/North_America/United_States/New_Mexico
Arts/Visual_Arts/Drawing/Artists/Portraits
Regional
Science/Instruments_and_Supplies/Laboratory_Equipment
Recreation/Collecting/Sports/Display
Health/Beauty/Schools/Cosmetology
Adult/Arts/Visual_Arts/Multiple_Media_Artists

Table 3: Results for query acrylic

- sets. In *CIKM*, pages 168–177, 2004.
- [6] L. Lakshmanan, J. Pei, and J. Han. Quotient cube: How to summarize the semantics of a data cube. In *VLDB*, pages 778–789, 2002.
- [7] D.J. Lawrie and W.B. Croft. Generating hierarchical summaries for web searches. In *SIGIR*, pages 457–458, 2002.
- [8] K.S. McCurley and N. Eiron. Untangling compound documents on the web. In *Proc. ACM Hypertext Conference*, pages 85–94, 2003.
- [9] S. Sarawagi. User-adaptive exploration of multidimensional data. In *VLDB*, pages 307–316, 2000.