The Data Warehouse of Newsgroups

Himanshu Gupta[1*] and Divesh Srivastava[2]

[1] hgupta@db.stanford.edu, Stanford University, Stanford, CA 94305, USA
[2] divesh@research.att.com, AT&T Labs – Research, Florham Park, NJ 07932

Abstract. Electronic newsgroups are one of the primary means for the dissemination, exchange and sharing of information. We argue that the current newsgroup model is unsatisfactory, especially when posted articles are relevant to multiple newsgroups. We demonstrate that considerable additional flexibility can be achieved by managing newsgroups in a data warehouse, where each article is a tuple of attribute-value pairs, and each newsgroup is a view on the set of all posted articles. Supporting this paradigm for a large set of newsgroups makes it imperative to efficiently support a very large number of views: this is the key difference between newsgroup data warehouses and conventional data warehouses. We identify two complementary problems concerning the design of such a newsgroup data warehouse. An important design decision that the system needs to make is which newsgroup views to eagerly maintain (i.e., materialize). We demonstrate the intractability of the general newsgroup-selection problem, consider various natural special cases of the problem, and present efficient exact/approximation algorithms and complexity hardness results for them. A second important task concerns the efficient incremental maintenance of the eagerly maintained newsgroups. The newsgroup-maintenance problem for our model of newsgroup definitions is a more general version of the classical point-location problem, and we design an I/O and CPU efficient algorithm for this problem.

1 Introduction

Electronic newsgroups/discussion groups are one of the primary means for the dissemination, exchange and sharing of information on a wide variety of topics. For example, `comp.databases` and `comp.lang.c`, typically contain articles relevant to computers and computation, while `soc.culture.indian` and `soc.culture.mexican` typically contain articles relevant to some world cultures.

In the current model of posting articles to electronic newsgroups, it is the responsibility of the *author* of an article to determine the newsgroups that are relevant to the article. Placing such a burden on the author of the article has many undesirable consequences, especially since there are thousands of newsgroups currently, and the author of an article may not always know all the relevant newsgroups: (a) articles are often cross-posted to irrelevant newsgroups, resulting in flame wars, and (b) articles that are obviously relevant to multiple newsgroups may not be posted to all of them, missing potentially relevant

* Supported by NSF grant IRI-96-31952.

readers. This unsatisfactory situation will only get worse as the number of newsgroups increase.

In this paper, we present a novel model for managing electronic newsgroups that does not suffer from the above mentioned problems; we refer to it as the *Data Warehouse of Newsgroups* (DaWN) model. In the DaWN model, the author of an article "posts" the article to the newsgroup management system, not to any specific newsgroups. Each newsgroup is defined as a view over the set of *all* articles posted to the newsgroup management system, and it is the responsibility of the system to determine all the newsgroups into which a new article must be inserted. Our *newsgroup views* allow the flexible combination of selection conditions on structured attributes of the article, such as its author and posting date, along with selection conditions on unstructured attributes of the article, such as its subject and body. For example, the newsgroup `soc.culture.indian` may be defined to contain all articles posted in the current year, each of whose bodies is similar to at least one of a set of, say, 100 articles that have been manually determined to be representative of this newsgroup. Similarly, the definition of the newsgroup `att.forsale` may require that the organization of the article's author be AT&T. The ability to automatically classify posted articles into newsgroups based on conditions satisfied by multiple structured and unstructured attributes of the article permits very flexible newsgroup definitions. Clearly, the DaWN model has the potential of bringing together the author and targeted readers of an article.

In this paper, we identify and address two complementary problems that arise when newsgroups are defined as views in the DaWN model.

Newsgroup-selection problem: Given a large set of newsgroups, which of these newsgroups should be eagerly maintained (materialized), and which of them should be lazily maintained, in order to conserve system resources while allowing efficient access to the various newsgroups?
We demonstrate that the general *newsgroup-selection problem* is intractable, which motivates a study of special cases that arise in practice. We consider many natural special cases, based on the DaWN model of articles and newsgroups, and present efficient exact/approximation algorithms for them.

Newsgroup-maintenance problem: Given a new article, in which of the possibly large number of materialized newsgroups must the article be inserted?
We expect any article to be contained in very few newsgroups, while the number of newsgroups supported by the data warehouse may be very large. Thus, an algorithm that iteratively checks whether the new article needs to be inserted in each of the newsgroups would be extremely inefficient. We devise an I/O and CPU efficient solution for the *newsgroup-maintenance problem.*

Both the above problems arise in any data warehouse that supports materialized views. What distinguishes DaWN from conventional data warehouses are the following characteristics: (i) the extremely large number of views defined in DaWN, and (ii) the simple form of individual newsgroup views as selections over the set of all posted articles. While we focus on the data warehouse of newsgroups in this paper, our techniques and solutions are more generally applicable to any data warehouse or multi-dimensional database with the above characteristics, such as data warehouses of scientific articles, legal resolutions, and corporate e-mail repositories.

The rest of this paper is organized as follows. In the next section, we briefly describe the DaWN model. In Section 3, we consider the problem of efficiently maintaining materialized newsgroup views, when new articles arrive into the newsgroup system. In Section 4, we discuss the problem of selecting an appropriate set of newsgroup views to eagerly maintain. We present related work for each of the two problems in their respective sections. We end with concluding remarks in Section 5.

2 The DaWN Model

A data warehouse is a large repository of information available for querying and analysis [IK93,HGMW+95,Wid95]. It consists of a set of materialized views over information sources of interest, using which a family of (anticipated) user queries can be answered efficiently. In this paper, we show the advantages of modeling the newsgroup management system as a data warehouse.

2.1 Article Store: The Information Source

A newsgroup article contains information of two types: *header fields*, and a *body*. The header fields of an article are each identified by a keyword and a value. In contrast, the body is viewed as unstructured text. For the purpose of this paper, we model articles as having d attributes, $A_1, A_2, \ldots, A_d$. For header fields of the article, the keyword is the name of the attribute; the body of the article can be treated as the value of an attribute called `Body`. The various examples in this paper use the commonly specified attributes `From`, `Organization`, `Date`, `Subject`, and `Body` of newsgroup articles, with their obvious meanings.

The *article store* is the set of all posted articles. To facilitate answering queries over the article store, various indexes can be built over the article attributes. For example, the article store may maintain an inverted index structure [Fal85] on the `Body` attribute, and a B-tree on the `Date` attribute. The article store along with the index structures is the *information source* for the data warehouse of newsgroups.

2.2 Newsgroup Views

An *electronic newsgroup* contains a set of articles. Current day newsgroup management systems support only newsgroups that contain articles that have been explicitly posted to those newsgroups. Here, we focus our attention on newsgroups that are defined as *views* over the set of all articles stored in the underlying article store. The articles in the newsgroups are determined automatically by the newsgroup management system based on the newsgroup definitions, and are *not* explicitly posted to the newsgroups by the authors. Conventional newsgroups can also co-exist with such automatically populated newsgroups.

In this paper, we consider newsgroups that are defined as selection views on the article attributes. The *atomic conditions* that are the basis of the newsgroup definitions are of the forms: (a) "attribute *similar-to* typical-article-body *with-threshold* threshold-value", (b) "attribute *contains* value", and (c) "attribute $\{\leq, \geq, =, \neq, <, >\}$ value".

Given an article attribute A_i, an *attribute selection condition* on A_i is an arbitrary boolean expression of atomic conditions on A_i. A *newsgroup-view* definition is a conjunction of attribute selection conditions on the article attributes, i.e., we consider newsgroups V defined using selection conditions of the form

$$\bigwedge_{j \in I}(f_j(A_j))$$

where $I \subseteq \{1, 2, \ldots, d\}$ is known as the *index set* of newsgroup V and $f_j(A_j)$ is an attribute selection condition on attribute A_j. We expect the size of the index set $|I|$ of typical newsgroups, that are defined as views, to be small compared to the total number of article attributes.

For example, the newsgroup `att.forsale` may be defined as "($\wedge$ (`Date` $\geq$ 1 Jan 1998) (`Organization` = AT&T) (`Subject` *contains* Sale))". A more interesting example defines the newsgroup `soc.culture.indian` as "($\wedge$ (`Date` $\geq$ 1 Jan 1998) ($\vee$ (`Body` *similar-to* B_1 *with-threshold* T_1) ... (`Body` *similar-to* B_{100} *with-threshold* T_{100})))", where the various B_i's are the bodies of *typical-articles* that are representative of the newsgroup, and the T_i's are the desired *cosine similarity match* threshold values [SB88]. Both these examples combine the use of conditions on structured and text-valued unstructured attributes.

2.3 The Data Warehouse and Design Decisions

The data warehouse of newsgroups, defined over the article store, allows users to request the set of all articles in any specific newsgroup; this request is referred to as a *newsgroup query* and is the only type of user query supported by the data warehouse. The newsgroup management system may decide to eagerly maintain (materialize) some of the newsgroups; these newsgroups are the materialized views in DaWN, and are kept up to date in response to additions to the article store. Newsgroup queries can be answered using the materialized views stored in the data warehouse and/or the article store. We use the term *newsgroup management system* to refer to the system consisting of the data warehouses and the article store.

Two of the most important decisions in designing DaWN, the data warehouse of newsgroups, are the following: (a) the selection of materialized views to be stored at the warehouse, for a given family of (anticipated) user queries; such a selection is important given limited amount of resources such as storage space and/or total view maintenance time, and (b) the efficient incremental maintenance of the materialized views, for a given family of information source data updates. The warehouse may maintain various indexes for efficient maintenance of the materialized newsgroups. We further discuss the nature and use of these index structures in Section 3.

In the following sections, we address the above two key issues in the design of DaWN, taking into account the special characteristics that distinguish it from conventional data warehouses: (i) the extremely large number of newsgroup views defined in DaWN, and (ii) the simple form of individual newsgroup views as selections over the set of all posted articles.

3 The Newsgroup-Maintenance Problem

In this section, we consider the newsgroup-maintenance problem, i.e., the problem of efficiently updating a large set of materialized newsgroups, in response to new articles being added to the article store. We start with a precise definition of our problem, and present some related work, before describing our I/O and CPU efficient solution for the problem.

3.1 The Problem Definition

We formally define the *newsgroup-maintenance problem* as follows. Let $V_1, \ldots, V_n$ be the large set of materialized newsgroups in the newsgroup system. Given a new article $m = (b_1, b_2, \ldots, b_d)$, we wish to output the subset of newsgroups that are affected by the posting of the article m to the article store, i.e., the set of newsgroups in which m needs to be inserted.

Given the large number of newsgroups in a typical newsgroup data warehouse, the brute force method of sequentially checking each of the newsgroup definitions to determine if the article needs to be inserted into the newsgroup could be very inefficient. The key challenge here is to devise a solution that takes advantage of the specific nature of our problem, wherein the size of the index set of newsgroups is small, compared to the number of attributes of the article.

3.2 Related Work

The work that is most closely related to our problem of newsgroup-maintenance is that on the classical *point-location problem*. The point-location problem is to report all hyper-rectangles from a given set, that contain a given query point. An equivalent problem that has been examined by the database community is that of the predicate matching problem in active databases and forward chaining rule systems [HCKW90]. However, our problem is more general because our problem combines conditions on ordered domains with conditions on unstructured text attributes. Also, each attribute selection condition on an ordered domain may be a general boolean expression of atomic conditions, for example, a *union* of interval ranges.

There has been a considerable amount of work on the point-location problem [EM81,Ede83a,Ede83b,Cha83] on designing optimal *main-memory* algorithms.

However, there hasn't been any work reported on designing secondary memory algorithms for the point-location problem, that have optimal worst-case bounds. In contrast, there has been some recent work [KRVV93,RS95,VV96] reported for the dual problem of range-searching. The secondary memory data structures developed for the point-location problem like various R-trees, cell-trees, hB-trees have good average-case behavior for common spatial database problems, but do not have any theoretical worst-case bounds. We refer the reader to [Sam89a,Sam89b] for a survey. We note that *none* of the previously proposed algorithms can take advantage of small index sets of newsgroup views, i.e., all of them treat unspecified selection conditions as intervals covering the entire dimension.

3.3 Independent Search Trees Algorithm

In this subsection, we present our I/O and CPU efficient approach called the *Independent Search Trees Algorithm* for solving the newsgroup-maintenance problem. For ease of understanding, we start with a description of the algorithm for newsgroup definitions where each attribute is an ordered domain, and each selection condition is an atomic condition. Thus, each attribute value is restricted to be in a single interval, i.e., each newsgroup is a hyper-rectangle. Later, we will extend it to handling unstructured text attributes and general boolean expressions in the attribute selection conditions.

Newsgroup-Maintenance of Hyper-Rectangles: Consider n newsgroups $V_1, \ldots, V_n$, where each newsgroup V_i is defined as

$$V_i = \bigwedge_{j \in I_i} (f_{ij})$$

and each f_{ij} is of the form $(A_j \in c_{ij})$ for some interval c_{ij} on the respective ordered domain D_j. The data structure we use consists of d external segment tree structures [RS94] $T_1, T_2, \ldots, T_d$, such that tree T_j stores the intervals $\{c_{ij} \mid j \in I_i, 1 \leq i \leq n\}$.

We compute the set of affected newsgroups (views) as follows. We keep an array I of size n, where $I[i]$ is initialized to $|I_i|$, the size of the index set of newsgroup V_i. When an article $m = (b_1, b_2, \ldots, b_d)$ arrives, we search for intervals in the external segment trees T_j that contain b_j, for all $1 \leq j \leq d$. While searching in the segment tree T_j for b_j, when an interval c_{ij} gets hit (which happens when $b_j \in c_{ij}$), the entry $I[i]$ is decremented by 1. If an entry $I[i]$ drops to zero, then the corresponding newsgroup V_i is reported as one of the affected newsgroups. These are precisely the newsgroups in which the article m will have to be inserted.

Handling the "contains" operator: We now extend our Independent Search Trees Algorithm to handle the "contains" operator for unstructured text-valued attributes such as `Subject`. Thus, the newsgroup definitions may now use the *contains* operator, i.e., f_{ij} can be (A_j *contains* s_{ij}) for some string s_{ij} and a text-valued attribute A_j. To incorporate the *contains* operator in our newsgroup definitions, we use *trie* data structures [Fre60] instead of segment trees for text-valued attributes. The question we wish to answer is the following. Given a set of strings $S_j = \{s_{ij} \mid j \in I_i\}$ (from the newsgroup definitions) and a query string b_j (the value of the article attribute A_j), output the set of strings $s_{i_1 j}, s_{i_2 j}, \ldots, s_{i_l j}$ such that b_j *contains* $s_{i_p j}$ for all $p \leq l$. The matching algorithm used in conjunction with the *trie* data structure can be easily modified to answer the above problem. We build a *trie* on the set S_j of data strings. On a query b_j, we search the *trie* data structure for *superstring* matches for each suffix of b_j. The search can be completed in $(|b_j|^2 + l)$ character comparisons, where $|b_j|$ is the size of the query string b_j and l is the number of strings reported. The space requirements of the *trie* data structure is $k|\Sigma|$ characters for storing k strings, where $|\Sigma|$ is the size of the alphabet. Note that the *trie* yields itself to an efficient secondary memory implementation, as it is just a special form of a B-tree.

Handling Selection Conditions on `Body` *Attribute:* We extend our techniques to general newsgroup definitions that may also use *similar-to with-threshold* predicates on the `Body` attribute, say A_d. In particular, for a view V_i, we consider

$$f_{id}(A_d) = \bigvee_k (C(A_d, B_{ik}, T_{ik})),$$

where $C(A_d, B_{ik}, T_{ik})$ is used to represent the predicate (A_d *similar-to* B_{ik} *with-threshold* T_{ik}). Each B_{ik} here is the body of a typical-article that is representative of the V_i newsgroup.

To handle maintenance of such newsgroup definitions, we build inverted lists $L_1, L_2, \ldots, L_p$, where p is the size of the dictionary (set of *relevant* words). Each typical-article body B_{ik} is represented by a *similarity-vector* $B_{ik} = (w_{ik1}, \ldots, w_{ikp})$, where w_{ikl} is the weight of the l^{th} word in B_{ik}. Let the set of all *distinct* similarity-vectors used in the view definitions be $\mathcal{W}_1, \mathcal{W}_2, \ldots, \mathcal{W}_m$. An inverted list L_l keeps a list of all similarity-vectors $\mathcal{W}_j$'s that have a non-zero weight of the l^{th} word. Also, with each similarity-vector $\mathcal{W}_j$, we keep a list, R_j, of all view definitions that use the similarity-vector $\mathcal{W}_j$ along with the corresponding threshold. In other words, $R_j = \{(i, T_{ik}) | \mathcal{W}_j = B_{ik}\}$, stored as an ordered list in increasing order of the thresholds. We also keep a dot-product integer P_j (initialized to zero).

Let $m = (b_1, b_2, \ldots, b_d)$ be a new article posted to the article store, whose `Body` attribute value $b_d = (w_{m1}, w_{m2}, \ldots, w_{mp})$ is the word-vector representing the article body. Each b_j for $j < d$ is searched in the external segment tree or trie T_j, as before, with entries in I decremented appropriately. To compute f_{id}, we sequentially scan the inverted list L_l, for each non-zero value w_{ml} in the new article's word-vector b_d. For each $\mathcal{W}_j$ in L_l, we increment the dot-product value P_j associated with W_j by $(w_{jl} * w_{ml})/|b_d||\mathcal{W}_j|$. After all required inverted lists have been scanned, for each $\mathcal{W}_j$, we scan its list R_j and for each $(i, T_{ik}) \in R_j$, such that $T_{ik} \leq P_j$, we decrement the value of $I[i]$ by 1, making sure that each $I[i]$ is decremented at most once.

Newsgroup-Maintenance of Boolean Expressions: When f_{ij}, the attribute selection condition involving an ordered attribute A_j, is $A_j \notin d_{ij}$ for some interval d_{ij} on domain D_j, we still store d_{ij} in the segment tree T_j. But, whenever d_{ij} is hit (which happens when $b_j \in d_{ij}$), we increase $I[i]$ to d (instead of decrementing by one), guaranteeing that newsgroup V_i is not output. Similarly, we handle f_{ij}'s of the form $\neg(A_j \;\; contains \; s_{ij})$ or $(\neg(C(A_j, B_{ik}, T_{ik})))$ for an unstructured text-valued attribute A_j. Also, in an entry $I[i]$ of array I, we store the size of *positive index set* of V_i instead of the size of V_i's index set. The positive index set I_i^+ of a view V_i is defined as $\{j \mid (j \in I_i) \wedge (f_{ij}$ is either of the form $(A_j \in c_{ij})$ or $(A_j \;\; contains \; s_{ij})$ or $(C(A_j, B_{ik}, T_{ik})))\}$. Similarly, the *negative index set* I_i^- is defined as $\{j \mid (j \in I_i) \wedge (f_{ij}$ is of the form $(A_j \notin d_{ij})$ or $\neg(A_j \;\; contains \; s_{ij})$ or $\neg(C(A_j, B_{ik}, T_{ik})))\}$.

The generalization to arbitrary boolean expressions for ordered domain attributes is achieved as follows. An arbitrary boolean expression f_{ij} for an arithmetic attribute A_j can be represented as $\bigvee_k (A_j \in c_{ijk})$ or as $\bigwedge_k (A_j \notin d_{ijk})$, for some set of intervals c_{ijk} or d_{ijk} on D_j. A segment tree T_j, corresponding to the attribute A_j, is constructed as including all the intervals c_{ijk} or d_{ijk} and corresponding entries in I are decreased by 1, or increased to d on hits to intervals appropriately. If A_j is an unstructured text-valued attribute, we can easily handle boolean expressions of the type $f_{ij} = (\bigvee_k (A_j \;\; contains \; s_{ijk}))$ or $f_{ij} = (\bigwedge_k \neg(A_j \;\; contains \; s_{ijk}))$ for a set of strings s_{ijk}. For the `Body` attribute A_d, we can handle expressions of the type $f_{id} = \bigvee_k (C(A_d, B_{ik}, T_{ik}))$, or $f_{id} = \bigwedge_k (\neg(C(A_d, B_{ik}, T_{ik})))$. Arbitrary boolean expressions in conjunctive normal form (CNF) for unstructured text attributes using *contains* or *similar-to with-threshold* predicate can also be handled by creating a duplicate attribute for each clause in the CNF boolean expression. We note that $\bigvee_k (C(A_d, B_{ik}, T_{ik}))$ is the only type of boolean expression involving the `Body` attribute, that we expect in practice.

For general boolean expressions, the definitions of I_i^+ and I_i^- are appropriately extended to $\{j \mid (j \in I_i) \wedge (f_{ij}$ is either of the form $\bigvee_k (A_j \in c_{ijk})$ or $\bigvee_k (A_j \;\; contains \; s_{ijk})$ or $\bigvee_k (C(A_d, B_{ik}, T_{ik})))\}$ and $\{j \mid (j \in I_i) \wedge (f_{ij}$ is either of the form $\bigwedge_k (A_j \notin d_{ijk})$ or $\bigwedge_k \neg(A_j \;\; contains \; s_{ijk})$ or $\bigwedge_k \neg(C(A_d, B_{ik}, T_{ik}))))\}$ respectively.[1]

Handling Unspecified Article Attributes: An article m is inserted into a view V_i based on the values of only those article attributes that belong to V_i's index set I_i. However, an attribute selection condition f_{ij} can be defined to accept or reject an unspecified attribute value. For example, it is reasonable for the selection condition (`Organization` = "AT&T") to reject articles that have the attribute `Organization` unspecified, while an unspecified attribute value should probably pass the selection condition `Organization` $\neq$ "AOL".

To handle such cases of unspecified attribute values in articles, we maintain two disjoint integer sets P_j and F_j for each attribute A_j, in addition to its index

[1] We assume that $|I_i^+| > 0$ for each i. Else, we will need to keep a list of views with zero $|I_i^+|$ and report them if the entry $I[i] = |I_i^+|$ remains unchanged.

structure. The *pass* list P_j is defined as $\{i \mid (j \in I_i^+) \wedge (f_{ij}$ accepts unspecified values$)\}$. Similarly, *fail* list F_j is defined as $\{i \mid (j \in I_i^-) \wedge (f_{ij}$ rejects unspecified values$)\}$. Thus, if an arriving article m has its A_j attribute's value unspecified, we decrement the entry $I[i]$ by one for each $i \in P_j$ and increment $I[i]$ to d for each $i \in F_j$, instead of searching in the index structure of A_j.

CPU Efficient Initialization of Array I: Whenever a new article arrives in the newsgroup management system, we need to initialize *each* entry $I[i]$ of the array I to $|I_i^+|$, the size of the positive index set of V_i. A simple scheme is to explicitly store (in persistent memory) $|I_i^+|$ for each V_i and initialize each of the n entries of I every time a new article arrives. However, since the article is expected to affect only a few newsgroups, initializing all n elements of the array I can be very inefficient. Below, we present an efficient scheme to find the initial values in I, only for potentially relevant newsgroups.

We number the views in such a way that $|I_j^+| \leq |I_k^+|$ for all $1 \leq j < k \leq n$. We define $d-1$ numbers $t_1, t_2, \ldots, t_{d-1}$, where t_j is such that $|I_{t_j}^+| < |I_{t_j+1}^+|$, i.e., these $d-1$ numbers define the transition points for the initial values in the array I. If no such t_j exists for some $j < d$, then $t_l = n+1$ for all $j \leq l < d$. We create a persistent memory array T of size d, where $T[0] = 0$ and $T[i] = t_i$ for $1 \leq i < d$. Since the array T contains only d elements, it is quite small and hence can be maintained in main memory. To find the initial value of $I[i]$, $|I_i^+|$, we find a number x such that $T[x-1] < i \leq T[x]$ in $O(\log d)$ main-memory time. It is easy to see that $|I_i^+| = x$.

When we have to decrement the value of $I[i]$ for the *first* time, we initialize $I[i]$ to $x-1$, else we reduce the *current* value of $I[i]$ by 1. How do we find out if $I[i]$ has been decremented before or not? We do this by keeping a bit vector H of size n, where $H[i] = 1$ iff $I[i]$ has been decremented before. Both arrays H and I can reside in main-memory, even when thousands of newsgroups are maintained as materialized views. For each new article that arrives into the newsgroup management system, only the bit vector H needs to be reset to 0, which can be done very efficiently in most systems.

I/O Efficiency: We now analyze the time taken by the above algorithm to output the newsgroups affected, when a new article arrives in the newsgroup management system.

Let B be the I/O block size. We define K_j as the number of intervals in the various newsgroup definitions involving an ordered-domain attribute A_j (equivalently, the number of entries in the segment tree T_j), and m_j as the maximum number of intervals in tree T_j that overlap. Using the optimal external segment tree structure of [RS94], we can perform a search in a segment tree T in $\log_B(p) + 2(t/B)$ number of I/O accesses, where p is the number of entries in T, and t is the number of intervals output. Let s_j be the number of similarity-vectors that have a non-zero weight in the j^{th} word, where $j \leq p$. Thus, the scan of the j^{th} inverted list L_j takes s_j/B disk accesses. Therefore, the overall query time complexity of the above algorithm is $O(\sum_{j=1}^{d-1} \log_B(K_j) + 2(\sum_{j=1}^{d-1} m_j)/B +$

$(\sum_{j=1}^{p} s_j)/B)$, assuming that there are no conditions involving the *contains* operator. An unspecified value in the attribute A_j of a new article m results in only $(|P_j| + |F_j|)/B$ disk accesses.

Theorem 1. *Consider n newsgroup views whose definitions are conjunctions of arithmetic attribute selection and unstructured attribute selection conditions. Each attribute selection condition on an ordered-domain attribute A_i is a boolean expression of atomic conditions on A_i.*

The Independent Search Trees Algorithm for newsgroup-maintenance is correct and has a maintenance time of $O(\sum_{j=1}^{d-1} \log_B(K_j) + 2(\sum_{j=1}^{d-1} m_j)/B) + (\sum_{j=1}^{p} s_j)/B)$ disk accesses, where K_j, m_j, and s_j are defined as above.

If an attribute selection condition for a text attribute A_s uses the contains *operator, the maintenance time required to search the index structure (trie) of A_s is $(|b_j|^2 + m_j)/B$, where $|b_j|$ is the length of b_j, the A_s attribute string value in the arriving article, and m_j is the number of newsgroups that match.*

The update time of the data structure due to an insertion of a new view is $O(\sum_{j=1}^{d} \log_B(K_j)) + s$ disk accesses, where s is the number of non-zero weights in the typical-article bodies of the added view.[2]

One of main features of the above described algorithm is that the time-complexity directly depends upon the total number of *specified* attribute atomic selection conditions, unlike any of the previously proposed algorithms for similar problems.

4 The Newsgroup-Selection Problem

An important decision in the design of the newsgroup system is to select an appropriate set of newsgroups to be eagerly maintained (materialized). The rest of the newsgroups are computed whenever queried, using the other materialized newsgroups and/or the article store. A natural optimization criterion is to minimize the storage space and/or newsgroup maintenance time, while guaranteeing that each newsgroup query can be answered within some threshold.

The query threshold of a newsgroup query is essentially the query-time a user request for the newsgroup can tolerate. Heavily accessed important newsgroups would have low query thresholds, while newsgroups with very low query frequencies could tolerate higher query times. The newsgroup-selection problem is to select the most "beneficial" newsgroups to materialize, so that *all* newsgroup queries can be answered within their respective query-time thresholds. Often, materializing only a small subset of newsgroups will be sufficient to answer each newsgroup query within its query-time thresholds. This conserves system resources and facilitates efficient maintenance of materialized newsgroups.

In this section, we first formulate the general problem of selecting newsgroups to be eagerly maintained (materialized), and show that it is, unfortunately, intractable. We then take advantage of our specific model of newsgroup definitions

[2] The array T can be maintained periodically.

as selection views, and present some efficient exact/approximation algorithms and complexity hardness results for the problems.

As with the previous problem of newsgroup-maintenance, the problems addressed here are more generally applicable to the selection of views to materialize in a data warehouse, when the queries are restricted to selections and unions over the underlying sources of information.

4.1 General Problem of Newsgroup-Selection

Consider a labeled bipartite hypergraph $G = (Q \cup V, E)$, where Q is the set of newsgroup queries and V is a set of candidate newsgroups (views) considered for materialization. The set E is the set of hyperedges, where each hyperedge is of the form $(q, \{v_1, v_2, \ldots, v_l\})$, $q \in Q$, and $v_1, v_2, \ldots, v_l \in V$. Each hyperedge is labeled with a *query-cost* of t, signifying that query q can be answered using the set of views $\{v_1, v_2, \ldots, v_l\}$ incurring a cost of t units. With each query node $q \in Q$, there is a query-cost threshold T_q associated, and with each view node $v \in V$, there is a weight (space cost) $S(v)$ associated. We refer to such a graph as a *query-view* graph. This notion of a query-view graph is similar to that used in [GHRU97], but more general. We now define the *newsgroup-selection* problem.

Newsgroup-Selection Problem: *Given a bipartite query-view hypergraph G defined as above, select a minimum weighted set of views $M \subseteq V$ to materialize such that for each query $q \in Q$ there exists a hyperedge $(q, \{v_1, v_2, \ldots, v_l\})$ in G, where views $v_1, v_2, \ldots, v_l \in M$ and the query-cost associated with the hyperedge is less than T_q.*

The above problem is trivially in `NP`. As there is a straightforward reduction from `minimum set cover` to a special case of the newsgroup-selection problem when G has only simple edges, the newsgroup-selection problem is also `NP`-hard. The newsgroup-selection problem is exactly the problem of minimizing the number of leaves scheduled in a 3-level AND/OR scheduling problem with internal-tree precedence constraints [GM97]. The 2-level version of the AND/OR scheduling problem with internal-tree precedence constraints is equivalent to `minimum set cover`, while the 4-level AND/OR scheduling problem with internal-tree constraints is as hard as the `LABEL-COVER` [ABSS93,GM97] problem making it quasi-`NP`-hard[3] to approximate within a factor of $2^{\log^{1-\gamma} n}$ for any $\gamma > 0$. To the best of our knowledge, nothing is known about the 3-level version of the AND/OR scheduling problem with internal tree constraints.

The intractability of the general problem leads us to look at some natural special cases that arise in practice in the context of newsgroup management, and we present efficient algorithms for each of them. Recall that, in Section 2, we allowed newsgroups to be defined only as selection views of a specified form. In such cases, a newsgroup needs only the union ($\cup$) and selection (σ) relational operators to be computed from a set of other newsgroups. However, the above formulation of the newsgroup-selection problem is much more general.

[3] That is, this would imply $\mathtt{NP} \subseteq \mathtt{DTIME}(n^{poly(log n)})$. "A proof of quasi-`NP`-hardness is good evidence that the problem has no polynomial-time algorithm" [AL95].

In the next subsection, we restrict the computation of a newsgroup from other newsgroups to just using the selection operator. We handle the case of using both the union and the selection operators in the subsequent subsection.

4.2 Queries as Selections over Views

In this subsection, we focus on the restricted newsgroup-selection problem where newsgroup queries are computed using only selections, either on some materialized newsgroup in the data warehouse or the article store. For example, if the data warehouse eagerly maintains `comp.databases`, then answering the newsgroup query `comp.databases.object` requires only a selection over the newsgroup `comp.databases`. Being a special case of the general newsgroup-selection problem, the above restricted version of the problem helps in better understanding of the general problem. Moreover, in some domains, the union operation may be very expensive or not feasible. For example, let the newsgroup V_1 contain all articles whose `Subject` *contains* "computer", and the newsgroup V_2 be the set of all articles whose `Subject` *contains* "compute". Though, V_2 can be computed using V_1 *and* the article store using the selection condition ($\wedge$ (`Subject` *contains* "computer") ($\neg$(`Subject` *contains* "compute"))) on the article store, it may be more efficient to compute V_2 using a simple selection over just the article store.

In the above restricted newsgroup-selection problem where newsgroup queries are computed using only the selection operator, a query uses exactly one view for its computation. Hence, the query-view graph defined earlier will have only simple edges. The restricted newsgroup-problem has a natural reduction from the `minimum set cover` problem and hence is also `NP`-complete. However, there exists a polynomial-time greedy algorithm that delivers a competitive solution that is within $O(\log n)$ factor of an optimal solution, where n is the number of newsgroup queries. The greedy algorithm used is almost the same as that used to approximate the `weighted set cover` problem [Chv79]. It can be shown that the solution delivered by the greedy algorithm is within $O(\log n)$ of the optimal solution.

So far, we have not taken any advantage of the specific nature of the atomic conditions used in the newsgroup definitions. We now do so, and restrict ourselves to newsgroups defined using arithmetic operators on the article attributes. As the arithmetic operators used in an atomic condition assume an order on the domain, all defined newsgroups form some sort of "orthogonal objects" in the multidimensional space of the article attributes. We take advantage of this fact and formulate a series of problems, presenting exact or approximate algorithms.

Newsgroup-Selection with Ordered Domains Consider an ordered domain D. Consider newsgroups (views or queries) that are ranges over D. In other words, views and queries can be represented as intervals over D. As we restrict our attention to using only the selection operator for computing a query, a query interval q can be computed using a view interval v only if v completely covers q. With each pair (q, v), where v completely covers q, there is a query-cost associated, which is the cost incurred in computing q from v.

We observe here that the techniques used to solve the various problems of one-dimensional queries/views addressed in this section can also be applied to the more general case when the queries involve intervals (ranges) along one dimension and equality selections over other dimensions.

Problem (One-dimensional Selection Queries) *Given interval views V and interval queries Q over an ordered domain D, select a minimum weighted set of interval views M such that each query $q \in Q$ has a view $v \in M$ that completely contains q and answers the query q within its query-cost threshold T_q.*

Let n be the number of query and view intervals. There is an $O(n^2)$ exact dynamic programming algorithm that delivers an optimal solution to the above problem. The algorithm appears in the full version of the paper.

The restricted version of the newsgroup-selection problem considered above is a special case of the view-selection problem in OR view graphs defined in [Gup97] with different optimization criteria and constraints. Gupta [Gup97] presents a simple greedy approach to deliver a solution that is within a constant factor of an optimal solution. In effect, we have taken advantage of the restricted model of the newsgroup definitions and shown that for this special case of the view-selection problem in OR graphs there exists a polynomial-time algorithm that delivers an optimal solution.

Multi-dimension Selection Queries The generalization of the above newsgroup-selection problem to *d-dimensional selection queries*, where each query and view is a d-dimensional hyper-rectangle, doesn't have any better than the $O(\log n)$ approximation algorithm. The newsgroup-selection problem for d-dimensional selection queries can be shown to be `NP`-complete through a reduction from `3-SAT`. In fact, the problem is a more general version of the classical age-old problem of covering points using rectangles in a 2-D plane [FPT81], for which nothing better than an $O(\log n)$ approximation algorithm is known.

Selection over `Body` Attribute Conditions Consider newsgroups defined by selection conditions of the form $f_{id} = \bigvee_k (C(A_d, B_{ik}, T_{ik}))$ over the `Body` attribute of the articles. So, an article m belongs to a newsgroup if m is *similar-to* one of the representative typical-article's body B_{ik} with a minimum threshold. A newsgroup query Q can be answered using another materialized newsgroup view V if the set of typical-article bodies of Q is a subset of the set of typical-article bodies of V. The newsgroup-selection problem in this setting can be shown to be exactly the same problem as the NP-complete `set cover` problem. Thus, allowing selections over the `Body` attribute makes the newsgroup-selection problem as difficult as the general newsgroup-selection problem with simple edges in the query-view graph.

4.3 Queries as Selections + Unions over Views

In this section, we look at some special cases of the general newsgroup-selection problem, while allowing both selection and union operators for computation of a newsgroup query from other materialized newsgroups. The use of both operators introduces hyperedges in the query-view graph. As mentioned before, the general newsgroup-selection problem involving hyperedges is intractable, hence we take advantage of the restricted model of our newsgroup definitions in designing approximation algorithms.

The newsgroup-selection problem with union and selection operators is a special case of the view-selection problem in AND-OR view graphs considered in [Gup97], with different optimization criteria and constraints. Gupta [Gup97] fails to give any approximation algorithms for the general view-selection problem in AND-OR graphs. We take advantage of the special nature of our problem, and present some polynomial-time approximation algorithms.

One-dimensional Selection/Union Queries Consider an ordered domain D, and let newsgroups be interval ranges over D. In other words, newsgroup views and queries can be represented as intervals over D and a newsgroup query interval can be answered using views $v_1, \ldots, v_l$ if the union of the view intervals covers the query interval completely. There is a query-cost associated with each such pair, which is the cost incurred in computing q from $v_1, v_2, \ldots, v_l$.

Problem (One-dimensional Selection/Union Queries) *Given a set of interval views V and interval queries Q in an ordered domain D, select a minimum weighted set of interval views M such that each query $q \in Q$ has a set of views $v_1, v_2, \ldots, v_l \in M$ that completely cover q and the query-cost associated is less than T_q.*

Consider the following cost model. In addition to a weight associated with each view, let there also be a cost $C(v)$ associated with each view. Let the cost of computing a query q using a set of views $\{v_1, v_2, \ldots, v_l\}$ that cover the query q be defined as $\sum_{i=1}^{l} C(v_i)$, i.e., the sum of the costs of the views used. The above cost model is general enough for all practical purposes.

The problem of one-dimensional selection/union queries with the above cost model can be shown to be `NP`-complete through a reduction from the `NP`-complete `Partition` [GJ79] problem. See the full version of the paper for the reduction. However, if we restrict our attention to the *index cost model* where the cost incurred in computing a query covered by l views is l units (a special case of the general cost model above, where each $C(v) = 1$), we show that there exists an $O(m^{k-1}n^2)$ dynamic programming solution, where m is the maximum overlap between the given queries and k is the maximum individual query-cost threshold (which we expect to be small). The details of the algorithm can be found in the full version of the paper.

The index cost model, where the cost of answering a query q using l views is l units, is based on the following very reasonable implementation. If all the materialized views are indexed along the dimension D, then the cost incurred in computing the query is proportional to l, the total number of index look-ups. Note that the query-costs associated with the edges may not be the actual query costs but could be the normalized query-cost “overheads”.

Average Query Cost Constraint A relatively easier problem in the context of the above cost model is when the constraint is on the *total* (or, *average*) query cost instead of having a threshold on each individual query. For such a case, there exists an $O(kn^3)$ time dynamic programming algorithm that delivers a minimum-weighted solution, where $k(\leq n)$ is the average query-cost constraint. The dynamic approach here works by maintaining for each interval $[1, i]$ a list of k solutions, where the j^{th} solution corresponds to the minimum-weighted set of views that covers the queries in $[1, i]$ under the constraint that the total query cost incurred is less than j.

Multi-dimensional Selection/Union Queries Consider next the newsgroup-selection problem where newsgroup queries and views are d-dimensional ranges. In other words, views and queries can be represented as hyper-rectangles in a d-dimensional space and a query hyper-rectangle can be answered using views $v_1, \ldots, v_k$ if the union of the view hyper-rectangles covers the query hyper-rectangle completely. We wish to select a minimum-weighted set of views such that all queries are covered. The simplest version of the problem has no threshold constraints and it is only required to cover all the query rectangles using the materialized views.

The above problem is `NP`-complete even for the case of two dimensions. We present here a polynomial-time (in n) $O(d \log n)$ approximation algorithm. The space of hyper-rectangular queries can be broken down into $O((2n)^d)$ elementary hyper-rectangles. Thus, the problem of covering the query hyper-rectangles can be reduced to covering the elementary hyper-rectangles with minimum-weighted set of views, which is equivalent to a `weighted set cover` instance having $O((2n)^d)$ elements; this has an $O(d \log n)$ approximation algorithm.

Selection/Union on `Body` Attribute Conditions In this subsection, we consider the case where newsgroup queries, having selection conditions of the form $f_{id} = \bigvee_k (C(A_d, B_{ik}, T_{ik}))$, can be computed using only selection and union over the `Body` attribute predicates of the materialized newsgroup views. Due to the same similarity-vector occurring in the definition of many newsgroups, a newsgroup V can be computed from the materialized newsgroups $V_1, V_2, \ldots, V_k$ if each similarity-vector of V is included in one of the materialized newsgroups.[4] The computation involves computing a selection over each of the relevant newsgroups followed by a union. If some of the similarity-vectors of a non-materialized view V are not covered by the other materialized views, then to answer the newsgroup query V, the article store needs to be accessed to select all articles whose bodies match (using cosine similarity function with the specified threshold) one of the uncovered similarity-vectors.

The cost of accessing a materialized view V, based on a selection over ordered domains, is proportional to the logarithm of the size of V, as it involves searching

[4] Here, for simplicity, we assume that the threshold corresponding to a particular similarity-vector is the same across different newsgroup views it is used in.

through efficient data structures like B-trees. In contrast, accessing V based on a selection over an unstructured attribute involves searching through the inverted index data structure. Therefore, when a query is computed as selection and union over the `Body` attribute using some materialized views and the article store, the cost incurred in accessing the article store is the dominant factor in the total query time. Since each similarity-vector is compared with the inverted index independently, the query cost is proportional to the number of similarity-vectors sent to the article store.

Thus, in the context of newsgroup queries computed as selection/union over the `Body` attribute, the natural optimization problem is to select a minimum weighted set of newsgroup views to materialize, such that the total number of uncovered similarity-vectors summed over all newsgroup queries is less than a given threshold. From the discussion in the previous paragraph, the threshold on the total number of similarity-vectors summed over all queries translates to a threshold on the average query-cost of a newsgroup query.

The above optimization problem has a greedy $O(\log n)$ approximation algorithm, that at each stage selects the newsgroup that decreases the total number of uncovered body articles (summed over all queries) by most. We omit the details of the proof here.

4.4 Related Work

The newsgroup-selection problem is similar to the view-selection problem defined in [Gup97]. The view-selection problem considered there was to select a set of views for materialization to minimize the query response time under the disk-space constraint. The key differences between the two problems are the different constraint and the minimization goal used.

Previous work on the view selection problem is as follows. Harinarayan et al. [HRU96] provide algorithms to select views to materialize for the case of data cubes, which is a special case of OR-graphs, where a query uses exactly one view to compute itself. The authors in [HRU96] show that the proposed polynomial-time greedy algorithm delivers a solution that is within a constant factor of the optimal solution. Gupta et al. [GHRU97] extend their results to selection of views *and* indexes in data cubes. Gupta [Gup97] presents a theoretical formulation of the general view-selection problem in a data warehouse and generalizes the previous results to general OR view graphs, AND view graphs, OR view graphs with indexes, and AND view graphs with indexes.

The case of one-dimensional selection queries considered here is a special case of the view-selection problem in OR view graphs for which we provided a polynomial time algorithm that delivers an optimal solution. Similarly, the case of one-dimensional selection/union queries is a special case of the view-selection problem in AND-OR view graphs ([Gup97]), which we observe can be solved optimally in polynomial-time for a reasonable cost model. The case of selection/union queries on newsgroups defined using similarity-vectors is also a special case of the general view-selection problem in AND-OR graphs, for which we have designed a provably good approximation algorithm.

In the computational geometry research community, to the best of our knowledge, the specific problems mentioned here haven't been addressed except for the preliminary work done on rectangular covers [FPT81].

5 Conclusions

We have proposed a novel paradigm for newsgroups, the DaWN model, where newsgroups are defined as selection views over the article store. The success of the DaWN model clearly depends on the efficiency with which news readers can continue to access newsgroups. In this paper, we have looked at the two complementary problems that are critical for this efficiency. The first problem is the efficient incremental maintenance of eagerly maintained newsgroups. We have designed an I/O and CPU efficient algorithm for this problem, based on external segment trees, tries, and inverted lists. The second problem is the choice of eagerly maintained (materialized) newsgroups. We have demonstrated the intractability of the general problem, and discussed various special natural cases of the general problem in the context of the DaWN model.

The success of the DaWN model also depends on the precision with which an article can be automatically classified into appropriate newsgroups. This precision will be determined by the newsgroup definitions, in particular by the choice of representative typical-articles. The problem of a good choice of typical-articles for a given newsgroup is orthogonal to the problems/issues addressed in this paper, and is an interesting open problem, where techniques from data mining and data clustering can play a significant role.

We believe that the DaWN model can also serve as the foundation for allowing individual users to enhance the standard newsgroups by defining their personal newsgroups. Such personal newsgroups can be specified using "profiles" of the users that are matched against the article store, and techniques and solutions from dissemination-based systems can be used to advantage here.

References

[ABSS93] S. Arora, L. Babai, J. Stern, and Z. Sweedyk. The hardness of approximate optima in lattices, codes, and systems of linear equations. In *Proceedings of the Foundations of Computer Science*, 1993.

[AL95] S. Arora and C. Lund. Hardness of approximations. Technical Report TR-504-95, Princeton University, Computer Science Department, 1995.

[Cha83] B. Chazelle. Filtering search: A new approach to query-answering. In *Proceeding of the Foundations of Computer Science*, 1983.

[Chv79] V. Chvatal. A greedy heuristic for the set covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[Ede83a] H. Edelsbrunner. A new approach to rectangle intersections, Part I. *International Journal of Computer Mathematics*, 13:209–219, 1983.

[Ede83b] H. Edelsbrunner. A new approach to rectangle intersections, Part II. *International Journal of Computer Mathematics*, 13:221–229, 1983.

[EM81] H. Edelsbrunner and H.A. Maurer. On the intersection of orthogonal objects. *Information Processing Letters*, 13(4):177–180, 1981.

[Fal85] C. Faloutsos. Access methods for text. *ACM Comp. Surveys*, 17(1), 1985.

[FPT81] R. J. Fowler, M. S. Paterson, and S. L. Tanimoto. Optimal packing and covering in the plane are NP-complete. *Info. Proc. Letters*, 12(3), 1981.

[Fre60] E. Fredkin. Trie memory. *Communications of the ACM*, 3(9), 1960.

[GJ79] M. R. Garey, D. J. Johnson. *Computers and Intractability: a Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.

[GHRU97] H. Gupta, V. Harinarayan, A. Rajaraman, and J. Ullman. Index selection in OLAP. In *Proceedings of the ICDE*, 1997.

[GM97] M. Goldwasser and R. Motwani. Intractability of assembly sequencing: Unit disks in the plane. In *Proceeding of the Workshop on Algorithms and Data Structures*, August 1997.

[Gup97] H. Gupta. Selection of views to materialize in a data warehouse. In *Proceedings of the ICDT*, Delphi, Greece., January 1997.

[HCKW90] E. Hanson, M. Chaabouni, C.-H. Kim, and Y.-W. Wang. A predicate matching algorithm for database rule systems. In *PODS*, 1990.

[HGMW+95] J. Hammer, H. Garcia-Molina, J. Widom, W. Labio, and Y. Zhuge. The Stanford Data Warehousing Project. *IEEE Data Engineering Bulletin, Special Issue on Materialized Views and Data Warehousing*, 18(2), 1995.

[HRU96] V. Harinarayan, A. Rajaraman, and J. Ullman. Implementing data cubes efficiently. In *SIGMOD*, 1996.

[IK93] W.H. Inmon and C. Kelley. *Rdb/VMS: Developing the Data Warehouse.* QED Publishing Group, Boston, Massachusetts, 1993.

[KRVV93] P.C. Kanellakis, S. Ramaswamy, D.E. Vengroff, and J.S. Vitter. Indexing for data models with constraints and classes. In *PODS*, 1993.

[Per94] M. Persin. Document filtering for fast ranking. *Proc. ACM SIGIR Conf.*, Dublin, Ireland, 1994.

[RS94] S. Ramaswamy and S. Subramanian. Path caching: A technique in optimal external searching. In *PODS*, 1994.

[RS95] S. Ramaswamy and S. Subramanian. The p-range tree: A new data structure for range searching in secondary memory. In *SODA*, 1995.

[SB88] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, 24(5), 1988.

[Sam89a] Hanan Samet. *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS.* Addison-Wesley, 1989.

[Sam89b] Hanan Samet. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, 1989.

[VV96] D.E. Vengroff and J.S. Vitter. Efficient 3-D searching in external memory. In *Proceeding of the STOC*, 1996.

[Wid95] J. Widom. Research problems in data warehousing. In *Proceedings of the Conference on Info. and Knowledge Management*, 1995.