

Factorization-based Lossless Compression of Inverted Indices

George Beskales
University of Waterloo
Waterloo, ON, Canada
gbeskale@cs.uwaterloo.ca

Marcus Fontoura
Google Inc.
Mountain View, CA, USA
marcusf@google.com

Maxim Gurevich
Yahoo! Labs
Santa Clara, CA, USA
maximg@yahoo-inc.com

Sergei Vassilvitskii
Yahoo! Labs
New York, NY, USA
sergei@yahoo-inc.com

Vanja Josifovski
Yahoo! Labs
Santa Clara, CA, USA
vanjaj@yahoo-inc.com

ABSTRACT

Many large-scale Web applications that require ranked top- k retrieval are implemented using inverted indices. An inverted index represents a sparse term-document matrix, where non-zero elements indicate the strength of term-document associations. In this work, we present an approach for lossless compression of inverted indices. Our approach maps terms in a document corpus to a new term space in order to reduce the number of non-zero elements in the term-document matrix, resulting in a more compact inverted index. We formulate the problem of selecting a new term space as a matrix factorization problem, and prove that finding the optimal solution is an NP-hard problem. We develop a greedy algorithm for finding an approximate solution.

A side effect of our approach is increasing the number of terms in the index, which may negatively affect query evaluation performance. To eliminate such effect, we develop a methodology for modifying query evaluation algorithms by exploiting specific properties of our compression approach.

Categories and Subject Descriptors

H.3.1 [Information Storage And Retrieval]: Indexing methods

General Terms

Algorithms, Experimentation, Performance

1. INTRODUCTION

Web search engines and other large-scale information retrieval systems typically have to process query workloads of thousands of requests per second over large collections of documents. Usually, the result of the retrieval is a ranked list of the top few (k) results.

Top- k retrieval is defined as follows. Given a query Q and a document corpus $Docs$, find the k documents $\{D_1, D_2, \dots, D_k\} \subset Docs$ that have the highest score, according to some scoring function $Score(D, Q)$. Both the query and the documents are sets of

terms from the same high-dimension space. Scoring is usually performed based on the overlapping terms between the query and the document. The document corpus $Docs$ can be represented as a two dimensional matrix, denoted as V , with m terms (rows) and n documents (columns). The values of elements in V reflect how strongly terms are associated with documents (e.g., tf-idf).

Inverted indices are the prevailing implementation of scalable top- k retrieval. In an inverted index, each term T appearing in the corpus $Docs$ is associated with a *posting list*, which enumerates the documents that contain T . Each posting list consists of a sequence of document identifiers, each of which associated with an optional payload (e.g., term frequency). An inverted index can also be viewed as a sparse representation of the matrix V that stores only non-zero matrix elements.

In several applications, top- k queries are processed while the user is waiting for the results, which imposes very strict bounds on query latency. Due to such requirements, memory-resident indices are becoming more popular. To reduce the amount of required memory, and hence the system cost, compression techniques (e.g., [6, 18, 21]) are heavily used to reduce the size of the inverted indices. Compression techniques are divided into two categories: lossy compression, where quality of the results might be affected by compression, and lossless compression, where the quality is not affected. In this work, we focus on lossless compression.

In this paper, we propose a novel lossless compression technique that holistically compresses multiple posting lists by taking advantage of similarities between them. This in contrast to the many existing lossless compression approaches that compress each posting list individually [16, 21]. The compression approach we propose can be applied before the standard per-posting-list compression in order to combine the benefits of both methods. In spirit, the technique presented in this paper is related to matrix factorization methods such as Non-negative Matrix Factorization [10, 13], Latent Dirichlet Allocation [4], Singular Value Decomposition [19], and Principal Component Analysis [11]. All of these techniques map the documents from the space of the original terms into a lower dimensional space. However, unlike previous factorization methods, we aim at providing an *exact* factorization of the input matrix in order to avoid any information loss, while reducing the number of non-zero elements in the resulting factors. Furthermore, we do not restrict the dimensionality of the factor matrices.

For example, the top term-document matrix in Figure 1 is factored into two matrices W and H such that: (1) the product of the factors is equal to the input matrix, and (2) the factor matrices contain fewer non-zeros than the input matrix. Note that the rank of

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'11, October 24–28, 2011, Glasgow, Scotland, UK.
Copyright 2011 ACM 978-1-4503-0717-8/11/10 ...\$10.00.

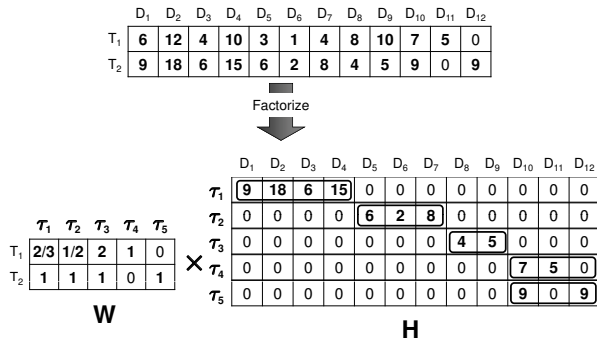


Figure 1: Factorization of term-document matrix

the second factor H (five), is higher than the rank of the input matrix (two). To answer queries, we use the first factor to map (i.e., rewrite) query terms from the original term space to a new space of meta-terms (e.g., T_1 is mapped to $\tau_1, \tau_2, \tau_3, \tau_4$ in Figure 1). The rewritten query is used for searching the index built from the second factor using any top- k search algorithms. In this paper, we prove that finding the optimal (i.e., the sparsest) factorization is NP-hard, and we develop a greedy algorithm that efficiently finds an approximate solution.

Although lossless compression helps reducing the amount of memory required for storing a given index, it usually incurs a computational overhead due to the need of decompressing the index data [18, 21]. Such overhead should be minimized in order to keep query latency small. The computational overhead in our approach is due to rewriting a query using a number of meta-terms that is greater than the number of the original terms. For example, the number of meta-terms in Figure 1 is five, while the original number of terms is two. We show how to eliminate such negative effect by exploiting some unique characteristics of our compression technique. More specifically, we show that standard query processing approaches such as No-Random-Access (NRA) algorithm [8] can be modified to search the compressed index as fast as the original algorithm searches the uncompressed index.

2. PRELIMINARIES

In this paper, we use the vector-space representation of documents and queries. That is, documents and queries are represented as vectors in a multidimensional space, where each term is a dimension. Let $\Omega = \{T_1, T_2, \dots, T_m\}$ be a set of terms. A document D_j is a vector $(d_j^1, d_j^2, \dots, d_j^m)$. When a term T_i occurs in a document D_j , the element d_j^i is non-zero, and its value is typically related to the number of times T_i occurs in D_j . Similarly, a query Q is represented by a vector (w_1, w_2, \dots, w_m) , where non-zero elements correspond to terms appearing in the query, and their values are term weights in the query.

Given a corpus of n documents $Docs = \{D_1, \dots, D_n\}$, and a query Q , a common task in many information retrieval systems is to retrieve the k documents with the highest score according to some scoring function $Score(D, Q)$. In this work, we assume the scoring function is defined as the inner product of document and query vectors: $Score(D_j, Q) = \sum_{i=1}^m d_j^i \cdot w_i$.

Many information retrieval systems use inverted indices as their main data structure for top- k retrieval. An inverted index is a collection of *posting lists* L_1, L_2, \dots, L_m : a list for each term in Ω . List L_i is a vector containing weights of term T_i in all documents (i.e., $L_i = (d_1^i, d_2^i, \dots, d_n^i)$). Usually, sparse representation of posting lists is used, where zero entries are omitted.

A naive top- k algorithm would examine all entries in the posting lists relevant to the query, compute the scores of found documents,

and return the top- k documents. However, the total number of documents in the relevant posting lists is typically much larger than k , especially when Q contains frequent terms. Many top- k search algorithms (e.g., [5, 8]) aim at retrieving the top- k documents while examining only a fraction of entries in the relevant posting lists.

3. INDEX COMPRESSION USING MATRIX FACTORIZATION

We represent an index by an $m \times n$ term-document matrix V (Figure 1). We denote by $V[T, D]$ the value of the element in V corresponding to a row T and a column D . We use $\|V\|_0$ to denote the number of non-zero elements in V . Top- k retrieval is defined as computing scores of all documents, represented by a size- n vector S , and picking the top- k documents with highest scores. The transpose of vector S is equal to $Q^T V$, where the superscript T denotes the transpose operator.

Typical document collections, such as Web corpora, contain redundant elements in their term-document matrices due to duplicate or near-duplicate contents. For example, news articles are usually shared across multiple Web sites. In this case, two documents D_x and D_y that refer to the same article would contain several identical sentences consisting of terms T_a, T_b, \dots, T_p . Consequently, the term-document matrix would contain two identical sets of values: $V[T_a, D_x], \dots, V[T_p, D_x]$, and $V[T_a, D_y], \dots, V[T_p, D_y]$. Another example that leads to redundancy in term-document matrix is co-occurrence of subsets of terms in multiple documents. For example, terms “Britney” and “Spears” usually co-occur in documents related to music.

A known technique for reducing redundancy in a matrix is matrix factorization. The simplest form of factorization is decomposing V into two matrices: an $m \times r$ matrix W and an $r \times n$ matrix H , such that $V = WH$. Note that since our goal is lossless index compression, we consider the exact formulation and not the approximate one ($V \approx WH$). Our objective is to minimize the total number of non-zero elements in W and H (i.e., $\|W\|_0 + \|H\|_0$).

Intuitively, factoring V into WH transforms the set of terms Ω into another space, denoted Θ , consisting of r meta-terms $\{\tau_1, \tau_2, \dots, \tau_r\}$. Matrix W linearly maps terms in Ω to meta-terms in Θ (and vice-versa), while matrix H represents an inverted index of *Docs* in space Θ . Figure 1 is an illustration of such a factorization, where terms $\{T_1, T_2\}$ are linearly mapped into meta-terms $\{\tau_1, \dots, \tau_5\}$ using matrix W , and documents are represented as combinations of these meta-terms in matrix H . Note that although $r > m$ (i.e., the number of rows in H is greater than the number of rows in V), the total number of non-zeros in W and H is less than the number of non-zeros in V .

Evaluation of query Q is performed on the inverted index represented by H , after rewriting Q according to W . Specifically, we rewrite the query vector Q into vector Q' such that $Q'^T = Q^T W$. In other words, each term T in Q is replaced by a set of meta-terms $\{\tau : W[T, \tau] \neq 0\}$, and the weight of each term τ in Q' is $w \cdot W[T, \tau]$, where w is the weight of T in Q . Any standard search algorithm can be used to retrieve the top- k documents from the compressed index H using query Q' . The following theorem proves that searching the original inverted index using Q is equivalent to searching the index represented by H using Q' .

THEOREM 1. *Let W and H be the result of factoring V (i.e., $VH = V$). Let $\mathcal{A}(V, Q, k)$ be the top- k documents for query Q using inverted index V and the scoring function in Section 2 (ties are broken by some predefined criteria). Let the rewritten query be Q' such that $Q'^T = Q^T W$. Then, $\mathcal{A}(V, Q, k) = \mathcal{A}(H, Q', k)$.*

The proof is described in [3]. An immediate consequence is that standard top- k algorithms can still be used for searching the compressed indices without any loss in quality.

REMARK It is possible to interpret the intermediate space Θ as a space of *meta-documents*, rather than *meta-terms*. In this case, the matrix W represents an inverted index of meta-documents in the original term space Ω , and H is a mapping from meta-documents to documents in *Docs*. However, existing top- k retrieval algorithms that employ early termination cannot be used on W , since top- k meta-documents do not necessarily contain the top- k documents.

4. SPARSE MATRIX FACTORIZATION

In this section, we consider the following exact sparse matrix factorization problem. Given a matrix V , obtain W and H that

$$\begin{aligned} & \text{minimize} && \|W\|_0 + \|H\|_0 \\ & \text{subject to} && WH = V. \end{aligned}$$

The following theorem states that the problem of obtaining the sparsest exact factorization is NP-hard.

THEOREM 2. *Given a matrix V , the problem of obtaining two matrices W and H , subject to the constraint $WH = V$, such that $\|W\|_0 + \|H\|_0$ is minimum is NP-hard.*

Proof of Theorem 2 is provided in [3]. Hardness of the problem is proved by a reduction from an NP-complete problem, namely the SPARSESTVECTOR problem [9].

Since obtaining the optimal factorization is computationally infeasible, we propose an iterative greedy algorithm for getting an exact factorization that might not have the minimal number of non-zeros. The key idea is to start with a trivial factorization $W_0 = I_m$ (I_m is the identity matrix of rank m) and $H_0 = V$, and iteratively improve the current solution (W_t, H_t) by a sequence of local transformations on W_t and H_t , obtaining W_{t+1} and H_{t+1} . At each step, we ensure that $\|W_{t+1}\|_0 + \|H_{t+1}\|_0 < \|W_t\|_0 + \|H_t\|_0$, and $W_{t+1}H_{t+1} = V$.

The transformation performed at each step is described as follows. At step t , given matrices W_t and H_t , the algorithm looks for a submatrix H_t^s of H_t defined by a subset R of H_t 's rows, and a subset C of H_t 's columns, such that the rank of H_t^s is one (i.e., rows are multiples of each others):

$$\forall(\tau_i, \tau_j) \in R \times R, \forall(D_p, D_q) \in C \times C \quad \left(\frac{V[\tau_i, D_p]}{V[\tau_j, D_p]} = \frac{V[\tau_i, D_q]}{V[\tau_j, D_q]} \right). \quad (1)$$

Clearly, keeping only one representative row from H_t^s , and encoding other rows in H_t^s as multiples of the representative row, would reduce the number of non-zero values in H_t . Unfortunately, identifying the largest rank-1 submatrix H_t^s is equivalent to the problem of finding the largest bi-cluster [15], which is an NP-hard problem. Thus, our algorithm considers only submatrices consisting of exactly two rows (i.e., $|R| = 2$) at each step.

For efficiency, in each iteration, our algorithm identifies z rank-1 submatrices that are composed of two rows $R = \{\tau_i, \tau_j\}$ and z sets of columns C_1, \dots, C_z (i.e., Equation 1 holds for submatrices (R, C_1) through (R, C_z)). Rows τ_i and τ_j can then be rewritten as linear combinations of a set of z new common subvectors, denoted $\tau_{r+1}, \dots, \tau_{r+z}$, and two new remainder vectors τ_{r+z+1} and τ_{r+z+2} that retain values of columns that are not in $C_1 \cup \dots \cup C_z$.

$$\begin{aligned} \tau_i &= \alpha_1 \cdot \tau_{r+1} + \alpha_2 \cdot \tau_{r+2} + \dots + \alpha_z \cdot \tau_{r+z} + \tau_{r+z+1} \\ \tau_j &= \beta_1 \cdot \tau_{r+1} + \beta_2 \cdot \tau_{r+2} + \dots + \beta_z \cdot \tau_{r+z} + \tau_{r+z+2} \end{aligned} \quad (2)$$

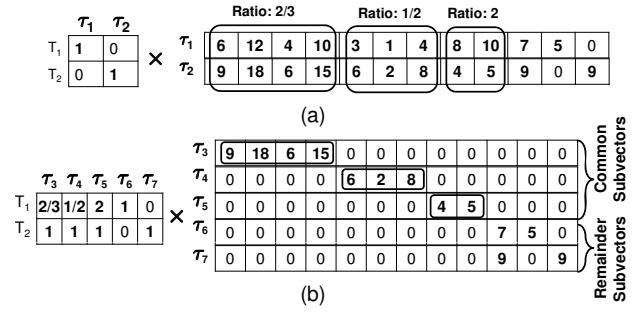


Figure 2: Combining term vectors (a) matrices W_t and H_t (b) matrices W_{t+1} and H_{t+1} after combining τ_1 and τ_2

The new vectors $\tau_{r+1}, \dots, \tau_{r+z+2}$ are appended to matrix H_t , and vectors τ_i and τ_j are removed from H_t , resulting in matrix H_{t+1} . Matrix W_t is modified by mapping the original terms τ_i, τ_j to $\tau_{r+1}, \dots, \tau_{r+z+2}$, obtaining W_{t+1} . Algorithm 1 describes the procedure in more details. Function *GetCorrelatedSubmatrices*(H_t), which we describe in Section 4.1, is responsible for extracting the sets R and C_1, \dots, C_z that maximize space saving.

Figure 2 shows an example of combining two meta-terms τ_1 and τ_2 into common subvectors τ_3, τ_4, τ_5 , and remainder vectors τ_6, τ_7 . Without loss of generality, we assume hereafter that $\beta_1 = \dots = \beta_z = 1$.

Algorithm 1 *ComputeFactorization*(V)

- 1: $W_0 \leftarrow I_m; H_0 \leftarrow V$
 - 2: $r \leftarrow m$
 - 3: $t \leftarrow 0$
 - 4: **repeat**
 - 5: $H_{t+1} \leftarrow H_t$
 - 6: $(R, C_1, \dots, C_z) \leftarrow \text{GetCorrelatedSubmatrices}(H_t)$
 - 7: **if** failed to find correlated submatrix **then**
 - 8: **break**
 - 9: remove rows $R = \{\tau_i, \tau_j\}$ from H_{t+1} , and add the new rows $\tau_{r+1}, \dots, \tau_{r+z+2}$ to H_{t+1}
 - 10: construct an $r \times (r+z+2)$ transformation matrix W_M that linearly maps τ_i and τ_j to $\tau_{r+1}, \dots, \tau_{r+z+2}$ using Equations 2 and 3, and trivially maps all other meta-terms in W_t to themselves.
 - 11: $W_{t+1} \leftarrow W_t W_M$.
 - 12: $r \leftarrow r + z + 2$
 - 13: $t \leftarrow t + 1$
 - 14: **until** $\|W_t\|_0 + \|H_t\|_0$ converges
 - 15: **return** W_t, H_t
-

4.1 Identifying Correlated Submatrices

The goal of function *GetCorrelatedSubmatrices*(H_t) is to return correlated submatrices, defined by $R = \{\tau_i, \tau_j\}$ and C_1, \dots, C_z . Our algorithm heuristically finds the submatrices that would result in the highest reduction of space. In the following, we first describe how to find the sets C_1, \dots, C_z given R , then formulate the potential saving from combining two meta-terms, and finally how to find R .

Denote by $X[p]$ the value of the element at index p in vector X . For two rows τ_i and τ_j in H_t , we compute vector γ of length n such that for $q \in \{1, \dots, n\}$, $\gamma[q] = \frac{\tau_i[q]}{\tau_j[q]}$ if $\tau_j[q] \neq 0$, and equals 0 otherwise. Each set C_p is a maximal subset of columns (documents) in H_t that have the same *non-zero* value in γ (which is eventually the value of the corresponding coefficient α_p in Equation 2). For example, in Figure 2, the first four cells have the same value in γ , namely 2/3, and thus constitute a common subvector (τ_3). It is clear that C_1, \dots, C_z are pairwise disjoint because each

set C_p corresponds to a unique non-zero ratio in γ . We rely on this property to improve query evaluation performance (Section 5).

The space saving resulting from combining τ_i and τ_j is computed as follows. Combining τ_i and τ_j in Algorithm 1 reduces the number of non-zero elements in H_t by $\sum_{p=1}^z |C_p|$ because each subvector corresponding to C_p is stored twice in H_t and only once in H_{t+1} . All the terms that were mapped to either τ_i or τ_j in W_t are now mapped to additional z meta-terms $\tau_{r+1}, \dots, \tau_{r+z}$ in W_{t+1} . For example, in Figure 2, T_1 is originally mapped to τ_1 . After combining τ_1 and τ_2 , T_1 is mapped to extra 3 meta-terms, namely τ_3, τ_4, τ_5 , (besides the remainder meta-term τ_6), which results in three additional elements in W . Formally, the overall space saving when combining τ_i and τ_j is:

$$\begin{aligned} \text{saving}(\tau_i, \tau_j, C_1, \dots, C_z, W_t) &= \sum_{p=1}^z |C_p| \\ &- z \cdot |\{T \in \Omega : W_t[T, \tau_i] \neq 0 \vee W_t[T, \tau_j] \neq 0\}|. \end{aligned} \quad (4)$$

In the following, we describe how to efficiently identify a pair of rows in H_t , with the highest potential savings. A straightforward approach is to compute the potential space saving, based on Equation 4, for all pairs of rows and return the pair with the highest savings. However, the complexity of such approach is quadratic in the number of rows in H_t , which is prohibitively expensive. To reduce the number of pair-wise comparisons, we use a *blocking* technique to heuristically prune a large number of pairs that have low potential space saving. Specifically, we partition the rows in H_t into several disjoint blocks, based on the number of non-zeros in each row, and we only compute the potential saving for pairs of rows within the same block. The main hypothesis here is that rows that have significantly different number of non-zero elements are uncorrelated, and they are unlikely to have overlapping elements.

4.2 Other Issues

Limiting the Number of Subvectors. Our compression approach iteratively reduces the index size at the cost of increasing the number of meta-terms. That is, there is a trade-off between the space savings and the increase in the number of meta-terms. In particular, it may not be beneficial to introduce new meta-terms whose space savings are below some threshold. The length of a new meta-term τ_{r+p} represents its maximum potential saving, according to Equation 4. Therefore, we modify algorithm *GetCorrelatedSubmatrices*(H_t) such that it generates a new meta-term only if its length is greater than or equal to a threshold μ . Consider the example depicted in Figure 2. Assuming that $\mu = 3$, only the first two meta-terms τ_3, τ_4 would be generated, while the third meta-term τ_5 would not be generated (i.e., it becomes part of the remainder meta-terms τ_6, τ_7).

MapReduce Implementation. To scale the algorithm to large matrices, we parallelize it according to the MapReduce model [12]. We notice that combining two rows of matrix H_t is done independently of other rows on H_t . Therefore, it is possible to combine multiple pairs of rows of H_t in parallel, as long as pairs are mutually disjoint. Such observations allow the following parallelization scheme: (1) map each row in H_t to a specific block, based on the number of non-zeros in each row, (2) compute the potential space saving for each pair of rows in the same block in parallel, (3) for each block, identify a set of disjoint pairs of rows that maximize the overall saving, (4) for each block, combine all pairs and output the (disjoint) parts of H_{t+1} and W_M , and (5) collect all the parts and construct H_{t+1} , and $W_{t+1} = W_t W_M$.

Updating the Compressed Index. Due to the dynamic nature of documents copra that are found in the Web, we must be able to update inverted indices with minimal computational cost, without

the need of reconstructing the entire index from scratch. In the following, we show how to append a new document to the index, and how to remove an existing document (updating a document can be decomposed into removing the old version of the document, and inserting the new version). We notice that at least one meta-term is a remainder meta-term, denoted $Rem(T_i)$, that is uniquely mapped to the original term T_i (i.e, $W[T_i, Rem(T_i)] = 1$, and $\forall j \neq i (W[T_j, Rem(T_i)] = 0)$). In order to add a new document D that mention term T_i , it is sufficient to add a new posting to the posting list of $Rem(T_i)$. To remove a document D from the index, we remove all elements in the column in H that corresponds to D . Note that the index should be reconstructed periodically (e.g., at idle times) to compress the newly inserted documents.

5. OPTIMIZING QUERY PROCESSING

A side-effect of our approach is having a number of meta-terms in the rewritten query that is larger than the number of terms in the original query, which can lead to longer query response times. In this section, we show how to mitigate this undesirable effect by exploiting unique characteristics of our compression scheme. As a case study, we show how to modify the Non-Random-Access algorithm (NRA) [8]. Note that there exist a plethora of search algorithm that might be more efficient than NRA, especially for memory-resident indices. However, we chose the NRA algorithm because of its simplicity.

We denote by L_1, \dots, L_h the posting lists corresponding to the terms with non-zero weight in query Q , where $h = \|Q\|_0$, and let w_1, \dots, w_h be the weights associated with L_1, \dots, L_h in Q . The score of a document D can be rewritten as $Score(D, Q) = \sum_{i=1}^h w_i \cdot L_i(D)$, where $L_i(D)$ denotes the weight of document D in list L_i . The NRA algorithm requires posting lists to be sorted in descending order of term frequencies.

The NRA algorithm retrieves documents from lists L_1, \dots, L_h in a round robin order. Having the lists sorted enables computing upper and lower bounds on document scores in each list. Every time a document is retrieved, the lower and upper bounds of retrieved documents, as well as unseen documents, are updated. Once k documents are found whose lower bounds are greater than or equal to the upper bounds of all other documents (including the unseen ones), the algorithm terminates.

Score bounds are computed as follows. Let \bar{x}_i denote the weight of the last document retrieved from list L_i , if L_i is not completely read by the algorithm, or 0 otherwise. During execution of the algorithm, the score upper bound of each retrieved document D , denoted $\overline{Score}(D, Q)$, is computed as follows:

$$\overline{Score}(D, Q) = \sum_{i=1}^h w_i \cdot \bar{L}_i(D). \quad (5)$$

where $\bar{L}_i(D)$ is the weight of D in list L_i if D has appeared in L_i , and \bar{x}_i otherwise. The upper bound for unseen documents is $\sum_{i=1}^h w_i \cdot \bar{x}_i$. Similarly, the lower bound of each retrieved document D , denoted $\underline{Score}(D, Q)$, is

$$\underline{Score}(D, Q) = \sum_{i=1}^h w_i \cdot \underline{L}_i(D). \quad (6)$$

where $\underline{L}_i(D)$ is the weight of D in list L_i , if D has appeared in L_i , and 0 otherwise.

In the following, we describe our modifications to the NRA algorithm. The score upper bound of each retrieved document D is computed by assuming that each undiscovered weight $L_i(D)$ is equal to \bar{x}_i (Equation 5). Recall that all meta-term lists corresponding to the same original term are disjoint (Section 4.1). Thus, it is

possible to compute a tighter score upper bound by setting undiscovered weight $L_i(D)$ to zero, instead of \bar{x}_i , if D has appeared in some list L_j such that L_i and L_j are disjoint.

Therefore, instead of considering lists of meta-terms independently, we create a two-level document retrieval scheme as follows. We create a *virtual list* for each original query term T_i . Each virtual list is traversed by probing the *disjoint* lists of the corresponding meta-terms. We use a priority queue PQ_i to implement retrieval from the virtual list of T_i . Let $M(T_i) \triangleq \{\tau : W[T_i, \tau] \neq 0\}$ be the set of meta-terms that term T_i is rewritten into. Initialization of a priority queue PQ_i is performed by inserting a pair (τ, D) for each meta-term τ in $M(T_i)$, where D is the document with the highest score in τ . The score of each pair (τ, D) in the queue is equal to the score of D in list τ multiplied by the weight $W[T_i, \tau]$. Retrieving next document from the virtual list of T_i is equivalent to retrieving the document D in the pair (τ, D) at the head of the priority queue. After each retrieval from PQ_i , we insert a new pair (τ, D') in PQ_i , where D' is the next document to be retrieved from the list of τ .

We modify the NRA algorithm to use virtual lists as follows. The algorithm initializes priority queues PQ_1, \dots, PQ_h for the original query terms T_1, \dots, T_h . Retrievals from each list L_i are replaced by retrievals from the corresponding virtual list. The following theorem states the correctness of the modifications, and gives an upper bound on the runtime overhead.

THEOREM 3. *Query results obtained by the NRA algorithm when processing query Q using the uncompressed index V are equal to the results obtained by the modified NRA algorithm when processing the same query Q using a compressed index H and a term rewriting matrix W such that $V = WH$. Furthermore, the modified NRA algorithm performs at most $\sum_{T_i \in Q} |M(T_i)|$ more probes than the unmodified NRA algorithm.*

The proof is provided in [3]. Some meta-terms in the rewritten query might be shared across multiple terms in the original query (i.e., $M(T_i) \cap M(T_j) \neq \emptyset$ for $T_i, T_j \in Q$ and $i \neq j$). We further reduce the number of probes by keeping each meta-term τ in exactly one priority queue PQ_i of a term T_i such that $\tau \in M(T_i)$, and removing occurrences of τ in other priority queues.

6. EXPERIMENTAL EVALUATION

In this section, we evaluate the effectiveness of our compression technique and its impact on query execution time. We show that our technique is orthogonal to the compression techniques that compress posting lists individually (e.g., var-byte encoding [16]), and thus can be integrated with such techniques as we demonstrate in this section. We perform our evaluation on memory-resident indices since this is the dominant approach in modern search engines due to the growing memory capacities of modern machines, as well as index partitioning techniques.

6.1 Setup

Our index factorization algorithm ran on a Hadoop cluster. Query evaluation latency was measured by a single-threaded Java process running on an Intel Xeon 2.00GHz 8-core machine with 32GB RAM. Both compressed and uncompressed indices were preloaded into RAM prior to query evaluation. We used TREC WT10g document corpus [1], which contains 1.7M documents. We indexed only the textual content of the documents and discarded HTML tags. We removed rare terms that appear in less than three documents, thus reducing the number of unique terms from 5.4M to 1.6M. These rare terms account for less than 1% of the index size. Each posting contains 4-byte integer for *docID* and 4-byte integer

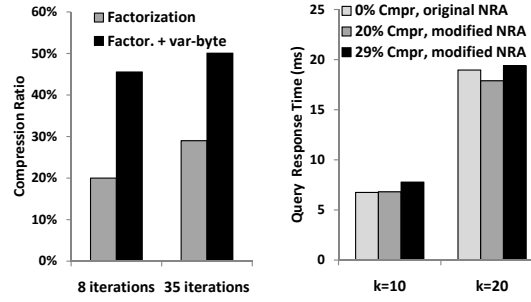


Figure 3: Compression ratio

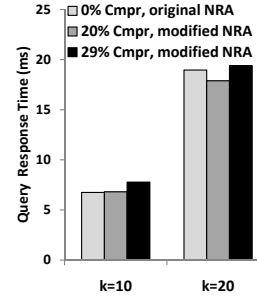


Figure 4: Average query response time

for term frequency. For query workload, we used 50,000 queries that are randomly selected from the AOL query log [17]. Unless specified otherwise, the minimum savings threshold μ is set to 100.

To measure the compression ratio, we use the relative reduction in the size of the inverted index: $(\text{Uncompressed Index Size} - \text{Compressed Index Size}) / (\text{Uncompressed Index Size})$. We consider both matrices W and H when computing the size of a compressed index. Note that for factorization-based compression only, the compression ratio is equal to $\frac{\|V\|_0 - (\|W\|_0 + \|H\|_0)}{\|V\|_0}$.

6.2 Results

Compression Performance. We selected two compressed indices obtained after 8 and 35 iterations of our algorithm. Figure 3 shows the compression ratio for the two indices. We observe that after 8 iterations, our algorithm compresses the index by 20%. When we additionally compress each posting list in the compressed index using the gap-based var-byte encoding [16], the overall compression reaches 46%. At iteration 35, the compression ratios further increase to 29% and 50%, respectively. The size of matrix W , which maps the original terms to the meta-terms, is less than 1% of the compressed index size in all iterations.

By lowering the saving threshold μ to 0, our approach archives a compression ratio of 35% after 30 iterations (Figure 7(b)).

When limiting the number of mappers/reducers to 100 per each job, each iteration took 22 minutes in average. The runtime of the first few iterations is slightly above average (e.g., the first iteration took 27 minutes, while the second iteration took 23 minutes).

Query Evaluation Latency. Figure 4 shows the average query latency for different numbers of retrieved documents (k) using the compressed indices at iterations 8 and 35. We do not show the latency of the unmodified NRA algorithm on the compressed indices as it is orders-of-magnitude higher and would distort the plot. In some cases, searching a compressed index outperforms searching the uncompressed index (e.g., for $k = 20$, the latency at compression ratio of 20% is 6% lower than the latency when using the uncompressed index).

Size of the Factor Matrices. Figure 5 depicts the relative number of non-zero elements in W and H compared to the number of non-zeros in V at various iterations of the compression algorithm. Observe the monotonicity of the curve due to the property of our algorithm that never increases the number of non-zero elements in the factors. Matrix W , which is used for query rewriting, is much smaller than H (e.g., $\|W\|_0$ is less than 1% of $\|V\|_0$).

Integration with Var-byte Encoding. In this experiment, we show the behavior of var-byte encoding [16] when applied to our compressed index. Figure 6 shows the effectiveness of the var-byte encoding at various compression ratios of our factorization algorithm. The relative stability of effectiveness of var-byte encoding

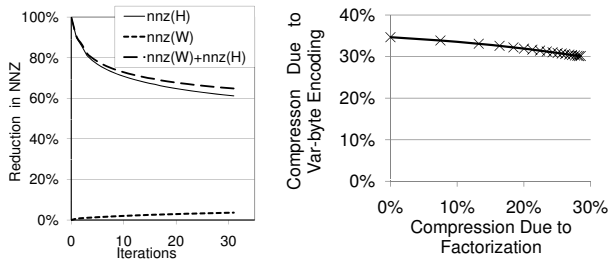


Figure 5: Relative reduction in non-zeros

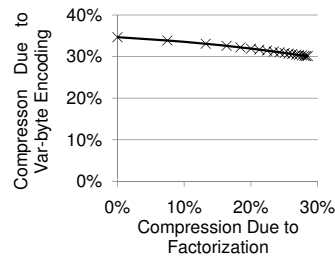


Figure 6: Effectiveness of var-byte encoding

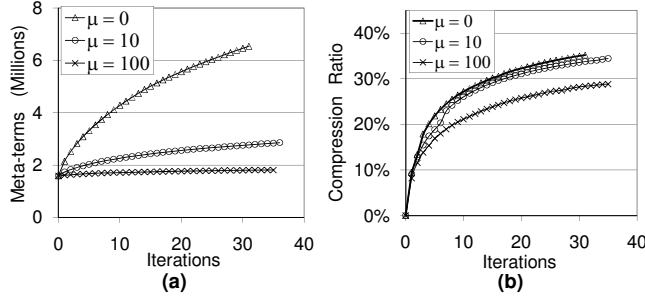


Figure 7: The effect of μ on (a) the number of meta-terms, and (b) the compression ratio

suggests that it is orthogonal to our compression technique, and thus both techniques can be used together to achieve higher compression ratios. For example, combining the two techniques enables compression ratio of 50% at iteration 35 (Figure 3), which is higher than using our compression technique alone (29%), or using var-byte encoding alone (34%).

The Saving Threshold μ . In this experiment, we analyze the effect of μ on the total number of meta-terms in the compressed index (Figures 7(a)), and on the compression ratio (Figures 7(b)). Changing μ from 0 to 100 reduces the total number of meta-terms in the compressed index at iteration 30 from 6.5M to 1.8M. At the same time, the compression ratio is reduced by only 6%.

7. RELATED WORK

Lossless compression of inverted indices has been an active topic for the past few years. Most of the developed techniques (e.g., variable-byte encoding, gamma-coding and delta-coding [16, 18]) aim at generating an efficient encoding of the entries in a posting list, and thus can be integrated with our approach (cf. Section 6). We also envision integrating our approach with lossy compression techniques, such as static pruning [6].

Several matrix factorization approaches have been proposed such as Non-negative Matrix Factorization [10, 13, 14], Principal Component Analysis [11], Latent Semantic Analysis [7], and Singular Value Decomposition [19]. Their goal is to factor a given matrix into two (or three) factor matrices that (optionally) exhibit some level of sparseness. Such techniques provide a *close approximation* of the input matrix, while our approach provides an *exact* factorization of the input matrix. Modifying such algorithms to be lossless is not straightforward. For example, one naïve approach is to compute the remainder matrix $R = V - WH$ so that V can be compactly represented using W , H , and R (i.e., $V = WH + R$). Unfortunately, there is no guarantee that sparseness of W and H would lead to sparseness of R . In fact, the size of R can be larger than the size of V because elements in V with values equal to zero may have non-zero values in the product WH . Another related work in the context of signal and image processing considers the

problem of representing a *signal* (i.e., a vector) using a linear combination of a small number of basis vectors from a *dictionary* (e.g., [2, 20]). The problem of selecting the optimal dictionary is similar to the problem we consider, with two main differences: (1) the dimensions of the factor matrices are selected in advance, and (2) the sparseness is required only for the encoding vectors (matrix W) and not for the basis vectors (matrix H).

8. CONCLUSION

We presented a novel approach for lossless compression of inverted indices based on exact matrix factorization. We proved that obtaining the optimal factorization is NP-hard, and developed an efficient greedy factorization algorithm. We described how to modify a typical top- k search algorithm to eliminate the query time overhead. Our experiments show that our technique achieves compression ratio of 35% while incurring negligible increase in the query evaluation time. Variable-byte encoding can be integrated with our approach to achieve overall compression ratios up to 50%.

9. REFERENCES

- [1] TREC WT10g Dataset, http://ir.dcs.gla.ac.uk/test_collections/wt10g.html.
- [2] On the uniqueness of overcomplete dictionaries, and a practical way to retrieve them. *Linear Algebra and its Applications*, 2006.
- [3] G. Beskales, M. Fontoura, M. Gurevich, S. Vassilvitskii, and V. Josifovski. Factorization-based lossless compression of inverted indices. Technical report, 2011. <http://arxiv.org/abs/1108.1956>.
- [4] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.
- [5] A. Z. Broder, D. Carmel, M. Herscovici, A. Soffer, and J. Zien. Efficient query evaluation using a two-level retrieval process. In *CIKM*, 2003.
- [6] D. Carmel, D. Cohen, R. Fagin, E. Farchi, M. Herscovici, Y. S. Maarek, and A. Soffer. Static index pruning for information retrieval systems. In *SIGIR*, 2001.
- [7] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of The American Society for Information Science*, 41(6):391–407, 1990.
- [8] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4):614–656, 2003.
- [9] S. Foucart and M.-J. Lai. Sparsest solutions of underdetermined linear systems via l_q -minimization for $0 < q \leq 1$. *Applied and Computational Harmonic Analysis*, 26(3):395 – 407, 2009.
- [10] P. O. Hoyer. Non-negative matrix factorization with sparseness constraints. *J. Mach. Learn. Res.*, 5:1457–1469, 2004.
- [11] I. Jolliffe. *Principal Component Analysis*. Springer Verlag, 1986.
- [12] R. Lämmel. Google’s mapreduce programming model — revisited. *Sci. Comput. Program.*, 68(3):208–237, 2007.
- [13] D. D. Lee and H. S. Seung. Algorithms for non-negative matrix factorization. In *In NIPS*, pages 556–562. MIT Press, 2001.
- [14] C. Liu, H.-c. Yang, J. Fan, L.-W. He, and Y.-M. Wang. Distributed nonnegative matrix factorization for web-scale dyadic data analysis on mapreduce. In *WWW*, 2010.
- [15] S. C. Madeira and A. L. Oliveira. Biclustering algorithms for biological data analysis: A survey. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 1(1):24–45, 2004.
- [16] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 1 edition, 2008.
- [17] G. Pass, A. Chowdhury, and C. Torgeson. A picture of search. In *Proc. 1st InfoScale*, 2006.
- [18] F. Scholer, H. E. Williams, J. Yiannis, and J. Zobel. Compression of inverted indexes for fast query evaluation. In *SIGIR*, 2002.
- [19] L. N. Trefethen and D. Bau. *Numerical Linear Algebra*. SIAM: Society for Industrial and Applied Mathematics, 1997.
- [20] J. A. Tropp. Greed is good: algorithmic results for sparse approximation. *IEEE Trans. on Info. Theory*, 2004.
- [21] H. Yan, S. Ding, and T. Suel. Inverted index compression and query processing with optimized document ordering. In *WWW*, 2009.