On the Size of Full Element-Indexes for XML Keyword Search

Duygu Atilgan[1,⋆], Ismail Sengor Altingovde[2], and Özgür Ulusoy[1]

[1] Computer Engineering Department, Bilkent University, Ankara, Turkey
{atilgan,oulusoy}@cs.bilkent.edu.tr
[2] L3S Research Center, Hannover, Germany
altingovde@L3S.de

Abstract. We show that a full element-index can be as space-efficient as a direct index with Dewey ids, after compression using typical techniques.

1 Full Element-Index for XML Keyword Search

Keyword search is a crucial operation that has to be supported on XML data. Earlier works attacking this problem from information retrieval (IR) perspective essentially consider disjunctive query semantics (e.g., see [2]); whereas works representing the database (DB) perspective mainly concentrate on Web-style conjunctive semantics (e.g., [1,4]). Typically, an inverted index is the preferred data structure for XML keyword search in both communities. In this respect, a straightforward approach is indexing each element in the XML data as a separate document, which is formed of the text contained in the element itself and that in all of its descendants [2]. This is called a *full (element-)index*.

While a full index can support both disjunctive and conjunctive keyword search semantics[1], the nested structure of XML data poses some efficiency challenges on its use in practice [1]. Most crucially, in a full index, a term t that is directly contained in an element at depth n is indexed n times, i.e., for each ancestor of that particular element (see Figure 1). This implies a non-trivial overhead in terms of storage space and query processing time.

To cope with the above problems of a full element-index, a key decision is indexing only direct textual content for each element, excluding the contents of its descendants. This so-called *direct* index remedies the redundancy inherent in the full index, and allows disjunctive query processing with a certain level of success (e.g., see [2]). However, for Web-style conjunctive query processing, such a direct index (in contrast to a full index) needs to explicitly capture the ancestor-descendant relationships among the elements. To this end, one of the most widely accepted solutions is labeling each element with Dewey IDs [1,4]. In Dewey ID representation, the label of a given node encodes the path from the document root down to the node so that the ancestor-descendant relationships between the nodes could be determined directly (see Figure 1).

⋆ Currently affiliated with Google Ireland.

[1] In this study, we focus on simple keyword queries without structural constraints.

R. Baeza-Yates et al. (Eds.): ECIR 2012, LNCS 7224, pp. 556–560, 2012.

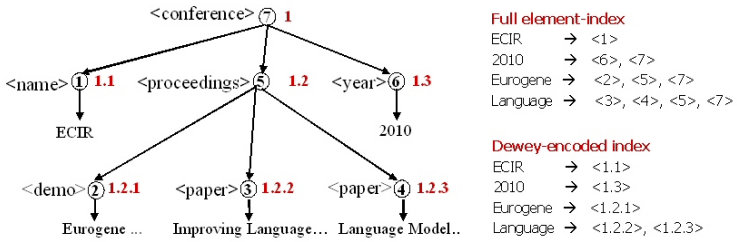

Fig. 1. An example XML tree and corresponding full and Dewey-encoded indexes

In this paper, we question one of the most important arguments against a full element-index, namely, index size. We advocate that, although a raw full index may be larger than a Dewey-encoded index, the size disadvantage may disappear after compression. Our claim is based on two key observations. First, the upper-level nodes of an XML document would be usually shared by many lower-level nodes, which would reduce redundancy in posting lists (e.g., in Fig. 1, since nodes 3 and 4 share the ancestor 5, it is enough for the posting list of term "language" to include only two ancestor nodes, namely, 5 and 7). Second, for typical tree-traversal orders, elements with ancestor-descendant relationships would be assigned very close ids, yielding smaller id gaps and higher compression ratio for a full index. In what follows, we justify our claims by a formal discussion and experimental results for three large datasets and different compression methods.

2 A Formal Comparison of Space Complexities

We provide a formal analysis for the space complexities of the full element-index and Dewey-encoded index. Without loss of generality, we use the well-known Elias-γ compression method [3] and restrict our discussion to compressing element ids (as they occupy the majority of the space in an index). We assume that the input XML tree to be indexed is a complete k-ary tree of depth d.

Space Complexity of the Dewey-Encoded Index (I_D). Dewey ID of a node at level m consists of m integers (where $1 \leq m \leq d$). That is, a node at level m is represented with the Dewey ID $a = a_1.a_2.a_3.\ldots.a_m$. In the worst case, only leaf nodes of an XML tree includes text, i.e., $m = d$. This is a viable assumption, since in a k-ary tree $(k-1)/k$ of the nodes are indeed leaves.

As each a_i is smaller than k, by using Elias-γ compression, a Dewey ID can be represented by at most $d(2\lg k + 1)$ bits[2]. Let's assume that the posting list of a term t in the direct index I_D consists of e number of elements. Then, the compressed size of the posting list of t would be $e \times d(2\lg k + 1)$ bits. Hence, the space complexity of a Dewey-encoded index is $O(ed \lg k)$.

[2] Recall that an integer x is encoded in $2\lg x + 1$ bits in Elias-γ compression [3].

Table 1. Dataset characteristics

	No. of Docs.	No. of Elem.	Max. Depth	Avg. Depth	Max Fan-out
DBLP	1	4.9 million	4	1.9	479,426
Wikipedia	659,388	7.4 million	47	2.6	5,621
XMark	1	1.6 million	11	4.5	10,000

Space Complexity of the Full Element-Index (I_F). Without loss of generality, the nodes of the input XML tree T are labeled with respect to the some tree traversal order of T. If T is a complete k-*ary* tree, these labels are smaller than the number of nodes in T, which is $K = 1+k+k^2+...+k^{d-1} = (k^d-1)/(k-1)$.

Assume that there are e elements in a term t's posting list in the Dewey-encoded index I_D, and e' elements in corresponding list in the full index I_F. As before, we also assume that all of the elements in a posting list of I_D are leaf elements at depth d. To compare the sizes of Dewey-encoded and full indexes, we have to estimate e'. We begin by analyzing two extreme cases: (i) If none of the leaf elements has a common ancestor except the root node, then they would have $e(d-1)+1$ distinct ancestors. In this case, the corresponding posting list in I_F would have $e' = e(d-1)+1+e = ed+1$ elements. (ii)All ancestors of these e leaf nodes are common. In this case, leaf elements would have $d-1$ ancestors and the corresponding posting list in I_F would have $e' = e+d-1$ elements.

However, both of these cases are quite rare. Therefore, we make an average case analysis and try to estimate a decay factor, α, which symbolizes the proportion of decrease in the number of ancestor nodes in consecutive levels. Assume that there are e_ℓ number of nodes at depth ℓ which contain term t directly and these elements have $e_{\ell-1} = \alpha e_\ell$ number of ancestors at depth $\ell-1$. Note that $\alpha \leq 1$ and hence, $e_{\ell-1} \leq e_\ell$.

Let's consider a practical case where $\alpha \leq 1/2$. In this case, since $e + e/2 + e/4 + ... + e/2^d \leq 2e$, e' is in the order of e. Recall that, for I_F, element id gaps are compressed instead of the actual element ids. Since the element ids are between 1 and K and there are e' elements in t's list in I_F, the average gap would be K/e'. Using Elias-γ method, a gap can be encoded in $2\lg(K/e')$ bits. Since $e' \leq 2e$, the compressed size of t's list with e' elements is:

$$e'2\lg\frac{K}{e'} = e'2\lg\frac{(k^d-1)/(k-1)}{e'} \quad < 2e2\lg\frac{(k^d-1)}{2e} < 4ed\lg k \quad = O(ed\lg k).$$

Hence, we conclude that for a typical XML tree where $\alpha \leq 1/2$, the space complexity of full and Dewey-encoded indexes are both $O(ed\lg k)$.

3 Experiments

We use two large real datasets, DBLP and Wikipedia, as well as a synthetically generated XMark dataset (Table 1). DBLP dataset contains a single XML document of size 207 MB obtained from the DBLP website. English Wikipedia XML collection is employed in INEX campaigns (in 2006-2008) and includes 52 million

Table 2. Storage requirements (in MBs) of full (I_F) and Dewey encoded (I_D) index files for DBLP, Wikipedia and XMark datasets (E-id: *element id*, D-id: *document id*, D: *depth*, F: *term frequency* fields as stored in the index)

		DBLP				**Wikipedia**					**XMark**			
		E-id	**D**	**F**	**Sum**	**D-id**	**E-id**	**D**	**F**	**Sum**	**E-id**	**D**	**F**	**Sum**
Elias-γ	I_F	32.2	0	3	35.2	0	389.8	0	103.4	493.2	40.1	0	6.9	47.0
	I_D	68.7	5.2	1.8	75.7	184.1	257.5	67.2	32.4	541.2	58.3	7.9	1.5	67.7
Elias-δ	I_F	28.5	0	3	31.5	0	351.2	0	114.6	465.8	36.7	0	7.1	43.8
	I_D	53.6	6.9	1.8	62.2	163.8	299.3	83.1	35.7	581.9	53.0	8.2	1.5	62.7
LT	I_D	50.3	5.3	7.1	62.9	441.3	443.1	132.4	264.8	1281.6	71.0	6.0	5.9	82.9

elements, from which 7.4 million retrievable (i.e., non-formatting) elements are indexed. We also generated an XMark dataset of around 100 MB.

We compare the storage requirements of the full element-index and Dewey-encoded index built for each dataset using three different compression methods. Elias-γ and Elias-δ are two well-known bit-aligned compression methods for compressing typical inverted files [3]. Level table (LT) method is specifically proposed for Dewey-encoded indexes [4].

In the experiments, we create index files only including pairs of <element id, frequency>. For full indexes, element id is assigned with postorder traversal of XML trees. We compress gaps between element ids, as usual. For Dewey encoded index, before each element id, we also store the *depth*, i.e., number of components in the following element id. Note that, for Wikipedia dataset, the first component of each Dewey id is the document id. In this case, we compress the gaps between the document id components of Dewey ids in a list. The succeeding components of the Dewey id (that correspond to the path within a document) are, of course, compressed as-is. Term frequency and depth values are also compressed as-is.

The compressed sizes are given for each dataset in Table 2. The results reveal that for both datasets, size of the full index is considerably smaller (ranging from 9% to 53%) than the corresponding Dewey-encoded index. The full index reserves more space for term frequencies, as they are repeated for ancestor nodes. Still, the element ids are compressed more efficiently for the full index than that for the Dewey-encoded index. This indicates that our assumptions used in the previous section hold for both datasets with different characteristics. That is, although a posting list in a full index stores ancestor elements redundantly, the gaps between their ids are small; and more crucially, an ancestor node at an upper level is indeed the ancestor of several nodes in the lower levels that include a particular term (i.e., implying a small decay factor as discussed in Section 2).

To sum up, our formal and experimental analyses show that a compressed full index can be as compact as a compressed Dewey-encoded index, and it can be safely employed in XML keyword search scenarios, which is our future work.

Acknowledgments. This work is partially supported by EU FP7 Project CUBRIK (contract no. 287704) and The Scientific and Technological Research Council of Turkey (TÜBİTAK) under the grant no 108E008.

References

1. Guo, L., Shao, F., Botev, C., Shanmugasundaram, J.: XRANK: Ranked keyword search over XML documents. In: Proc. ACM SIGMOD, pp. 16–27 (2003)
2. Lalmas, M.: XML Retrieval. Synthesis Lectures on Information Concepts, Retrieval, and Services. Morgan & Claypool Publishers (2009)
3. Witten, I.H., Moffat, A., Bell, T.C.: Managing Gigabytes: Compressing and Indexing Documents and Images, 2nd edn. Morgan Kaufmann (1999)
4. Xu, Y., Papakonstantinou, Y.: Efficient keyword search for smallest LCAs in XML databases. In: Proc. ACM SIGMOD, pp. 537–538 (2005)