Compression and an IR Approach to XML Retrieval

Vo Ngoc Anh Alistair Moffat

Department of Computer Science and Software Engineering
The University of Melbourne
Victoria 3010, Australia
www.cs.mu.oz.au/˜{vo,alistair}

Abstract

A two-phase evaluation scheme is proposed for XML retrieval. In the first phase, a modified vector space model is employed to obtain similarity scores for the textual nodes of XML trees. In the second stage, the scores are propagated upward in the XML trees, with scores of the textual nodes being modified and scores of other nodes being generated. As a result, while a vector space ranking is used, the final scores computed do not truly reflect the vector space scores. In addition to the two-phase evaluation, an integrated compressed file system is proposed for both storing and retrieving XML documents. This leads to an efficient representation of XML repositories.

1 Introduction

Applying IR techniques to XML retrieval is undoubtedly an interesting and promising approach. Conventional IR techniques, however, cannot be employed directly because of the need to handle content-and-structure queries. To accept this kind of query, retrieval systems must capture the structure of the documents and queries, and carry out some computation over these structures. In this paper we focus on two of the various aspects of the task. The first focus is on an alternative method to extend the vector space model to XML retrieval. The second is a unified compression scheme that supports both the retrieval model and efficient decompression of any part of an XML document. While the first goal is core to the *INEX* project, the second goal should as well be regarded as important. XML document collections can be large. Moreover, retrieval of XML elements involves not only the document content but also its structure, potentially consuming more disk space than retrieval of flat documents would.

A number of techniques to extend the vector space model to XML retrieval have been presented. Three main approaches are worth commenting on. Fuhr et al. [1998], Fuhr and Großjohamn [2001] explicitly indicate *indexing nodes*, each of which is a group of XML nodes. Indexing is then done for these nodes. This static index is used directly for query processing. Grabs and Schek [2002] proposed to generate vector space statistics on-the-fly during query processing. In this approach, a static index is built only for *basic indexing nodes*, which can be defined manually or automatically. At query time, the basic index is used to derive appropriated vector space statistics depending on the query scope. Carmel et al. [2002] chose to index the pairs *(path, word)* (as opposed to the conventional indexing of *words* only), where *path* is the XML path of the node that contains *word*.

A common property of these techniques is that they are tightly bound to the vector space model. During the evaluation of a query, the statistics are retrieved or generated for all nodes that are in the query's scope. These statistics are then used to compute similarity scores and rank the nodes. The common property likely guarantees the correctness of applying a vector space ranking, since otherwise there would be serious problems with ranking inconsistency. On the other hand, semi-structured XML documents are quite different from flat documents for which vector space ranking is good, and an alternative formulation of the similarity score might be preferable. Moreover, it is still not clear how to fairly combine different kinds of XML node according to a common statistical scale.

We use a vector space ranking technique because of its efficiency and effectiveness for flat text retrieval. But we do not rely exactly on the vector space score. Instead we adjust the scores, possibly more than once. The "right" statistics for an appearance of a word are counted once for the node that contains the word directly. Only these nodes are then processed through the conventional vector space ranking, regardless of whether they are compatible with the structural conditions of the query. Even at this stage, the scores computed are not exactly vector space scores – they are augmented according to the structural conditions. After the IR stage has been done, a second stage is conducted where the scores are propagated upward in the XML tree, and then the top nodes are selected as answers.

For our second goal – providing a compression framework for XML retrieval, we mainly rely on the existing work. Our contribution here is extending the current compression framework for flat text retrieval to XML retrieval. We introduce additional files to keep the XML collection in the compressed form, allowing effective decompression of any XML node.

The remainder of the paper is organized as follows.

```
<article>
    <atl> XML Retrieval </atl>
    <au sequence="first">
        <fnm> First N. </fnm>
        <ref> Surname </ref>
    </au>
    <sec> <st> Everything </st>
        <p>
            Everything about XML </it>
            and XML retrieval </it>.
        </p>
    </sec>
</article>
```

Figure 1: Example of XML document.

Section 2 introduces some concepts of XML documents and presents our opinion on query format and interpretation of queries. Then, section 3 describes the data structures employed for compressing XML collections. Section 3 also introduces a general scheme for query evaluation with these structures. Sections 4 and 5 describe the main techniques employed for the two phases of evaluation. Section 6 outlines the experiments we undertook.

2 Documents and queries

Documents A simplified example of an XML document is provided in Figure 1 and is used throughout this text to illustrate the concepts introduced.

It is convenient to list some of the standard definitions here. Thus, an XML document is a set of *nodes* or *elements* such as <*article*> and <*p*>. Each node is associated with a *path*, for example, */article* and */article/sec/p*. The exact location and content of a node is defined by its *positional path*. For example, if the above XML document is the first one in a collection, then */article[1]/sec[1]/p[1]/it[1]* and */article[1]/sec[1]/p[1]/it[2]*, respectively, is used to indicate XML </*it*> and XML retrieval </*it*>.

The following concepts are introduced for this paper. A node is called *textual* if and only if it has some proper free text which does not belong to any of its children or descendants. Otherwise, the node is called *skeleton* and it contains no proper text. In the above document, for example, textual nodes are

/article[1]/atl[1],
/article[1]/au[1]/fnm[1],
/article[1]/au[1]/ref[1],
/article[1]/sec[1]/st[1],
/article[1]/sec[1]/p[1],
/article[1]/sec[1]/p[1]/it[1],
/article[1]/sec[1]/p[1]/it[2];
and the skeletal nodes are
/article[1],
/article[1]/au[1],
/article[1]/sec[1].

```
<query>
    <te> article </te>
    <ce>
        <cp> bdy/sec </cp>
        <cw> nonmonotonic reasoning </cw>
    </ce>
    <ce>
        <cp> hdr//yr </cp>
        <cw> 1999 2000 </cw>
    </ce>
    <ce>
        <cp> tig/atl </cp>
        <cw> <nw> calendar </nw> </cw>
    </ce>
    <cw> belief revision </cw>
    <kw>
        nonmonotonic reasoning belief revision
    </kw>
</query>
```

Figure 2: Example of query: the reformatted version of topic 09.

Note that normally in an XML tree, leaf nodes are textual, and internal nodes are skeletal, but this cannot be assumed. A counter-example is the */article[1]/sec[1]/p[1]*, which is an internal node, but containing some proper text. This type of node is popular in the *INEX* collection.

The textual part of a textual node, including any accompanying punctuation, is called a *textual item* of the node *wrt* the XML collection. Thus, the textual item of */article[1]/au[1]/ref[1]* is Surname, while that of */article[1]/sec[1]/p[1]* is Everything about and.

Queries We appreciated the straightforward query format supplied by the *INEX* organizer and described by Fuhr et al. [2002]. In our opinion, the format (of course, after removing *Description* and *Narrative* fields) is simple and powerful enough, at least for the purpose of IR approaches.

To make the queries more consistent, we introduced a couple of small changes to the initial format. Firstly, words appearing in a *ce* field are included inside the field itself. Secondly, a formal element <*nw*> ... </*nw*> is added to surround negative words in queries. For example, topic 09 is now reformatted as shown in Figure 2.

We believe that the *Keywords* of the original *INEX* queries is unnecessary and it would better be removed totally from the query format, making queries simpler and shorter. However, to be consistent with the settlement of this round of *INEX*, this element is left in this format with the new name of <*kw*>.

There is a number of points that should be made clear. Firstly, the *Title* field in this format is removed since we consider that field the main part of queries. As the field is in fact a structured node, it is simply removed.

Secondly, the format is used for both content-only and content-and-structure queries, and we also recommend the use of queries which have no *te* field but contain *ce* fields. Thirdly, it is easy to build a script to transfer all *INEX* queries to the new format automatically. And last, except for the *te* field, all other information should be considered by a retrieval system as inexact constraints as is also the case in conventional IR ranking. For example, the first *ce* element in 2 should be interpreted as the desire of having the sections discussing *about* "non-monotonic reasoning", and it does not necessarily mean that the sections must contain these word. In the same manner, a retrieved article for the query, for example, might not be published in 1999-2000 as required by the topic's author.

3 System Architecture

Backbone Our system is based on the *MG* system (see `http://www.cs.mu.oz.au/mg/`). The main feature of *MG* for text retrieval is that it applies compression to the documents as well as to the index. This feature is especially suitable for our task of building a compact repository for XML retrieval. We report here only the changes made specifically for this task.

File system *Textual and related files:* All textual items of the XML documents are gathered together in a data structure, called *textual file*. That is, each item in this file corresponds to one textual node of a certain XML document. This file is compressed and is accompanied with some auxiliary files supporting direct access to, and decompression of, each of of the textual items. An illustration of textual files is given at the bottom left of Figure 3. Information about text compression methods employed, as well as about the auxiliary data structures, can be found in [Witten et al., 1999].

Structural files: Each node (either textual or structural) of any XML document has an entry in a structural file. In this data structure, entries are stored in the order of their appearance (or, more correctly, of the appearance of their opening markups) in the XML collection. An entry of the structural file describes a node's structure and its position in the parent's node. The entry includes

- the opening markup of the node (including the accompanying parameters, if any);
- distance to the parent node (that is, number of nodes between the node and its parent, which is 0 if the current node is a root node);
- byte-offset position of the beginning of the node relative to the (end of) its immediately preceding markup;
- pointer to the textual item of the node, that is, to the corresponding item in the textual file (the value is 0 if the node is a skeleton).

The bottom right block in Figure 3 illustrates the content of a structural file. Note that for each node, the closing markup is not stored.

To the structural items, random access is needed. Since all the numerical values of the file is generally small, and the texts (that is, the markups) are generally repeated, the file can be compressed effectively even with the random access requirement. Our ad-hoc solution is to use a dictionary for all the text parts, then to replace each text with the pointer to the corresponding element in the dictionary, hence transforming each structural item to a quadruple of numbers prior to the compression. Conventional compression techniques for small integers are then applied.

Note that with support of the textual files, which allow direct access and decompression of any textual item, the structural file can be used to build back any node of the original XML document collection. An example of this process is given in Figure 3. Truly, the compression is lossy: when there is no text between two consecutive markups, the punctuation between them (if any) is not stored anywhere. However, as the primary purpose of the XML documents is to have the structure of documents along with their texts, not to render them, the compression scheme can be considered as lossless.

Text-structure mapping files: A text-structure mapping file is illustrated at the top of Figure 3. The file maps any item in textual file to its corresponding entry in the structural file. During query processing it is better to have the mapping resided in the memory, so the random access to the file is not required. Hence, the numbers indicating absolute position of a structural node (in the structural file) are replaced by the gaps between it and its preceding . That is, run-length coding is applied. In our current implementation, Elias's Gamma code Elias [1975] is used for this purpose.

Index files: Changes have been made to *MG* to suit our needs, in both the indexing and the querying modules. While the changes are already reported in Anh and Moffat [2002], it is worth reiterating that the weighting scheme for terms of the textual items is integrated into the index, and that during query processing, the calculation of cosine measure for these items is not required.

Remark: It might be arguable about the need to divide the XML collection into textual, structural and the mapping files since keeping them in one file might be better for compression. The point is that during query evaluation the structural parts are needed anyway, when the whole textual parts are needed only when the documents need to be rebuilt to present as the answer. Another argument might be that it would probably better to insert empty items to the textual file to represent the structural nodes, and hence exclude the mapping file from consideration. However, number of such empty items is relatively high, making the compression of inverted files ineffective.

Query evaluation After an query has been parsed, information about each of its distinct terms is stored in a general list data structure. This information includes

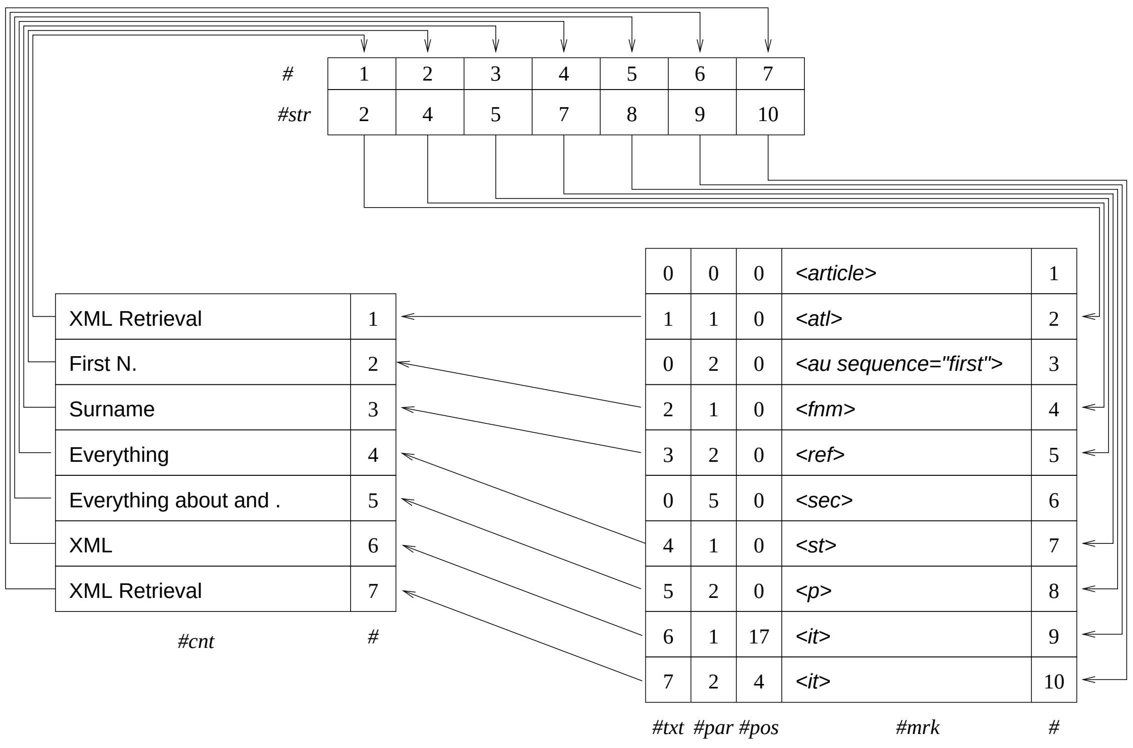

Figure 3: Example of textual, structural and their mapping files. The picture conceptually describes a database which has only one XML document, namely, the document presented in Figure 1. The arrows represent explicit or implicit links from a file to another. To each file, the field # is added to show the item number. The contextual file is shown at the bottom left. It contains 7 items, each for one textual node of the document. The ranking is first done in the IR manner for these items. In this process, the system can use the mapping file (top) to map the items with their path in the structural file (bottom right). In the structural file, the *#par* link a node to its parent. For example, the value 2 of *#par* for the last item (item number 10) means that its parent is at 2 positions ahead, that is, is the item number 10-2=8. The columns *#txt* and *#pos* are used to rebuild nodes. For example, rebuilding of item number 8 begins from building its initial string value <*p*> Everything about and . </*p*>, then the next structural items are taken to insert into this string since they are the item's children. Value *#par = 17* of item 9 shows that the corresponding node should be inserted to position 17 after <*p*> (its preceding markup), making the above string become <*p*> Everything about XML </*it*> and . </*p*>. Similarly, item number 10 should be inserted to this string at position 4 after </*it*>.

representation of the term itself, list of the query's <*ce*> paths that contains the term (with a special value to represent "any path"), and the frequency for each of these paths. The evaluation then involves the following main steps:

1. *Scoring:* Conventional vector space technique, with an adjustment to take into account structural conditions of queries, is employed to calculate similarity score for each textual item. The weighting scheme for this step is reported in section 4.

2. *Propagation:* The scores obtained are propagated upward in the XML tree, hence awarding the internal (not necessarily being structural) nodes with some scores. The techniques for doing this step is shown in section 5.

3. *Selection:* After the previous step we come up with a list of nodes with non-zero scores. The task of the selection step is a) to delete some anomalies, and b) to select the nodes with the top scores. There are three situations where a node is considered abnormal. The first case is when the node or any of its descendants has negative score. The second case happens when the parent of the node scores higher than it as well as any of its siblings. The motivation behind this case is to avoid retrieving the descendants of retrieved nodes. The third case is applied only to content-and-structure queries. It involves the clearing of scores of the nodes that do not belong to the <*te*> list.

4. *Presentation:* The list of the nodes with the top scores is now used to retrieve the actual nodes. In this step, we use information from the structural file to rebuild the full node. Figure 3 serves as an example for this process.

For simplicity, the first and second steps, and only them, are referred to as the first and second, respectively, phase of the query evaluation process.

4 Weighting Textual Items

The weighting scheme employed for the textual items is based on our impact transformation technique [Anh and Moffat, 2002]. The weight is an integer number and computed as

$$S_{d,q} = \sum\nolimits_{t \in q \cap d} p_{d,q,t} \cdot \omega_{d,t} \cdot \omega_{q,t}$$

where $p_{d,q,t}$ is *cross-structural importance* of t relative to d and q, $\omega_{d,t}$ and $\omega_{q,t}$ are *quantized impact* of term t in textual item d and query q, respectively.

The cross-structural importance is defined by

$$p_{d,q,t} = c_w \cdot p^{w}_{d,q,t} + c_{\bar{w}} \cdot p^{\bar{w}}_{d,q,t} + c_e \cdot p^{e}_{d,q,t} + c_{\bar{e}} \cdot p^{\bar{e}}_{d,q,t} \; .$$

Here c_w, $c_{\bar{w}}$, c_e and $c_{\bar{e}}$ are constants and, in this series of experiments, are set to 1, 10, -10 and -20, respectively. Other values are generally 0 except for the following special cases:

- $p^{w}_{d,q,t}$ is set to 1 if t appears in either */query/cw* or */query/kw*, and d is any textual item,
- $p^{\bar{w}}_{d,q,t} = 1$ if t appears in */query/cw/nw*, and d is any textual item,
- $p^{e}_{d,q,t} = 1$ if t appears in an */query/ce/cw* field and the parent of this field contains at least one item that is the same as, or the ancestor of, the path name of the textual element d,
- $p^{\bar{e}}_{d,q,t} = 1$ if t appears in an */query/ce/cw/nw* field, and the ancestor */query/ce* of this field contains at least one item that is the same as, or the ancestor of, the path name of the textual element d.

Each of the quantized impacts $\omega_{d,t}$ and $\omega_{q,t}$ is in the range 1 to 2^b, with (in these experiments) $b = 5$. Each of them is calculated in two steps. First, a normal cosine similarity is used to compute $w^*_{d,t}$ and $w^*_{q,t}$:

$$\begin{aligned}
w_{d,t} &= (1 + \log_e f_{d,t}) \\
W_d &= \sqrt{\sum_{t \in d} w^2_{d,t}} \\
W^*_d &= 1/((1-s) + s \cdot W_d / W^a) \\
w^*_{d,t} &= w_{d,t}/W^*_d \\
w^*_{q,t} &= \log_e \left(1 + \frac{f^m}{f_t}\right) \cdot (1 + \log_e f_{q,t})
\end{aligned}$$

where $f_{d,t}$ is the term frequency in the textual item, $f_{q,t}$ is frequency of t in the textual part of the query q (that is, $f_{q,t}$ is calculated without considering the markups); f_t is the number of textual items that contain term t; f^m is the greatest value of f_t in the textual file; W_d is length of the textual item d; W^a is the average value of W_d over all items of the textual file; and W^*_d represents the normalized item length using pivoted normalization [Singhal et al., 1996] with a slope of $s = 0.7$.

Then, a small enough positive value L and a large enough positive value U are chosen such that all of the $w^*_{d,t}$ lie between L and U, thereby allowing the following transformation to be calculated:

$$\begin{aligned}
\omega_{d,t} &= \left\lfloor 2^b \cdot \frac{\log w^*_{d,t} - \log U}{\log U - \log L + \epsilon} \right\rfloor + 1 \\
\omega_{q,t} &= \left\lfloor 2^b \cdot \frac{\log w^*_{q,t} - \log U}{\log U - \log L + \epsilon} \right\rfloor + 1
\end{aligned}$$

in which $B = (U/L)^{L/(U-L)}$, and ϵ is a small positive quantity, and the impact values are recorded and used as integers.

Our experiments made use of two different types of transformation, characterized by the choice of L and U. In the first, referred to as *global*, the values of L and U respectively are the minimum and maximum values of $w^*_{d,t}$ over the whole textual file. In the second, referred to as *local*, each textual item or query x is associated with its own L and U, which are the minimum and maximum among all of the values $w^*_{x,t}$ generated from x. That is, in the local transformation, a value $w^*_{x,t}$ is transformed with respect to the values of L and U of x – the textual item or query it belongs to.

5 Propagating Scores

After having the scores of the textual nodes, the next step is to propagate the scores upward in the XML trees (or tree). Two methods are investigated in our experiments. In the description of the methods (below) it is supposed that the propagation is being done for a node b whose parent is a, and that a has totally n children, of them m have non-zero (possibly negative) score.

The first method is called *maximum-by-category*. Here, each distinct term is called a *category*. For this method, whenever a score is computed, regardless of whether the computation belongs to the first or the second phase of the evaluation process, it is calculated separately and kept separately for each category. A real score of an item is then the sum of its scores over the categories. Hence the categorical score of b can be represented as $(s_1(b), s_2(b), \ldots, s_{|q|}(b))$, and the real score for b is

$$s(b) = \sum_{i=1}^{|q|} s_i(b)$$

where $|q|$ is number of distinct terms of query q. The score $s(a)$ of a is computed based on

$$s_i(a) = s_i(a) + \text{sign } s_i(b) \cdot \alpha \cdot \max_b |s_i(b)|,$$

where $|s_i(b)|$ is the absolute value of $s_i(b)$, α is a constant and is set to 0.8 in these experiments.

Label	Characteristics
um_mgx21_short	
	Queries: not having <*kw*> elements
	Type of transformation: global
	Propagation method: summation
um_mgx2_long	
	Queries: having <*kw*> elements
	Type of transformation: global
	Propagation method: maximum-by-category
um_mgx26_long	
	Queries: having <*kw*> elements
	Type of transformation: local
	Propagation method: maximum-by-category

Table 1: Settlement of the experiments

The second method of propagation is called *summation*. It involves not only the calculation of $s(a)$ but also the re-scoring of $s(b)$. $s(a)$ is computed as

$$s(a) = s(a) + \sum_b (\beta \cdot s(b)/n + \gamma \cdot s(b)/m)$$

and $s(b)$ is redefined as

$$s(b) = s(b) - (\beta \cdot s(b)/n + \gamma \cdot s(b)/m).$$

where β and γ are constants. Both of them are set to 0.5 in the experiments reported below.

6 Experiments

Hardware The experiments were carried out on a 933 MHz Intel Pentium III with 1 GB RAM, a 9 GB SCSI disk for system needs, and four 36 GB SCSI disks in a RAID-5 configuration for data. The indicative times reported below are for experiments in which there was no other activity on the hardware.

Experiment parameters Three experiments were conducted. Their labels and settings are listed in Table 6.

References

V. N. Anh and A. Moffat. Impact transformation: effective and efficient web retrieval. In M. Beaulieu, R. Baeza-Yates, and S. H. Myaeng, editors, *Proc. 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 3–10, Tampere, Finland, Aug. 2002. ACM Press, New York.

D. Carmel, N. Efraty, G. M. Landau, Y. S. Maarek, and Y. Mass. An extention of the vector space model for querying XML documents via XML fragments. In *Proc. SIGIR'2002 Workshop on XML and Information Retrieval*, pages 14–25, Tampere, Finland, Aug 2002.

W. Croft, D. Harper, D. Kraft, and J. Zobel, editors. *Proc. 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, New Orleans, Louisiana, USA, Sept. 2001. ACM Press, New York.

P. Elias. Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory*, 21(2): 194–203, Mar. 1975.

N. Fuhr, N. Gövert, G. Kazai, and M. Lalmas. INEX: Initiative for the Evaluation of XML Retrieval. In *Proc. SIGIR'2002 Workshop on XML and Information Retrieval*, pages 62–70, Tampere, Finland, Aug 2002.

N. Fuhr, N. Gövert, and T. Rölleke. Dolores: A system for logic-based retrieval of multimedia objects. In W. B. Croft, A. Moffat, C. J. van Rijsbergen, R. Wilkinson, and J. Zobel, editors, *Proc. 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 257–265, Melbourne, Australia, Aug. 1998. ACM Press, New York.

N. Fuhr and K. Großjohamn. XIRQL: a query language for information retrieval in XML. In Croft et al. [2001], pages 172–180.

T. Grabs and H. Schek. Generating vector spaces on-the-fly for flexible XML retrieval. In *Proc. SIGIR'2002 Workshop on XML and Information Retrieval*, pages 4–13, Tampere, Finland, Aug 2002.

A. Singhal, C. Buckley, and M. Mitra. Pivoted document length normalization. In H.-P. Frei, D. Harman, P. Schäuble, and R. Wilkinson, editors, *Proc. 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 21–29, Aug. 1996.

I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann, San Francisco, second edition, 1999.