XML Information Retrieval through Tree Edit Distance and Structural Summaries

Cyril Laitang, Mohand Boughanem, and Karen Pinel-Sauvagnat

IRIT-SIG,
118 route de Narbonne,
31062 Toulouse Cedex 9,
France

Abstract. Semi-structured Information Retrieval (SIR) allows the user to narrow his search down to the element level. As queries and XML documents can be seen as hierarchically nested elements, we consider that their structural proximity can be evaluated through their trees similarity. Our approach combines both content and structure scores, the latter being based on tree edit distance (minimal cost of operations to turn one tree to another). We use the tree structure to propagate and combine both measures. Moreover, to overcome time and space complexity, we summarize the document tree structure. We experimented various tree summary techniques as well as our original model using the SSCAS task of the INEX 2005 campaign. Results showed that our approach outperforms state of the art ones.

1 Introduction

XML documents are organized through semantically meaningful elements. As content is distributed over different levels of the document structure, using this information should improve the overall search process as well as it enables the returned information to be more focused on the expressed needs. In this context, Semi-Structured information retrieval (SIR) models aim at combining content and structure search processes.

XML documents are structured through nested tags. This hierarchical organization is naturally expressed through a specific graph representation, i.e. trees[1], in which nodes are elements and edges hierarchical dependencies. In an XML tree, the text is located in the *leaves* which are the bottom nodes of the hierarchy. In structured retrieval, queries can be expressed using either *Content Only* constraints (CO) or both *Content And Structure* constraints (CAS). As for XML documents, the structural constraints expressed in CAS queries can be visualized through a tree representation, and might contain two parts: the *target element* indicates the tag element we want to retrieve and the rest of the structural constraints is called *support* or *environment*.
Figure 1 shows a conversion example of an XML document and a query[2]. The

[1] Trees are a particular type of graphs which do not contain any cycles. Cycles are paths starting and ending with the same node.

[2] This query is expressed in the NEXI [20] (Narrowed Extended XPath) language used in the context of the INEX evaluation campaign.

M.V.M. Salem et al. (Eds.): AIRS 2011, LNCS 7097, pp. 73–83, 2011.

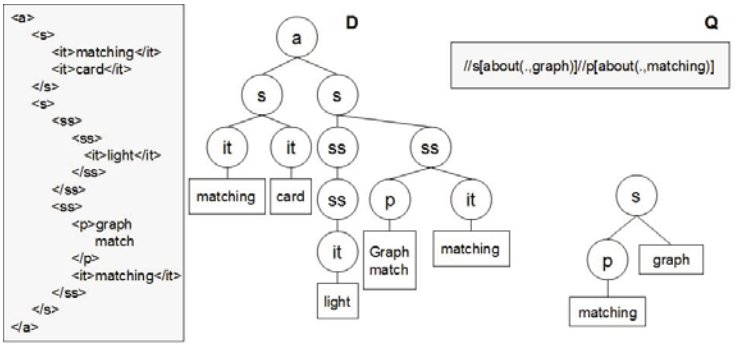

Fig. 1. Tree representation of an XML document and a query in which *we want a "p" element about "matching"' contained in an "s" element about "graph"*

target element is "p". For clarity reasons we shorten the tags. In real case the "p" is equivalent to the semantically richer "paragraph".

Based on these representations we propose a SIR model based on both graph theory properties and content scoring. To our knowledge, only few SIR models explicitly use tree matching algorithms between documents and queries. Most of them reduce the structure of documents to strings which impoverish the amount of information available to the matching process. Some authors such as Alilaouar et al. [1], Ben Aouicha et al. [2] and Popovici et al. [16], lose a high level of details to get back to their domain known approaches. Our approach uses the well-known tree edit distance combined with a traditional IR model for content. This allows to use the whole document structure to select relevant elements. Moreover to improve space and time complexity we propose three methods to summarize documents.

The rest of this paper is organized as follows: Section 2 presents existing tree matching algorithms and XML retrieval approaches using trees; Section 3 presents our model and finally Section 4 discusses the experiments and results obtained by our approach using the SSCAS task of the INEX 2005 campaign.

2 Related Works

In this section we will first overview some tree matching algorithms and then give a brief survey on some SIR approaches based on tree matching.

2.1 Structural Similarities between Trees

Two graphs are called isomorphic if they share the same nodes and edges. Evaluating how isomorphic are two graphs is called graph matching. We make the distinction between approximate matching and exact matching. The first one attempts to find a degree of similarity between two structures while exact matching tries to validate the similarity. Because of the context of our work, we will focus here on approximate tree matching. There are three main families of approximate tree matching: *alignment*, *inclusion*, and *edit distance*. As the later offer

the most general application we will focus on this one. Tree edit distance algorithms [18] generalizes Levenshtein *edit distance*[14] to trees. The similarity is the minimal set of operations (adding, removing and relabeling) to turn one tree to another. Given two forests (set of trees) F and G, Γ_F and Γ_G their rightmost nodes, $T(\Gamma_F)$ the tree rooted in Γ_F and the cost functions $c_{del}()$ and $c_{match}()$ for removing (or adding) and relabeling, the distance $d(F, G)$ is evaluated according to the following recursive lemma:

$$\begin{aligned} d(F,\emptyset) &= d(F-\Gamma_F,\emptyset) + c_{del}(\Gamma_F) \\ d(\emptyset,G) &= d(\emptyset,G-\Gamma_G) + c_{del}(\Gamma_G) \\ d(F,G) &= min \begin{cases} (a)\ d(F-\Gamma_F,G) + c_{del}(\Gamma_F) \\ (b)\ d(F,G-\Gamma_G) + c_{del}(\Gamma_G) \\ (c)\ \begin{array}{l} d(T(\Gamma_F)-\Gamma_F, T(\Gamma_G)-\Gamma_G) \\ +d(F-T(\Gamma_F), G-T(\Gamma_G)) + c_{match}(\Gamma_F,\Gamma_G) \end{array} \end{cases} \end{aligned} \quad (1)$$

Operations *(a)* and *(b)* are respectively the cost $c_{del}()$ of removing Γ_F or Γ_G while *(c)* is the cost $c_{match}()$ of relabeling the Γ_F by Γ_G. Later, Klein et al. [13] reduced the overall complexity in time and space by splitting the tree structure based on the heavy path (defined in Section 3.3). Demaine et al. [6] further improved this algorithm by storing substrees scores in order to reduce calculation time. Finally Touzet et al. [8] used a decomposition strategy to dynamically select always the best nodes to recurse on between rightmost and leftmost which reduce the number of subtrees in memory. The best tree edit distance algorithms uses $\Theta(nm)$ on space complexity and between $\Theta(n \times log(n).m \times log(m))$ [8] and $\Theta(n.m(1 + log(\frac{n}{m})))$ [6] on time for n and m the respective sizes in nodes of two trees T_1 and T_2. Tree edit distance algorithms are efficient but slow on large trees. One way to overcome this issue in SIR is to reduce the document tree size by pruning or summaries. Used mainly on indexing and clustering, the latter is based on the intuition that the document underlying structure can be captured efficiently with a smaller size representation. As we will see in Section 2.2, the summarizing algorithm proposed in [4] provides satisfying results in improving the runtime. In this paper, we will apply this algorithm to XML retrieval.

2.2 Semi-structured Information Retrieval

XML documents as well as CAS queries can be represented as trees. It then seems natural to apply tree matching algorithms for the document-query matching process. However, following INEX proceedings overviews ([7] and [9]), tree matching is uncommon in SIR. One can however find two main categories of retrieval approaches in the literature using XML trees matching.

The first one uses *relaxation*, which means reducing the links or constraints. Relaxing is a research space expansion process. The general idea is to translate the tree structure into a set of binary weighted edges. This simplified representation allows to use traditional IR models. For example, Alilaouar et al. [1] combined relaxation and minimal covering trees. Similarly Ben Aouicha et al. [2] relaxed the whole hierarchical constraints by adding virtual edges. These edges are weighted based on the initial hierarchical distance in the document structure.

The resulting arcs are then projected on a vector space and the document-query matching is done using a traditional *Vector Space Model*. Some other approaches as for example [5] use *fuzzy closing* in which a set of virtual edges representing all tree's transitive relationships is created based on the closing property. One can also found approaches that directly use *edit distance algorithms*. Popovici et al. [16] translated documents tree structure in a set of paths and evaluated the document-query similarity using the Levenshtein [14] string edit distance. Other works using tree matching algorithms can be found in IR related domains, like error detecting and clustering. In their papers, Boobna [3] and Rougemont [17] check XML documents conformity to their DTD thought tree edit distance. Similarly, Dalamagas et al. [4] summarize documents before applying Tai [18] algorithm to cluster them.

The fact that there are relatively few SIR models using explicitly tree matching algorithms could be due to the complexity of such approaches. All of the previously presented models always applies trees conversion to restrict the problem to a smaller search space. This removes a lot of details. In this paper we attempt to overcome the complexity drawback by summarizing the structure without loosing too much of the structural information.

3 Tree-Edit Distance for Structural Document-Query Matching

We assume that a query is composed of content (keywords) and structure conditions, as shown in Figure 1. The document-query similarity is evaluated by considering content and structure separately, and we then combine these scores to rank relevant elements. In this section, we first describe the content evaluation. We then present our subtree extraction and summary algorithms. Finally we detail our structure matching algorithm based on tree edit distance.

3.1 Content Relevance Score Evaluation

First, we used a $tf \times idf$ (Term Frequency $\times$ Inverse Document Frequency [11]) formula to score the document leaf nodes according to query terms contained in content conditions. To score inner nodes, our intuition is that a node score must depend on three elements. First, it should take into account its leaves scores, that form what we call intermediate score. Second we should score higher a node located near a relevant element than a node located near an irrelevant one. Finally, there must be a way to balance the hierarchical effect on the node score. Based on these constraints we define the content score $c(n)$ of an element n as the *intermediate content score of the element itself plus its father's intermediate score plus all its father's descendants score*. Recursively, and starting from the document root:

$$c(n) = \begin{cases} \underbrace{\frac{p(n)}{\mid leaves(n) \mid}}_{(i)} + \underbrace{\frac{p(a_1) - p(n)}{\mid leaves(a_1) \mid}}_{(ii)} + \underbrace{\frac{c(a_1) - \frac{p(a_1)}{\mid leaves(a_1) \mid}}{\mid children(a_1) \mid}}_{(iii)} & \text{if } n \neq root \\ \underbrace{\frac{p(n)}{\mid leaves(n) \mid}} & \text{otherwise} \end{cases} \quad (2)$$

(i) is the *intermediate content score* part with $\mid leaves(n) \mid$ the number of leaf nodes descendants of n and $p(n)$ the intermediate score of the node based on the sum of the scores of all its leaf nodes: $p(n) = \sum_{x \in leaves(n)} p(x)$, with $p(x)$ evaluated using a $tf \times idf$ formula.
(ii) is the *neighborhood score* part which allows us to convey a part of the relevance of a sibling node through its father a_1. $p(a_1)$ is the intermediate score of a_1 and $\mid leaves(a_1) \mid$ the number of leaves of a_1.
(iii) is the *ancestor scores*, with $c(a_1)$ the final content score of the father a_1 minus its intermediate score.

3.2 Extracting and Summarizing Subtrees

For each relevant leaf node (i.e. with a score $p(x) > 0$), we extract all the subtrees rooted by all its ancestors starting from the first one whose tag is similar to a tag in the query. In Figure 2, five subtrees are extracted.

Even the best edit distance algorithm runs with the minimal time complexity $O(n^3)$ [8], it thus remains costly on large trees. In order to reduce the matching space, we choose Dalamagas [4] summary rules which seem to be the best compromise between preserving the overall structure and speed. As illustrated in Figure 3 these rules are "remove nesting" which is equivalent to move the subtree of a node having the same label than one of its ancestor; and "remove duplicates from father-child relationship" which removes the siblings having the same labels. Based on these rules we create four different summary versions. The first one, illustrated on top, is the strict adaptation of these rules to all nodes while the second version, illustrated at the bottom, apply the summary rules only for the nodes which labels are not in the query. Thanks to this version

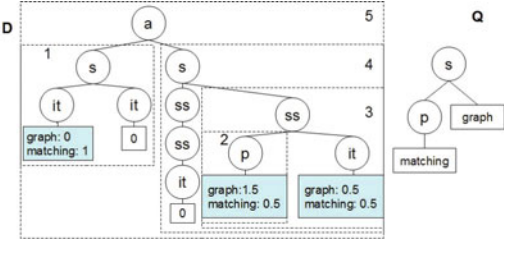

Fig. 2. Subtrees extraction based on relevant leaf nodes

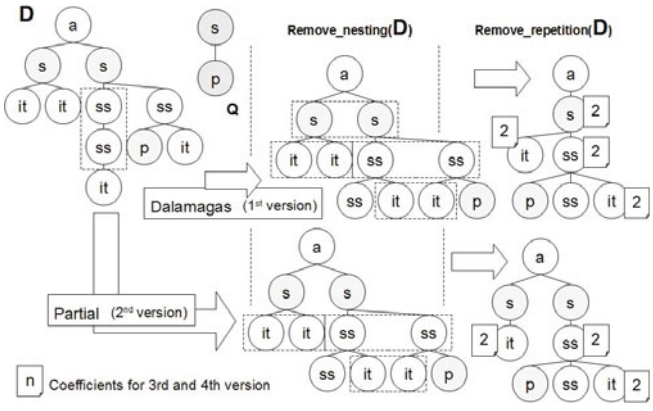

Fig. 3. The original document tree D and it's tree summaries. On top the rules applied for all node, at the bottom the rules applied only for tags that are not in the query.

we keep as many relevant nodes as possible while reducing the document size. Finally, in the third and fourth versions, we adapt the first two ones by keeping a record of the number of removed nodes. This will be used to multiply the edit costs $c_{del}()$ or $c_{match}()$ during the recursion (eg. $c_{del}(it)$ will be multiplied by 2).

3.3 Structure Relevance Score Evaluation

As seen in Section 2.1, the tree edit distance is a way of measuring similarity based on the minimal cost of operations to transform one tree to another. The number of subtrees stored in memory during this recursive algorithm depends on the direction we choose when applying the operations. Our algorithm is an extension of the optimal cover strategy from Touzet et al. [8]. The difference is that the optimal path is computed with the help of the *heavy path* introduced by Klein et al. [13]. The heavy path is the path from root to leaf which pass through the rooted subtrees with the maximal cardinality. This means that selecting always the most distant node from this path allows us create the minimal set of subtrees in memory during the recursion creating the *optimal cover strategy*. Formally a heavy path is defined as a set of nodes $[n_1, ..., n_i, ..., n_z]$ satisfying:

$$\forall (n_i, n_{i+1}) \in heavy \begin{cases} n_{i+1} \in children(n_i) \\ \forall x \in children(n_i), x \notin n_{i+1}, \mid T(n_{i+1}) \mid \geq \mid T(x) \mid \end{cases}$$

This strategy is used on the document and the query as input to our tree edit distance algorithm (Algorithm 1). F, G are two forests (i.e. the document and the query as first input), and p_F and p_G are positions in O_F and O_G the *optimal paths* (i.e. paths of the *optimal cover strategy*). Function $O.get(p)$ returns the node in path O corresponding to position p. .

To evaluate the final structure score of a node n, we average the tree edit distances between its subtrees and all of its ancestor ones and the query. These distance are averaged by their cardinality in order to reduce the gap size between the subtrees. With $Anc(n)$ the set of n ancestors; $a \in Anc(n)$; $T(a)$ the subtree

rooted in a; $d(T(a), Q)$ the edit distance between the tree rooted in a $T(a)$ and Q, the structure score $s(n)$ is formally:

$$s(n) = \frac{\sum_{a \in \{n, Anc(n)\}} (1 - \frac{d(T(a),Q)}{|T(a)|})}{\mid Anc(n) \mid} \tag{3}$$

Algorithm 1. Edit distance using optimal paths

```
d(F, G, p_F, p_G) begin
    if F = ∅ then
        if G = ∅ then
            return 0;
        else
            return d(F - O_F.get(p_F)), ∅, p_F++, p_G) + c_del (O_F.get(p_F);
        end
    end
    if G = ∅ then
        return d(∅, G - O_G.get(p_G)), p_F, p_G++) + c_del (O_G.get(p_G);
    end
    a = d(F - O_F.get(p_F), G, p_F++, p_G) + c_del (O_F.get(p_F);
    b = d(F, G - O_F.get(p_F), p_F, p_G++) + c_del (O_G.get(p_G));
    c = d(F - O_F.get(p_F), G - O_G.get(p_G), p_F++, p_G++) + d(F -
    T(O_F.get(p_F)), G - T(O_G.get(p_G))) + c_match (O_F.get(p_F), O_G.get(p_G));
    return min(a, b, c);
end
```

3.4 Final Combination

The final score $score(n)$ for each candidate node n extracted as explained in Section 3.2 is evaluated through the combination of the previously normalized scores $\in [0, 1]$. Then the elements corresponding to the target nodes are filtered and ranked. Formally, with $\lambda \in [0, 1]$:

$$score(n) = \lambda \times c(n) + (1 - \lambda) \times s(n). \tag{4}$$

4 Experiments and Evaluation

We evaluated our approach on both summarized and unsummarized subtrees on the INEX 2005 collection and compared our results with the official participants.

4.1 INEX Collection

INEX (*Initiative for the Evaluation of XML Retrieval*) is the reference evaluation campaign for XML retrieval. To evaluate our approach we used the 2005 collection which is composed of 16000 XML documents from the IEEE Computer Society scientific papers. These documents have an average of 1500 elements for a hierarchical depth of 6.9.

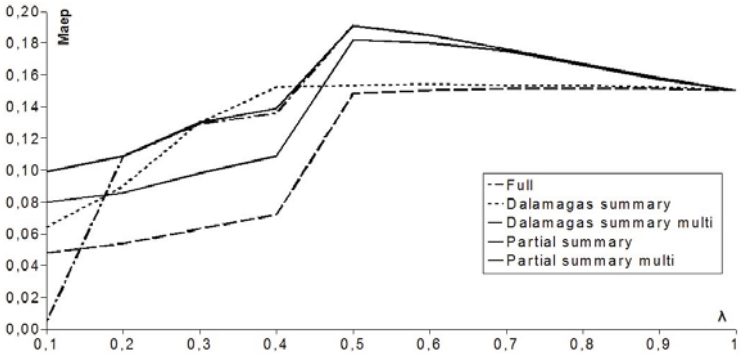

Fig. 4. MAeP score evolution over λ parameter variation for content and structure

Two main types of queries are available, namely *Content Only* (CO) and *Content And Structure* (CAS). Tasks using CAS queries are centered on structural constraints and were not reconducted in later campaigns[3]. Four subtasks were proposed. To evaluate queries in which structural constraints are semantically relevant, we use in our experiments the SSCAS subtask, in which constraints are strict on the target element and its environment.
There are two measures for the CAS subtasks [12]: *Non-interpolated mean average effort-precision* (MAeP) which is used to average the effort-precision measure at each rank and *Normalized cumulated gain* (nxCG). This last one corresponds to the cumulative gain at a threshold and is based on the ideal ranking over the sum of all score to that threshold.
Finally we will use the "strict" quantization to aggregate specificity and exhaustivity relevance judgments as it only takes into account fully relevant elements.

4.2 Experiments

We run our algorithm with four summary versions of document and query trees. Our baseline is unsummarized version (*full*). Regarding the summary versions, *dalamagas summary* corresponds to the run for the exact summary rules while in *partial summary* nodes containing tags from the query are not summarized. The *multi* extension for the both previous versions are for the cases in which we keep a record of the number of removed nodes as a coefficient in the edit distance. We set the removing cost $c_{del}()$ to 0.5 for a label in the query and 1 otherwise and the relabeling cost $c_{match}()$ to 0 for similar tags (eg: *section* and *subsection* are considered as semantically similar) and 1 otherwise.

Figure 4 shows the results for the *MAeP* metric depending on the λ parameter of equation (4). The best tuning for our system is around 0.5 which means an

[3] Since 2010 a new task called *Datacentric* appeared. Centered on rich structure it is based on the IMDB database which contains around 1 590 000 documents about movies and several millions about actors, producers, etc. Our next experimentations will of course use this collection.

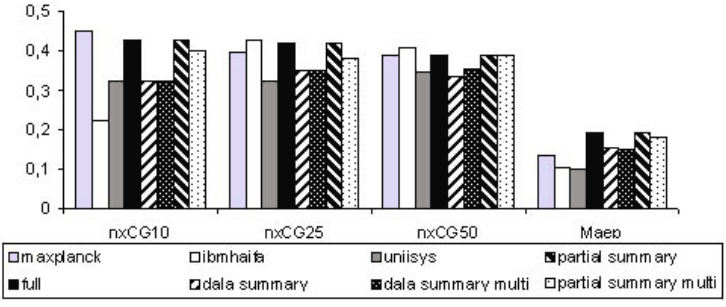

Fig. 5. Results for $\lambda = 0.5$ compared to INEX 2005 participants

equal part of content and structure. However, if all of our solutions return better results around the same value of λ it is clear that two of them are well over the others (more than 80% for the *MaeP*): our full tree algorithm and the *partial summary* algorithm in which nodes that have tags equivalent to the query ones are not summarized. We further notice that the results are equivalent for these two cases which means that we can gain on runtime while keeping the score of our intial solution. Moreover, keeping track of the removed nodes does not improve our results. Even if it seems natural for the *dalamagas summary* in which the tree structure is strongly altered through the removal of duplicate node descendants, it seems more surprising for our *partial summary*. It can however be explained by the fact that it artificially increases the number of nodes for a substitution score equal to 0. It means that it gives the same score for a summarized tree with several noisy children nodes than for a subtree with only the correct number of relevant nodes.Considering now efficiency, *Dalamagas summary* reduces the number of nodes to 48%, while our *partial summary* version reduces their number to 45%. Regarding the running time, it is improved 20 times on the average.

Figure 5 shows the results for $\lambda = 0.5$ over the various INEX 2005 metrics compared to the best participants, namely the Max Planck institute with TopX [19] a database-centered approach; IBM Haifa Research Lab [15] with a vector space model and the University of Klagenfurt [10] which also uses a vector space model. While our *full tree* and *partial summary* runs score slightly the same than the first participants for the nxCG (an average of 5% under the first while 13% over the second), our *MAeP* results are overall better (+45 % compared to the first one and +90% for the second) for all our runs.

4.3 Conclusions and Future Work

In this paper we presented an XML retrieval approach whose main originality is to use tree edit distance. This solution allowed us to outperform other approaches on the *MAeP* measure by 45% while proving the usefulness of structural information in SIR process. However its main drawback, i.e. complexity, was time consuming on the runs. We overcame this issue with a new set of summary rules

which allowed us to keep the effectiveness of our solution while improving efficiency though search space reduction. In future work we will improve our content score as well as the overall edit distance cost system in order to better use the tag semantics. Finally we are currently working on the INEX 2010 *Datacentric* collection which will allow us to confirm our results on a bigger and semantically richer collection.

References

1. Alilaouar, A., Sedes, F.: Fuzzy querying of XML documents. In: International Conference on Web Intelligence and Intelligent Agent Technology, Compigne, France, pp. 11–14. IEEE/WIC/ACM (September 2005)
2. Ben Aouicha, M., Tmar, M., Boughanem, M.: Flexible document-query matching based on a probabilistic content and structure score combination. In: Symposium on Applied Computing (SAC), Sierre, Switzerland. ACM (March 2010)
3. Boobna, U., de Rougemont, M.: Correctors for XML Data. In: Bellahsène, Z., Milo, T., Rys, M., Suciu, D., Unland, R. (eds.) XSym 2004. LNCS, vol. 3186, pp. 97–111. Springer, Heidelberg (2004)
4. Dalamagas, T., Cheng, T., Winkel, K.-J., Sellis, T.: A methodology for clustering XML documents by structure. Information Systems 31, 187–228 (2006)
5. Damiani, E., Tanca, L., Arcelli, F.: Fuzzy XML queries via context-based choice of aggregation. Kybernetika 36, 635–655 (2000)
6. Demaine, E.D., Mozes, S., Rossman, B., Weimann, O.: An optimal decomposition algorithm for tree edit distance. ACM Trans. Algorithms 6, 2:1–2:19 (2009)
7. Demartin, G., Denoyer, L., et al.: Report on INEX 2008. SIGIR Forum 43, 17–36 (2009)
8. Dulucq, S., Touzet, H.: Analysis of tree edit distance algorithms. In: Proceedings of the 14th Annual Symposium of Combinatorial Pattern Matching, pp. 83–95 (2003)
9. Geva, S., Kamps, J., Lethonen, M., Schenkel, R., Thom, J.A., Trotman, A.: Overview of the INEX 2009 Ad Hoc Track. In: Geva, S., Kamps, J., Trotman, A. (eds.) INEX 2009. LNCS, vol. 6203, pp. 4–25. Springer, Heidelberg (2010)
10. Hassler, M., Bouchachia, A.: Searching XML Documents – Preliminary Work. In: Fuhr, N., Lalmas, M., Malik, S., Kazai, G. (eds.) INEX 2005. LNCS, vol. 3977, pp. 119–133. Springer, Heidelberg (2006)
11. Sparck Jones, K.: Index term weighting. Information Storage and Retrieval 9(11), 619–633 (1973)
12. Kazai, G., Lalmas, M.: INEX 2005 Evaluation Measures. In: Fuhr, N., Lalmas, M., Malik, S., Kazai, G. (eds.) INEX 2005. LNCS, vol. 3977, pp. 16–29. Springer, Heidelberg (2006)
13. Klein, P.N.: Computing the Edit-Distance Between Unrooted Ordered Trees. In: Bilardi, G., Pietracaprina, A., Italiano, G.F., Pucci, G. (eds.) ESA 1998. LNCS, vol. 1461, pp. 91–102. Springer, Heidelberg (1998)
14. Levenshtein, V.I.: Binary Codes Capable of Correcting Deletions, Insertions and Reversals. Soviet Physics Doklady 10, 707 (1966)
15. Mass, Y., Mandelbrod, M.: Component Ranking and Automatic query Refinement for XML Retrieval. In: Fuhr, N., Lalmas, M., Malik, S., Szlávik, Z. (eds.) INEX 2004. LNCS, vol. 3493, pp. 73–84. Springer, Heidelberg (2005)

16. Popovici, E., Ménier, G., Marteau, P.-F.: SIRIUS: A Lightweight XML Indexing and Approximate Search System at INEX 2005. In: Fuhr, N., Lalmas, M., Malik, S., Kazai, G. (eds.) INEX 2005. LNCS, vol. 3977, pp. 321–335. Springer, Heidelberg (2006)
17. Rougemont, M., Vieilleribière, A.: Approximate schemas, source-consistency and query answering. J. Intell. Inf. Syst. 31, 127–146 (2008)
18. Tai, K.-C.: The tree-to-tree correction problem. J. ACM 26, 422–433 (1979)
19. Theobald, M., Schenkel, R., Weikum, G.: TopX and XXL at INEX 2005. In: Fuhr, N., Lalmas, M., Malik, S., Kazai, G. (eds.) INEX 2005. LNCS, vol. 3977, pp. 282–295. Springer, Heidelberg (2006)
20. Trotman, A., Sigurbjörnsson, B.: Narrowed Extended XPath I (NEXI). In: Fuhr, N., Lalmas, M., Malik, S., Szlávik, Z. (eds.) INEX 2004. LNCS, vol. 3493, pp. 16–40. Springer, Heidelberg (2005)