Edit Distance for XML Information Retrieval : Some Experiments on the Datacentric Track of INEX 2011

Cyril Laitang, Karen Pinel-Sauvagnat, and Mohand Boughanem

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

Abstract. In this paper we present our structured information retrieval model based on subgraphs similarity. Our approach combines a content propagation technique which handles sibling relationships with a document query matching process on structure. The latter is based on tree edit distance (TED) which is the minimum set of insert, delete, and replace operations to turn one tree to another. As the effectiveness of TED relies both on the input tree and the edit costs, we experimented various subtree extraction techniques as well as different costs based on the DTD associated to the Datacentric collection.

1 Introduction

XML documents can be naturally represented through trees in which nodes are elements and edges hierarchical dependencies. Similarly structural constraints of CAS queries can be expressed though trees. Based on these common representations we present a SIR model using both graph theory and content scoring.
This paper is organized as follows: Section 2 presents work related to the different steps of our approach; Section 3 presents our approaches and finally Section 4 discusses the results obtained in the Datacentric track of INEX 2011.

2 Related Works

As our algorithm is based on the structure of documents we will first overview document structure extraction techniques. We will then give a brief survey on tree-edit distance algorithms.

2.1 Document Structure Representation and Extraction

In the literature we identify two families of approaches regarding how to handle document structure regardless of content. The first one is relaxation [1] [3] [5]. In these approaches, the main structure is atomized into a set of node-node relationships. The weight of these relationships is then the distance between nodes in the original structure. The second family is linked to subtree extraction.

The *lowest common ancestor* (LCA) is the tree rooted by the first common ancestor of two or more selected nodes [4]. In IR it aims at scoring the structure by finding the subtrees in which all the leaves contain at least on term of the query [2].

2.2 Edit distance

Two graphs are isomorphic if they share the same nodes and edges. Finding a degree of isomorphism between two structures is called approximate tree matching. There are three main families of approximate tree matching: *inclusion*, *alignment* and *edit distance*. We choose to use the latter as it offers the most general application. The original tree edit distance (TED) algorithms [14] generalize Levenshtein *edit distance*[10] to trees. The similarity is the minimal set of operations (adding, removing and relabeling) to turn one tree to another. Later, Klein et al. [9] reduced the overall complexity in time and space by splitting the tree structure based on the heavy path (defined in Section 3.2). Finally Touzet et al. [6] used a decomposition strategy to dynamically select the best nodes to recurse on between rightmost and leftmost. Regarding the costs, a common practice is to use *apriori* fixed costs for the primitive operations [11]. However as these costs impact the isomorphism evaluation one can find some approaches that try to estimate these costs. Most of the non-deterministic approaches are based on learning and training techniques [13] [12].

3 Tree-Edit Distance for Structural Document-Query Matching

In our approach the document-query similarity is evaluated by scoring content and structure separately. We then combine these scores to rank relevant elements. In this section, we first describe the content evaluation and then 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 [8]) to score the document leaf nodes according to query terms contained in content conditions. We propose two approaches. In the first one (which we call *Vague*) content parts of the query are merged and the content score is evaluated in a one time pass. In our second approach which we define as *Strict*, the content conditions are considered separately and summed at the end of the process.

Our propagation algorithm is based on the idea that the score of an inner node must depend on three elements. First, it must contain its leaves relevance (this is what we call the intermediate score). Second we should score higher a node located near a relevant element. Finally, the node score must depend on the score of its ancestors. 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:

$$c(n) = \begin{cases} \underbrace{\frac{p(n)}{\mid leaves(n) \mid}}_{(i)} + \underbrace{\frac{p(a_1) - p(n)}{\mid leaves(a_1) \mid}}_{(ii)} + \underbrace{\frac{c(a_1) - \frac{p(a_1)}{\mid leaves(a_1) \mid}}{\mid children(a_1) \mid}}_{(iii)} & \text{if } n \neq root \\ \overbrace{\frac{p(n)}{\mid leaves(n) \mid}} & \text{otherwise} \end{cases} \quad (1)$$

(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*, evaluated with $c(a_1)$ the final score of the father a_1 minus its intermediate score.

3.2 Structure Relevance Score Evaluation

Our structural evaluation process follows three steps : subtree selection and extraction, structure score through tree-edit distance, and the score combination. As the final part is strongly related to the type of subtree extracted we will postpone our explanations subtree extraction to the end of this section.

Optimal Path of Tree-Edit Distance As seen in section 2.2, the TED measures 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. [6]. The difference is that the optimal path is computed with the help of the *heavy path* introduced by Klein et al. [9]. The heavy path is the path from root to leaf which passes through the rooted subtrees with the maximal cardinality. Selecting always the most distant node from this path allows us to create the minimal set of subtrees in memory during the recursion : this is the *optimal cover strategy*. Formally a heavy path is defined as a set of nodes $[n_1, ..., n_z]$ with $T(x)$ the rooted tree in x, satisfying:

$$\forall (n_i, n_{i+1}) \in heavy \begin{cases} n_{i+1} \in children(n_i) \\ \forall x \in children(n_i), x \notin n_{i+1}, \mid T(n_{i+1}) \mid \geq \mid T(x) \mid \end{cases} \quad (2)$$

This strategy is used on the document and the query as input of our following TED algorithm (Algorithm 1) in which F, G are two forests (i.e. the document and the query as first input), 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.

Algorithm 1: Tree-Edit Distance using Optimal Paths

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

Tree-edit distance costs evaluation TED operation costs are generally set to 1 for removing, to 0 for relabeling similar tags and to 1 otherwise [14] which is sufficient for evaluating relatively similar trees. However in our approach document trees are larger than query trees which means that the edit costs must be less discriminative. There is two constraints in estimating these costs. First, as relabeling is equivalent to removing and then adding a node, its cost should be at most equivalent to two removings. Second, we need to reduce the estimation of these costs to the minimum computation cost. For these reasons we propose to use the DTD of the considered collection to create an undirected graph representing all the possible transitions between elements. The idea behind is that the lower degree a node have the less its cost must be. As the Datacentric collection comes up with two distinct DTDs (respectively *movie* and *person*) we choose to create three graphs : one for each DTD and a last one merged on the labels equivalent in the two. In order to process the substitution cost $c_{match}(n_1, n_2)$ of a node n_1 by a node n_2, respectively associated with the tags t_1 and t_2, we seek the shortest path in these DTD graphs through a Floyd-Warshall [7] algorithm which allows to overcome the cycle issues. We divide this distance by the longest of all the shortest paths that can be computed from this node label to any of the other tags in the DTD graph. Formally, with $sp()$ our shortest path algorithm :

$$c_{match}(n_1, n_2) = \frac{sp(t_1, t_2)}{max(sp(t_1, t_x))} \forall x \in DTD \tag{3}$$

The removing cost is the highest cost obtained from all the substitution costs between the current document node and all of the query nodes :

$$c_{del}(n_1) = max(\frac{sp(t_1, t_y)}{max(sp(t_1, t_x))}) \forall x \in DTD; \forall y \in Q \quad (4)$$

Subtree Extraction and Evaluation In our *Strict* model we use the minimal subtree representing all the relevant nodes labeled with a label contained in the query as input for the matching process. In our *Vague* algorithm we extract all the subtrees rooted from the first node with a label matching a label in the query to the documents root. Formally, for the *Vague* approach, with $Anc(n)$ the set of n ancestors; $a \in Anc(n)$; $T(a)$ the subtree rooted in a; $d(T(a), Q)$ the TED between $T(a)$ and Q, the structure score $s(n)$ is :

$$s(n) = \frac{\sum_{a \in \{n, Anc(n)\}} (1 - \frac{d(T(a),Q)}{|T(a)|})}{\mid Anc(n) \mid} \quad (5)$$

The idea behind this extraction is that a node located near another one matching the structural constraint should get an improvement to its score.

For our *Strict* algorithm the subtree S is created from the combination of all the paths from the deepest relevant nodes which contain a label of the query to the higher in the hierarchy. The subtree is then the merged paths rooted by a node having the same label than the query root. Formally it is composed of all the nodes a extracted as $\{a \in G \mid a \in \{n, Anc(n)\}, \forall n \in leaves \wedge p(n) \neq 0\}$

As we apply one TED for all the nodes in the created subtree, the final score is then :

$$s(n) = \frac{d(S, Q)}{\mid S \mid} \quad (6)$$

3.3 Final Structure and Content Combination

For both models, the final score $score(n)$ for each candidate node n is evaluated through the linear combination of the previously normalized scores $\in [0, 1]$. Formally, with $\lambda \in [0, 1]$:

$$score(n) = \lambda \times c(n) + (1 - \lambda) \times s(n). \quad (7)$$

4 Experiments and Evaluation

In order to evaluate both the efficiency of our algorithms as well as the usefulness of the DTD based costs we run various combinations of our *Strict* and *Vague* algorithms. These runs are *with split DTD* in which we used the three DTD graphs; *with no DTD* for our solution in which the TED operation costs are fixed to 1 for removing a node not in the query, 0.5 for a node with a tag in the query, 0 for a relabeling of one node with another if their label are equivalent and 1 otherwise. λ parameter from the equation (7) is set to 0.4 for *Strict* approachand 0.6 for the *Vague* one. Finally *baseline* are the runs in which we only use the content part ($\lambda = 1$). As the task is to rank documents, and as our method retrieve elements, we decided to score documents as the score of

their best element. Finally it is important to notice the presented runs are not the official ones as the submitted runs for the INEX 2011 Datacentric track were launched over a corrupted index missing around 35% of the documents. Results are presented in table 1. Our *Strict* algorithm scores overall better that the *Vague* version. It tends to demonstrate that using the content location over the structure improves the results. However our TED structure scoring process doesn't seem to improve the search process even if the score tend to drop less with the DTD costs than with the empirically set costs for our *Strict* algorithm. Regarding the *Vague* version we cannot conclude are the results are very similar between our three versions.

Runs	**MAP**	**P@5**	**P@10**	**P@20**	**P@30**
Strict with split DTD	0.1636	**0.2667**	0.2361	**0.2208**	0.2213
Strict with no DTD	0.125	0.2167	0.1917	0.2069	0.2185
Strict baseline	**0.1756**	0.25	**0.2389**	0.2125	**0.23**
Vague with split DTD	0.105	0.1842	0.1789	0.1605	0.1439
Vague with no DTD	0.1083	0.1895	0.1658	0.1421	0.1289
Vague baseline	0.1104	0.1737	0.1579	0.1382	0.1351

Table 1. Our corrected INEX 2011 results with λ set to 0.4 for our *Strict* method and 0.6 for our *Vague* approach compared to our baselines with no use of the DTD.

Queries of the Datacentric track were of four types [15]. These types are *Known-item* in which the aim is to retrieve a particular document; *Informational* in which the user tries to find information about a subject or event; *List* for which the aim is to retrieve a list of documents matching a particular subject; and finally *Others* for the queries not listed in one of the previous categories. Results against these types are shown in table 2. It appears that the ranking between the different versions of our algorithm stays the same. However we notice that our *Strict* algorithm scores significantly for the *Know-item* queries. As these types of queries are particularly focused on finding only a few relevant documents this tends to demonstrate that our strict algorithm is more relevant in the context of a focus search process.

Runs	**Known-item**	**Informational**	**List**	**Others**
Strict with split DTD	0.3439	0.1082	0.1473	**0.1122**
Strict with no DTD	0.1796	0.1093	0.1043	0.1097
Strict Baseline	**0.3637**	**0.1416**	**0.1646**	0.1002
Vague with split DTD	0.1498	0.0262	0.138	0.086
Vague no DTD	0.1978	0.0254	0.1404	0.0695
Vague baseline	0.1848	0.027	0.1421	0.0772

Table 2. Our corrected INEX 2011 results for the four different types of queries.

4.1 Conclusions and Future Work

In this paper we presented two XML retrieval models whose main originality is to use graph theory through tree edit distance (TED). We proposed a way

of estimating the TED operation costs based on the DTD. It appears that the use of the DTD as well as the edit distance doesn't improve our results for this particular track. In future work we plan to modify our final scoring formula to score documents. Then we will update our leave scoring process by using a more up-to date algorithm such as a language model. Finally we will conduct future studies on Datacentric 2010 to determine if our DTD based edit distance system work betters on element retrieval than it does for document retrieval.

References

1. A. Alilaouar and F. Sedes. Fuzzy querying of XML documents. In *International Conference on Web Intelligence and Intelligent Agent Technology, Compigne, France*, pages 11–14. IEEE/WIC/ACM, septembre 2005.
2. Evandrino G. Barros, Mirella M. Moro, and Alberto H. F. Laender. An Evaluation Study of Search Algorithms for XML Streams. *JIDM*, 1(3):487–502, 2010.
3. M. Ben Aouicha, M. Tmar, and M. Boughanem. Flexible document-query matching based on a probabilistic content and structure score combination. In *Symposium on Applied Computing (SAC), Sierre, Switzerland*. ACM, mars 2010.
4. Michael A. Bender and Martin Farach-Colton. The lca problem revisited. In *Proceedings of the 4th Latin American Symposium on Theoretical Informatics*, LATIN '00, pages 88–94, London, UK, 2000. Springer-Verlag.
5. E. Damiani, B. Oliboni, and L. Tanca. Fuzzy techniques for XML data smushing. In *Proceedings of the International Conference, 7th Fuzzy Days on Computational Intelligence, Theory and Applications*, pages 637–652, London, UK, 2001. Springer-Verlag.
6. S. Dulucq and H. Touzet. Analysis of tree edit distance algorithms. In *Proceedings of the 14th annual symposium of combinatorial pattern matching*, pages 83–95, 2003.
7. Robert W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5:345–, June 1962.
8. K. Sparck Jones. Index term weighting. *Information Storage and Retrieval*, 9(11):619–633, 1973.
9. P N. Klein. Computing the edit-distance between unrooted ordered trees. In *Proceedings of the 6th Annual European Symposium on Algorithms*, ESA '98, pages 91–102, London, UK, 1998. Springer-Verlag.
10. VI Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707, 1966.
11. Y. Mehdad. Automatic cost estimation for tree edit distance using particle swarm optimization. In *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, ACLShort '09, pages 289–292, 2009.
12. M. Neuhaus and H. Bunke. Automatic learning of cost functions for graph edit distance. *Information Science*, 177(1):239–247, 2007.
13. J. Oncina and M. Sebban. Learning stochastic edit distance: Application in handwritten character recognition. *Pattern Recogn.*, 39:1575–1587, September 2006.
14. K-C. Tai. The tree-to-tree correction problem. *J. ACM*, 26:422–433, July 1979.
15. Qiuyue Wang, Georgina Ramírez, Maarten Marx, Martin Theobald, and Jaap Kamps. Overview of the INEX 2011 data centric track. In Shlomo Geva, Jaap Kamps, and Ralf Schenkel, editors, *INEX 2011 Workshop Pre-proceedings*, pages 88–106, 2011.