Nearest Neighbour Search For XML Trees

László Kovács, Tibor Répási, Erika Baksa-Varga

kovacs@iit.uni-miskolc.hu, repasi@iit.uni-miskolc.hu, iitev@uni-miskolc.hu

Abstract: There is a significant research effort on efficient computing of similarities between objects of non traditional data types as strings, documents, sound tracks or pictures. It is reasonable to use the results of these efforts in the problem of XML tree matching, too. As an XML document has a tree structure and the trees can be transformed into a linear structure, a tree can be regarded as a special kind of string. In our approach, the results of string comparison will be extended to measure the similarity between XML trees.

Keywords: Tree edit distance, Nearest neighbour searching

1 Introduction to XML and XPath

The Extensible Markup Language (XML) is a simple, very flexible text format derived from SGML (Standard Generalized Markup Language) [1]. XML provides a way of describing data in a rich, flexible and efficient way by marking up data with descriptive tags. However, XML does not provide a way to locate specific pieces of structured data within a given document. Instead, the XML Path Language (XPath) provides a syntax for locating specific parts of an XML document effectively and efficiently. It gets its name from its use of a path notation as in URLs for navigating through the hierarchical structure of an XML document. In XPath, an XML document is viewed conceptually as a tree in which each part of the document is represented as a node [2]. The standard XPath model assumes random access to XML data and also the ability to query XML data at any time.

XPath is not a structural language like XML; rather, it is a string-based language of expressions. An expression is evaluated to yield an object, which has one of the following four basic types: node-set (an unordered collection of nodes without duplicates); boolean; number or string. XPath expressions often occur in XML attributes. Expression evaluation occurs with respect to a context. The context consists of: a node (the context node), a pair of non-zero positive integers (the context position and the context size), a set of variable bindings, a function library and the set of namespace declarations in scope for the expression [3].

An important kind of expression is a location path. There are two kinds of location paths: relative location paths and absolute location paths. (1) A relative location path consists of a sequence of one or more location steps separated by /, and each step in turn selects a set of nodes relative to the context node. A location step has three parts: an axis, which specifies the tree relationship between the nodes selected by the location step and the context node; a node test, which specifies the node type and expanded-name of the nodes selected by the location step; and zero or more predicates, which use arbitrary expressions to further refine the set of nodes selected by the location step. The node-set selected by the location step is the node-set that results from generating an initial node-set from the axis and node-test, and then filtering that node-set by each of the predicates in turn. (2) An absolute location path consists of / optionally followed by a relative location path. A / by itself selects the root node of the document containing the context node. If it is followed by a relative location path, then the location path selects the set of nodes that would be selected by the relative location path relative to the root node of the document containing the context node [3].

Thus a location path selects a set of nodes. The result of evaluating an expression that is a location path is the node-set containing the nodes selected by the location path. The initial node-set consists of the nodes having the relationship to the context node specified by the axis, and having the node type and expanded-name specified by the node test. The initial node-set is filtered by the first predicate to generate a new node-set; this new node-set is then filtered using the second predicate, and so on. The final node-set is the node-set selected by the location step. The axis affects how the expression in each predicate is evaluated and so the semantics of a predicate is defined with respect to an axis. Every axis has a principal node type. For the attribute axis, the principal node type is attribute; for the namespace axis, the principal node type is namespace; for other axes, the principal node type is element [3].

The disadvantage of XPath is that it is based on partial matching. It means that some parts of the tree must match the given pattern exactly, while others are totally ignored. The resulted measure does not take into account the deviation from the pattern. However, this deviation value would carry useful information for users. Our aim is to find an efficient algorithm that measures the difference between the pattern and the selected subtrees. Therefore allowing searching for nearest neighbour approximations of the pattern XML document.

2 The distance between two XML trees

The problem of object matching is an interdisciplinar area as it collects methods from three current disciplines: computer science, statistics and operations research. There is a significant research effort on efficient computing of

similarities between objects of non traditional data types as strings, documents, sound tracks or pictures. It is reasonable to use the results of these efforts in the problem of XML tree matching, too. As an XML document has a tree structure and the trees can be transformed into a linear structure, a tree can be regarded as a special kind of string. In our approach, the results of string comparison will be extended to measure the similarity between XML trees. The strings and XML trees can be considered as objects located in metric space. A space *X* is metric if there is a real non-negative function of two objects *d(A, B)* defined. The function is known as the distance between the two points. It is characterized by the following properties. For *AX*, *BX*, and *CX*

1. d(A, B) = 0 if and only if A = B (the distance is 0 if and only if the points coincide);
2. d(A, B) = d(B, A) (the distance from A to B is the same as the distance from B to A);
3. d(A,B) + d(B,C) > d(A,C) (the sum of two sides of a triangle is never less than the third side).

The best known string matching algorithms in computer science are the Russel Soundex coding and the Levenshtein edit distance methods. The Soundex method is used in record linkage methods mainly for blocking the master record set. The Levenshtein edit distance is defined as the smallest number of insertions, deletions, and substitutions required to change one string or tree into another. The efficiency of the algorithm to compute the distance between strings is *O(mn)*, where *m* and *n* are the lengths of the strings. The Levenshtein distance is defined for strings of arbitrary length. It counts the differences between two strings, where we would count a difference not only when strings have different characters but also when one has a character whereas the other does not.

In general, the transformation of strings can be performed between strings of different character sets. Thus, the input parameter for string distance problem is a triple *(A,B,c)*, where *A* is the character set of the input string, *B* is the character set of the output string, and *c* is the cost function. The transformation may usually consists of the following elementary operations: insertion, deletion and substitution. There are several variants for elementary cost functions, among which the simplest one is the unit-cost function. In this case the insertion, the deletion of a single character has the value of 1, and the substitution of two different characters is equal to 1, too.

For a string *s*, let *s(i)* stand for its i^{th} character. For two characters *a* and *b*, define

c(0,a) = 1 (insertion)
c(a,0) = 1 (deletion)
c(a, b) = 0 if a = b; 1, otherwise (substitution)

Assume we are given two strings *s* and *t* of length *n* and *m*, respectively. The length value is denoted by an upper index: s^n, $t^{m.}$. The distance, i.e. the minimal

cost of transformation from *s* to *t* is usually calculated with using a dynamic programming algorithm. The distance value is defined with a recursive formula:

$$d(s^n,t^m) = \min \begin{cases} c(s_n,t_m) + d(s^{n-1},t^{m-1}) \\ c(s_n,0) + d(s^{n-1},t^m) \\ c(0,t_m) + d(s^n,t^{m-1}) \end{cases}$$

where s_i denotes the charater at the position *i* in string *s*.

Basically, edit distance can be computed in $O(n\ m)$ or $O(n\ m\ /\ log(n))$ time if the cost weights are rational [7]. In a lot of applications, a more precise distance value can be achieved with the normalized edit distance measure. This measure takes not only the number of elementary transformations into account, but also the length of the input and output strings. The normalized edit distance is the ratio of the total weight to the length of the sequence. The complexity of the normalized edit distance is $O(n\ m\ log(n))$ [7].

The distance value for trees can be defined similarly to the distance of strings. An important assumption in our investigation is that the labels of the trees are ordered. This assumption corresponds to the XML standard. Usually, like in [4], the following elementary erations are defined for tree objects:

- relabel: assigns a new node name to the root of the tree
- insert: inserting a new node into the children of the root node
- delete: deleting a node from the children of the root node
- insert tree: inserting a tree under the root node
- deleting tree: deleting a tree from the children of the node

In our approach, the tree insertion or tree deletion operators are replaced with a list of single node operations. According to [7], the tree distance value can be calculated using the following recursive formula:

$$\begin{aligned} &d(0,0) = 0 \\ &d(F,0) = && d(F\text{-}v,0) + c(v,0) \\ &d(0,F) = && d(0,F\text{-}v) + c(0,v) \\ &d(F_1,F_2) = \min\{ && d(F_1\text{-}v,F_2) + c(T(v),0) \\ & && d(F_1,F_2\text{-}v) + c(0,T(v)) \\ & && d(F_1\text{-}T(v),F_2\text{-}T(w)) + c(T(v),T(w)) \end{aligned}$$

where *F* denotes a tree and $T(v)$ denotes a tree with root element *v*.

The ordered tree edit distance problem was deeply investigated by Tai [9]. In that proposal a distance calculation algorithm with $O(|F_1|\ |F_2|\ [L_1|^2\ |L_2|^2)$ complexity was presented. In the literature, we can find some improvements proposed in the past years for this kind of problem.

The most recent one is the algorithm of Chen [8] with a complexity of $O(|F_1|\ |F_2| + [L_1|^2\ |F_1| + |L_1|^{2.5}|L_2|)$. The $|F_i|$ symbol denotes the number of nodes in the tree F_i. The $|L_i|$ is a symbol for the number of leaves in the tree. The problem area of tree edit distance calculation is analyzed very deeply in the last decades, so there

exist some efficient algorithms for comparing two XML trees or documents. In our approach, the tree distance calculation algorithm works as follows:

```
DistF(F1, F2) {
        int M1 = number of children nodes of the root in F1
        int M2 = number of children nodes of the root in F2
        int dist = new dist[M1][M2]
        dist[0][0] = c(root(F1), root(F2))
        for (i =1; i<M1;i++)  dist[i][0] = dist[i-1][0] + distF(0, F2^i)
                for (i =1; i<M2;i++)  dist[0][i] = dist[0][i-1] + distF(F1^i,0)
                        for (i =1; i<M1;i++)  {
                                for (j =1; j<M2;j++) {
                                        dist[i][j] = min (dist[i-1][j-1] +
                                        distF(F1^i,F2^j), dist[i][j-1] +
                                        distF(0,F2^j), dist[i-1][j] + distF(F1^i,0))
                                }
                        }
        return dist[M1][M2]
}
```

where F_i^j denotes the j^{th} node in tree indexed by *i*.

The main goal of our investigation is not to compare two XML trees, but to find an XML subtree in a base tree with the best matching to an input sample tree. For example, the base XML document may be contained by the Reuters articles and the sample XML tree is a fragment of an article. The user may want to find articles with similar structure or similar content. In this case, a best matching subtree should be selected. The most naive approach is to compare the sample tree with every subtree of the base tree. This means that the tree distance routine should be repeated $|F_1|$ times, if F_1 is the base tree and F_2 is the sample tree. The rough estimation of the total cost is equal to $O(|F_1|^2 |F_2|)$. In our investigation, we have focused on the case when the base table is a stabile, fixed tree. This is true for a lot of information systems where the base tables are updated very rarely. In these cases, some kind of pre-computation can be performed to reduce the cost of incoming, on-line comparisons. The main ideas for cost reduction can be summarized as follows:

- building a VP- tree for the subtrees in the base tree;
- introducing a multi-level distance measure with a low-cost distance approximation layer and a high-cost exact distance layer.

The VP- tree is a search tree for objects in metric space. The subtrees of the base tree are considered as the objects in the metric space. The distance between the subtree objects is defined as the tree edit distance measure given previously. The VP tree can be generated in a recursive way. At each level, the tested subtree objects are separated into two or more groups, based on the distance value to a

pre-selected object (called ventage object). The generated groups may be divided into subgroups at lower levels.

The low-cost distance approximation is based on a cheksum function. Each subtree is assigned to a vector value. The applied distance function is based on the n-gram distance for strings. In this approach, the vector value for bigrams is calculated as follows. First an integer code value is assigned to every node, based on the label of the node. Then every parent-child node is counted. The result is a vector, where a dimension corresponds to a parent-child relation and the dimension value corresponds to the count value. Thus every tree is mapped to a vector. If the two trees are the same, the vectors are the same, too. Similar trees have similar vector representations. The comparison of the two vectors has a significantly lower cost than the edit distance computation.

3 Applicability in search engines

Some news portals are using the NewsML standard for representing news stories. The NewsML standard was created as a standard format for news stories capable for covering the needs of data and meta-data storage over the whole lifecycle of the documents. The standard itself, especially the system of topic indexes was invented to support the most common type of search engines answering keyword-based queries. These search engines are extremely efficient if the query fits the consideration of the original indexing. However, the human intellect has often the need of more information about the topic of the recently read article. Subsequently this leads to a query like "Show me similar articles!". Passing the same keywords to a search engine found in an article, most probably the result set will contain the same article. This is because the categorization of articles is subjective to the opportunity of writing the article.

A search engine based on the proposed method will be able to find articles which are simmilar to the currently read article or a part of it. The query will be a part of a structured document, some sentences, paragraphs or the whole document. Representing the document as a graph or tree of expressions which consists of homographs will create the possibility of selecting a part of a document i.e. a subtree and searching for similar subtrees in a set of documents. The search engine will try to find any document with parts most similar to the structure of the recent document, e.g. reading a story about a storm in Florida can raise the interests on weather phenomena, but searching in the same topic will only lead to articles about storms in Florida. Only the search for similar document parts can lead to documents about other weather phenomena independent from Florida.

An other disadvantage of strict keyword based searching is that keywords and expressions used in journalism may change over the decades or centuries. Only

the analysis of the document structure can help to find documents according to a sample document, e.g. searching for scientific publications in quantum physics about the "state matrix of particles" is not as obvious as it seems to be, since the expression for state matrix was "energy table" 4 decades ago. Documents in the same topic from that time will have totally different keywords, and only the analysis of the structure and context will result in similar documents to the searched topic.

3 Conclusion

The main goal of our investigation was not to compare two XML trees, but to find an XML subtree in a base tree with the best matching to an input sample tree. In our investigation, we have focused on the case when the base table is a stabile, fixed tree. Using the VP-tree and distance approximation methods every subtree is mapped to a vector. If the two subtrees are the same, the vectors are identical, too. Similar trees have similar vector representations. The comparison of the two vectors has a significantly lower cost than the edit distance computation.

References

[1] Extensible Markup Language (XML), W3C Recommendation 04 February 2004, http://www.w3.org/TR/2004/REC-xml11-20040204/

[2] Deitel,Deitel,Nieto,Lin & Sadhu: XML How to Program, Prentice Hall 2001

[3] XML Path Language (XPath) Version 1.0 W3C Recommendation 16 November 1999, http://www.w3.org/TR/xpath

[4] Nierman, Jagadish: Evaluating Structural Similarity in XML Documents, Proceedings of the Fifth International Workshop on the Web and Databases (WebDB 2002)

[5] Arslan, Egecioglu: An Efficient Uniform-Cost Normalized Edit Distance Algorithm, Proc. 6-th String Processing and Information REtrieval Conference (SPIRE'99), IEEE Computer Society, Sep. 1999, Cancun, Mexico, pp. 8-15.

[6] Shasha, Wang, Giuno: Algorithms and Applications of Tree and Graph Searching, In Proc. PODS'02, 2002, pp 39-52

[7] Bille: Tree Edit Distance, Alignment Distance and Inclusion, IT University Technical Reports, Copenhagen, 2003, ISSN 1600-6100

[8] Chen: New Algorithm for ordered tree-to-tree correction problem, Journal of Algorithms, 2001, pp 40:135-158

[9] David T. Barnard, Gwen Clark, N. Duncan: Tree-to-tree Correction for Document Trees, Technical Report 95-372, January 1995, pp 8-10