A similarity-based approach to match elements across versions of XML documents

Fernando Campello[1], Bruno Pinto[1], Gabriel Tessarolli[1], Alessandreia Oliveira[1,2], Carlos Oliveira[2], Marcio Oliveira Junior[2], Leonardo Murta[1], Vanessa Braganholo[1]

[1]Universidade Federal Fluminense
[2]Universidade Federal de Juiz de Fora

{fernandocampello, brunoferreirapinto}@id.uff.br,
{carlosroberto, mtojunior}@ice.ufjf.br,
{alessandreia, gtessarolli, leomurta, vanessa}@ic.uff.br

Abstract. *XML documents are often used to provide inter-system interoperability. A related problem is that XML documents evolve over time, so identifying and understanding the changes they undergo become crucial. Some diff approaches based on syntactic and semantic analysis of the documents have been developed to address this problem. The strategy is to find data fragments that are identical in both versions of an XML document and match the corresponding elements through the use of context keys. However, depending on how XML documents are managed, there is no guarantee that the values of these keys remain the same across versions. Thus, differently from existing approaches, this paper proposes the use of similarity to match corresponding elements across XML versions, rather than key equality. It also shows how this can be applied to support both syntactic and semantic XML diff applications.*

1. Introduction

XML is being adopted as a standard language by many industries and research communities to provide inter-system interoperability, including healthcare [Argüello *et al.* 2009; Thuy *et al.* 2013], legislative houses [Hallo Carrasco *et al.* 2013], and Web Science [Getov 2008]. In this context, users are often not only interested in the current value of data but also in their changes. This explains a renewed interest for managing document evolution. It is not a trivial task, though. The challenge resides in the fact that these documents contain a hierarchical structure and user-defined tags [Bray *et al.* 2008]. While allowing flexibility in data representation, this complicates the monitoring of its evolution, especially in large data repositories. To deal with this problem, some *diff* approaches specific to XML documents have been developed [Cobena *et al.* 2002; Wang *et al.* 2003; Santos e Hara 2004]. Based on syntactic analysis, the main strategy on these approaches is to find data fragments that are identical in both versions of the XML document and match the corresponding elements through the use of context keys [Buneman *et al.* 2003].

Although effective, the use of context keys may not be suitable in all scenarios. Depending on how XML documents are managed, there is no guarantee that the values of the attributes and elements that represent these keys remain the same across versions. For example, there may be a typing error on the key value for an element in version *v1* of an XML document that is corrected in version *v2* of the same document. Even if all the remaining data in that element is kept unchanged, the approaches that rely on keys cannot uniquely identify this element across the versions.

In this paper we propose the use of a similarity-based approach to match elements across versions. It overcomes the problem illustrated before, since it does not rely on key equality to identify elements. Instead, it uses a set of characteristics of the elements, employing different strategies according to the element type to determine the similarity degree. According to this strategy, elements are matched if their similarity degree is greater than a given threshold. Additionally, our matching approach aims at obtaining a global optimum matching instead of a local optimum matching.

Both syntactic and semantic XML diff applications can benefit from this similarity-based matching approach. In this paper, we present Phoenix and XChange [Oliveira *et al.* 2014], two applications that use our similarity-based matching approach for both syntactic and semantic XML diff problems, respectively. Phoenix is a novel syntactic XML diff tool that compares two XML documents, which may be two revisions of the same document or two variants with little shared content. XChange is a semantic XML diff tool that helps understanding the evolution of XML documents. It was originally developed to work with a key-based matching approach, but was adapted to work with our similarity-based approach.

The remaining of this paper is organized as follows. Section 2 describes the similarity matching in XML documents. Section 3 presents examples of applications that use this similarity-based approach. Section 4 discusses related work. Finally, Section 5 presents conclusions and suggestions for future work.

2. Similarity-based Matching in XML Documents

The basic task for every diff algorithm is to identify corresponding elements from two documents. For instance, text-based diff basically adopt LCS [Cormen *et al.* 2009] algorithms for this purpose, detecting the longest common subsequence of lines of text between two files. However, depending on the type of document, such task becomes trickier. XML document is one of the types that brings extra complexity to the matching task, due to its tree-based semistructured nature and flexible compositions rules.

There are different ways to identify corresponding elements in XML documents. Among them, we can cite context key approaches and similarity-based approaches. A context key guarantees the uniqueness of an element throughout versions, using key values. It assumes the key does not change, thus, the matching across documents is fairly simple and relies on the equality of such key. However, this precondition may not hold or, even worse, the elements may not have keys. A related problem is that depending on how XML documents are managed, there may be no guarantee that the value chosen as key remains the same across different versions. For such situations, a similarity-based approach fits better, since it does not rely on key attributes. Similarity-based approaches investigate characteristics of the elements under comparison in order to quantify how much alike they are. Thus, the comparison is no longer binary (equal or not equal), but proportional, returning a similarity degree.

The similarity-based approach for XML elements matching proposed by this paper recursively evaluates the similarity between the information contained within the structures of the XML elements and expresses it through a real value. This value ranges from zero to one, zero representing total inequality and one representing the total equality of the elements.

It is important to note that our similarity technique considers XML documents as unordered trees. That is, only the ancestor-descendant relationship is significant, as opposed to ordered trees in which child ordering is also significant. Although ordering amongst elements in the same level is considered in XML specification, there are several scenarios where it should not impact the use of the document. In fact, several approaches in the literature consider unordered XML trees [Chawathe e Garcia-Molina 1997; Wang *et al.* 2003].

Figure 1 represents our similarity-based approach. Starting from the root elements of the documents under comparison, the similarity-based approach analyzes their similarity based on four features: (1) element name; (2) textual content; (3) element attributes; and (4) subelements. For every feature, the approach evaluates its values on both elements under comparison and compares them to produce a similarity component related to that feature. At the end, the similarity components are combined, resulting in a real value that indicates the similarity degree between the elements under comparison.

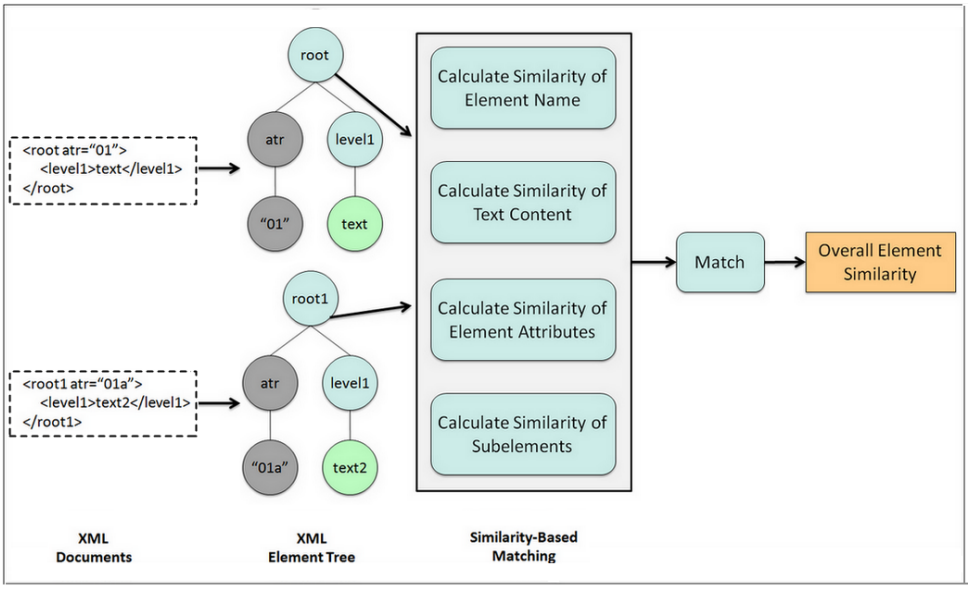

Figure 1. Similarity-based approach to XML matching

In order to calculate each similarity component, our approach uses different algorithms, chosen specifically to deal with the characteristics of each component type. These algorithms and the rationale on their use are briefly described in the following:

- *Name similarity component*. Since element names are strings, the Longest Common Subsequence (LCS) algorithm [Cormen *et al.* 2009] is applied over the names of the elements under comparison. The length of the resulting LCS sequence is then divided by the average length of the two input strings. The result indicates the similarity degree between the element names.
- *Textual content similarity component*. Since the textual content is also a string, the LCS algorithm is applied once more to calculate this similarity component. First, it identifies the LCS sequence and then divides its length by the average length of the textual content from both elements.
- *Attributes similarity component*. Attributes are compared in the similarity-based approach by the following steps:

1. Extract the complete set of attributes names used in both elements under comparison;
2. For each identified attribute, compare its value in the first element with its value in the second element using the LCS algorithm, as already explained, and keep the resulting similarity; if the attribute is not present in one of the elements, this similarity is set to zero. It is worth mentioning that we only compare values when there is a match in the attribute name, i.e., if the name changes, we consider it a new attribute;
3. Sum all the similarities calculated in the previous step and divide by the number of attributes in the complete set;
4. The resulting value is the attributes similarity component.

- *Subelements similarity component*. To calculate this component, another multi-step procedure is used:
  1. Each subelement from the first element is compared to each subelement of the second element using the same similarity calculation algorithm (thus, characterizing recursion), and the results are registered in a matrix;
  2. The all-to-all similarity matrix calculated in the previous step is then provided to the Hungarian algorithm [Kuhn 1955], which outputs the global optimum match amongst the subelements;
  3. From this best match, the subelements similarity component is calculated by dividing the sum of all the similarities in the match by the number of subelements.

All the similarity components are real numbers ranging from zero to one, where zero means total dissimilarity in the given feature, and one means total similarity (equality). With all the similarity components calculated, the overall similarity is calculated using a weighted average.

The weight of each similarity component in the overall similarity calculation is configurable, and should be carefully chosen, depending on the application using the approach. For instance, it may be suitable in some applications to give more importance to text content. That is the case when it is known that the documents being compared use the same schema (thus, the same element names). There might be cases where the application does not use one of the components (by giving it a weight of zero).

3. Similarity-based Matching Applications

Similarity algorithms for comparing XML documents are important in various applications that manipulate semistructured data. The first application that comes to mind in this case is querying XML documents using similarity instead of exact matches [Cohen *et al.* 2003; Dorneles *et al.* 2004; Lima *et al.* 2004]. In this section, however, we focus on other types of applications, such as syntactic and semantic diff. The use of similarity functions to evaluate comparisons in these applications has the potential of making them more generic, as no key is necessary to be set beforehand. The remaining of this section introduces two diff tools that used our similarity-based matching algorithm.

3.1 Phoenix

Phoenix is a diff tool that uses our similarity-based approach to identify and present the differences between two or more XML documents. When used to compare two XML documents, Phoenix calculates the similarities between the elements in these documents and composes a diff document containing the resulting information. This document is presented to the user using a tree view, where each node represents one element from the resulting diff document. Figure 2 presents such visualization. A color scheme is applied to each node, depending on the calculated similarity for the represented element. If the element is only found in the first document, the red color is used for its node, representing that this element was “removed” in the second document. If it is only found in the second document, the green color is applied, meaning that the element was “added”. When the resulting similarity of an element is greater than a given threshold, this element is represented in the tree using a gray scale color scheme. The lighter the color, the greater the similarity found for that element.

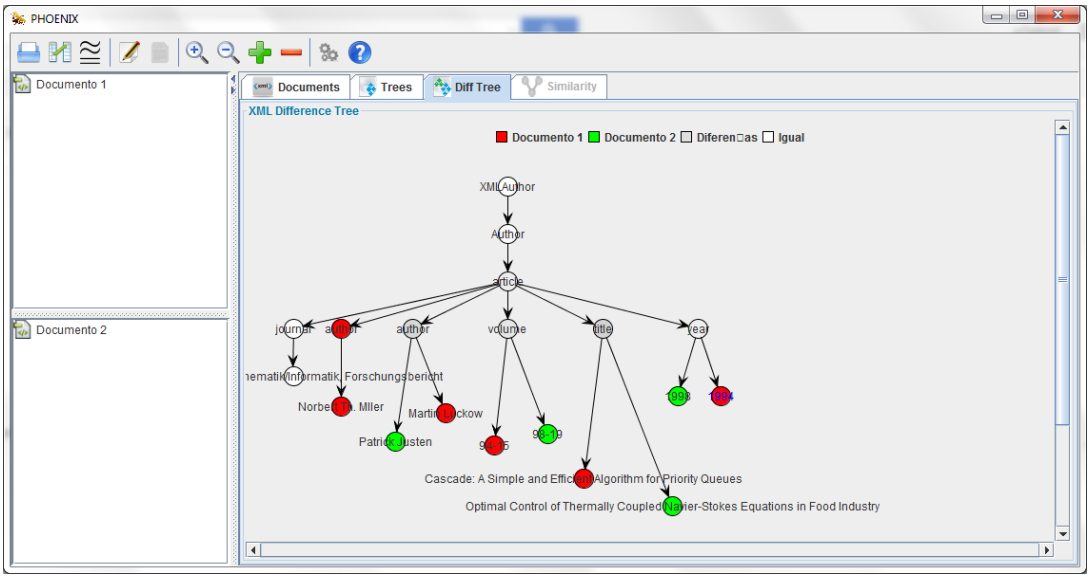

Figure 2. Phoenix diff tree visualization

In addition to the already presented configurable weights of the similarity-based approach, Phoenix tool allows the user to set extra parameters that adapt the similarity calculation to specific application scenarios. A situation that deviates the overall similarity calculation is the presence of trivial similarities, that is, total similarities (100%) pointed by similarity component calculation where the considered feature (attributes, for instance) is non-existent in both elements being compared. To illustrate this problem, consider, for instance, two elements that have no attributes. The attribute similarity component in this case would be 100%. This may not be suitable in certain situations, especially if we would like to focus on the differences instead of on the equalities. For that, a parameter *ignore trivial similarities* was created. If set, the similarity calculation algorithm will simply discard the similarity components identified as trivial during evaluation. It is worth mentioning that the weight of discarded similarity components will also be left out of the similarity formula.

When comparing XML documents within the same XML schema, it is already expected that the element names will be equal. If they are not equal, we should consider

the documents dissimilar. For these scenarios, a parameter called *name similarity required* was created. If set, the similarity calculation will stop if the name similarity component is not 1.0, that is, if the names are not equal. It is also worth mentioning that if this parameter is set, the name similarity component is excluded from the overall similarity formula. With that, the weights from the remaining similarity components shall sum up to 1.0.

Another parameter is the *automatic similarity weight*, which equally distributes the similarity weights, considering the other parameters. That is, if *name similarity required* is set, it will set the other three similarity component weights to 0.33. Also, if a trivial similarity is found and the *ignore trivial similarity* parameter is set, the approach will again equally distribute the weights to the remaining similarity components.

Finally, the last parameter allows the configuration of the *similarity threshold*, that is, the lower limit for the resulting overall similarity in which the approach considers the elements to match (have some similarity involved). If the found similarity is lower than the threshold, the elements are not matched.

3.2 XChange

XChange is an XML semantic diff tool. Its main contribution is to support the understanding of XML documents evolution [Martins *et al.* 2013; Oliveira *et al.* 2014]. Initially, XChange used context keys to support the match of the corresponding elements in different versions of the same XML document. The current tool, however, uses our similarity-based approach.

The input data consists of two versions (revisions) of an XML document and a set of rules. These rules are divided into two categories: match rules and semantic enrichment rules. The match rules are used to identify the corresponding elements in different versions. The rules of semantic enrichment try to understand the meaning of these changes, i.e., to produce a semantic diff [Mens 2002].

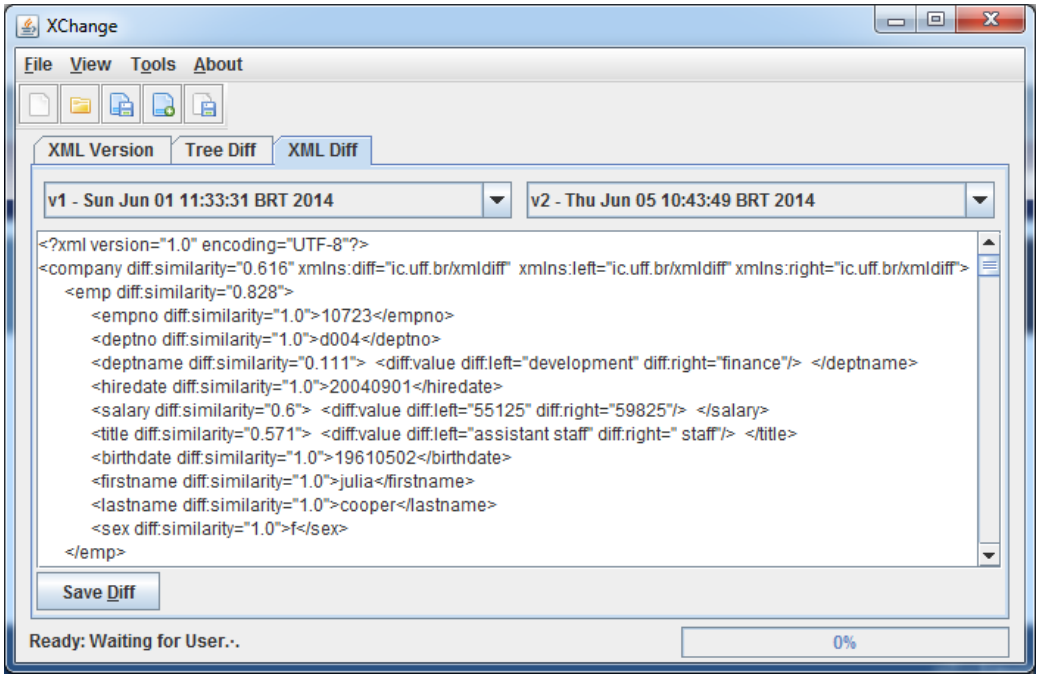

Figure 3. Similarity results

XML document versions are pre-processed and transformed into a set of Prolog facts [Bratko 2001]. However, before generating the Prolog facts, it is necessary to calculate the similarity between all the elements that compose the XML document versions to support the matching of the elements. This task is performed using the described similarity-based approach. A domain expert specifies the similarity weight of each component, the options to use with such weights and sets the minimum similarity threshold. The result is an XML document that lists the elements of the two versions and their respective similarities, as shown in Figure 3. It is worth mentioning that, in this application, we only compare the elements values when there is a match in their names. This is performed by configuring the similarity-based approach underneath. Our reasoning for this is that we respect the schema and assume that elements or attributes with different names should not be matched, regardless of the content.

This output is used to modify the original versions of the XML document, adding an *<id>* element to the corresponding pairs of elements, matched by the similarity-based approach rate. For elements with similarity below the minimum similarity threshold, and also for those that do not have a corresponding match element, we add different *<id>* tags. When all the similarity calculation is finished, the translation process generates several Prolog facts, transforming elements into predicates and their contents into constants (shown in Figure 4). Finally, the Prolog inference engine applies the match rules and the semantic rules on the generated Prolog facts and returns the high level meaning of the changes. In other words, it is able to identify the reason of the evolution of the XML document from an earlier version to a later one.

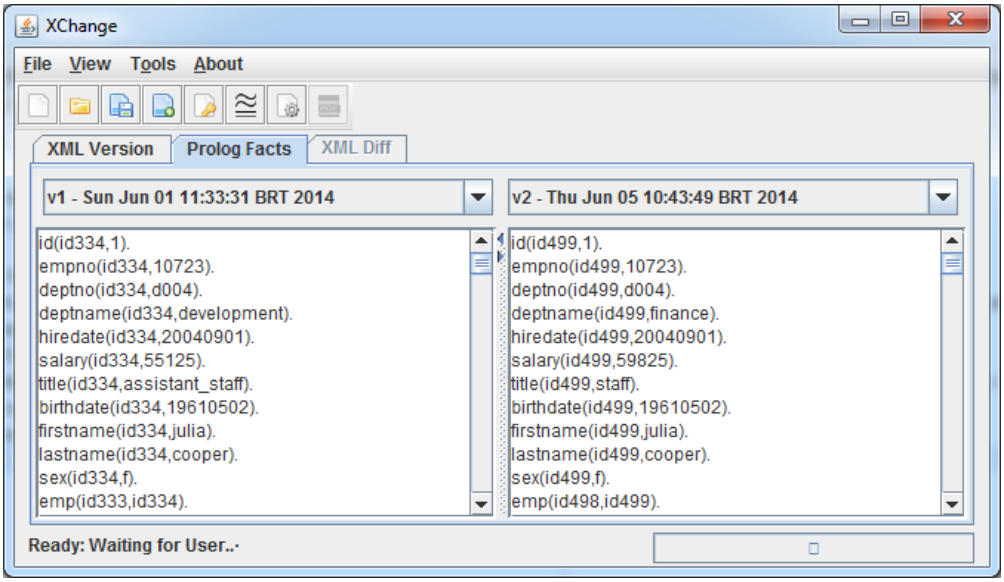

Figure 4. Prolog facts

4. Related Work

During XML documents evolution, it is important to highlight which were the changes that took place to transform from an old version into a new version. To deal with this problem, some diff approaches were developed. There are a number of techniques that identify corresponding elements in XML documents based on context keys. There are also algorithms that support change detection and subtree matching based on similarity techniques. Following, we discuss some approaches that use these strategies.

XyDiff [Cobena *et al.* 2002] detects the differences between versions of XML documents using a key-based approach. It uses ordered trees. Its strategy is to identify large subtrees unchanged between two versions of the XML document, thus reducing the amount of data to compare. Its running time is linear in the size of the document, but it cannot guarantee an optimal result. In certain cases, the change set is not minimal. It is worth mentioning that it considers the operations of insert, delete, and update of elements. It also considers the move operation on subtrees that transfers a node or a subtree from one position to another. X-Diff [Wang *et al.* 2003] is also a key-based approach to detect changes between versions of XML documents. It focuses on ensuring the optimal delta, i.e., the minimum operation sequence that can transform the XML tree from one version to the other. Moreover, it considers only the standard operation of insert, delete, and update of elements in diffs. XKeyDiff [Santos e Hara 2004] is a diff algorithm that uses XML keys to match elements that refer to the same entity in two versions of the document. With that approach, it is able to find matches that may not be possible using solely the structural analysis of XML documents. It uses the XyDiff algorithm [Cobena *et al.* 2002] to find additional matches, and generates the edit script corresponding to the correct order of operations that transforms a previous XML version into a new one. MH-DIFF [Chawathe e Garcia-Molina 1997] detects changes between two structured data snapshots, or trees. It shows the changes as an edit script and it considers the standard operation, plus move and copy operations on subtrees. According to the authors, MH-Diff transforms the change detection into a problem of computing a minimum-cost edge cover of a bipartite graph. Although key-based techniques are effective, they do not work on cases where the elements don't present a key or if the elected key changes across versions. Our similarity-based approach handles such situations.

There are also algorithms that already apply similarity across XML documents. Tekli *et al.* [2009] discusses and classifies several algorithms according to the technique they use: (1) Edit Distance (ED), for algorithms that use edit distance between the trees representing the documents to derive similarity; (2) Information Retrieval, for those used in query/document matching, generally applying approximations to reduce the response time of queries, at the expense of decreasing the exactness and quality of the matching; (3) Other application-specific techniques, such as structure matching and path similarity. Our work provides an alternative to ED-based techniques, which are focused on document/document structure and content matching. XML-SIM-CHANGE [Viyanon e Madria 2010] detects XML similarity using change detection mechanism to join XML document versions. Keys in subtrees play an important role in order to avoid unnecessary comparisons of subtrees within different XML versions of the same document. It uses a relational database to store XML versions and apply SQL for detecting similarities. Differently from these approaches, our similarity-based approach is able to provide a similarity degree amongst attributes, elements, and documents as a whole, allowing a precise comparison of versions. We believe that this characteristic, combined with the fine tuning of the weight in each similarity component, can increase the applicability of our approach. Although not entirely related to our approach, it is worth mentioning that there are initiatives to XML document/grammar matching, such as [Bertino *et al.* 2004; Xing 2006; Tekli *et al.* 2007]. These approaches try to identify how similar to the grammar a document is. Thuy *et al.* [2013] also proposes metrics and an approach to calculate XML schema similarity, useful in data integration.

5. Conclusion

This work introduced an approach for matching XML documents through an analysis of similarity. Traditional approaches compare XML documents versions using, for example, context keys to match elements. In some situations this is not feasible. There are cases where it is not possible to define a context key. In other situations, when the elements are edited, there is no guarantee that the key values will remain the same across versions. Our approach provides an alternative to element matching during XML document comparison that is applicable even in those situations. Furthermore, with the flexibility achieved through the configurable weights for each similarity component and the parameters, our approach allows applications to obtain different similarity degrees from the document comparison. Future work may contribute to enhance the user application interaction, so it becomes more intuitive in the context of setting the parameters. We will perform independent sensibility tests to find out the best values for each parameter in each context. Also, we plan to conduct an experimental evaluation to measure the runtime and the quality of the match when compared to existing approaches.

Acknowledgement

We would like to thank CNPq and FAPERJ for the financial support.

References

ARGÜELLO, M., DES, J., FERNANDEZ-PRIETO, M. J., PEREZ, R., PANIAGUA, H. Executing Medical Guidelines on the Web: Towards Next Generation Healthcare. In *Applications and Innovations in Intelligent Systems XVI,* Springer London, p. 197–210, 2009.

BERTINO, E., GUERRINI, G., MESITI, M. A matching algorithm for measuring the structural similarity between an XML document and a DTD and its applications. *Inf. Syst.* 29(1): 23–46, 2004.

BRATKO, I. *Prolog programming for artificial intelligence*. Addison Wesley, Harlow, England; New York, 2001.

BRAY, T., PAOLI, J., SPERBERG-MCQUEEN, C. M., MALER, E., YERGEAU, F. Extensible Markup Language (XML) 1.0 (Fifth Edition)., 2008.

BUNEMAN, P., DAVIDSON, S., FAN, W., HARA, C., TAN, W.-C. Reasoning about keys for XML. *Inf. Syst.* 28(8): 1037–1063, 2003.

CHAWATHE, S. S., GARCIA-MOLINA, H. Meaningful Change Detection in Structured Data. In *ACM SIGMOD International Conference on Management of Data*, ACM, New York, USA, p. 26–37, 1997.

COBENA, G., ABITEBOUL, S., MARIAN, A. Detecting changes in XML documents. In *International Conference on Data Engineering (ICDE)*, IEEE Computer Society, San Jose, California, USA, p. 41–52, 2002.

COHEN, W. W., RAVIKUMAR, P., FIENBERG, S. E. A Comparison of String Metrics for Matching Names and Records. In *KDD Workshop on Data Cleaning and Object Consolidation*, p. 73–78, 2003.

CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., STEIN, C. *Introduction to Algorithms*. The MIT Press, 2009.

DORNELES, C. F., HEUSER, C. A., LIMA, A. E. N., SILVA, A. S. DA, MOURA, E. S. DE. Measuring Similarity Between Collection of Values. In *Proceedings of the 6th Annual ACM International Workshop on Web Information and Data Management* WIDM '04., ACM, New York, NY, USA, p. 56–63, 2004.

GETOV, V. e-Science: The Added Value for Modern Discovery. *Computer* 41(11): 30–31, 2008.

HALLO CARRASCO, M., MARTÍNEZ-GONZÁLEZ, M. M., LA FUENTE REDONDO, P. DE. Data Models for Version Management of Legislative Documents. *J. Inf. Sci.* 39(4): 557–572, 2013.

KUHN, H. W. The Hungarian Method for the Assignment Problem. *Nav. Res. Logist.* 2(1-2): 83–97, 1955.

LIMA, A. E. N., DORNELES, C., HEUSER, C. Eris: Um protótipo para consulta por similaridade a bases de dados XML. In *Sessão de Demos do Simpósio Brasileiro de Banco de Dados*, SBC, Brasília, DF, p. 13–18, 2004.

MARTINS, G., LARCHER, J., OLIVEIRA, A., MURTA, L., BRAGANHOLO, V. XChange: Compreensão de Mudanças em Documentos XML. In *Sessão de Demos do Simpósio Brasileiro de Banco de Dados*, SBC, Recife, PE, p. 31–36, 2013.

MENS, T. A State-of-the-Art Survey on Software Merging. *IEEE Trans. Softw. Eng.* 28(5): 449–462, 2002.

OLIVEIRA, A., MURTA, L., BRAGANHOLO, V. Towards Semantic Diff of XML Documents. In *Symposium on Applied Computing (SAC)*, ACM, Gyeongju, Korea, p. 833–838, 2014.

SANTOS, R. C., HARA, C. S. XKeyDiff - Um algoritmo semântico para detecção de mudanças entre documentos XML. *Rev. Eletrônica Iniciaç. Científica REIC* 4(3), 2004.

TEKLI, J., CHBEIR, R., YETONGNON, K. Structural Similarity Evaluation Between XML Documents and DTDs. In *Web Information Systems Engineering – WISE 2007* Lecture Notes in Computer Science., Springer Berlin Heidelberg, p. 196–211, 2007.

TEKLI, J., CHBEIR, R., YETONGNON, K. An overview on XML similarity: Background, current trends and future directions. *Comput. Sci. Rev.* 3(3): 151–173, 2009.

THUY, P. T. T., LEE, Y.-K., LEE, S. Semantic and structural similarities between XML Schemas for integration of ubiquitous healthcare data. *Pers. Ubiquitous Comput.* 17(7): 1331–1339, 2013.

VIYANON, W., MADRIA, S. K. XML-SIM-CHANGE: Structure and Content Semantic Similarity Detection among XML Document Versions. In *On the Move to Meaningful Internet Systems, OTM 2010* Lecture Notes in Computer Science., Springer Berlin Heidelberg, p. 1061–1078, 2010.

WANG, Y., DEWITT, D. J., CAI, J.-Y. X-Diff: an effective change detection algorithm for XML documents. In *International Conference on Data Engineering (ICDE)*, IEEE Computer Society, Bangalore, India, p. 519 – 530, 2003.

XING, G. Fast Approximate Matching Between XML Documents and Schemata. In *Frontiers of WWW Research and Development - APWeb 2006* Lecture Notes in Computer Science., Springer Berlin Heidelberg, p. 425–436, 2006.