Change Detection on JATS Academic Articles: An XML Diff Comparison Study

Milos Cuculovic, Frederic Fondement, Maxime Devanne, Jonathan Weber, Michel Hassenforder

Change Detection on JATS Academic Articles: An XML Diff Comparison Study

Milos Cuculovic
milos.cuculovic@uha.fr
IRIMAS, University of Haute-Alsace
Mulhouse, France
MDPI
Basel, Switzerland

Frederic Fondement
frederic.fondement@uha.fr
IRIMAS, University of Haute-Alsace
Mulhouse, France

Maxime Devanne
maxime.devanne@uha.fr
IRIMAS, University of Haute-Alsace
Mulhouse, France

Jonathan Weber
jonathan.weber@uha.fr
IRIMAS, University of Haute-Alsace
Mulhouse, France

Michel Hassenforder
michel.hassenforder@uha.fr
IRIMAS, University of Haute-Alsace
Mulhouse, France

ABSTRACT

XML is currently a well established and widely used document format. It is used as a core data container in collaborative writing suites and other modern information architectures. The extraction and analysis of differences between two XML document versions is an attractive topic, and has already been tackled by several research groups. The goal of this study is to compare 12 existing state-of-the-art and commercial XML diff algorithms by applying them to JATS documents in order to extract and evaluate changes between two versions of the same academic article. Understanding changes between two article versions is important not only regarding data, but also semantics. Change information consumers in our case are editorial teams, and thus they are more generally interested in change semantics than in the exact data changes. The existing algorithms are evaluated on the following aspects: their edit detection suitability for both text and tree changes, execution speed, memory usage and delta file size. The evaluation process is supported by a Python tool available on Github.

CCS CONCEPTS

• **General and reference → General conference proceedings**; **Evaluation**; • **Applied computing → Extensible Markup Language (XML)**; **Version control**.

KEYWORDS

XML diff; semantic diff; change control; document comparison; academic publishing; JATS

ACM Reference Format:
Milos Cuculovic, Frederic Fondement, Maxime Devanne, Jonathan Weber, and Michel Hassenforder. 2020. Change Detection on JATS Academic Articles: An XML Diff Comparison Study. In *ACM Symposium on Document Engineering 2020 (DocEng '20), September 29-October 2, 2020, Virtual Event, CA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3395027.3419581

DocEng '20, September 29-October 2, 2020, Virtual Event, CA, USA

ACM ISBN 978-1-4503-8000-3/20/09.
https://doi.org/10.1145/3395027.3419581

1 INTRODUCTION

Extracting and understanding the differences between two versions of an academic article has practical usage within the scientific community. Currently, reviewers and editorial teams must manually compare two versions of the same manuscript when author corrections have to be evaluated. More widely, readers are interested in comparing different versions of the same article (e.g., two preprint versions or a preprint with its later published article version). There are several main document types in which a scientific article can be written. Authors mostly use tex[1], docx[2] and odt[3], which are well-known and established formats used by the main typesetter tools. Office suites (Open Office and MS Office) provide two interesting functions—change tracking and document compare. Although very useful, the change tracking function lies in the author's hands and can be disabled at any time. The document compare function can be applied regardless of the change tracking function; however, it relies on the typesetter tool. For both, changes are represented only visually and cannot be extracted for further processing. From the reader's perspective, having direct access to changes made by the author (differences between the original and the modified version, called deltas) is more convenient than reading both versions of the article and visually evaluating the differences in order to match them with the improvement suggestions. Standardising the delta would not only speed up the process, but also increase the revision evaluation quality; thus, there is a need for a dedicated and powerful document comparison tool able to compare academic articles.

Since the beginning of the digital age in the 1970s, researchers have expressed their interest in comparing textual documents with the purpose of extracting, analysing and understanding differences. Text diff algorithms [16, 27] have been studied and described. Some of them are still in use, like Hunt–McIlroy's [10] algorithm (currently used in the GNU Diff utility) and Myers' [17] algorithm. Most of these algorithms are line-based and rely on two edit operations:

[1] foldoc.org/TeX
[2] https://loc.gov/preservation/digital/formats/fdd/fdd000397.shtml
[3] https://loc.gov/preservation/digital/formats/fdd/fdd000428.shtml

Insert and Delete. The difference is calculated by comparing each line of the original with the corresponding line of the modified text file. The range of applications for text diff algorithms is wide; they are present in version control systems (Git, Apache Subversion) and many other tools like IDEs (Eclipse Compare) or text editors (Notepad++ Compare), where tracking textual differences is important. Starting from the late 1990s to the early 2000s, with the explosion of the World Wide Web and semi-structured text documents, several research groups had their focus on a specific type of text document—XML [28]. The main reason for this was the promising future of XML, adopted widely in a short period by many key players in the domain of high technology. Compared with plain text documents, the particularity of XML resides in its hierarchical structure, also called tree structure, and the existence of its parent–child node relations. The tree-to-tree editing problem defined by Selkow [26] makes existing text diff algorithms unsuitable for comparing XML documents. This was further demonstrated by several research groups [2, 3, 7]. Several projects [29] also developed HTML-based diff algorithm implementations providing content and styling differences. In our environment, we are only interested in content changes, so this approach is not considered.

Academic publishers convert articles from tex, docx and odt to JATS (Journal Article Tag Suite) XML developed by NISO (National Information Standards Organization). JATS is the de-facto standard for the XML representation of journal articles. It has the advantage of being machine readable and independent of typesetter tools. JATS has no styling/layout information, and carries only the article data and structure. The styling, depending on the publisher preferences, is applied at a later stage, while the JATS is converted to LaTeX, HTML and other formats. Due to the text-centric nature of JATS, important information resides in text nodes, tree elements and attributes (Figure 1). The data are divided into the following main tree elements: front, body and back. The front contains subtree elements about the metadata information: the journal, the title, the authors, etc. The body contains the article content structured with sections, subsections and paragraphs. It is the largest part of the document, dominated by paragraphs representing text blocks in a similar way to HTML <p> tags. The back contains references, figures, tables, notes and acknowledgments. We are interested in analysing the existing XML diff algorithms by evaluating their compatibility with JATS documents. This is one of the possible metrics we are using to measure the quality of a delta, and is closely related to our initial goal of comparing JATS article versions.

Table 1 shows 12 implementations of XML diff algorithms: 11 are state-of-the-art algorithms from the scientific literature and one is a commercial implementation. Their analyses will be presented as follows: In Section 2, we describe each of the algorithms individually. In Section 3, we present the analysis approach and the XMLDiffAnalyzer script we developed in order to automate one part of the analysis. In Section 4, we do an initial high-level performance and suitability analysis of the algorithms for JATS document comparison. The performance is evaluated in terms of execution time, average and maximal memory used and resulting delta file size. The suitability depends on the results obtained by analysing the delta files for specific author edit operations which impact the XML document in both the text and tree structure. In Section 5, we focus on the three algorithms we identified as most suitable and three algorithms identified as low-performing in JATS comparison. A deeper analysis of the delta outputs they generated is performed in order to identify their strengths and weaknesses. Finally, in Section 6, we present the need for a new JATS diff algorithm.

Table 1: XML diff algorithms

Algorithm	Language	Link	Last update
XmlDiff [3]	Python	Pypi.org	present
DeltaXML	Java	Commercial	present
XyDiff [7]	C++	Github	2015
Xdiff [31]	C++	Github C++	2015
DiffXml [4]	Java	Github	2018
XOp [11]	Java	Living-pages.de	2009
FC-XmlDiff [12]	Java	Github	2009
DiffMK [18]	Java	Sourceforge	2015
JXyDiff [24]	Java	Github	2009
XCC [22]	Java	Launchpad	2009
JNDiff [5]	Java	Sourceforge	2014
Node-delta [13]	JavaScript	Github	present

2 RELATED WORK

As seen in Table 1, we identified 12 XML diff algorithms. In these algorithms, edit operations specific to the tree structure were introduced: Insert, Delete, Update and Move. Deleting a node implies the same action on all of its child nodes, which means it is no longer a line-based approach, but becomes a tree-structure-based approach. While moving a tree has no impact on the text content, it does move the content by changing its position within the tree. In the real-world scenario of an academic article, inserting an author that is composed of multiple child nodes representing the author's name, email and affiliation number would result in a node Insert action where the entire author tree substructure is impacted. The Update operation is also important in order to minimise the overuse of Insert/Delete operations. Ronnau et al. [23] explain that in case of an attribute change on the root of the document tree, without Update it is necessary to represent the change with a full Delete operation followed by a full Insert operation. The Move operation consists of changing the position of a given child node among the other child nodes of a specific parent, and can be used for author ordering. Without the Move action, changing the node order would result in removing and re-inserting them in the correct order. Attribute editing is another specificity of XML documents which the text diff algorithms are not able to deal with. By using all previously mentioned edit operations, XML diff algorithms should be able to detect both text and tree changes.

XML documents, being simple and general in nature, are suitable for both text and data, and so there are two main categories of XML documents: text-centric (or document-centric) and data-centric, as described in [5] and [15] (see chapter *Text-centric vs. data-centric XML retrieval*). The size of individual text nodes is usually larger in text-centric XML documents, while data-centric nodes are smaller in size but higher in number. Most of the early XML diff algorithms were developed for data-centric XML documents with the main focus on execution time, memory usage and delta size efficiency.

Several research groups [5, 21, 25] demonstrated that XML diff algorithms for data-centric documents are not suitable for text-centric documents, and there is a need for specific XML diff algorithms adapted for text-centric XML documents.

Here, we will briefly describe each of these algorithms and the research work done around them in chronological order:

- **XMLDiff** is the oldest among the algorithms we analysed. It was published in 1996 [3]. However, the XMLDiff implementation we tested dated from 2004. The paper was published two years before the first XML 1.0 Specification and did not directly mention the XML format. It was related to hierarchically structured data, which is closely related to data-centric XML documents. Four edit actions, all related to tree structure changes, were defined (Insert, Delete, Update and Move). The algorithm was written with the purpose of being able to compute the minimum-cost edit script" between two documents. It is still maintained and is at version 2.4.
- **DeltaXML** is a commercial suite of products that started in 2001. One of their solutions is XML Compare, which can be used as a command line or GUI to compare XML files. The delta results are passed through a pipeline so the output can be adapted to different needs. DeltaXML is still maintained, and their R&D teams have published several white papers [30] regarding new approaches in XML diffing.
- **XyDiff** was developed by Gregory Cobena within an PhD project called Xyleme. The publication describing the algorithm was published in 2002 [7]. XyDiff was developed in C++ with the purpose of indexing and analysing the changes on the web where parts of French websites were stored as XML documents. The algorithm parses each document twice: the first time for assigning so-called "XID persistent identifiers" to each node and then to compute the difference. Differences, called XyDelta, are all related to the XIDs. An additional .xidmap file is created grouping XIDs depending on the node differences between the two documents. XyDiff supported four edit actions: Insert, Delete, Update and Move. Its implementation has not been maintained since 2015.
- **XDiff** was published in 2003 [31]. The paper explains that for data-centric XML documents, the order of the elements is not important. This involved the vanishing of the Move edit action. This hypothesis is not suitable in our JATS environment, as sections, authors and others are ordered elements. The complexity of comparing XML documents within the so-called "unordered model" was described as higher than comparing within the "ordered model". The algorithm implementation was written in C++ and was maintained until 2015.
- **DiffXML** was published in 2004 [4]. The algorithm maps the XML DOM tree structure to a relational database using SQL operations to detect changes. DiffXML supports four edit actions: Insert, Delete, Update and Move. Its implementation has not been maintained since 2018.
- **XOp** was developed in 2004 by Living Pages Research GmbH as part of the Ercato project. There is no research literature describing XOp; however, the Ercato project concept was published in 2004 [11] and one year later analysed in a XML Diff comparison thesis [9] that evaluated, among others, the XOp algorithm. The Ercato project was based on "thing-oriented programming" with the so-called ercatons representing "things" (i.e., XML documents). In order to represent object-oriented inheritance, the XOp was developed within the project to compute the difference between two ercatons (XML documents) and represent those as algebraic operations on XML trees. Its implementation has not been maintained since 2009.
- **FC-XmlDiff**, also called faxma, was published in 2006 [12]. The first implementation of the algorithm dates from 2008. In the publication, FC-XmlDiff was compared with other existing algorithms and was presented as being fast and simple. The algorithm uses a greedy heuristic approach by transforming the XML to the domain of sequence alignment, computing the difference and transforming it back to the tree domain. FC-XmlDiff supports four edit actions: Insert, Delete, Update and Move. The algorithm implementation has not been maintained since 2009.
- **DiffMK** implementation was done by Norman Walsh in 2007. There was no research literature describing DiffMK, but there was one paper [12] and one PhD thesis [6] evaluating the algorithm. DiffMK uses the Unix diff algorithm and works in the sequence domain, which makes the tree move detection impossible. Thus, there are three possible edit actions: Insert, Delete and Update. The algorithm implementation has not been maintained since 2015.
- **JXyDiff** is a Java implementation of the XyDiff algorithm developed by Adriano Bonat in 2009. The author claimed this implementation had some improvements compared to the original C++ implementation developed by Gregory Cobena in 2002. This implementation has not been maintained since its publication in 2009.
- **XCC** was published in 2012 [22]. However, the implementation dated from 2009 and has the purpose of comparing office documents where the content is saved in XML format—precisely, the OpenDocument format. Context fingerprints were introduced in order to identify the edit operation in a highly reliable way. XCC supports four edit actions: Insert, Delete, Update and Move. The algorithm implementation has not been maintained since 2009.
- **JNDiff** was published in 2016 [5], however, its implementation was done in earlier in 2009. Compared to most of the XML diff algorithms, the JNDiff authors made a clear difference between text- and data-centric XML documents and set the focus on the delta output quality (human readability, accuracy and clearness) rather than high execution performance. Their approach concentrated on comparing textual documents in XML and representing the differences in a similar way to change-tracking tools in different word processors. JNDiff supports four edit actions: Insert, Delete, Update and Move. The research paper also mentioned the need of additional level 2 edit operations to detect paragraph split, etc. The algorithm implementation has not been maintained since 2014.
- **Node-delta** was part of the Delta.js JavaScript project developed by Lorenz Schori in 2012. There was no research

literature describing the algorithm. However, its development was part of a BSc. thesis [13]. The main purpose of the Delta.js project was the implementation of a version control system for structured documents. The algorithm was inspired by the XCC we mentioned earlier. Node-delta supports four edit actions: Insert, Delete, Update and Move. The algorithm also uses fingerprints, and its implementation is still maintained in 2020.

In the above, we described 12 existing XML diff algorithms. There are probably many more that are less well-known, proprietary or with no scientific literature. Each of these algorithms has its own way of describing differences with no universal delta model. This makes it challenging to compare them and measure the quality of their delta outputs, as described in [1].

3 CRITERIA

In previous data-centric XML diff comparison studies [8, 9], the main criteria for an appropriate algorithm were set based on its execution time, delta size, CPU and memory usage. In our case, we are mainly focused on the delta output—that is, all the differences between two versions of an academic article should be detected, correctly interpreted and represented. In order to assess the delta output, we need to understand the modifications an author is making during the revision process and correlate those to the changes observed on JATS.

3.1 Author Modifications

During the revision rounds, the following author modification actions are seen as common:

- **Paragraph correction** is the most common author modification we observed. This is because paragraphs represent the largest part of the article. Authors modify paragraph content, and move, merge or split paragraphs. Content changes include text additions, removals, moves and style changes. Paragraphs are also composed of other objects such as mathematical formulas and citations that are subject to editing. Smaller corrections can be interpreted as updates, while larger corrections can be seen as rewrites.
- **Section correction** is mostly about article structural change. A section is composed of other sub-elements such as its title, subsections, paragraphs, figures, etc. Authors modify section sub-elements, merge and split sections or upgrade subsections into sections or downgrade sections into subsections.
- **Author correction** is observed while modifying author information, adding or deleting an author or changing an author's position within the authors list.
- **Title correction** is observed in the article title, which is a relatively short portion of text.
- **Citable object correction** is observed on objects that can be cited within the article (i.e., references, figures, tables, sections, algorithms, etc.). The particularity of those objects is that their order of appearance within the article is used to generate their incremental citation number; moving a table, reference or figure impacts the number with which this object is cited within the article.
- **Embedded object correction** is observed on objects that are externally produced and inserted in the article. The most common examples are figures and images.

By further analysing the previously mentioned modifications observed on a corpus of academic articles, we identified nine general edit actions that an author can apply: Add, Remove, Update, Move, Merge, Split, Upgrade, Downgrade and Styling. Add and Remove are the two main modifications, and the others can be reinterpreted with their sequence. Figure 1 shows the correlation between the typesetter version of an academic article and its JATS representation. As mentioned in the introduction, JATS has structured data, and important information resides in text nodes, tree element structures and attributes. The following list shows a short description of each edit action:

- **Add** is the first main modification observed on all levels of the article. Authors can add characters, sentences, paragraphs, sections, authors, affiliations, references, etc.
- **Remove** is the opposite of Add.
- **Update** is observed where changes are minor. Instead of representing a typo correction or a rephrasing in a paragraph as a full Remove/Add sequence, Update is used as a more fine-grained approach.
- **Move** is observed where the order of items within the article is changed. It has no impact on the content itself, only on its position within the article.
- **Merge** is observed on paragraphs, subsections and sections. For paragraphs, it usually consists of removing the line breaks between two paragraphs in order to form one larger paragraph. For subsections and sections, the line breaks are removed and, additionally, their titles are merged.
- **Split** is the opposite of Merge.
- **Upgrade** is observed on subsections and consists of changing a subsection to a section.
- **Downgrade** is observed on sections and consists of changing a section to a subsection.
- **Styling** is observed on portions of text, mostly within paragraphs. Authors can add or remove styling elements such as italic, bold, subscript, superscript, etc. Those styling elements can also have their range changed while extending or shrinking the styled portion.

Each of the previously described modifications that an author applies to an article is a sequence of the general edit actions we identified. Paragraph correction, for example, can be composed of text Insertions, Deletions, Updates, Moves, Merges with other paragraphs, Splits or Styling changes.

3.2 JATS Edit Actions

In order to understand how these common author modifications impact the JATS versions of the article, we analysed each of them by observing the changes from an XML perspective. This gave us the opportunity to define 16 edit actions produced by the author and reflected on the JATS. Those actions are divided into two edit groups: six text and ten tree edits. Text edits are actions observed on text nodes, mostly paragraphs, but also article title, author name, etc. Tree edit actions are observed on the tree structure of the JATS

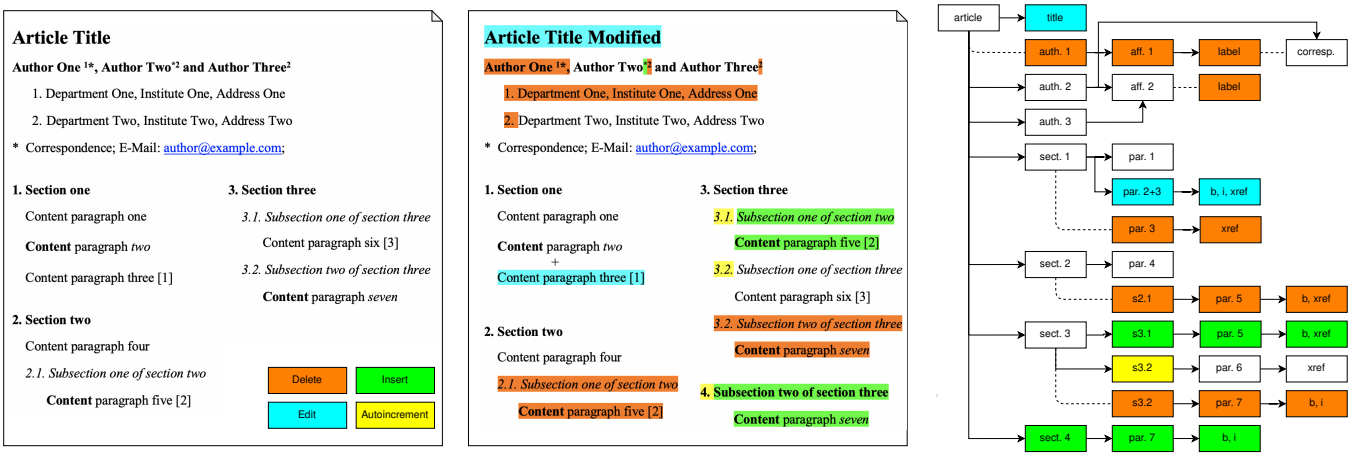


Figure 1: Author modifications' impact on JATS article version

Left, the original typesetter article; middle, the modified version; right, the impact on JATS

while adding or deleting elements, moving paragraphs or sections.

Text edits:

- **Delete** is a basic edit operation that can be executed on any text node within the article. We can find text deletion in paragraphs, sections, authors, title and other nodes.
- **Insert** is another basic edit operation that can be executed on any text node within the article.
- **Move** is a composite operation that is often seen in paragraphs. Entire sentences or parts of sentences are moved in order to correct the grammar or improve the writing clarity.
- **Update** is another composite edit type that can be seen in any of the XML nodes. It makes sense while doing relatively small modifications to a given text. Having a text edit action with a high volume of inserted/deleted text should not be considered as an update, but rather as a full rewrite using Delete–Insert actions.
- **Style** is a specific edit type where styling elements range change while extending or shrinking the styled portion. One example is the extension of a portion of text that is in italics.
- **Complex** is a specific text content (usually paragraphs) that has the specificity of embedding other markups such as styling, cross-reference, tables, mathematical formulas, etc. This is very common in academic articles, and edits on complex content should be implemented in the same way as on non-complex content.

Tree edits:

- **Delete** is produced by deleting specific parts of the article, for example an author, section, paragraph, reference or figure. It is specific enough that when a parent node is removed, all child nodes are also removed.
- **Insert** is the opposite of Delete.
- **Update attribute** is very specific to XML documents. It is observed when changing the correspondence of an author, the license information of the article or the reference type within the reference.
- **Move** is observed in the reordering of authors, references, figures and other elements. It is an important edit action for text-centric XML documents and is reflected by changing the order of child nodes within their parent node.
- **Merge** is observed when paragraphs or sections are merged. This implies the merging of multiple parent nodes with all their child nodes into a single parent node containing all child elements from each merged parent node.
- **Split** is the opposite of Merge.
- **Upgrade** is observed when subsections are changed to sections.
- **Downgrade** is the opposite of Upgrade and is observed when sections are changed to subsections.
- **Style** consists of adding or removing styling information such as bold or italics to some portion of text.
- **Complex** tree nodes are nodes that contain text nodes embedding other markups.

After observing a sample batch of 50 real-life author JATS articles[4], we found that the frequency of complex paragraphs was much higher compared to plain text paragraphs. The average ratio is 80% vs. 20%. The largest edited part of a JATS document is the article body, composed mostly of paragraphs. We conclude that edits on complex text and trees is important.

3.3 XMLDiffAnalyzer Script

In order to automate the execution and the results collection for the 12 algorithms we are comparing, we developed the XMLDiffAnalyser script available online[5]. The results collected and JATS testing files prepared from the original research paper [14] are also available[6]. Algorithm 1 presents the pseudo code of the script written in Python. The script embeds the executable of 12 algorithm implementations, the compared articles and the resulting delta files per algorithm per article. It requires the user to input the number

[4] github.com/milos-cuculovic/XMLDiffAnalyzer/tree/master/Sup/XML
[5] github.com/milos-cuculovic/XMLDiffAnalyzer
[6] github.com/milos-cuculovic/XMLDiffAnalyzer/tree/master/TestingCorpus

of times each algorithm will be executed, and which articles the algorithms will run on. It measures and returns the time, memory and delta size performance data, writing them to a CSV file and generating SVG vector figures. In case of further tests, the algorithm can be easily reused on other XML documents.

Algorithm 1 XMLDiffAnalyzer

```
procedure XMLDiffAnalyzer()
    tools ← XyDiff, JNDiff, JXyDiff, ..., DiffXml
    rounds ← 5
    articles ← { orig_1  new_1
               { orig_2  new_2
               { ...     ...
               { orig_n  new_n
    delta_dir ← Full path of the delta save directory
    START(rounds, articles, delta_dir)
    function START(rounds, articles, delta_dir)
        build the CSV file
        for article in articles do
            for round in rounds do
                for tool in tools do
                    result ← PROC(tool, rounds, article)
                    for result_item in result do
                        generate result_item pyplot graph
                        save result_item into SVG vector figures
                    end for
                    results[] ←result
                end for
            end for
        end for
        write results[] in the CSV file
    end function
    function PROC(tool,rounds,article)
        build and execute the command
        results.execution_time ← Execution time
        results.memory ← Max and average memory used
        results.delta_file_size ← Size of the delta output
        return result
    end function
end procedure
```

4 COARSE-GRAINED

The initial evaluation phase consists of a high-level performance and suitability analyse of the 12 XML diff algorithms. Here, we carry out a coarse-grained evaluation with the purpose of identifying the potential suitable algorithms for JATS article comparisons. These will be further analysed in Section 5 with a more fine-grained approach. The coarse-grained evaluation is divided in two parts, first the performance evaluation and then the delta output analysis.

4.1 Performance Evaluation

JATS articles are large text-centric XML files that may vary from 100 KB to 400 KB. In order to calculate the minimum performance requirements for suitable algorithms, we measure the time and memory needed per algorithm for a comparison. The tests[7] were run on two different edit scenarios. The first on a JATS representing one minimal change on the title, and the second on a JATS representing real-life author changes to the article title, authors, affiliations, paragraphs, figures, tables, references, etc.

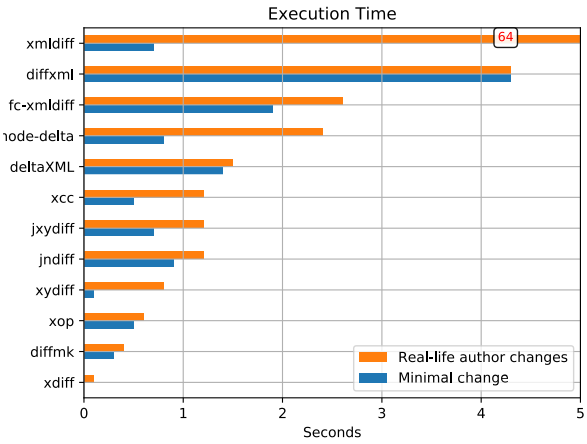


Figure 2: Time—minimal vs. real-life author changes

Time measured per algorithm in two edit scenarios: one with minimal text edit and the other with real-life author changes. In both scenarios, the algorithms were run to compare the same original file with its modified version depending on the two edit scenarios

Figure 2 shows the execution time each algorithm takes on average to perform a comparison of the XML file pairs. The average was calculated on five comparison round executions per algorithm. Except for XMLDiff, where the execution time scales exponentially while the number of changes increases, the rest of the algorithms are able to do the comparison in under three seconds (under five seconds for DiffXML), which is acceptable within our environment.

Figure 3 represents the average and maximum RAM usage per algorithm. The results range from 5 MB for XDiff to 120 MB for DiffMK in terms of average RAM used, which is more than acceptable within our environment. We noticed that DiffMK has an imbalance between the average and maximum RAM used, where the maximum peaks are up to six times larger than the average.

The purpose of the delta file size evaluation is to get initial insights on the number of potential operations that each algorithm will produce, and how those are stored. For one minimal change comparison, Figure 4a shows that three algorithms, DeltaXML, DiffMK and XMLDiff, produce large delta files compared to others due to the fact these algorithms represent differences by annotating one of the two compared XML files. While evaluating the real-life author changes comparison, Figure 4b shows that DiffXML produces the largest delta file with over 10k edit actions, which seems to be oversized. JXyDiff and XCC are both heavily affected by the number of changes. Overall, with the exception of DiffXML, the algorithms produce delta files with acceptable sizes within our environment.

[7] The evaluation was done on an Apple MacBook Pro (15-inch, 2016); Processor: 2.7 GHz Quad-Core Intel Core i7; Memory: 16 GB 2133 MHz LPDDR3; SSD Hard Drive

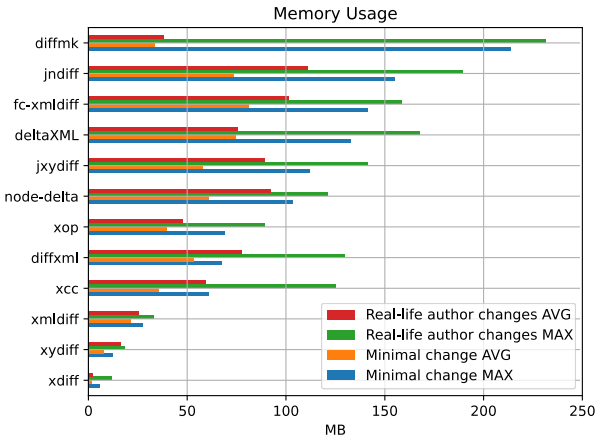


Figure 3: Memory—minimal vs. real-life author changes

Average and maximum memory measured per algorithm in two edit scenarios: one with minimal text edit and the other with real-life author changes. In both scenarios, the algorithms were run to compare the same original file with its modified version depending on the two edit scenarios

4.2 Delta Output

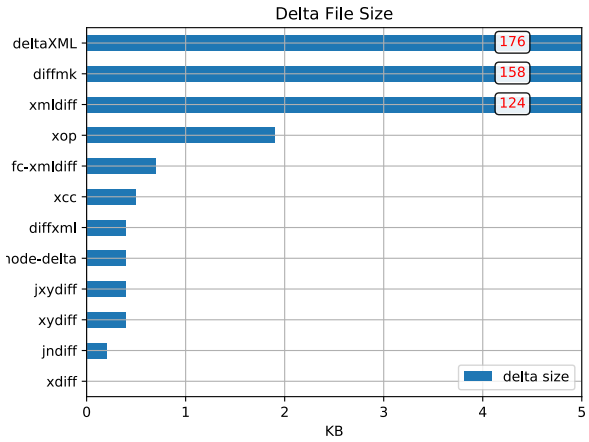
The evaluation of delta outputs is done within the two modification groups we have presented—text and tree edits—with a total of 16 edit actions:

- **Text edits:**
  - Delete
  - Insert
  - Move
  - Update
  - Style edit
  - Complex edit
- **Tree edits:**
  - Delete
  - Insert
  - Attribute
  - Move
  - Merge
  - Split
  - Upgrade
  - Downgrade
  - Style edit
  - Complex edit

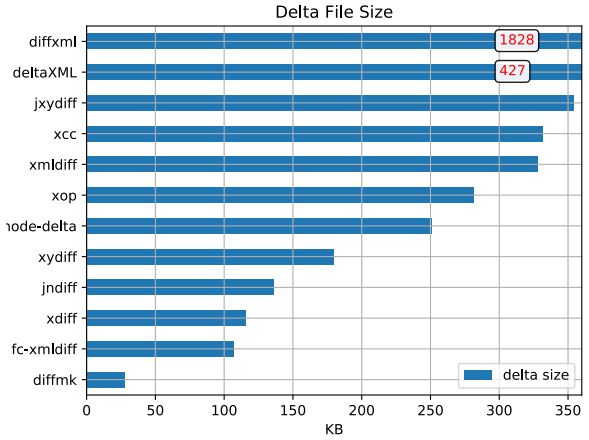
Using the XMLDiffAnalyzer, we applied the algorithms to 16 different JATS file pairs, each pair reflecting one of the edit actions. The full comparison results are shown in the supplementary supporting information[8] organised into individual tables, each table describing our observation per edit action. The scoring system gives between zero and two points per edit action by comparing the expected and obtained delta results. The number of points is weighted by a factor two for complex edit actions, both on text and trees. This decision is supported by the importance of the author edits on complex content, as mentioned in 3.2. An active Excel sheet was created [9] in order to interactively observe the scoring changes by editing the weights.

We present the delta output results in Table 2. The total score is calculated by summing the successful matching of expected versus obtained delta outputs per edit action. Based on this, we can rate the JNDiff algorithm as first; XyDiff sharing the second place with XMLDiff; and XCC as third.

[8] github.com/milos-cuculovic/XMLDiffAnalyzer/blob/master/Sup/Supplementary.pdf
[9] github.com/milos-cuculovic/XMLDiffAnalyzer/blob/master/Sup/Weight_Editor.xlsx



(a) Minimal change



(b) Real-life author changes

Figure 4: Delta size—minimal vs. real-life author changes

Delta size per algorithm applied to the same original JATS—once on the modified JATS with minimal text edit and once with real-life author changes

By further analysing the obtained results, we observe that none of the algorithms are able to deal with Text move nor Style addition or removal detection. In the best-case scenario, those are considered as full text node updates, and in other cases as a suite of Delete–Insert actions. Operations on complex text nodes are common in text-centric XML documents, and only one of the 12 algorithms, JNDiff, is able to successfully treat complex text edits. Regarding complex tree operations, the best-scoring algorithm is XDiff because it only detects tree edits and is not affected by analysing text edit operations. Tree move is only supported by jXyDiff, XyDiff and JNDiff. Most of the remaining algorithms present this change as a sequence of tree Delete–Insert operations. Tree merge, split, upgrade and downgrade are not fully supported by any of the tested algorithms. Node-Delta, DeltaXML, XyDiff, XCC

Table 2: Delta output analysis

Coarse-grained analysis of delta output per edit action for each of the 12 XML diff algorithms. The results are divided into two groups: text and tree edits

	Text edits						Tree edits										
Algorithm	Del	Ins	Mv	Upd	Style	Complex	Del	Ins	Attr	Mv	Merge	Split	Upgr	Dwngr	Style	Complex	Res/36
XOP																	0
DiffXml	1	1															2
FC-XmlDiff								2	1								3
DiffMK		1		1				2									4
Xdiff							2		2							2	6
Node-delta	1	1					2		1		1	1	1			2	10
DeltaXML	2	2					2	2				1	1	1			11
jXyDiff	2	2		1	1		1	2	2	2							13
XCC	2	2		1	1			2	2		1	1				2	14
XMLDiff	2	2		2	2	4		1					1			1	15
XyDiff	2			1	1		2	2	2	2	1	1	1				15
JNDiff	2	2		1	1	4	2	1	2	2	1	1	1	1		4	25

and JNDiff are partly able to represent some of those operations as a short sequence of tree Delete–Insert operations.

5 FINE-GRAINED ANALYSIS

As seen in Table 2, none of the tested XML diff algorithms are able to fully fulfil our expectations for comparing JATS documents. Moreover, XMLDiff is too slow within our environment (see Figure 2). However, three of them (JNDiff, XyDiff and XCC) are interesting, as their delta outputs could be eventually improved or post-processed in order to obtain the desired results; we decided to further analyse these three algorithms in order to understand their functional principles and identify their strengths and weaknesses. Moreover, we also perform a quick analysis of the three algorithms that scored the lowest in order to identify the principal reasons of their low performance in our testing scenario. The analysis is more technical and allows us to extract important aspects for a suitable JATS XML diff algorithm.

5.1 JNDiff

JNDiff is able to fulfil 69% of our expectations and is the only algorithm able to deal with text changes in complex environments—one of the reasons it rates as top performing. Complex environments are common in JATS documents, as the majority of the text content contains bold, italics, xref and other styling or reference nodes. The algorithm is also able to make distinctions between text updates and replacements (Delete–Insert), depending on the size of the modification compared to the original text. While editing short text sequences such as the author name, JNDiff considers this action as a full replace. The article title being longer, the same text insertion is considered as a text update. All tree edits are well or partly detected, more than any of the 11 other algorithms. As a concrete example, JNDiff was able to detect all modifications to article authors: insertions and removals, changes in the correspondences via attribute updates and position/ordering changes.

Although scoring higher than others regarding the ability to detect changes in JATS documents, there are several aspects we would like to mention where the algorithm was not able to fulfil our expectations. One of them is the missing ability to represent text Move operations. When an author moves large portions of text, it is considered as a complete rewrite using the Delete–Insert edit sequence. Tree Moves are presented for the parent but also for all its child nodes, which represents an author Move operation containing three child nodes (firstname, lastname and email) as four Move operations. Although the distinction between text Update and text Replace has its strengths, it also presents a weakness where small changes have to be detected. This can be the case for text corrections where changes have been made on the character level. JNDiff will, in this case, consider those edits as complete word rewrites. Tree Merge, Split, Upgrade and Downgrade operations are represented as Delete–Insert sequences. In this way, merging or splitting two paragraphs and upgrading or downgrading sections within a document are not interpreted as we would expect. When on Style edit operations, styling a text is represented as a complete deletion of the existing text followed by a complete insertion of the new text containing the styling node. One last note about JNDiff is the fact that the documentation is written in Italian, which could present further difficulties for future improvements.

5.2 XyDiff

XyDiff is one of the fastest XML diff algorithms, able to fulfil 42% of our expectations and to deal with most of the tree edits (Delete, Insert, Attribute Update and Move) with the exception of Split, Merge and Downgrade, which are only partly carried out. Tree downgrade is, for example, represented as four move actions, which makes post-processing difficult. On the other hand, the text edits score is much lower for XyDiff. Text Update precision is fine-grained, which is good for small text changes like corrections; however, in our test environment, this is more often considered as a weakness for larger text changes, as the algorithm calculates the longest common substring (LCS) and tries to minimise the edit distance at the cost of increasing the difficulty of post-processing the delta representations. Comparisons of complex text nodes are not optimal. Minor text changes are represented as very large due to the fact the change is shown on the entire non-complex part of the text node; changing one letter in such an environment results in a multi-sentence change. Note that XyDiff also considers as complex

those text nodes containing HTML character representations. In the same way as other algorithms, text moves are represented as complete Delete–Insert rewrites. The same behaviour occurs in tree merges, where paragraph/section merges are not represented as we would expect.

5.3 XCC

XCC is able to fulfil 39% of our expectations. It always presents old and new values, which could be very useful for eventual post-treatment of the delta results. XCC is also able to detect text Updates. Tree Insert is represented as one edit action, which is easier to interpret compared to JNDiff. Furthermore, attribute updates are also well represented.

However, there are several important edit actions that are not represented as we would expect. The most important is the detection of all edit operations in complex environments. Changing one character in a large paragraph containing one bold node will result in a large update of the entire text content outside of the bold node. Text Move operations are not detected and are represented as an Update. XCC is also not able to deal with Style changes; they are represented as two actions—an Insert of the new style element with the following text node, and an Update of the original edited element in order to remove the duplicated text. Tree Move detection is another weakness of the XCC algorithm. This edit action is represented as two Inserts, two Deletes and four Updates. Tree Delete impacts the parent, but also all its child elements, similar to how JNDiff interprets tree Move, making the further processing of this edit action difficult. Tree Upgrade and Downgrade are represented by complete Delete–Insert actions, followed by some Updates.

5.4 Low-Performing Algorithms

XOP, DiffXML and FC-XmlDiff (faxma) are the lowest scoring among the tested algorithms. XOP is not able to represent text-node differences but only tree differences. DiffXML is able to represent text Insert, delete and Move operations. While performing tree edits, a tree Upgrade operation results in close to 1000 edit operations, mostly composed of Move actions. FC-XmlDiff, also called faxma, presents the difference with a diff-copy XML tag and there is no information about the edit type.

6 DISCUSSION

The previous section illustrates that several algorithms are able to correctly detect differences between two JATS documents. Unfortunately, none of them are able to fulfil all of our expectations—that is, all differences between two versions of an academic article should not only be detected, but also correctly interpreted and represented. One of the reasons for this is the limited portfolio of possible edit actions, as the current XML diff algorithms are only able to represent Insert, Delete, Move and Update on tree nodes, and Insert, Delete and Update on text nodes. We will list the requirements for a new algorithm able to fulfil the remaining requirements. Figure 1 represents a simplified version of an academic article with the following author modifications: article title edited; author 1 deleted together with its affiliation, implying a correspondence information change and a reorder of the remaining authors; paragraphs 2 and 3 in Section 1 merged into a unique paragraph; Subsection s2.1 from Section 2 moved to Section 3, implying the reorder of the two existing subsections in Section 3; and Subsection s3.2 from Section 3 upgraded to Section 4. We were able to identify the following additional edit actions that would be useful for JATS comparisons:

- **Style edit** (bold, italic, etc.): interpreted and represented differently compared to other nodes. Adding or removing a styling element is not the same as adding or removing a paragraph or a section. These edits should be seen as style edits and not as tree edits.
- **Citable object edit** (section, table, figure, reference, mathematical formula, etc.): interpreted and represented differently compared to other nodes. These objects have several specific aspects: complex tree structure, auto-incremental behaviour while being cited and common elements such as caption nodes or citable object identifiers.
- **Text move**: considered as a move action instead of rewrites, after a certain size.
- **Tree merge** (paragraphs, sections, subsections): considered as a merge action instead of an Insert–Delete sequence.
- **Tree split** (paragraphs, sections, subsections): opposite to the tree merge.
- **Tree upgrade** (subsection): considered as an upgrade action instead of an Insert–Delete sequence.
- **Tree downgrade** (section): opposite to the tree upgrade.
- **Semantics**: additional information on what was changed and how.

In addition to the need for new edit actions, another improvement which could be made is the specific treatment of text (mostly paragraph) nodes. Style, citable objects and mathematical formulas can all be seen differently from other tree nodes, and thus the text change detection could be more accurate. We would also like to discuss the need for semantic change detection. Some recent studies [19, 20] were done on algorithms for semantic change detection on XML documents. This approach is also interesting for scientific articles. The following semantic analyses would be useful while reviewing changes:

Edit Area : Provide semantic information on the area of the article where the change was made (title, author, section, table, etc.).
Edit Type: Provide semantic information on the update type, for example an addition, deletion, correction, reorder, styling, rephrasing, etc.
Edit Score: Number or percent of changes for the entire document or per edit area and edit type.

Knowing that a new author was added, a specific section was changed by X% or one part of a section was used to create a new section is often initially more interesting for the reviewer than the exact change that was applied.

7 CONCLUSION

In this article, we tested 12 implementations of existing state-of-the-art and commercial XML diff algorithms (Table 1) for their suitability in detecting changes in JATS versions of academic articles. We started with a performance evaluation with the XMLDiffAnalyzer

script we developed, followed by a manual evaluation of the resulting deltas. The overall performance was rather acceptable among the algorithms. XMLDiff presents large execution times for real-life JATS documents and would not be suitable for regular usage. Concerning memory, all 12 tested algorithms have acceptable performance. The delta file size allowed us to identify three algorithms (DeltaXML, DiffMK and XMLDiff) annotating changes on one of the compared XML files. Xdiff is not able to detect text edits and works only on tree edit detections. XOp prints the entire parent tree of a modified text element, which is not convenient for interpretation. The delta file size of DiffXML scales exponentially as the number of changes increase because this algorithm interprets most of the changes as move operations. In the second part of the evaluation, we analysed the delta output produced by each algorithm for specific types of edit operations an author usually performs while revising an article. The results show that none of the algorithms are fully compliant with our expectations. The top three algorithms have success scores of 25/36 for JNDiff, 15/36 for XyDiff and 14/36 for XCC. The main weakness shown by most of the algorithms, except JNDiff, is the inability to represent edit operations in complex environments. Moreover, the existing edit operations detected while comparing two XML articles (Insert, Delete, Update, Move) are not sufficient to efficiently represent the differences between two versions of an academic article. In order to implement a suitable JATS diff algorithm, it would be necessary to detect, in addition to the existing edits, style changes, citable object changes, text moves, tree splits and merges, tree upgrades and downgrades and semantic differences. Once these are added, it would be possible to properly detect, represent and store most of the edit operations presented in Section 3. Having such a mechanism in place would be useful for an eventual future versioning system for academic articles.

ACKNOWLEDGMENTS

We would like to thank Dr. Shu-Kun Lin and MDPI for their financial and technical support regarding our study. We would also like to thank DeltaXML for providing us with a free trial license in order to perform the testing of their XMLCompare algorithm.

REFERENCES

[1] Gioele Barabucci. 2013. Introduction to the Universal Delta Model. In *Proceedings of the 2013 ACM Symposium on Document Engineering* (Florence, Italy) *(DocEng '13)*. Association for Computing Machinery, New York, NY, USA, 47–56. https://doi.org/10.1145/2494266.2494284

[2] Sudarshan S Chawathe and Hector Garcia-Molina. 1997. Meaningful change detection in structured data. *ACM SIGMOD Record* 26, 2 (1997), 26–37.

[3] Sudarshan S Chawathe, Anand Rajaraman, Hector Garcia-Molina, and Jennifer Widom. 1996. Change detection in hierarchically structured information. *Acm Sigmod Record* 25, 2 (1996), 493–504.

[4] Yan Chen, Sanjay Madria, and Sourav Bhowmick. 2004. DiffXML: Change Detection in XML Data. In *Database Systems for Advanced Applications*, YoonJoon Lee, Jianzhong Li, Kyu-Young Whang, and Doheon Lee (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 289–301.

[5] Paolo Ciancarini, Angelo Di Iorio, Carlo Marchetti, Michele Schirinzi, and Fabio Vitali. 2016. Bridging the gap between tracking and detecting changes in XML. *Software: Practice and Experience* 46, 2 (2016), 227–250.

[6] Greégory Cobena. 2003. *Change management of semi-structured data on the Web.* Ph.D. Dissertation. Institut Polytechnique de Paris.

[7] Gregory Cobena, Serge Abiteboul, and Amelie Marian. 2002. Detecting changes in XML documents. In *Proceedings 18th International Conference on Data Engineering.* IEEE, San Jose, CA, USA, 2002, 41–52.

[8] Grégory Cobéna, Talel Abdessalem, and Yassine Hinnach. 2004. A comparative study of XML diff tools.

[9] Daniel Hottinger and Franziska Meyer. 2005. XML-diff-algorithmen. (2005).

[10] James Wayne Hunt and M Douglas MacIlroy. 1976. *An algorithm for differential file comparison.* Bell Laboratories Murray Hill, USA.

[11] Falk Langhammer. 2004. Bauen statt modellieren. *iX* 2 (2004), 100–103. https://shop.heise.de/katalog/bauen-statt-modellieren

[12] Tancred Lindholm, Jaakko Kangasharju, and Sasu Tarkoma. 2006. Fast and Simple XML Tree Differencing by Sequence Alignment. In *Proceedings of the 2006 ACM Symposium on Document Engineering* (Amsterdam, The Netherlands) *(DocEng '06)*. Association for Computing Machinery, New York, NY, USA, 75–84. https://doi.org/10.1145/1166160.1166183

[13] Lorenz Schori. 2020. Delta.js - A JavaScript diff and patch engine for DOM trees. http://znerol.github.io/node-delta/

[14] Salvatore Manfreda, Matthew F. McCabe, Pauline E. Miller, Richard Lucas, Victor Pajuelo Madrigal, Giorgos Mallinis, Eyal Ben Dor, David Helman, Lyndon Estes, Giuseppe Ciraolo, Jana Müllerová, Flavia Tauro, M. Isabel De Lima, João L. M. P. De Lima, Antonino Maltese, Felix Frances, Kelly Caylor, Marko Kohv, Matthew Perks, Guiomar Ruiz-Pérez, Zhongbo Su, Giulia Vico, and Brigitta Toth. 2018. On the Use of Unmanned Aerial Systems for Environmental Monitoring. *Remote Sensing* 10, 641 (2018), 1–28. https://doi.org/10.3390/rs10040641

[15] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval.* Cambridge University Press, USA.

[16] Webb Miller and Eugene W Myers. 1985. A file comparison program. *Software: Practice and Experience* 15, 11 (1985), 1025–1040.

[17] Eugene W Myers. 1986. AnO (ND) difference algorithm and its variations. *Algorithmica* 1, 1-4 (1986), 251–266.

[18] Norman Walsh. 2015. DiffMK. https://sourceforge.net/projects/diffmk/

[19] Alessandreia Oliveira, Leonardo Murta, and Vanessa Braganholo. 2014. Towards Semantic Diff of XML Documents. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing* (Gyeongju, Republic of Korea) *(SAC '14)*. Association for Computing Machinery, New York, NY, USA, 833–838. https://doi.org/10.1145/2554850.2554893

[20] Alessandreia Oliveira, Gabriel Tessarolli, Gleiph Ghiotto, Bruno Pinto, Fernando Campello, Matheus Marques, Carlos Oliveira, Igor Rodrigues, Marcos Kalinowski, Uéverton Souza, et al. 2018. An efficient similarity-based approach for comparing XML documents. *Information Systems* 78 (2018), 40–57.

[21] Sebastian Rönnau and Uwe M Borghoff. 2009. Versioning XML-based office documents. *Multimedia Tools and Applications* 43, 3 (2009), 253–274.

[22] Sebastian Rönnau and Uwe M Borghoff. 2012. XCC: change control of XML documents. *Computer Science-Research and Development* 27, 2 (2012), 95–111.

[23] Sebastian Rönnau, Christian Pauli, and Uwe M. Borghoff. 2008. Merging Changes in XML Documents Using Reliable Context Fingerprints. In *Proceedings of the Eighth ACM Symposium on Document Engineering* (Sao Paulo, Brazil) *(DocEng '08)*. Association for Computing Machinery, New York, NY, USA, 52–61. https://doi.org/10.1145/1410140.1410151

[24] Sebastian Rönnau, Geraint Philipp, and Uwe M. Borghoff. 2009. Efficient Change Control of XML Documents. In *Proceedings of the 9th ACM Symposium on Document Engineering* (Munich, Germany) *(DocEng '09)*. Association for Computing Machinery, New York, NY, USA, 3–12. https://doi.org/10.1145/1600193.1600197

[25] Sebastian Rönnau, Jan Scheffczyk, and Uwe M. Borghoff. 2005. Towards XML Version Control of Office Documents. In *Proceedings of the 2005 ACM Symposium on Document Engineering* (Bristol, United Kingdom) *(DocEng '05)*. Association for Computing Machinery, New York, NY, USA, 10–19. https://doi.org/10.1145/1096601.1096606

[26] Stanley M Selkow. 1977. The tree-to-tree editing problem. *Information processing letters* 6, 6 (1977), 184–186.

[27] Walter F Tichy. 1984. The string-to-string correction problem with block moves. *ACM Transactions on Computer Systems (TOCS)* 2, 4 (1984), 309–321.

[28] W3C. 2016. Extensible Markup Language (XML). https://www.w3.org/XML

[29] W3C. 2018. HtmlDiff. https://www.w3.org/wiki/HtmlDiff

[30] W3C. 2019. HtmlDiff. https://docs.deltaxml.com/support/latest/articles-and-papers-9340757.html

[31] Yuan Wang, David J DeWitt, and J-Y Cai. 2003. X-Diff: An effective change detection algorithm for XML documents. In *Proceedings 19th International Conference on Data Engineering (Cat. No. 03CH37405)*. IEEE, Bangalore, India, 519–530.