20th IMEKO TC4 International Symposium and
18th International Workshop on ADC Modelling and Testing
Research on Electric and Electronic Measurement for the Economic Upturn
Benevento, Italy, September 15-17, 2014

Measuring the domain-oriented quality of diff algorithms

Gioele Barabucci[1], Paolo Ciancarini[2], Angelo Di Iorio[2], Fabio Vitali[2]

[1]*Cologne Center for eHumanities, Universität zu Köln, Germany*
[2]*Department of Computer Science and Engineering, University of Bologna, Italy*

Abstract –

Software changes over time: not only the source code but also its models and documentation, configuration files, messages payloads, and so on. Diff algorithms help users to track such evolution. These algorithms vary a lot in terms of efficiency, resources consumption, internal strategies and final results. The focus of this paper is on measuring and comparing the quality of the output produced by these algorithms. There is no univocal definition of quality in this context. We propose a top-down approach to characterise deltas, in order to investigate which is the most suitable algorithm for a given domain and a given class of users. Some measuring experiments on XML diff algorithms are presented too.

I. INTRODUCTION

Diff tools are widely used to support software developers, architects, and maintaners. These tools take two files as input and compute their difference, according to a given set of change operations. They can be evaluated mainly by comparing their time and space performance. Almost all the experiments in the literature follow the same pattern: the authors first compare the computational complexity and the execution time of the algorithms, then they evaluate the quality of the results, see for instance [7][9][10].

Surprisingly only a few quality measures have been defined and applied, mostly on the size of the produced deltas[4]. There is now a growing interest in characterizing more precisely the quality of diff deltas, in order to design algorithms that produce an output that is *easier to interpret* and that *works better in specific domains*, for instance on ontological data [8] or on generic textual documents [5].

The focus of this work is to define metrics able to capture peculiarities of deltas and that can be exploited to select the most appropriate algorithm for a given domain. We introduce a framework for measuring the quality of diffs through an objective evaluation process. The basic idea consists of extracting numerical indicators from deltas (such as the number of detected changes, the number of high-level changes, the number of elements listed in the description of each change) and aggregating them into more complex quantitative metrics that, in turn, can be associated to quality requirements.

The framework gives users a tool to analyze in a more precise way the behaviour of heterogeneous algorithms, using their own internal conceptual model and strategies. It is also worth remarking that this model is general enough to deal with streams of text, lists, trees or graphs, so that the same evaluation process can be applied to multiple algorithms and domains. Here we present some experiments on XML diff tools.

The paper is structured as follows. Section ii. describes some related works. Section iii. discusses how the same concept of 'quality' can have different meanings in different domains. Section iv. describes our approach. The application of the metrics is presented in Section v., before concluding in Section vi..

II. RELATED WORK

It is hard to compare the quality of the output of diff algorithms. First of all, because different algorithms usually produce different deltas that are all correct. There is a further tricky issue. As highlighted by [9] "*all approaches make use of different delta models, which makes it difficult to measure the quality of the resulting deltas*". In fact, each algorithm uses its own internal model and recognizes its own set of changes.

The most used parameter to measure the quality of diff algorithms is its ability to reduce the dimension of the delta. There are two main approaches to calculate such a dimension: either measuring the size of the file[4] or measuring the *edit distance*, i.e. the number of edits listed in the delta [9], [6].

A refinement of the first approach has been proposed for measuring the quality of Faxma[7]. The authors compared the size of compressed deltas, since "*[they] expect to get results that are less dependent on the encoding and more closely related to the amount of actual information*". The minimization of the edit distance was further refined by deploying *edit cost models*[3]. The idea is to pre-define a cost for each type of change and to measure the overall cost of the delta as the sum of the costs of each detected change. In the same paper the authors introduce an algorithm that minimizes this cost.

In other cases, the quality of a delta has been associated to the capability of humans to interpret and exploit the changes it contains. This quality is much more difficult to define, as it involves the nature of changes and the human analysis of the output. In [5] the authors introduced the notion of *naturalness* of a diff algorithm, i.e. the "*capability of producing an edit script that an author would recognize as containing the changes she/he effectively performed when editing a document*". The authors presented a taxonomy of natural operations on literary documents and an algorithm, called JNDiff, able to capture most of those operations. The importance of letting users tune the quality of a diff algorithm in relation to the set of detectable high-level changes was also stressed by [8]. The authors, in fact, introduced the idea of *viewpoints* (i.e., each ontology designer has her/his own needs and should be able to define the set of complex changes she/he is interested in).

III. QUALITY OF DELTAS IN DIFFERENT DOMAINS

Users in different domains have different requests and expectations on deltas and their quality. In this section we sketch out some scenarios in which such heterogeneity is evident. The scenarios are identified with labels S1, S2, . . . , Sn that will be used throughout the paper.

S1: Programmer developing code. Some verbosity is appreciated by developers who review source code during the development. Contextual information helps developers to understand what has changed and where.

S2: Programmer browsing code history. In other cases a programmer could prefer not having contextual information, for instance when looking for unchanged components in a large code base.

S3: Sysadmin diffing files. Minimizing the size of the delta (between different versions of files) is an important requirement for sysadmins, since they expect to store a lot of differences and do not want to waste space and bandwidth.

S4: XML database admin comparing records. An admin that compares versioned dumps of a XML database also needs to minimize the size of the delta and to tame the computational complexity. Contextual information is also less relevant here than in other cases.

S5: Author revising text-centric documents. There are many other situations where detected changes are mainly meant to be interpreted by human readers, for instance when diffing textual documents, reports, articles, and so on. In those cases a more precise delta – able to capture frequent operations on those documents – is preferable.

Though incomplete, this discussion shows that there is not a single definition of what makes a diff algorithm *good*. The suitability of a diff algorithm depends on the deltas it creates and how fit these deltas are for use in a certain scenario. There is, thus, the need to create metrics to measure multiple qualities of a delta.

IV. A TOP-DOWN APPROACH TO MEASURE THE QUALITY OF DELTAS

The goal of this work is to define some metrics that capture peculiarities of deltas and can can be used to select the most appropriate algorithm for a given domain. We propose a top-down approach inspired by Goal-Question-Metric (GQM) [2]. In this section we introduce our metrics: we first identify which information is needed to measure them and then we give a formal definition of each metric.

Throughout the section, we will use two XML documents shown in Figure 1 and some possible deltas generated by algorithms with different characteristics, shown in Figure 2.

The definition of the metrics actually requires a preliminary step: we need to establish a common conceptual schema on top of which metrics will be built and that allows us to apply the same evaluation process to algorithms very different among each other. This work is built on top of UniDM [1], a unified conceptual model able to abstract the characteristics of deltas. Space limits prevent us to go into details of UniDM. Here, we describe very briefly what a delta is in the model and which information it may contain.

In UniDM a Delta is a collection of changes that an algorithm detects, together with some relations between these changes. There are two kinds of changes: *atomic* and *complex*. Complex changes are obtained by aggregating atomic ones into more meaningful structures. An example is given in Figure 2: the change C.1 of Delta 2 can be seen as the aggregation of the changes B.1, B.2 and B.3 of Delta1bis. The detection of moves is a further very common example of complex change. Moving a paragraph from a document, for instance, can be expressed as a single high-level change or as a combination of the deletion of the original paragraph and the combination of another one that is an exact clone but in a different location. A key aspect of the model is the distinction between the top level changes and the other ones, that have been grouped in other changes.

There are many kinds of relations between changes that can be stored in a delta. The two most common kinds are the *application order* relation, used to define a partial or total order in which changes should be applied, and the *grouping* relations, used to bundle different changes that are somehow related. These relations are used to express various properties of a delta, such as its reversibility or the fact that they express higher-level changes taken together.

A. *Delta quantitative indicators*

There are various objective features that can be extracted from a delta and the changes it contains. The most basic one is the number of changes produced by an algorithm; for algorithms that are able to recognize complex changes, one can also extract the number of such changes and rate

Listing 1. Source document (S)

```
<book>
   <info>
      <author>John Doe</author>
      <rights licence="cc-by-sa"/>
   </info>
</book>
```

Listing 2. Target document (T)

```
<book>
   <info>
      <author><name>John</name>
      <surname>Doe</surname></author>
      <rights licence="cc-by-sa"/>
   </info>
</book>
```

Fig. 1. Example documents.

Listing 3. Delta 1

```
A.1: REMOVE-TEXT(John Doe)
A.2: ADD-ELEM(<name>,<author>)
A.3: ADD-TEXT(John,<name>)
A.4: ADD-ELEM(<surname>,
                              <author>)
A.5: ADD-TEXT(Doe,<surname>)
```

Listing 4. Delta 1bis

```
B.1: REMOVE-TEXT(John)
B.2: ADD-ELEM(<name>,<author>)
B.3: ADD-TEXT(John,<name>)
B.4: REMOVE-TEXT(Doe)
B.5: ADD-ELEM(<surname>,<author>)
B.6: ADD-TEXT(Doe,<surname>)
B.7: REMOVE-CHAR(' ')
```

Listing 5. Delta 2

```
C.1: WRAP(John, <name>)
C.2: WRAP(Doe, <surname>)
```

Listing 6. Delta 3

```
D.1: IDENTIFY-NAME(John, <author>)
D.2: IDENTIFY-SURNAME(Doe, <author>)
```

Fig. 2. Possible deltas for <author>

their complexity. Structural information about the hierarchical organization of changes can also be measured automatically. The main properties that can be extracted from each change expressed in a UniDM delta are:

population the total number of changes of which a change is composed of, including itself and the recursive closure of the encapsulated changes;

depth the length of the longest path from the change to an atomic change, in the graph of its encapsulated changes;

width the number of distinct changes encapsulated directly inside the change;

num touched elements the number of distinct items that are included as part of the change or of the encapsulated changes; these items might be included because they have actually changed or as contextual information; an item might be a single character, a text node, an XML element according to the concrete application domain;

num modified elements the minimum number of elements that must be modified by the change to fulfill its purpose; in other words, the number of elements included in the change since they were actually modified.

There are also features that can be calculated on the whole delta, with a direct count or by combining measures of individual changes:

num top-level the number of changes that are not encapsulated in any other change;

population the sum of the population property of all changes;

num touched elements the sum of the touched elements of all changes;

num modified elements the minimum number of distinct pieces of information that must be modified in order to turn the original document into the modified one.

From these basic properties, other specialized properties can be derived taking into account only certain kinds of changes:

prop*$_{\mathbf{k}}$** is the property ***prop calculated taking into account only changes of kind ***k***. For example num-top-level$_{complex}$ is the number of *complex* changes that are not encapsulated in any other change.

Table 1 reports the values of these properties on the four sample deltas shown in Figure 2. Most of them have been calculated by aggregating the values of individual changes composing the delta.

	population	# touched	# modified	# top-level (complex)
Delta 1	5	17	3	5 (0)
Delta 1bis	7	17	3	7 (0)
Delta 2	5	11	3	2 (2)
Delta 3	7	11	3	2 (2)

Table 1. Measuring delta indicators on the sample deltas.

B. Metrics

Now that we have a reference formalization of deltas and their properties, we can give brief description of our metrics. A complete exposition can be found in [1].

B..1 Length

The dimension of a delta has been widely used to evaluate the quality of diff algorithms. In our model, that dimension is captured by the *Length* metric, indicating the number of changes listed in the top level of a delta.

$$Length\,(\delta) = \text{num-top-level}\,(\delta)$$

The values of Length for the deltas in Figure 2 are: 5 for delta 1, 7 for delta 1bis, 2 for delta 2 and 2 for delta 3, corresponding to the values in the last column of Table 1.

B..2 Terseness

Another interesting aspect is to know whether or not the delta contains redundant information. That is why we introduce the concept of *Terseness*. The *Terseness* measures the ratio between the number of elements that have been included or referred to in the delta because they have changed and the number of context elements that appear in the delta but were not actually modified. It basically measures the amount of contextual information.

$$Terseness\,(\delta) = \frac{\text{num-modified-elements}\,(\delta)}{\text{num-touched-elements}\,(\delta)}$$

Considering the modified and touched elements reported in Table 1, the values of Terseness are: 0.17 ($\frac{3}{17}$) for delta 1 and delta 1bis, 0.27 ($\frac{3}{11}$) for delta 2 and delta 3.

B..3 Conciseness

It is also interesting to discover which strategies an algorithm uses to detect a given number of edits and, in particular, its capability of aggregating atomic changes into complex ones. The *Conciseness* of a delta indicates how much the delta has been simplified by the encapsulation of changes. It indicates the ratio between the number of changes in the top level of a delta and the overall number of changes. The formula has been designed so to return values ranging from 0, for deltas that are not concise, to 1, for very concise deltas.

$$Conciseness\,(\delta) = 1 - \frac{\text{num-top-level}\,(\delta)}{\text{population}\,(\delta)}$$

The values of Conciseness are: 0 $(1 - \frac{5}{5})$ for delta 1, 0 $(1 - \frac{7}{7})$ for delta 1bis, 0.6 $(1 - \frac{2}{5})$ for delta 2 and 0.71 $(1 - \frac{2}{7})$ for delta 3.

B..4 Compositeness

A further dimension is needed to measure if an algorithm tends to prefer complex changes more than atomic ones. The *Compositeness* of a delta was introduced to show how much of its conciseness is due to the use of complex changes. It indicates the ratio between the number of complex changes and the overall number of changes in the top level of the delta.

$$Compositeness\,(\delta) = \frac{\text{num-top-level}_{complex}\,(\delta)}{\text{num-top-level}\,(\delta)}$$

The values of Compositeness are: 0 ($\frac{0}{5}$) for delta 1, 0 ($\frac{0}{7}$) for delta 1bis, 1 ($\frac{2}{2}$) for delta 2 and 1 ($\frac{2}{2}$) for delta 3.

B..5 Deep Compositeness

The compositeness of a delta does not take into account the depth of complex changes. The *Deep Compositeness* was introduced to measure how an algorithm encapsulates complex changes into other complex changes. The deep compositeness values ranges from 0, for deltas that contain no complex changes and thus no deep compositeness, up to 1, for deltas with with many deep changes. The formula has been designed to asymptotically approach the maximum value, so to accommodate deltas with arbitrary average depth $\bar{d}$.

$$DeepCompositeness\,(\delta) = 1 - \frac{1}{1 + \log\left(\bar{d}\right)}$$

where

$$\bar{d} = \frac{\sum_{c \in \hat{C}} \text{depth}\,(c)}{\text{num-top-level}}$$

The values of Deep compositeness are: 0 $(1 - \frac{1}{1+\log(1)})$ for delta 1 and delta 1bis, 0.52 $(1 - \frac{1}{1+\log(3)})$ for delta 2 and 0.58 $(1 - \frac{1}{1+\log(4)})$ for delta 3.

V. APPLYING THE METRICS

In this section we explain how to apply our metrics to existing algorithms and we present the results of their application on three well-known XML diff tools.

A. *A two-phase process to apply metrics*

The delta model and metrics are independent from a specific data format. Such a level of abstraction makes it possible to capture relevant peculiarities of changes and to compare heterogeneous input. On the other hand, a further step is required for measuring actual delta files: these files need to be translated into UniDM, on top of which metrics can be applied.

In fact the process of applying metrics to study algorithms can be refined in two steps: (i) **interpretation**: the pre-processing analysis in which the algorithm's internal model is made explicit and mapped into UniDM; (ii) **evaluation**: the actual measurement of atomic indicators and aggregated metrics on the pre-processed delta. While the second step can be generalized and automated, the first one is different for each algorithm and requires users to understand the basic functioning of that algorithm. However this interpretation is to be done once for each algorithm and, from then on, the quality of [the output of] the algorithm can be evaluated in a fully objective manner.

B. *Experimental results on XML diff*

In order to test the applicability of the metrics on real deltas we studied three well-known XML diff tools: JNDiff [5], XyDiff [4] and Faxma [7]. We ran experiments on the same dataset used to evaluate the 'naturalness' of JNDiff[1]. It consists of five documents, each available in two versions. These documents are very heterogeneous for size, internal structure and source.

We first interpreted the results of each algorithm, in order to map them into our model. For JNDiff and XyDiff we constructed augmented output from the output generated by the reference implementations. For Faxma we patched the code to produce a low-level dump of the changes detected internally [2].

The evaluation phase was fully automated. We also included an implementation of the trivial algorithm for diffing XML files that, when diffing documents A and B, produces a delta with two operations: the deletion of the whole document A and the insertion of the whole document B. Our goal is to verify if the metrics highlight the 'bad' behavior of such algorithm: even if correct, in fact, the output of this algorithm provides too little information and is very difficult to use.

The overall value of terseness has been approximated. An exact value of terseness could be calculated by knowing exactly what changed and by looking manually at exact modified nodes for each algorithm. A faster but still reliable process consists of considering the minimum amount of modified text as calculated by external binary diff tools on the isolated content.

Some normalization was also needed to make the values of length comparable. In fact, absolute values of deltas produced by different algorithms vary a lot and cannot be compared directly. To solve the problem, we first calculated the magnitude of each length, as $L_{algorithm} = \log_{10}(Length(\delta))$ and then calculated:

$$LengthNormalized(\delta) =$$
$$= 1 - \frac{L_{algorithm}}{\max(L_{JNDiff}, L_{XyDiff}, L_{faxma}, L_{trivial})}$$

Thus, we obtained an *length normalized* score between 0 and 1, with higher values for lower (normalized) values of length. This normalization is also the reason why, for each document, there is always an algorithm with length normalized value equal to zero (the one with highest length).

To better highlight differences between algorithms and some common behaviors we summarized the results in radar charts[3]. The results on documents BIBLIO, PROTOCOL and DL2221 are graphically shown in Figure 3. These documents differ a lot in terms of format and dimensions. Nonetheless all the web graphs for a certain algorithm look similar, while there are big differences in the graphs generated by different algorithms. This is an important finding: the algorithms show a quite regular behavior and the metrics are able to capture that behavior.

The other interesting point is that these plots highlight clearly some peculiarities of each algorithm. First of all, consider the results of the trivial algorithm. It scored a very high length normalized (since only two changes were detected) but obtained a zero score for conciseness, compositeness and deep compositeness since it does not try to give a higher-level interpretation of changes. Note that a mere evaluation of the number of edits, i.e. of the length metric, would have given a very high score to the delta produced by this algorithm.

Consider also the behavior of Faxma with regard to conciseness, compositeness and deep compositeness. These dimensions are related to each other and capture whether or not an algorithm is able to aggregate changes into complex ones. Conciseness is high for Faxma since the algorithm builds large complex changes in the form of "move" changes. Faxma also tries hard in generating as few changes a possible. These two factors lead Faxma to produce deltas with low values for compositeness and for deep compositeness, though these values are higher than other

[1]The test-suite is available at http://diff.cs.unibo.it/jndiff/tests/

[2]The original code is available at https://github.com/ept/fuego-diff; the new code is available at https://github.com/gioele/fuego-diff

[3]Full results in tabular form are also available at http://diff.cs.unibo.it/metrics/experiments/

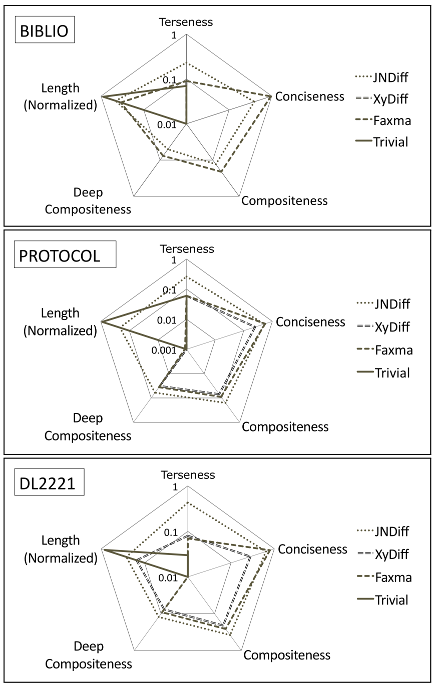

Fig. 3. Applying metrics to deltas of XML diff algorithms

algorithms that aggregate less changes into more complex ones. However, this helps Faxma to obtain high performance and low consumption of space.

The low values of XyDiff in almost all metrics also confirm some of its characteristics. The algorithm, in fact, gives much importance to performance and does only consider the size of the delta as indicator of quality. Being a greedy algorithm, it is not able to refine the already generated changes: for example the deletion of three sibling nodes would appear as three separate atomic change deleting one node, not as a complex change enclosing all the three siblings; this reduces compositeness, deep compositeness and conciseness.

The results on JNDiff are also insightful. The algorithm, in fact, works very well on textual changes and is able to aggregate fine-grained modifications on text nodes into complex changes. On the other hand, such JNDiff is not equally precise on elements and some structural changes that could be aggregated are left disjoint. This is the reason why results are generally good but there is no clear dominance in any dimension. It is also interesting to note that JNDiff tries to limit the amount of nodes involved in each change: this is confirmed by the value of terseness, that is the highest in all cases.

The experiments on the other two documents were consistent with what we discussed so far about metrics and algorithms' peculiarities. We do not present them here for the sake of brevity.

VI. CONCLUSIONS AND FUTURE WORKS

In this paper we presented a set of metrics and a methodology for the evaluation of the deltas produced by diff algorithms. The results in Section v. show that the values of these metrics reflect properties of the analysed algorithms. In our opinion the current set of properties is good enough to be used to evaluate the deltas and, consequently, the algorithms that produced them. The overall approach and the UniDM model, on the other hand, can be used straightforwardly to investigate additional metrics.

REFERENCES

[1] G. Barabucci. Introduction to the universal delta model. In *Proceedings of the 2013 ACM Symposium on Document Engineering*, DocEng '13, pp. 47–56, New York, NY, USA, 2013. ACM.

[2] V. R. Basili, G. Caldiera, and H. D. Rombach. The goal question metric approach. In *Encyclopedia of Software Engineering*. Wiley, 1994.

[3] S. S. Chawathe, A. Rajaraman, H. Garcia-Molina, and J. Widom. Change detection in hierarchically structured information. In *SIGMOD Conference*, pp. 493–504. ACM Press, 1996.

[4] G. Cobéna, S. Abiteboul, and A. Marian. Detecting changes in XML documents. In *Proceedings of the 18th International Conference on Data Engineering, San Jose, CA, USA, February 26 - March 1, 2002*, pp. 41–52. IEEE Computer Society, 2002.

[5] A. Di Iorio, M. Schirinzi, F. Vitali, and C. Marchetti. A natural and multi-layered approach to detect changes in tree-based textual documents. In *Enterprise Information Systems, ICEIS 2009*, Vol. 24 of *Lecture Notes in Business Information Processing*, pp. 90–101. Springer, 2009.

[6] E. Leonardi and S. S. Bhowmick. Xandy: A scalable change detection technique for ordered XML documents using relational databases. *Data & Knowledge Engineering*, 59(2):476–507, 2006.

[7] T. Lindholm, J. Kangasharju, and S. Tarkoma. Fast and simple XML tree differencing by sequence alignment. In *ACM Symposium on Document Engineering*, pp. 75–84. ACM, 2006.

[8] P. Plessers, O. D. Troyer, and S. Casteleyn. Understanding ontology evolution: A change detection approach. *J. Web Sem.*, 5(1):39–49, 2007.

[9] S. Rönnau, G. Philipp, and U. M. Borghoff. Efficient change control of XML documents. In *ACM Symposium on Document Engineering*, pp. 3–12. ACM, 2009.

[10] Y. Wang, D. J. DeWitt, and J. yi Cai. X-diff: an effective change detection algorithm for XML documents. In *Proceedings of the 19th International Conference on Data Engineering, March 5-8, 2003, Bangalore, India*, pp. 519–530. IEEE Computer Society, 2003.