Int J Software Informatics, Volume 9, Issue 2 (2015), pp. 259–277
International Journal of Software and Informatics, ISSN 1673-7288
©2015 by ISCAS. All rights reserved.

E-mail: ijsi@iscas.ac.cn
http://www.ijsi.org
Tel: +86-10-62661040

Similarity-Based Diff, Three-Way Diff and Merge

Serge Autexier

(German Research Centre for Artficial Intelligence (DFKI), Bremen, Germany)

Abstract Semi-structured documents and data pervade modern workflows in all areas. Collaborative work and version management rely on effective, automatic difference analysis and three-way difference analysis tools. In our effort to develop a three-way difference analysis for tree-structured documents we developed a kernel three-way difference algorithm which extends the equality-based procedures, such as GNU diff3, by considering the similarity of documents in the difference analysis as well as to ignore the order of data if that is semantically suitable. As a result we obtain difference analysis algorithms that can be more fine-tuned to the application domain. Moreover, the equality-based counter-parts of our three-way difference analysis algorithms has the idempotency property, which the current three-way diff algorithms lacks.

Key words: version control; collaborative work; similarity of sequences and multisets; semi-structured data and documents

Autexier S. Similarity-based diff, three-way diff and merge. ***Int J Software Informatics*, Vol.9, No.2 (2015): 259–277. http://www.ijsi.org/1673-7288/9/i217.htm**

1 Introduction

Collaborative working on and version management of digital documents and data requires good automatic difference analysis and merge algorithms. Standard algorithms for difference analysis for text documents and sequences of atomic data elements use dynamic programming[8] to compute common subsequences between two or more subsequences. Similarly, the problem has been studied for tree-structured and even general graph structured models in the context of programming languages[9] and domain-specific models[11,14,18]. In previous work[3] we developed difference analysis algorithms for XML documents based on the similarity of subtrees for which the notion could be customized. That has been extended to develop a three-way difference algorithm for XML-trees, similarily based on the similarity of subtrees, and which has been applied and tested in Refs. [1,2]. The central parts of this algorithm are similarity-based three-way difference analysis procedures for mixed ordered and unordered sequences, which we present in this article and compare with existing three-way difference analysis algorithms for sequences.

The most well known implementation is the GNU diff-algorithm[7] based on Ref. [16]. Merging of documents is considered either as merging variants of a document based on a common ancestor document or simply merging two documents

This work is sponsored by German Federal Ministry of Education and Research (BMBF) under grant FKZ 01IW13001 (SPECIFIC)
Corresponding author: Serge Autexier, Email: serge.autexier@dfki.de
Received 2015-01-11; Revised 2015-04-29; Accepted 2015-05-15.

without a common ancestor. The first problem class is known as a *three-way diff* and the methods to tackle the problem fall into two categories: the *operational* approach tries to merge the sequence of edit-operations performed to obtain the different variants and the *state-based* approach only considers the documents and is typically based on the difference analysis between the documents. The most well-known problem here is the GNU diff3 algorithm[17]. Though it has been and still is extensively used its properties have only been studied in 2007 in Ref. [10] showing that three properties one may intuitively expect to hold for diff3 actually do not hold. Furthermore, as digital data and documents beyond pure sequences of atomic elements are more and more widespread, tree-based difference and merge algorithms were developed[5,13,12]. In Ref. [3] we considered the explicit use of similarities between semi-structured documents to compute subsequences of two documents that are most similar to each other and does not require the identified elements to be equal. In that work we also addressed the similarity-based difference analysis problem for sequences, where the order of the elements does not matter, i.e. for multisets. In neither work we considered the similarity-based three-way difference analysis. The contributions of this article are (1) an intuition how matters change when moving from equality to similarity, (2) a coherent, uniform solutions for similarity-based difference analysis, three-way difference analysis and merge for lists and multisets, (3) solutions for mixed sequences where some parts are to be considered as lists and others as multisets, and (4) we show that our equality-based three-way diff algorithms for sequences has the idempotency properties, which does not hold for GNU diff3 and alike.

The article is organised as follows: In Section 2 we provide some motivations for using similarity instead of equality for difference analysis of sequences of structured objects. Section 3 provides intuitions how the problem changes when considering similarities and introduces coherent formulations of the difference analysis between two versions of documents depending on whether they are lists or multisets and show that the similarity-based approach is a conservative extension of the equality-based approach. Section 4 presents the four basic three-way difference analysis algorithms as well as the mixed version and the derivation of a merge algorithm for documents without common ancestor. It also includes the proof of the idempotency of the equality-based three-way difference analysis for lists.

2 Motivation to Use Similarity instead of Equality

When comparing two unstructured objects, typically their syntactic equality is considered because it is sufficient to entail semantic equality. For instance, two numbers are semantically equal, if they are syntactically equal; similarly for characters. When dealing with structured objects, things change: syntactical equality still entails semantic equality, but two semantically equal objects are not necessarily syntactically equal. For instance, consider arithmetic expressions: with some background knowledge about $+$ the two expressions $1 + 2$ and $2 + 1$ are semantically equal, but not syntactically equal. In this paper we are interested in developing difference analysis algorithms, that do not distinguish semantically equal structured objects. For instance, consider SVG[6] expressions describing rectangles X and Y

```
X: <rect width="300" height="100" style="fill:red;stroke−width:3;stroke:black"/>
Y: <rect width="300" height="100" style="fill:blue;stroke−width:3;stroke:black"/>
Z: <rect width="200" height="100" style="fill:green;stroke−width:3;stroke:black"/>
```

If we are only interested in the shapes, then the first two rectangles are equal. If we are also interested in the colors, then they are no longer equal, but still more similar than they are with the rectangle Z. In that case, when comparing two documents wrt. their differences, similar structures must be recognised. We will now briefly formalize the properties and relationships between syntactic and semantic equality and similarity.

Given two objects o, o', we denote the *syntactic (resp. semantic) equality* of these objects by oo' (oo'). Both relations are reflexive, commutative, and transitive. Moreover, it must hold that if oo' implies oo'. The similarity of two objects o, o' is $oo' \in [0 \dots 1]$ such that oo' implies $oo' = 1$. If $oo' = 0$ then *o and o' are dissimilar.*

3 SDiff: Most Similar Subsequences and Multi-Subsets

To determine the longest common subsequence of two sequences s and s' is a standard problem which has an efficient solution using dynamic programming. The result is a sequence c which is contained as a subsequence in both s and s'. Moving from equality of objects in the sequence to similarity changes the problem slightly, because there is no longer one sequence which occurs in both s and s', but subsequences c and c' which are respectively subsequences of s and s', and which have a maximal similarity. As an example consider the two sequences

$$\begin{aligned} s &:= \text{ all yellow birds can fly} \\ s' &:= \text{ some red bird can fly} \end{aligned}$$

The longest common subsequence of both is “can fly”. The edit script from s to s' is

$$-\text{all}, +\text{some}, -\text{yellow}, +\text{pink}, -\text{birds}, +\text{bird}, \text{can}, \text{fly}$$

where $-W$ denotes to remove the word W, $+W$ denotes to add word W and words not prefixed by $+$ or $-$ remain in place.

Now we define the similarity of two words based on twice the length of the longest subsequence of characters divided by the sums of the lengths of both words. Then the similarity of fly and fly is 1.0, the similarity of yellow and red is $\#(\text{e})/(\#(\text{yellow}) + \#(\text{red})) = 1/9 = 0.11$, where $\#(w)$ denotes the length of the word w, and the similarity of bird and birds is $2 \times \#(\text{bird})/(\#(\text{bird}) + \#(\text{birds})) = 8/9 = 0.88$. Note that although the lengths of the common subsequences of both red and some with yellow are both 1, red is preferred because of the smaller denominator $\#(\text{yellow}) + \#(\text{red})$ compared to $\#(\text{yellow}) + \#(\text{some})$. Now the subsequence of s that is most similar to a subsequence of s' is “yellow birds can fly” and the respective subsequence of s' is “red bird can fly”. The edit script from s to s' now also has to accommodate those words, that are similar but not equal. In the edit script we prefix these with U followed by the new version. This results in the following edit script

$$-\text{all}, +\text{some}, \text{Ured}, \text{Ubird}, \text{can}, \text{fly}$$

The result of applying this edit script to s is a sequence which is syntactically equal to s'. This is because syntactic and semantic equalities coincide for the words in this example. The situation changes for instance if we add arithmetic expressions to the language and consider the sequences

$$s := (1 + 3) \text{ equals } 4$$
$$s' := (3 + 1) \text{ equals } 4$$

Here $(1 + 3)$ and $(3 + 1)$ are objects and we include as semantics the commutativity property of $+$. Now both expressions are semantically equal but syntactically different. As similarity notion we use the same for words as before and the largest common subset of operands of two arithmetic expressions to acknowledge the commutativity of $+$. As a result, the subsequences of s and s' that are most similar are s and s' themselves. The edit script is also empty if we consider semantic equality of expressions, namely:

(1+3), equals, 4

Not considering the semantic equality, but keeping the same similarity notion would result in the edit script

U(3+1), equals, 4

to obtain a sequence which is syntactically equal to s'. Depending on the situation and the analysis to conduct either approach can be sensible.

Summarizing this discussion, when moving from equality to similarity when comparing two sequences s and s' one has two degrees of freedom: (i) the choice of the similarity notion, and (ii) the choice to consider semantic equality compatible with the similarity notion or not. Furthermore, the goal is not to find one sequence which is a subsequence of both s and s'. Instead, the goal is to find subsequences of s and s' of equal length and which are maximally similar. Finally, if one would opt for using semantic equality when comparing s and s', then the application of the computed edit script to s results in a sequence that is semantically equal, but not necessarily syntactically equal to s'. Throughout the rest of this paper we stick to syntactic equality, though.

3.1 Formalization

In this section we formalize sequences and the similarity-based common subsequences depending on whether they are lists or multisets. Our notion of sequences is explicitly formulated over an arbitrary subset of the natural numbers as index sets rather than just lists of elements of some basic domain. This subsumes the standard approach and enables for a uniform treatment using morphisms identifying elements in the sequences.

Definition 1 (Sequence). Let I be a set of positive natural numbers and $s_i, i \in I$ be elements of a given domain D. Then $s := (s_i)_{i \in I}$ denotes the *sequence* $s_{i_1} \dots s_{i_n}$ such that $|I| = n$ and $\forall 1 \leqslant k < l \leqslant n$ holds $i_k < i_l$. The length of the sequence is the cardinality of I. The set of sequences is $\mathcal{S}$.

The equality of two sequences now depends on whether they are lists or multisets. We also provide the equality for mixed sequences, where some parts are to be considered as a list and other parts a multiset. This shows the benefit of building the notion of sequences over arbitrary index sets. Examples for sequences, where the subsequence comparison can benefit from a mixed treatment are:

- In LaTeX documents mostly the order of the lines and environments matter. However, for environments that can float like figures or tables explicitly declared as such, the order and relation to the rest of the text could be ignored in order to find a common subdocument
- Similarly, in the programming language Scala in class declarations, variables can be declared and assignment throughout the whole class and the order matters. However, the method declarations are independent of these declarations and assignments and their order does not matter.

We now define the different equality notions for sequences, depending on whether we consider them as lists, as multisets or as being mixed of both.

Definition 2 (Sequence Equalities). Let $s := (s_i)_{i \in S}$ and $t := (t_j)_{j \in T}$ be sequences.

(i) *Sequence Equality:* s and t are *sequence equal*, iff there exists a total bijection $\mu : S \to T$ such that $\forall i \in S . s_i = t_{\mu(i)}$ and $\forall i, j \in S . i < j \Rightarrow \mu(i) < \mu(j)$.

(ii) *Multiset Equality:* s and t are *multiset equal*, iff there exists a total bijection $\mu : S \to T$ such that $\forall i \in S . s_i = t_{\mu(i)}$.

(iii) *Mixed Equality:* Let $S_L \uplus S_M = S$ and $T_L \uplus T_M = T$ be partitionings. Then s and t are *mixed equal* wrt. that partitioning iff $(s_i)_{i \in S_L}$ and $(t_j)_{j \in T_L}$ are sequence equal and $(s_i)_{i \in S_M}$ and $(t_j)_{j \in T_M}$ are multiset equal.

Subsequently, we also define the similarities of two sequences.

Definition 3 (Similarity). A similarity on elements of a domain D is a total binary function $\mathrm{sim} : D^2 \to [0, 1]$ such that for all $a, b \in D$ with $a = b$ holds $\mathrm{sim}(a, b) = 1$. The equality on D induces the *canonical similarity* on D defined by

$$\mathrm{sim}^{=}(u, v) := \begin{cases} 1 & \text{if } u = v \\ 0 & \text{otherwise} \end{cases}$$

Definition 4 (Similarity of Sequences). Let $s := (s_i)_{i \in S}$ and $t := (t_j)_{j \in T}$ be sequences, sim a similarity notion on elements, and $\mu : S \hookrightarrow T$ a partial bijection of domain $\mathrm{Dom}(\mu)$ and image $\mathrm{Im}(\mu)$. The *similarity* of s and t wrt. μ and sim is

$$\mathrm{sim}_{\mu}(s, t) := \frac{2}{|S| + |T| + 2} \left(1 + \sum_{i \in \mathrm{Dom}(\mu)} \mathrm{sim}(s_i, t_{\mu(i)}) \right)$$

Having set the stage, we define the common subsequence notions dependent on whether we consider equality or similarity and whether the sequences are lists or multisets (see also Fig. 1).

	Equality	Similarity
Lists	longest common subsequence, Definition 5	most similar subsequence, Definition 6
Multisets	largest multi-subset, Definition 7	most similar multi-subset, Definition 8
Mixed Lists/Multisets	largest mixed common subsequence, Definition 9	most similar common subsequence, Definition 10

Figure 1. The four common subsequence notions

Definition 5 (Longest Common Subsequence). Let $s := (s_i)_{i\in S}$ and $t := (t_j)_{j\in T}$ be sequences. A partial mapping $\mu : S \hookrightarrow T$ is a *common subsequence* of s and t if it is a partial bijection of length $|Dom(\mu)|$ such that $\forall i, j \in Dom(\mu)$ holds: $i < j$ iff $\mu(i) < \mu(j)$. μ is a *longest common subsequence* iff all other common subsequences have size less or equal to $|Dom(\mu)|$.

Definition 6 (Most Similar Subsequence). Let $s := (s_i)_{i\in S}$ and $t := (t_j)_{j\in T}$ be sequences and sim a similarity notion on elements. A partial mapping $\mu : S \hookrightarrow T$ is a *similar subsequence* of s and t if it is a partial bijection of similarity $\mathrm{sim}_\mu(s,t)$ such that (i) $\forall i \in Dom(\mu)$ holds: $sim(s_i, t_{\mu(i)}) > 0$, and (ii) $\forall i, j \in Dom(\mu)$ holds: $i < j$ iff $\mu(i) < \mu(j)$. μ is a *longest common subsequence* iff all other similar subsequences have similarities less or equal to $sim(\mu)$.

Definition 7 (Largest Multi-Subset). Let $s := (s_i)_{i\in S}$ and $t := (t_j)_{j\in T}$ be sequences. A partial mapping $\mu : S \hookrightarrow T$ is a *multi-subset* of s and t if it is a partial bijection of length $|Dom(\mu)|$. μ is a *largest multi-subset* iff all other common subsequences have size less or equal to $|Dom(\mu)|$.

Definition 8 (Most Similar Multi-Subset). Let $s := (s_i)_{i\in S}$ and $t := (t_j)_{j\in T}$ be sequences and sim a similarity notion on elements. A partial mapping $\mu : S \hookrightarrow T$ is a *similar multi-subset* of s and t if it is a partial bijection of similarity $sim_\mu(s,t)$ such that $\forall i \in Dom(\mu)$ holds: $sim(s_i, t_{\mu(i)}) > 0$. μ is a *most similar multi-subset* iff all other similar multi-subsets have similarities less or equal to $sim(\mu)$.

Based on the notions for list-like sequences and multiset sequences, we can define the notions for mixed sequences which have a list-like and a multiset part.

Definition 9 (Largest Mixed Common Subsequence). Let $s := (s_i)_{i\in S}$ and $t := (t_j)_{j\in T}$ be sequences with partitions $S_L \uplus S_M = S$ and $T_L \uplus T_M = T$. A partial mapping $\mu : S \hookrightarrow T$ is a *mixed common subsequence* of s and t if $\mu_{|S_s}$ (resp. $\mu_{|S_M}$) is a common subsequence of $(s_i)_{i\in S_L}$ and $(t_j)_{j\in T_L}$ (resp. a common multi-subset of $(s_i)_{i\in S_M}$ and $(t_j)_{j\in T_M}$) of size $Dom(\mu)$. It is a *largest mixed common subsequence* if any other mixed common subsequence has smaller or equal size.

Definition 10 (Most Similar Mixed Common Subsequence). Let $s := (s_i)_{i\in S}$ and $t := (t_j)_{j\in T}$ be sequences with partitions $S_L \uplus S_M = S$ and $T_L \uplus T_M = T$ respectively, and sim a similarity notion on elements. A partial mapping $\mu : S \hookrightarrow T$ is a *similar mixed common subsequence* of s and t if $\mu_{|S_L}$ (resp. $\mu_{|S_M}$) is a similar subsequence of $(s_i)_{i\in S_L}$ and $(t_j)_{j\in T_L}$ (resp. a similar multi-subset of $(s_i)_{i\in S_M}$ and $(t_j)_{j\in T_M}$) of similarity $sim_\mu(s,t)$. It is a *most similar mixed subsequence* of any other similar mixed subsequence has smaller or equal similarity.

Finally, we show that the similarity-based notions are conservative extensions of the equality-based notions, which enables us to consider the equality-based versions as special cases of the similarity-based versions.

Lemma 1 (Conservativity). The similarity-based notions of common subsequences are conservative extensions of the equality-based notions. That is let $s := (s_i)_{i \in S}$ and $t := (t_j)_{j \in T}$ be sequences and $\mu : S \hookrightarrow T$ a partial bijection. Then it holds:

(i) If μ is common subsequence of s and t, then it is a similar subsequence of s and t of similarity $\frac{2+2|Dom(\mu)|}{2+m+n}$ wrt. the canonical similarity.

(ii) If μ is multi-subset of s and t, then it is a similar multi-subset of s and t of similarity $\frac{2+2|Dom(\mu)|}{2+m+n}$ wrt. the canonical similarity.

(iii) If μ is a mixed common subsequence μ of s and t, then it is a similar mixed common subsequence of s and t of similarity $\frac{2+2|Dom(\mu)|}{2+m+n}$ wrt. the canonical similarity.

Proof: (i) First, it holds for all $i \in Dom(\mu)$ that $s_i = t_{\mu(i)}$. Thus, by definition $\mathrm{sim}(s_i, t_{\mu(i)}) = 1$, which is greater than 0. Second:

$$\begin{aligned}\mathrm{sim}(s,t) &= \frac{2}{m+n+2}\left(2 + \sum_{i \in Dom(\mu)} \mathrm{sim}(s_i, t_{\mu(i)})\right) = \frac{2}{m+n+2}\left(1 + \sum_{i \in Dom(\mu)} 1\right) \\ &= \frac{2}{m+n+2}(1 + |Dom(\mu)|) = \frac{2+2|Dom(\mu)|}{m+n+2}\end{aligned}$$

□

Corollary 1. Let s and t be two sequences of equal length n and μ be their longest common subsequence (resp. largest multi-subset and longest mixed common subsequence wrt. fixed partitions) such that $|Dom(\mu)| = n$. Then μ is a most similar subsequence (resp. most similar multi-subset and most similar mixed common subsequence) of s and t and of similarity 1.

Proof: From Lemma 1 follows that $\mathrm{sim}_\mu(s,t) = \frac{2+2|Dom(\mu)|}{2+n+n}$. Since $Dom(\mu) = n$ it follows $\mathrm{sim}_\mu(s,t) = \frac{2+2n}{2+n+n} = 1$. □

4 SDiff3: Similarity-Based State-Based Synchronizer

Typically the result of three-way difference analysis between two variants a and b with repsect to a common ancestor document o is a merged document o' possibly containing conflict information. We follow the approach taken in[10] and consider a three-way difference analysis algorithm to have three outputs: a new version a' for a, where all non-conflicting changes from a are included in b' and o', all non-conflicting changes of b included in o' and a'. o' contains all non-conflicting changes from a and b and the old state from o for conflicting parts. We denote the triples (a, o, b) and $(a', o'b')$ as *configurations* and a three-way difference analyser computing (a', o', b') from (a, o, b) is a *synchronizer*.

Formally, following[10], a *configuration* is a triple $(a, o, b) \in \mathcal{S} \times \mathcal{S} \times \mathcal{S}$ usually written in the more suggestive notation $a \leftarrow o \rightarrow b$ to emphasize, that o is the common origin of a and b.

A *synchronizer* is a function that maps an input configuration $a \leftarrow o \rightarrow b$ to a new configuration $a' \leftarrow o' \rightarrow b'$. $a \leftarrow o \rightarrow b \Rightarrow a' \leftarrow o' \rightarrow b'$ is a *run* of the synchronizer; it is *conflict-free* if $a' = o' = b'$.

We will introduce four synchronizers (see Fig. 2), depending on whether the sequences are lists or multisets and whether the elements are to be compared by equality or by similarity. The algorithm sDiff3$^{=}_{L}$ for lists and equality comparison is comparable to the standard diff3-algorithm studied in [10]. The way our synchronization works is to take into account parts of the similarity between the two variants. It is thus different from the standard diff3 and in contrast to that sDiff3$^{=}_{L}$ is idempotent.

	Equality	Similarity
Lists	sDiff3$^{=}_{L}$ Section 4.1	sDiff3$^{\sim}_{L}$ Section 4.3
Multisets	sDiff3$^{=}_{S}$ Section 4.2	sDiff3$^{\sim}_{S}$ Section 4.4

Figure 2. The four synchronizer functions

4.1 Equality-based sequence synchronizer

This is a synchronizer which is a variant of the standard diff3 algorithm from[17]. In contrast to this ours uses the similarity between the two variants a and b as guidance for the common subsequence mapping between o and a resp. o and b. This is achieved by taking the longest common subsequence c_{ab} between a and b and computing the longest common subsequence c_{init} between o and c_{ab}. Now we consider only common subsequence between o and a (resp. b), which contain c_{init}. This is in contrast to diff3, which considers the common subsequence between o and a and resp. o and b independently. Of course the longest common subsequence obtained in our approach may be shorter. On the other hand we gain the advantage that the alignment of parts that have remained unchanged between o, a and b is better in general. Once the alignment is obtained, we identify conflicts and the elements added or deleted locally in one of the variants without conflicting with anything else. The conflicts are those subsequence that have to belong together and which are different in a and b and were different in o.

This whole approach of obtaining the alignment is crucial to our variant of three-way diff and formalized as follows:

Definition 11 (Subsequence Alignment). Let $a = (a_j)_{j \in A}, o = (o_i)_{i \in O}, b = (b_k)_{k \in B}$ be sequences. An *alignment* $\alpha = \langle \mu_O, \mu_A, \mu_B \rangle$ *from* o *to* a *and* b consists of three strictly increasing morphisms $\mu_O : O \rightarrow \mathbb{N}$, $\mu_A : A \rightarrow \mathbb{N}$ and $\mu_B : B \rightarrow \mathbb{N}$ such that it holds

$$\mu_O(i) = \mu_A(j) \Rightarrow o_i = a_j$$
$$\mu_O(i) = \mu_B(k) \Rightarrow o_i = b_k$$

$$\mu_A(j) = \mu_B(k) \Rightarrow a_j = b_k$$

The alignment image $\mathrm{Im}(\alpha)$ is $\mathrm{Im}(\mu_O) \cup \mathrm{Im}(\mu_A) \cup \mathrm{Im}(\mu_B)$ and its *size* is the cardinality of its image.

The *conflict sets* $\mathrm{Conflicts}(\alpha)$ of the alignment are all largest convex subsets of $\mathrm{Im}(\alpha)$ which are unions of pairwise disjoint non-empty convex subsets of $\mathrm{Im}(\mu_O)$, $\mathrm{Im}(\mu_A)$, and $\mathrm{Im}(\mu_B)$ *(AddAddConflicts).*

The *local additions* $\mathrm{add}_A(\alpha)$ *of A (resp.* $\mathrm{add}_B(\alpha)$ *for B)* is the subset of $\mathrm{Im}(\mu_A)$ (resp. $\mathrm{Im}(\mu_B)$) which is disjoint from $\mathrm{Im}(\mu_O)$, $\mathrm{Im}(\mu_B)$ (resp. $\mathrm{Im}(\mu_A)$), and any conflict set.

The *deletion set* $\mathrm{del}_O(\alpha)$ *of O* is the subset of $\mathrm{Im}(\mu_O)$ which is disjoint from $\mathrm{Im}(\mu_A)$ and $\mathrm{Im}(\mu_B)$.

The induced configuration $a' \leftarrow o' \rightarrow b'$ is defined by

$$\begin{aligned}
M &= (\mathrm{Im}(\mu_O) \cap \mathrm{Im}(\mu_A) \cap \mathrm{Im}(\mu_B)) \cup \mathrm{add}_A(\alpha) \cup \mathrm{add}_B(\alpha)\\
O' &= M \cup \bigcup_{C \in \mathrm{Conflicts}(\alpha)} (C \cap \mathrm{Im}(\mu_O))\\
A' &= M \cup \bigcup_{C \in \mathrm{Conflicts}(\alpha)} (C \cap \mathrm{Im}(\mu_A))\\
B' &= M \cup \bigcup_{C \in \mathrm{Conflicts}(\alpha)} (C \cap \mathrm{Im}(\mu_B))
\end{aligned}$$

and $o' = (o'_i)_{i \in O'}$, $a' = (a'_j)_{j \in A'}$ and $b' = (b'_k)_{k \in B'}$ where

$$o'_i = \begin{cases} o_{\mu_O^{-1}(i)} & \text{if } i \in \mathrm{Im}(\mu_O) \cap \mathrm{Im}(\mu_A) \cap \mathrm{Im}(\mu_B) \\ o_{\mu_O^{-1}(i)} & \text{if } i \in \bigcup_{C \in \mathrm{Conflicts}(\alpha)} (C \cap \mathrm{Im}(\mu_O)) \\ a_{\mu_A^{-1}(i)} & \text{if } i \in \mathrm{add}_A(\alpha) \\ b_{\mu_B^{-1}(i)} & \text{if } i \in \mathrm{add}_B(\alpha) \end{cases}$$

$$a'_j = \begin{cases} o'_j & \text{if } j \in M \\ a_{\mu_A^{-1}(j)} & \text{if } j \in \bigcup_{C \in \mathrm{Conflicts}(\alpha)} (C \cap \mathrm{Im}(\mu_A)) \end{cases}$$

$$b'_k = \begin{cases} o'_k & \text{if } k \in M \\ b_{\mu_B^{-1}(k)} & \text{if } k \in \bigcup_{C \in \mathrm{Conflicts}(\alpha)} (C \cap \mathrm{Im}(\mu_B)) \end{cases}$$

Based on that notion of alignment we can define our equality-based three-way diff for lists as follows.

Definition 12 ($\mathrm{sDiff3}_L^=$ **for Sequences).** Let $a = (a_j)_{j \in A}, o = (o_i)_{i \in O}, b = (b_k)_{k \in B}$ be sequences. Let μ_{AB} be the most similar subsequence for a and b, and μ_{OSubA} (resp. μ_{OSubB}) be the most similar subsequences of o and $a_{|\,\mathrm{Dom}(\mu_{AB})}$ (resp. $b_{|\,\mathrm{Im}(\mu_{AB})}$). Furthermore, let μ_{OA} (resp. μ_{OB}) be the most similar subsequence of O and A (resp. B) which extends μ_{OSubA} (resp. μ_{OSubB}). Finally, let $\alpha = \langle \mu_O, \mu_A, \mu_B \rangle$ be the smallest alignment such that

$$\mu_O(i) = \mu_A(j) \Leftrightarrow \mu_{OA}(i) = j$$

$$\mu_O(i) = \mu_B(k) \Leftrightarrow \mu_{OB}(i) = k$$
$$\mu_A(j) = \mu_B(k) \Leftrightarrow \mu_{AB}(j) = k$$

The result of the SDiff3 algorithm for sequences is the configuration induced by α.

Example 4.1. As a first example consider der sequences $a = [1, 4, 3, 2, 9]$, $o = [1, 2, 3, 7, 9]$ and $b = [1, 3, 6, 8, 9]$. The application of the standard diff3 algorithm yields the following alignment

a	1	4, 3, 2	9
o	1	2, 3, 7	9
b	1	3, 6, 8	9

This has a large conflict between $4, 3, 2$ in a, $2, 3, 7$ in o and $3, 6, 8$ in b due to the switch of 2 and 3 from o to a. The reason is, that the 3 is not part of the three-way matching, because matching o and a choses the "wrong" subsequence $[1, 2, 9]$, while from o to b only $[1, 3, 9]$ can be selected.

In our sDiff3$^{=}_{L}$ considering the longest common subsequence $[1, 3, 9]$ between a and b is used to guide the mapping from o to a and b. This is done by first computing the common subsequence between o and $[1, 3, 9]$, which is in this case again $[1, 3, 9]$. This is considered as the fixed part to compute the mappings from o to a and b. This results in the following alignment

a	1	–	4	3	–	2	–	–	9
o	1	2	–	3	7	–	–	–	9
b	1	–	–	3	–	–	6	8	9

This creates only a conflict between 7 in o, 2 in a and $6, 8$ in b yielding the induced configuration $1, 4, 3, 2, 9 \leftarrow 1, 4, 3, 7, 9 \rightarrow 1, 4, 3, 6, 8, 9$. The edit script for o is

$$1, -2, +4, 3, -7, \mathrm{C} < [2], [6, 8] >, 9$$

where $\mathrm{C} < [2], [6, 8] >$ denotes the conflict of having the sequence $[2]$ in one variant and the sequence $[6, 8]$ in the other variant.

As a second example consider the sequences $a = [1, 4, 3, 6, 9]$, $o = [1, 2, 3, 9]$ and $b = [1, 3, 8, 9]$. The run of the synchronizer sDiff3$^{=}_{L}$ on $a \leftarrow o \rightarrow b$ yields the alignment

a	1	–	4	3	6	–	9
o	1	2	–	3	–	–	9
b	1	–	–	3	–	8	9

This creates no conflict between 6 in a and 8 in b yielding the edit script

$$1, -2, +4, 3, +6, +8, 9$$

Note that the diff3 algorithm returns a conflict between 6 and 8 and the non-conflicting result of our synchronizer yields the analogous results than the ELAM algorithm[9].

4.1.1 Properties

The standard diff3 algorithm is not idempotent as shown in Ref. [10]. An example are the sequences $a = [1,2,4,6,8]$, $o = [1,2,3,4,5,5,5,6,7,8]$, and $b = [1,4,5,5,5,6,2,3,4,8]$. Using the standard diff3 twice results in

$$\begin{aligned} &[1,2,4,6,8] \leftarrow [1,2,3,4,5,5,5,6,7,8] \rightarrow [1,4,5,5,5,6,2,3,4,8]\\ \Rightarrow&[1,2,4,6,8] \leftarrow [1,2,3,4,6,7,8] \rightarrow [1,4,6,2,3,4,8]\\ \Rightarrow&[1,4,6,2,4,6,8] \leftarrow [1,4,6,2,4,6,7,8] \rightarrow [1,4,6,2,4,8] \end{aligned}$$

which shows, that diff3 is not idempotent. Using sDiff3$_L^=$ we have a conflict-free run

$$\begin{aligned} &[1,2,4,6,8] \leftarrow [1,2,3,4,5,5,5,6,7,8] \rightarrow [1,4,5,5,5,6,2,3,4,8]\\ \Rightarrow&[1,4,6,2,3,4,8] \leftarrow [1,4,6,2,3,4,8] \rightarrow [1,4,6,2,3,4,8] \end{aligned}$$

The reason for the conflict-free run is again because the longest common subsequence $[1,4,6,8]$ is used as a guidance for the mapping between o and a resp. b.

As an example illustrating that it also holds for non-conflict-free runs consider the configuration from the first example in Example 4.1. A second run of sDiff3$_L^=$ is

$$1,4,3,2,9 \leftarrow 1,4,3,9 \rightarrow 1,4,3,6,8,9 \Rightarrow 1,4,3,2,9 \leftarrow 1,4,3,9 \rightarrow 1,4,3,6,8,9$$

which leaves the configuration unchanged.

Theorem 4.1. (Idempotency) The algorithm sDiff3$_L^=$ is idempotent.

P roof: Let $a \leftarrow o \rightarrow b \Rightarrow a' \leftarrow o' \rightarrow b' \Rightarrow a'' \leftarrow o'' \rightarrow b''$. We show that $a'' = a'$, $o'' = o'$ and $b'' = b'$.

First, for $a' \leftarrow o' \rightarrow b'$ being obtained from $a \leftarrow o \rightarrow b$ by an alignment α we know that the longest common subsequence $\mu_{A'B'}$ between a' and b' is the one with indexes from M by construction. Furthermore, the longest common subsequence of o and $\mu_{A'B'}$ also has domain M. Hence, for the smallest subsequence alignment α' holds that (i) the conflicts sets Conflict(α') are identical to Conflicts(α), and (ii) the local additions of A' and B' are empty. Hence, A'', O'' and B'' are identical to A', O' and B' and thus $o'' = o'$, $a'' = a'$, and $b'' = b'$. □

4.2 Equality-based multiset synchronizer

The next synchronizer is for multisets. The key idea adapted from the previous list-like version consists of using the intersection between the variants and then consider the intersection between o and that intersection. This is actually the intersection of o, a and b. Since we have multisets, there are no conflicts when computing the merge. However, in order to emphasize the relationship to the solution for list-like sequences we use an analogous alignment formulation. This will also be useful when considering similarities, where we will have conflicts in general and the mapping obtained from the alignment is necessary to determine the new configuration.

Definition 13 (Multi-Subset Alignment). Let $a = (a_j)_{j\in A}, o = (o_i)_{i\in O}, b = (b_k)_{k\in B}$ be sequences. A *multi-subset alignment* $\alpha = \langle\mu_O, \mu_A, \mu_B\rangle$ *from* o *to* a *and* b consists of three morphisms $\mu_O : O \rightarrow \mathbb{N}$, $\mu_A : A \rightarrow \mathbb{N}$ and $\mu_B : B \rightarrow \mathbb{N}$ such that it holds

$$\mu_O(i) = \mu_A(j) \Rightarrow o_i = a_j$$

$$\mu_O(i) = \mu_B(k) \Rightarrow o_i = b_k$$
$$\mu_A(j) = \mu_B(k) \Rightarrow a_j = b_k$$

The alignment image $\mathrm{Im}(\alpha)$ is $\mathrm{Im}(\mu_O) \cup \mathrm{Im}(\mu_A) \cup \mathrm{Im}(\mu_B)$ and its *size* is the cardinality of its image.

The *local additions* $\mathrm{add}_A(\alpha)$ *of* A *(resp.* $\mathrm{add}_B(\alpha)$ *for* B*)* is the subset of $\mathrm{Im}(\mu_A)$ (resp. $\mathrm{Im}(\mu_A)$) which is disjoint from $\mathrm{Im}(\mu_O)$ and $\mathrm{Im}(\mu_B)$ (resp. $\mathrm{Im}(\mu_A)$).

The *deletion set* $\mathrm{del}_O(\alpha)$ *of* O is the subset of $\mathrm{Im}(\mu_O)$ which is disjoint from $\mathrm{Im}(\mu_A)$ and $\mathrm{Im}(\mu_B)$.

The induced configuration $a' \leftarrow o' \rightarrow b'$ is defined by

$$O' = (\mathrm{Im}(\mu_O) \cap \mathrm{Im}(\mu_A) \cap \mathrm{Im}(\mu_B)) \cup \mathrm{add}_A(\alpha) \cup \mathrm{add}_B(\alpha)$$
$$A' = B' = O'$$

and $o' = (o'_i)_{i \in O'}$, $a' = (a'_j)_{j \in A'}$ and $b' = (b'_k)_{k \in B'}$ where

$$o'_i = \begin{cases} o_{\mu_O^{-1}(i)} & \text{if } i \in \mathrm{Im}(\mu_O) \cap \mathrm{Im}(\mu_A) \cap \mathrm{Im}(\mu_B) \\ a_{\mu_A^{-1}(i)} & \text{if } i \in \mathrm{add}_A(\alpha) \\ b_{\mu_B^{-1}(i)} & \text{if } i \in \mathrm{add}_B(\alpha) \end{cases}$$

$$a'_j = b'_j = o'_j \text{for all } j \in O'$$

Based on that we can define the equality-based three-way diff for multisets.

Definition 14 ($\mathrm{sDiff3}_S^=$ **for Multi-Subsets).** Let $a = (a_j)_{j \in A}$, $o = (o_i)_{i \in O}$, $b = (b_k)_{k \in B}$ be sequences. Let μ_{AB} be the largest multi-subset for a and b, and μ_{OSubA} (resp. μ_{OSubB}) be the largest multi-subset of o and $a_{|\,\mathrm{Dom}(\mu_{AB})}$ (resp. $b_{|\,\mathrm{Im}(\mu_{AB})}$). Furthermore, let μ_{OA} (resp. μ_{OB}) be the largest multi-subset of O and A (resp. B) which extends μ_{OSubA} (resp. μ_{OSubB}). Finally, let $\alpha = \langle \mu_O, \mu_A, \mu_B \rangle$ be the smallest multi-subset alignment such that

$$\mu_O(i) = \mu_A(j) \Leftrightarrow \mu_{OA}(i) = j$$
$$\mu_O(i) = \mu_B(k) \Leftrightarrow \mu_{OB}(i) = k$$
$$\mu_A(j) = \mu_B(k) \Leftrightarrow \mu_{AB}(j) = k$$

The result of the SDiff3 algorithm for sequences is the configuration induced by α.

Example 4.2. To illustrate the equality-based multiset synchronizer consider the following number sequences $a = [1, 2, 5, 4]$, $o = [1, 2, 3, 4]$, and $b = [1, 4, 5]$. A run of the synchronizer $a \leftarrow o \rightarrow b$ finds the equality multi-subset alignment

a	1	2	–	4	5
o	1	2	3	4	–
b	1	–	–	4	5

Here 5 and 4 in a had to exchange positions. The resulting edit script for o is

$$1, -2, -3, 4, +5.$$

4.3 Similarity-based sequence synchronizer

This is the first extension to take similarities into account. The basic principle is the same as for equality-based sequence to use the most similar subsequences between a and b and again with o as a guidance to determine the similar subsequence between o and a resp. b. As in the equality case, the obtained subsequences between o and a resp. b are not the most similar subsequences, but that way the alignment identifying the most similar subsequences between all three is better. Regarding the resulting configuration, the use of similarities induces a few more conflicts and modifications compared with the equality case. Indeed, conflicts may now also arise between elements which have been identified in all three sequences, but are not equal. To illustrate that we consider *colored numbers* (n, c) where n is a number and $c \in \{red, green, blue\}$ a color. The similarity between colored numbers is defined by

$$(n,c)(n',c') = \begin{cases} 0 & \text{if } n \neq n' \\ 0.5 & \text{if } n = n', c \neq c' \\ 1.0 & \text{if } n = n', c = c' \end{cases}$$

Now, we may have in o a red 1, which has been changed to a blue 1 in a and a green 1 in b. Thus, all three are identified and part of the most similar subsequence. However, this is a conflict which version to chose, blue or green, for the new configuration. A similar conflict occurs, if a number has been changed color in one variant but deleted in the other. We denote the first kind of conflicts as *UpdateUpdate*-conflicts and the latter as *UpdateDelete*-conflicts.

Finally, consider a colored number being unchanged in one variant, but which changed color in the other variant. In this case this is a *local modification* and we have to include it in the merge along with the local additions as before.

As before, we formalize first the similarity-based alignment of subsequences.

Definition 15 (Similar Subsequence Alignment). Let $a = (a_j)_{j \in A}, o = (o_i)_{i \in O}, b = (b_k)_{k \in B}$ be sequences. An *alignment* $\alpha = \langle \mu_O, \mu_A, \mu_B \rangle$ *from o to a and* b consists of three strictly increasing morphisms $\mu_O : O \to \mathbb{N}$, $\mu_A : A \to \mathbb{N}$ and $\mu_B : B \to \mathbb{N}$ such that it holds

$$\begin{aligned} \mu_O(i) = \mu_A(j) &\Rightarrow o_i a_j > 0 \\ \mu_O(i) = \mu_B(k) &\Rightarrow o_i b_k > 0 \\ \mu_A(j) = \mu_B(k) &\Rightarrow a_j b_k > 0 \end{aligned}$$

The alignment image $\mathrm{Im}(\alpha)$ is $\mathrm{Im}(\mu_O) \cup \mathrm{Im}(\mu_A) \cup \mathrm{Im}(\mu_B)$ and its *size* is the cardinality of its image.

The *conflict sets* $\mathrm{Conflicts}(\alpha)$ of the alignment are

- all largest convex subsets of $\mathrm{Im}(\alpha)$ which are unions of pairwise disjoint non-empty convex subsets of $\mathrm{Im}(\mu_O)$, $\mathrm{Im}(\mu_A)$, and $\mathrm{Im}(\mu_B)$ *(AddAddConflicts)*
- the *UpdateDeleteSets* of A and B

$$\left\{ n \in \mathrm{Im}(\mu_o) \cap \mathrm{Im}(\mu_A) | n \notin \mathrm{Im}(\mu_B) \text{ and } o_{\mu_O^{-1}}(n) \neq a_{\mu_A^{-1}} \right\}$$
$$\left\{ n \in \mathrm{Im}(\mu_o) \cap \mathrm{Im}(\mu_B) | n \notin \mathrm{Im}(\mu_A) \text{ and } o_{\mu_O^{-1}}(n) \neq b_{\mu_B^{-1}} \right\}$$

– the *UpdateUpdateSet* of A and B which are those $n \in \mathrm{Im}(\mu_A) \cap \mathrm{Im}(\mu_B) \cap \mathrm{Im}(\mu_O))$ such that $a_{\mu_A^{-1}(n)} \neq o_{\mu_O^{-1}(n)}$, $b_{\mu_B^{-1}(n)} \neq o_{\mu_O^{-1}(n)}$ and $a_{\mu_A^{-1}(n)} \neq b_{\mu_O^{-1}(n)}$.

The *local modifications* $\mathrm{mod}_A(\alpha)$ *of* A *(resp.* $\mathrm{mod}_B(\alpha)$ *for* B*)* are those $n \in \mathrm{Im}(\mu_A) \cap \mathrm{Im}(\mu_B) \cap \mathrm{Im}(\mu_O))$ such that $a_{\mu_A^{-1}(n)} \neq o_{\mu_O^{-1}(n)}$ and $b_{\mu_B^{-1}(n)} = o_{\mu_O^{-1}(n)}$ (resp. $b_{\mu_B^{-1}(n)} \neq o_{\mu_O^{-1}(n)}$ and $a_{\mu_A^{-1}(n)} = o_{\mu_O^{-1}(n)}$).

The *local additions* $\mathrm{add}_A(\alpha)$ *of* A *(resp.* $\mathrm{add}_B(\alpha)$ *for* B*)* is the subset of $\mathrm{Im}(\mu_A)$ (resp. $\mathrm{Im}(\mu_B)$) which is disjoint from $\mathrm{Im}(\mu_O)$, $\mathrm{Im}(\mu_B)$ (resp. $\mathrm{Im}(\mu_A)$), and any conflict set.

The *deletion set* $\mathrm{del}_O(\alpha)$ *of* O is the subset of $\mathrm{Im}(\mu_O)$ which is disjoint from $\mathrm{Im}(\mu_A)$ and $\mathrm{Im}(\mu_B)$.

The induced configuration $a' \leftarrow o' \rightarrow b'$ is defined by

$$\begin{aligned}
M &= (\mathrm{Im}(\mu_O) \cap \mathrm{Im}(\mu_A) \cap \mathrm{Im}(\mu_B)) \cup \mathrm{add}_A(\alpha) \cup \mathrm{add}_B(\alpha) \cup \mathrm{mod}_A(\alpha) \cup \mathrm{mod}_B(\alpha)\\
O' &= M \cup \bigcup_{C \in \mathrm{Conflicts}(\alpha)} (C \cap \mathrm{Im}(\mu_O))\\
A' &= M \cup \bigcup_{C \in \mathrm{Conflicts}(\alpha)} (C \cap \mathrm{Im}(\mu_A))\\
B' &= M \cup \bigcup_{C \in \mathrm{Conflicts}(\alpha)} (C \cap \mathrm{Im}(\mu_B))
\end{aligned}$$

and $o' = (o'_i)_{i \in O'}$, $a' = (a'_j)_{j \in A'}$ and $b' = (b'_k)_{k \in B'}$ where

$$o'_i = \begin{cases} o_{\mu_O^{-1}(i)} & \text{if } i \in \mathrm{Im}(\mu_O) \cap \mathrm{Im}(\mu_A) \cap \mathrm{Im}(\mu_B)\\ o_{\mu_O^{-1}(i)} & \text{if } i \in \bigcup_{C \in \mathrm{Conflicts}(\alpha)} (C \cap \mathrm{Im}(\mu_O))\\ a_{\mu_A^{-1}(i)} & \text{if } i \in \mathrm{add}_A(\alpha) \cup \mathrm{mod}_A(\alpha)\\ b_{\mu_B^{-1}(i)} & \text{if } i \in \mathrm{add}_B(\alpha) \cup \mathrm{mod}_B(\alpha) \end{cases}$$

$$a'_j = \begin{cases} o'_j & \text{if } j \in M\\ a_{\mu_A^{-1}(j)} & \text{if } j \in \bigcup_{C \in \mathrm{Conflicts}(\alpha)} (C \cap \mathrm{Im}(\mu_A)) \end{cases}$$

$$b'_k = \begin{cases} o'_k & \text{if } k \in M\\ b_{\mu_B^{-1}(k)} & \text{if } k \in \bigcup_{C \in \mathrm{Conflicts}(\alpha)} (C \cap \mathrm{Im}(\mu_B)) \end{cases}$$

Based on that we define the similarity-based three-way diff for sequences as follows.

Definition 16 ($\mathrm{sDiff3}_L^{\sim}$ **for Similar Sequences).** Let $a = (a_j)_{j \in A}, o = (o_i)_{i \in O}, b = (b_k)_{k \in B}$ be sequences. Let μ_{AB} be the most similar subsequence for a and b, and μ_{OSubA} (resp. μ_{OSubB}) be the most similar subsequences of o and $a_{|\mathrm{Dom}(\mu_{AB})}$ (resp. $b_{|\mathrm{Im}(\mu_{AB})}$). Furthermore, let μ_{OA} (resp. μ_{OB}) be the most similar subsequence of O and A (resp. B) which extends μ_{OSubA} (resp. μ_{OSubB}). Finally, let $\alpha = \langle \mu_O, \mu_A, \mu_B \rangle$ be the smallest similar subsequence alignment such that

$$\mu_O(i) = \mu_A(j) \Leftrightarrow \mu_{OA}(i) = j$$

$$\mu_O(i) = \mu_B(k) \Leftrightarrow \mu_{OB}(i) = k$$
$$\mu_A(j) = \mu_B(k) \Leftrightarrow \mu_{AB}(j) = k$$

The result of the SDiff3 algorithm for similar sequences is the configuration induced by α.

Example 4.3. To illustrate the similarity-based synchronizers we consider the following numbered color sequences $a = [(1, red), (2, blue), (4, red), (4, red)]$, $o = [(1, red), (2, red), (3, red), (4, red)]$, and $b = [(1, red), (4, red), (4, red)]$. A run of the synchronizer for similar subsequences on $a \leftarrow o \rightarrow b$ finds the similar subsequence alignment

a	$(1, green)$	$(2, blue)$	$-$	$(4, red)$	$(4, red)$
o	$(1, red)$	$(2, red)$	$(3, red)$	$(4, red)$	$-$
b	$(1, blue)$	$-$	$-$	$(4, red)$	$(4, red)$

Here we have an UpdateUpdateConflict in the first position and an UpdateDeleteConflict in the third position. The resulting edit script for o is

$$C([(1, green)], [(1, blue)]), C([(2, blue)], []), -(3, red), (4, red), +(4, red).$$

4.4 Similarity-based multiset synchronizer

The fourth three-way diff algorithm is the extension of the equality-based three-way diff for multisets to a similarity-based version. Analogously, we start with the most similar multi-subset c_{ab} of a and b, which no longer is an intersection. From that we determine the most similar multi-subset c_{init} between o and c_{ab}, which again is no longer an intersection. Then we determine the most similar multi-subsets between o and a (resp. b) which contain c_{init}. As before, this is in general not a most similar multi-subset of o and a (resp. b), but eases the alignment of all three sequences and subsequently the merge.

In contrast to the equality case, there may well be conflicts in the similarity cases. Indeed, as for sequences, we also have *UpdateUpdate*-conflicts and *UpdateDelete*-conflicts in the multiset case, where one element has been changed in different ways in both variants. However, these are the only conflicts. As for sequences we also have local modifications, where one element has been changed in one variant and remained equal in the other variant. These are included in the merge.

The similarity-based alignment for multisets is defined as follows.

Definition 17 (Similar Multi-Subset Alignment). Let $a = (a_j)_{j \in A}, o = (o_i)_{i \in O}, b = (b_k)_{k \in B}$ be sequences. A *multi-subset alignment* $\alpha = \langle \mu_O, \mu_A, \mu_B \rangle$ *from* o *to* a *and* b consists of three morphisms $\mu_O : O \rightarrow \mathbb{N}$, $\mu_A : A \rightarrow \mathbb{N}$ and $\mu_B : B \rightarrow \mathbb{N}$ such that it holds

$$\mu_O(i) = \mu_A(j) \Rightarrow o_i a_j > 0$$
$$\mu_O(i) = \mu_B(k) \Rightarrow o_i b_k > 0$$
$$\mu_A(j) = \mu_B(k) \Rightarrow a_j b_k > 0$$

The alignment image $\mathrm{Im}(\alpha)$ is $\mathrm{Im}(\mu_O) \cup \mathrm{Im}(\mu_A) \cup \mathrm{Im}(\mu_B)$ and its *size* is the cardinality of its image.

The *conflict sets* Conflicts(α) of the alignment is the *UpdateUpdateSet* which are those $n \in \mathrm{Im}(\mu_A) \cap \mathrm{Im}(\mu_B) \cap \mathrm{Im}(\mu_O))$ such that $a_{\mu_A^{-1}(n)} \neq o_{\mu_O^{-1}(n)}$, $b_{\mu_B^{-1}(n)} \neq o_{\mu_O^{-1}(n)}$ and $a_{\mu_A^{-1}(n)} \neq b_{\mu_O^{-1}(n)}$.

The *local modifications* $\mathrm{mod}_A(\alpha)$ *of* A *(resp.* $\mathrm{mod}_B(\alpha)$ *for* B*)* are those $n \in \mathrm{Im}(\mu_A) \cap \mathrm{Im}(\mu_B) \cap \mathrm{Im}(\mu_O))$ such that $a_{\mu_A^{-1}(n)} \neq o_{\mu_O^{-1}(n)}$ and $b_{\mu_B^{-1}(n)} = o_{\mu_O^{-1}(n)}$ (resp. $b_{\mu_B^{-1}(n)} \neq o_{\mu_O^{-1}(n)}$ and $a_{\mu_A^{-1}(n)} = o_{\mu_O^{-1}(n)}$).

The *local additions* $\mathrm{add}_A(\alpha)$ *of* A *(resp.* $\mathrm{add}_B(\alpha)$ *for* B*)* is the subset of $\mathrm{Im}(\mu_A)$ (resp. $\mathrm{Im}(\mu_A)$) which is disjoint from $\mathrm{Im}(\mu_O)$ and $\mathrm{Im}(\mu_B)$ (resp. $\mathrm{Im}(\mu_A)$).

The *deletion set* $\mathrm{del}_O(\alpha)$ *of* O is the subset of $\mathrm{Im}(\mu_O)$ which is disjoint from $\mathrm{Im}(\mu_A)$ and $\mathrm{Im}(\mu_B)$.

The induced configuration $a' \leftarrow o' \rightarrow b'$ is defined by

$$\begin{aligned}
M &= (\mathrm{Im}(\mu_O) \cap \mathrm{Im}(\mu_A) \cap \mathrm{Im}(\mu_B)) \cup \mathrm{add}_A(\alpha) \cup \mathrm{add}_B(\alpha) \cup \mathrm{mod}_A(\alpha) \cup \mathrm{mod}_B(\alpha) \\
O' &= M \cup (\bigcup_{C \in \mathrm{Conflicts}(\alpha)} (C \cap \mathrm{Im}(\mu_O))) \\
A' &= M \cup (\bigcup_{C \in \mathrm{Conflicts}(\alpha)} (C \cap \mathrm{Im}(\mu_A))) \\
B' &= M \cup (\bigcup_{C \in \mathrm{Conflicts}(\alpha)} (C \cap \mathrm{Im}(\mu_B)))
\end{aligned}$$

and $o' = (o'_i)_{i \in O'}$, $a' = (a'_j)_{j \in A'}$ and $b' = (b'_k)_{k \in B'}$ where

$$o'_i = \begin{cases} o_{\mu_O^{-1}(i)} \text{ if } i \in \mathrm{Im}(\mu_O) \cap \mathrm{Im}(\mu_A) \cap \mathrm{Im}(\mu_B) \\ o_{\mu_O^{-1}(i)} \text{ if } i \in \bigcup_{C \in \mathrm{Conflicts}(\alpha)} (C \cap \mathrm{Im}(\mu_O)) \\ a_{\mu_A^{-1}(i)} \text{ if } i \in \mathrm{add}_A(\alpha) \cup \mathrm{mod}_A(\alpha) \\ b_{\mu_B^{-1}(i)} \text{ if } i \in \mathrm{add}_B(\alpha) \cup \mathrm{mod}_B(\alpha) \end{cases}$$

$$a'_j = \begin{cases} o'_j & \text{if } j \in M \\ a_{\mu_A^{-1}(j)} & \text{if } j \in \bigcup_{C \in \mathrm{Conflicts}(\alpha)} (C \cap \mathrm{Im}(\mu_A)) \end{cases}$$

$$b'_k = \begin{cases} o'_k & \text{if } k \in M \\ b_{\mu_B^{-1}(k)} & \text{if } k \in \bigcup_{C \in \mathrm{Conflicts}(\alpha)} (C \cap \mathrm{Im}(\mu_B)) \end{cases}$$

From that notion of alignment we can now define the similarity-based three-way diff for multisets.

Definition 18 ($\mathrm{sDiff3}_S^{\sim}$ **for Multi-Subsets).** Let $a = (a_j)_{j \in A}$, $o = (o_i)_{i \in O}$, $b = (b_k)_{k \in B}$ be sequences. Let μ_{AB} be the most similar multi-subset for a and b, and μ_{OSubA} (resp. μ_{OSubB}) be the most similar multi-subset of o and $a_{|\,\mathrm{Dom}(\mu_{AB})}$ (resp. $b_{|\,\mathrm{Im}(\mu_{AB})}$). Furthermore, let μ_{OA} (resp. μ_{OB}) be the most similar multi-subset of O and A (resp. B) which extends μ_{OSubA} (resp. μ_{OSubB}). Finally, let $\alpha = \langle \mu_O, \mu_A, \mu_B \rangle$ be the most similar multi-subset alignment such that

$$\mu_O(i) = \mu_A(j) \Leftrightarrow \mu_{OA}(i) = j$$

$$\mu_O(i) = \mu_B(k) \Leftrightarrow \mu_{OB}(i) = k$$
$$\mu_A(j) = \mu_B(k) \Leftrightarrow \mu_{AB}(j) = k$$

The result of the SDiff3 algorithm for sequences is the configuration induced by α.

Example 4.4. Consider the following multisets of colored numbers: $a = [(1, red), (2, blue), (4, red), (4, green)]$, $o = [(1, red), (2, red), (4, red), (4, red)]$ and $b = [(1, red), (4, blue), (4, red)]$. Running the synchronizer sDiff3$^{\sim}_{S}$ on $a \leftarrow o \rightarrow b$ yields the smallest multi-subset alignment

a	$(1, red)$	$(2, blue)$	$(4, green)$	$(4, red)$
o	$(1, red)$	$(2, red)$	$(4, red)$	$(4, red)$
b	$(1, red)$	$-$	$(4, blue)$	$(4, red)$

In order to obtain the alignment, the $(4, red)$ and $(4, green)$ switch positions in a. This creates an UpdateUpdate conflict between $(4, green)$ in a, $(4, blue)$ in b and the second $(4, red)$ in o. Furthermore, there is an UpdateDelete conflict between $(2, red)$ being deleted in b and updated to $(4, blue)$ in a. The resulting edit script is

$$(1, red), C < (2, blue), - >, C < (4, green), (4, blue) >, (4, red)$$

5 Discussion

It is easy to see that the similarity-based three-way diffs for sequences and multisets are conservative extensions of the equality-based notions. The algorithms are parametric in the algorithm used to determine longest common subsequences. The current implementation uses, for ordered sequences, a classical dynamic programming approach[8], which was the easiest to adapt to take into account pre-fixed mappings (i.e. c_{init} in the introductory discussion of Section 4.1). Thus, in principle other LCS algorithms could be used, such as the edit-cost minimizing algorithms of Refs. [19,4], but it would have to be adapted to take into account pre-fixed mappings. Analogously, for unordered sequences, edit-cost optimizing algorithms can be developed, but have not been considered yet.

Furthermore, like we have defined a most-similar mixed common subsequence, a similarity-based three-way difference analysis for mixed sequences can be defined: first, the three lists of a configuration are partitioned into the list-like subsequence and the multiset subsequence to obtain two configurations. Then apply the respective three-way diff synchronizer on each and subsequently merge the results back into a single configuration.

Finally, as the synchronizers all operate on the principle to first compute the similar subsequence between both variants, it is easy to derive the merge algorithms for sequences by only computing the alignment between a and b and then compute the induced configuration. This way we immediately obtain similarity-based merge algorithms for sequences and multisets.

The similarity-based synchronizers are the central part of our own solution for three-way difference analysis for tree-structured documents, like XML. The principle here is to compute the three-way difference analysis between two lists of subdocuments and consider the edit script: if in the edit script UpdateUpdate-conflicts occur, then we have to recursively apply the three-way diff on the conflicting variants to come up with a synchronization and hence an edit-script for these two conflicts. The results can be assembled into a single configuration and an edit-script can be generated for the different subtrees. Here other aspects are relevant, which are the level of granularity, at which edit-scripts are returned. A current limitation is that we cannot find subtrees that are moved across levels in the tree or to different subtrees, which is, e.g., handled in Ref. [14]. Furthermore, we currently do not handle renamings, such as in Ref. [15]. However, detected renamings could be handled by our approach by prefixing mappings between detected renamings in the subsequence/subset computations as is done in the synchronizers. Employing ideas from Ref. [15] or quantifiers and scoping as used in formal logic languages to identify renamings in a pre-processing phase is an avenue to explore in future work.

A key aspect of generic similarity-based difference analysis algorithms for programming and domain-specific languages (e.g., Refs. [9,14,11,18]) is how easily they can be customized for a specific domain. Addressing this issue is beyond the scope of this article. However in Ref. [3] we have proposed a first solution to support customization, which has been extended even further in Refs. [1,2]. In contrast to the often weight-based and recursive or iterative approaches that can be found in the literature, we are considering also more qualitative customizations such as alternatives based on available attributes and subtrees, mandatory and optional attributes and subtrees as well as customizable, context-dependent depth-limits. This shows promising results and will be reported in a different paper.

The similarity-based synchronizers and merge algorithms for sequences, multisets and mixed sequences as well as for trees have been implemented in Java and successfully tested. They subsume and extend the implementation of the most similar subsequence algorithms in Ref. [3] and have been successfully tested to heterogeneous document collections in Refs. [1,2]. They have been successfully with XML-files and are routinely used with XML files of up to 200 kb in size.

6. Conclusion

We have presented diff, three-way diff and merge algorithms that analyse sequences based on the similarity of the elements and can also abstract from the order of the elements. This is more suitable for collections of structured documents and data as they occur in modern, digital data and document processing based workflows. It allows to define document and data specific similarity notions that can be informed by a domain and application specific knowledge. Moreover, it provides tree-based difference analysis algorithms different from those currently found in the literature. A comparison with these is beyond the scope of this article and future work. All algorithms have been successfully implemented and tested. The equality-based synchronizer for lists is different from the quasi-standard diff3 and in contrast to that is idempotent. Future work will consist of analysing the other algorithms wrt. that property and other properties, such as a locality property as considered in[10]. Further future work beyond the future work presented in the previous section will be devoted to have semantic-based difference analysis that goes beyond similarity-based analysis, even if the similarity is determined using semantic information.

References

[1] Autexier S, David C, Dietrich D, Kohlhase M, Zholudev V. Workflows for the management of change in science, technologies, engineering and mathematics. In: Davenport JH, Farmer W, Rabe F, Urban J, eds. Proc. of Calculemus/MKM 2011. LNAI. Springer-Verlag Berlin Heidelberg. july 2011. number 6824. 164–179.

[2] Autexier S, Dietrich D, Hutter D, Lüth C, Maeder C. Smartties - management of safety-critical developments. In: Margaria T, Steffen B, eds. Proc. 5th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISoLa'12). LNCS. Amirandes, Heraclion, Crete. Springer. october 2012.

[3] Autexier S, Müller N. Semantics-based change impact analysis for heterogeneous collections of documents. In: Gormish M, Ingold R, eds. Proc. of 10th ACM Symposium on Document Engineering (DocEng2010). UK. 2010.

[4] Ann HY, Yang CB, Peng YH, Liaw BC. Efficient algorithms for the block edit problems. Information and Computation, 2010, 208(3): 221–229.

[5] Chawathe SS, Rajamaran A, Garcia-Molina H, Widom J. Change detection in hierarchically structured information. ACM SIGMOD Record, 1996, 25(2): 493–504.

[6] Dahlström E, Dengler P, Grasso A, Lilley C, McCormack C, Schepers D, Watt J, Ferraiolo J, Jun F, Jackson D. Scalable Vector Graphics (SVG) 1.1 (Second Edition). W3C, August 2011.

[7] Eggert P, Haertel M, Hayes D, Stallman R, Tower L. Gnu diff, April 1988. Version 2.8.1, April 2002; distributed with GNU diffutils package.

[8] Hunt JW, McIlroy MD. An algorithm for differential file comparison. Computing Science Technical Report 41, Bell Laboratories, June 1976.

[9] Hunt JJ, Tichy WF. Extensible language-aware merging. 18th International Conference on Software Maintenance (ICSM 2002), Maintaining Distributed Heterogeneous Systems. 3-6 October 2002. Montreal, Quebec, Canada. IEEE Computer Society. 2002. 511–520.

[10] Khanna S, Kunal K, Pierce BC. A formal investigation of diff3. In: Arvind V, Prasad S, eds. FSTTCS 2007: Foundations of Software Technology and Theoretical Computer Science. Lecture Notes in Computer Science. Springer Berlin Heidelberg. 2007, volume 4855. 485–496.

[11] Lin YH, Gray J, Jouault F. DSMDiff: a differentiation tool for domain-specific models. EJIS, 2007, 16(4): 349–361.

[12] Lindholm T. A three-way merge for xml document. Proc. of the 2004 ACM symposium on Document engineering. ACM Press. New York, NY, USA. 2004. 1–10.

[13] Lanham M, Kang A, Hammer J, Helal A, Wilson J. Format-independent change detection and propoagation in support of mobile computing. Brazilian Symposium on Databases (SBBD). Gramado, Brazil. October 2002. 27–41.

[14] Melnik S, Garcia-Molina H, and Erhard Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In: Agrawal R, Dittrich KR, eds. Proc. of the 18th International Conference on Data Engineering. San Jose, CA, USA. February 26 - March 1, 2002. IEEE Computer Society. 2002. 117–128.

[15] Malpohl G, Hunt JJ, Tichy WF. Renaming detection. Automated Software Engg., April 2003, 10(2): 183–202.

[16] Myers EW. An o(nd) difference algorithm and its variations. Algorithmica, 1986, 1(2): 251–266.

[17] Smith R. Gnu diff3, April 1988. Version 2.8.1, April 2002; distributed with GNU diffutils package.

[18] Treude C, Berlik S, Wenzel S, Kelter U. Difference computation of large models. In: Crnkovic I, Bertolino A, eds. Proc. of the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2007, Dubrovnik, Croatia, September 3-7, 2007. ACM. 2007. 295–304.

[19] Tichy WF. The string-to-string correction problem with block moves. ACM Trans. Comput. Syst., 1984, 2(4): 309–321.