A Comparative Evaluation of XML Difference Algorithms with Genomic Data

Cornelia Hedeler and Norman W. Paton

School of Computer Science, The University of Manchester, Oxford Road, Manchester M13 9PL, UK
{chedeler,npaton}@cs.manchester.ac.uk

Abstract. Genome sequence data and annotations are subject to frequent changes resulting from re-assembly and re-annotation, or community feedback based on experimental evidence, giving rise to new data releases. These releases are rarely accompanied by a description of the changes, making it difficult for biologists working with the data to identify and work through the consequences of the changes that have taken place. This paper explores the extent to which existing XML difference algorithms, namely X-Diff, JXyDiff and 3DM, can be used to identify and document genome changes, in particular investigating: (i) their ability to detect typical changes in genome sequence documents; and (ii) the ease with which the difference report can be used to determine whether genes of interest are affected by changes to the genome. The evaluation compares the performance of the algorithms both with synthetic modifications and for detecting changes in a public genomic database. Typical behaviours of the algorithms are identified and a root cause analysis carried out.

1 Introduction

Genomic data, including the annotation of predicted genes and proteins, is available for an increasing number of genomes. The genomic data for each of those genomes undergoes regular re-annotation, and in many cases even re-assembly of the sequence data, resulting in new releases that replace previous versions. In particular in the early stages of a genome release, re-annotation generally involves automatic annotation of the (re-assembled) sequence from scratch using a computational analysis pipeline (e.g., the Ensembl analysis pipeline [1,2]). In the later stages, i.e., a few years after the initial genome release, this process might be complemented or increasingly replaced by manual changes to the annotation based on experimental evidence provided by the community. In contrast to the manual annotation, which explicitly introduces a change to the previous annotation and therefore, if captured appropriately, provides a delta description between two different releases of the same genome (e.g., the summary of chromosome sequence and annotation updates provided by the Saccharomyces Genome Database (SGD) [3]), the automatic re-annotation of a genome does not result in such a delta description. Therefore, very little or no information about the changes carried out is made available by providers of genomic data. For example, Ensembl [4] only provides very general information on the type of data updated as news for each new release, but no detailed information on gene level changes. In contrast, the Fungal Genome Initiative (FGI) at the Broad Institute[1] provides a list of the changes between two releases for some genomes. The information provided on the changes, however, is limited and only lists the identifiers of newly predicted, updated, split and deleted genes, but no further details. The limited information on changes between genome data releases presents significant challenges to biologists and bioinformaticians working with the genomic data and the corresponding annotation, in some cases even resulting in the continued use of an out-of-date genome release.

A number of data providers make previous genome releases available in archives (e.g., Ensembl, EMBL [5,6]), provide tools for converting data from one release to another release of a genome (Ensembl), or provide tools for comparison of two genome versions, the result of which can be inspected manually (EMBL). Those tools, however, are not generally applicable. The converter provided by Ensembl is currently only available for converting mouse assembly data between the current release and the previous one, and is not available for conversion between other combinations of releases or other genomes. The comparison tool provided by EMBL can be used to compare any two versions of a genome, but it compares the files line by line and highlights lines that have been removed or inserted between the two versions. Lines that contain changes are also presented as removed and inserted. The user has to inspect the highlighted file manually to identify the consequences for genes of interest, a labour-intensive task when carried out for a whole genome and/or a number of releases. Changes between two releases of the same genome can be of varied nature (see the following section). Without a history of changes, however, it is hard for users of the data to track genes of interest through time, i.e., various genome releases and determine, e.g., whether a missing gene has been removed, renamed, moved to a different location, or merged with a neighbouring gene.

Genomic data is represented in a variety of formats, amongst others XML (e.g., EMBL XML Schema), and converters are available to convert data between different formats and also to convert it into the EMBL XML format (e.g., converters are provided by EMBL [5] and BioJava[2]). A number of XML difference tools and algorithms have also been published previously (e.g., X-Diff [7], XyDiff [8], 3DM [9]). In the original publications these algorithms have mainly been applied to merging of XML documents that have been edited by different people (3DM), and for detecting changes of XML documents on the web (X-Diff, XyDiff). In this paper we evaluate the applicability of XML difference algorithms on XML documents representing genomic data and associated annotation. In so doing, we seek both to obtain insights into the algorithms and to identify an effective means of understanding the changes that have been made to genomes.

[1] http://www.broad.mit.edu/annotation/fgi/

[2] http://biojava.org

The paper is structured as follows. Section 2 provides a detailed description of the problem, including an introduction to the format in which genomic data is represented. Section 3 introduces the XML difference algorithms evaluated here. Section 4 describes the experimental setup and discusses the results. This is followed by an evaluation of the XML difference algorithms on real data in Section 5. Section 6 concludes by reviewing the lessons learned.

2 Problem Description

In this section the XML representation of genomic data is introduced. Furthermore, a number of typical changes to genomic data are presented.

Genomic data is represented in a number of flat file formats established by the major data providers, including Genbank [10] and EMBL [5]. For both Genbank and EMBL formats, XML representations of the data are available, namely Insd XML and EMBL XML, respectively. Due to its slightly more intuitive representation of genomic data, we have chosen to use the EMBL XML representation. The majority of genomic data is made available separately for each chromosome of a genome. For this reason, we have chosen a chromosome as the unit in which we analyse changes in genomic data. Therefore, we do not consider changes that affect multiple chromosomes, such as a move of a predicted gene between chromosomes; such changes would be detected as a deletion in one chromosome and an insertion in another.

A gene is a defined strand of DNA that contains regions that code for a protein (exons) and those that do not (introns). The complete sequence of exons for a gene is also called a coding sequence (CDS). The whole DNA sequence of a gene is transcribed from DNA to (pre-)mRNA, followed by a process called splicing during which the introns are removed and the exons spliced together to form the messenger RNA (mRNA). The mRNA is then translated into a protein.

In both the well established flat file formats and the corresponding XML representations a predicted gene is represented as follows: (i) an element capturing information on the gene; (ii) an element describing the corresponding mRNA; (iii) an element containing information on the coding sequence (CDS); and (iv) elements with information on the corresponding exons (elements `<feature name="gene">`,`<feature name="mRNA">`,`<feature name="CDS">`, and `<feature name="exon">`, respectively, in Figure 1). Usually, information on the predicted genes appear in the order of the genes on the chromosome, i.e., the order among siblings is important, and the genomic sequence is included as a whole for the chromosome and not for each gene separately. However, to ease the identification of changes in the sequence of a gene, we retrieve the sequence for each gene and include it alongside the corresponding gene information. An example of the resulting XML representation of the elements describing the gene, its mRNA, CDS, genomic sequence and exon is provided in Figure 1. Usually, the descriptions of all exons for all genes can be found at the end of the document (indicated by '...' in the example). As can be seen in the example, all the elements corresponding to a gene are located at the same level in the hierarchical structure

```
<EMBL>
  <entry accession="chromosome:SGD1.01:I:1:230308:1" lastUpdated="8-APR-2007" name="I">
    ...
    <feature name="gene">
      <qualifier name="gene">YAL003W</qualifier>
      <qualifier name="note"> Elongation factor 1-beta (EF-1-beta) </qualifier>
      <location complement="false" type="single">
        <locationElement complement="false" type="range">
          <basePosition type="simple">142176</basePosition>
          <basePosition type="simple">142255</basePosition>
        </locationElement> </location> </feature>
    <feature name="mRNA">
      <qualifier name="gene">YAL003W</qualifier>
      <qualifier name="note">transcript_id=YAL003W</qualifier>
      <location complement="false" type="single">
        <locationElement complement="false" type="range">
          <basePosition type="simple">142176</basePosition>
          <basePosition type="simple">142255</basePosition>
        </locationElement> </location> </feature>
    <feature name="CDS">
      <dbreference db="RefSeq_peptide" primary="NP_009398.1"/>
      <qualifier name="gene">YAL003W</qualifier>
      <qualifier name="protein_id">YAL003W</qualifier>
      <qualifier name="note">transcript_id=YAL003W</qualifier>
      <qualifier name="translation"> MASTDFSKIETLKQLNASLADKSYIEGTAVSQA...</qualifier>
      <location complement="false" type="single">
        <locationElement complement="false" type="range">
          <basePosition type="simple">142176</basePosition>
          <basePosition type="simple">142255</basePosition>
        </locationElement> </location> </feature>
    <sequence length="987" type="DNA" version="0.0"> agttgcgcatgaatttctcc...</sequence>
    ...
    <feature name="exon">
      <qualifier name="note">exon_id=YAL003W.1</qualifier>
      <location complement="false" type="single">
        <locationElement complement="false" type="range">
          <basePosition type="simple">142176</basePosition>
          <basePosition type="simple">142255</basePosition>
        </locationElement> </location> </feature>
    ...
  </entry>
</EMBL>
```

Fig. 1. Example of genomic data represented using a variant of EMBL XML [5,6]

of the XML representation. There are no parent elements for each gene that contain all the associated elements; instead the parent element of all elements describing all the genes is the element describing the chromosome on which the genes are located. It can also be seen that the elements contain partly redundant information, such as the location and the identifier of the gene.

With increasing knowledge of a genome, its genomic data and associated annotation can change in a number of different ways. Changes include modifications of the genomic sequence itself, the identification of new genes with their associated proteins, and the removal of a previously predicted gene that is no longer thought to be a gene. Further changes include the merging of two neighbouring genes into one gene or the splitting of one gene into two neighbouring genes. Examples of such changes can be found in the change history provided by SGD for the yeast genome.[3] In the remainder of the paper we focus on five types of

[3] http://www.yeastgenome.org/cache/genomeSnapshot.html#ChrSeqAnnotUpdates

changes, which are explained in more detail below. A number of other changes, such as the update of the name or identifier of a gene or the update of its location, are essentially updates to the value of a text node. As is shown later, an update to the value of a text node does not present a challenge to the XML difference algorithms.

Change to the genomic sequence. The genomic sequence can undergo changes, for example, to correct mistakes introduced in an earlier release. To change the sequence of a gene, the value of the corresponding text node with name `sequence` is updated.

Identification of a new gene. When a new gene that has been missed previously is identified, all elements capturing information on the gene, its mRNA, CDS, sequence and exons are inserted. As the genes appear in the document in the same order as on the chromosome, the first four elements (gene, mRNA, CDS, sequence) for the new gene are inserted after the element containing the sequence of the preceding gene and before the elements describing the following gene. As the exons are listed at the end of the document in the same order as on the chromosome, the elements with information on the exons of the new gene are inserted between the exons of the preceding gene and those of the following gene.

Removal of a previously predicted gene. Analysis of the genome and its sequence can reveal that a previously identified gene isn't actually a gene. Thus, all elements describing this gene, its mRNA, its CDS, its sequence and its exons need to be removed.

Both identification and removal of a gene result in a change of the positions of elements following the inserted or removed elements within the sequence of elements describing all the genes on a chromosome.

Merging of two neighbouring genes, gene1 and gene2, into one gene. Biological experiments can reveal that two separate genes actually correspond to a single gene. In such a case the annotation needs to be updated as follows (see also Figure 2):

– Update of the elements corresponding to *gene1*. Changes include (i) update of the identifiers of the elements describing the gene, its mRNA and CDS; (ii) update of the location of *gene1* and insert of additional location elements for the exons of *gene2*, which after the merge belong to *gene1*; and (iii) update of the sequence and translation of *gene1* by appending the corresponding sequence and translation of *gene2*.
– Update the identifiers of all the exons belonging to *gene1* and *gene2* before the merge to reflect that they belong to the merged gene.
– Delete the elements (with information on gene, mRNA, CDS, sequence) corresponding to *gene2*.

Splitting of one gene into two neighbouring genes, gene1 and gene2. As the complementary change to merging of two neighbouring genes, a single gene can

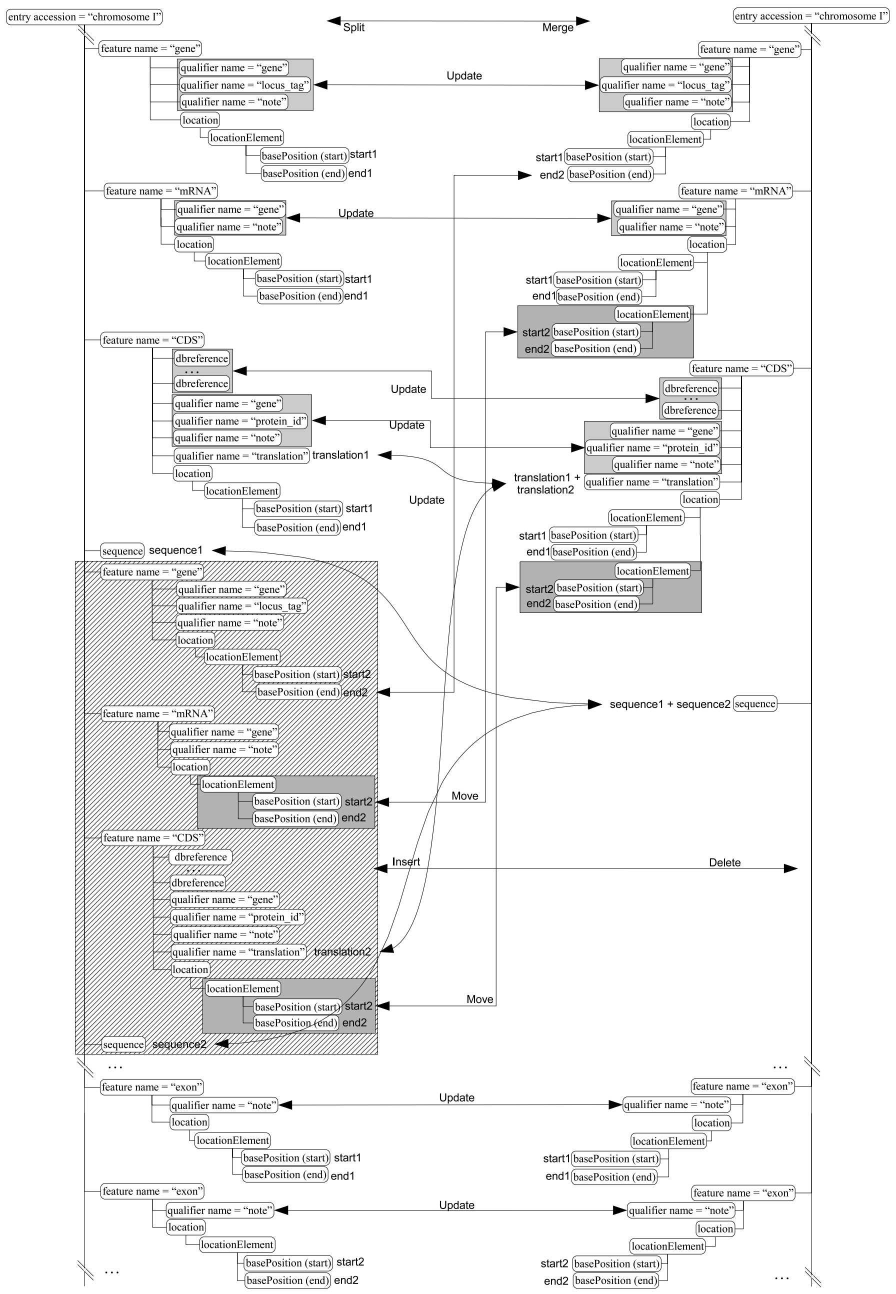

Fig. 2. Split of one gene into two neighbouring genes and merge of two neighbouring genes into one gene

also be split into two neighbouring genes. The annotation is updated as follows (see also Figure 2):

– Insert elements (gene, mRNA, CDS, sequence) corresponding to *gene2* with the appropriate part of the translation and sequence that belongs to *gene2* after the merge and the appropriate location elements. This information can be obtained from the single gene that is to be split.
– Update of the elements corresponding to the single gene to reflect that it represents *gene1* after the split. Changes include (i) update of the identifiers of the elements describing the gene, its mRNA and CDS; (ii) update of the location of the gene and delete of the additional location elements for the exons that belong to *gene2* after the split; and (iii) update of the sequence and translation of *gene1* by removing the part of the sequence and translation that belongs to *gene2* after the split.
– Update of the identifiers of all the exons belonging to the gene to be split to reflect that they belong to either *gene1* or *gene2*. If there is only one exon, this needs to be split by updating the existing exon appropriately to reflect that it belongs to *gene1* and inserting a new element with information on the exon belonging to *gene2*.

As shown above, changes to genomic sequence data and its annotation generally lead to a number of (different) changes to the document, not something the techniques evaluated here were designed to support.

3 XML Difference Algorithms

Three published XML difference algorithms for which working Java implementations were obtained are evaluated in this paper. The key properties of the algorithms are summarised in Table 1 and described below.

X-Diff [7]. The X-Diff algorithm involves the following steps:

– *Preprocessing:* In the preprocessing phase, both XML documents are parsed into tree representations and using XHash (a special hash function similar to DOMHash [13], but working on unordered trees), hash values for all nodes in both trees are calculated. The hash value of an element node a is calculated based on the hash values of its child nodes and the resulting value, therefore, represents the entire subtree rooted at the node a.

Table 1. Summary of key properties of XML difference algorithms

Algorithm	**Source**	**Ordered/ Unordered Tree**	**Changes detected**
X-Diff	[7]	Unordered	Insert, Delete of leaf nodes or subtrees; Update of values of text- or attribute nodes
JXyDiff	XyDiff [8,11]	Ordered	Insert, Delete, Move of leaf nodes or subtrees; Update of values of text- or attribute nodes
3DM	[9,12]	Ordered	Insert, Delete, Move of leaf nodes; Update of values of text- or attribute nodes

– *Matching:* The matching step consists of the following three steps.
  - To reduce the search space, subtrees with the same hash value are filtered out.
  - Starting from the leaf node pairs and moving upwards, nodes of the same type (i.e., text-, element-, or attribute node) and with matching ancestor names are matched, and their edit distance computed using a cost model with a uniform distance of 1 for update, insert and delete. To compute the edit distance between subtrees, the minimum-cost maximum flow algorithm [14,15] is used to find the minimum-cost bipartite matching. Both matching and edit distance are stored.
  - Starting from the root node and using the matchings and edit distances calculated in the previous step, create minimum-cost matchings between nodes of the two trees, allowing only one-to-one matchings and matching only child nodes of parents that are matched.

– *Edit script:* Starting from the root nodes and based on the minimum-cost matching and the edit distance, a minimum-cost edit script is generated. Nodes or subtrees found in the base document, but not found in the matching, are marked as deleted; nodes or subtrees found in the updated document, but not in the matching, are marked as inserted; and leaf nodes that are found in the matching but have different values are marked as updated.

JXyDiff (Java implementation of XyDiff [8,11]). The algorithm, called Bottom-Up, Lazy-Down (BULD) propagation, consists of the following steps:

– *Preprocessing:* Starting from the leaf nodes, hash values are calculated based on content of the node itself and the hash values of its children. Similar to X-Diff, the hash value of a node represents the entire subtree rooted in that node. In addition to the hash value, a weight is calculated for each node as follows: for a text node the weight is the size of the content and for an element node the weight is the sum of the weights of its child nodes. Subtrees represented by their root nodes are inserted into a priority queue where they are ranked by their weight.

– *Matching:* Starting with the heaviest subtree of the updated document (when there are several subtrees with the same weight, the first one in the queue is chosen), nodes with the same hash value (representing the entire subtree rooted at that node) are identified in the base document. If there is only one node in the base document with the same hash value, they are matched. If there are no nodes with the same hash value and the node is an element node, its children are inserted into the priority queue. If there are several nodes with the same hash value, the node whose parent matches the parent of the node from the updated document is chosen. The matching is followed by an optimisation phase in which already matched nodes are used to propagate matches further to nodes not matched in the previous matching step. During this phase, nodes are matched when their parents and/or children are matched and they have the same label. Bottom-up propagation of the matchings is controlled by the weight of the matching subtrees, i.e., the heavier/larger a subtree the further the matching is propagated.

– *Edit script:* Based on the matchings, the edit script is generated. Unmatched nodes in the old document are marked as deleted, and those unmatched in the new document are marked as inserted. Matched text nodes with changed content are marked as updated. Nodes that are matched, but without matching parents, are marked as moved. JXyDiff also detects moves of nodes within the same parent, i.e., changes to the order of matched siblings under matched parents. This is done by finding the largest order-preserving subsequence and adding move operations for the remaining pairs of nodes. As this step is expensive for large sequences of elements, in such cases the sequence is cut into smaller subsequences of a fixed maximum length and the same process applied to all the smaller subsequences. This improves the performance, but does not guarantee the optimal number of moves.

3-Way Merge and Diff (3DM) [9,12]. 3DM is a merge and diff tool that can merge 3 documents, the base document and two updated versions of the document for the purpose of reintegrating changes from two independently modified copies into a single document containing all the modifications. It can, however, also be used to find differences between two documents by providing two copies of the same document, e.g., the base document and the updated document as input. The algorithm consists of the following steps:

– *Matching:* For each node in the base document, find exact or close matching nodes in the updated document. The similarity of close matching nodes is based on the q-gram string distance measure [16]. Q-grams are substrings of length q and the q-gram distance is the number of q-grams that appear in only one of the two strings. For all pairs of matched nodes, match the subtrees by depth-first traversal starting with the two matched nodes. Continue as long as the child nodes are matched too. Select the best matching subtree.
– *Post-processing:* The postprocessing phase can be divided into the following steps:
  • Remove matches of small copies: All nodes in the updated document whose matching node in the base document has several matches in the updated document are checked, and matches to nodes that are part of a subtree containing only little information are removed to avoid copying small amounts of data.
  • Propagate matches: Using the structure of the document, nodes so far not matched are matched if their parents and left or right siblings are matched.
  • Set type of match: All matches are classified as structural, content or full (structural and content) matches.
  • Merge documents: Starting from the root node and based on the matchings between the base document and each of the two updated documents, the merged document and the edit script is created. This is done by pairing up children of matched nodes, determining the sequence of these pairs according to any moves made, and merging the contents of the matched pairs. The merged node is added to the merge tree.

– *Edit script:* During the merging step, the edit script is created. Nodes found in either of the updated copies but not the base version of the document are marked as inserted; matched nodes with different contents are marked as updated; nodes that are matched, but appear in a different order are marked as moved; and nodes not found in either of the two updated copies but in the base document are marked as deleted.

4 Experiments Involving Controlled Modifications

In this section the experimental setup is introduced and the results of the experiments are discussed. The experiment consists of two parts: (i) evaluation of the XML difference algorithms on synthetic modifications to enable controlled exploration of a number of different kinds of change; and (ii) exploration of the use of the algorithms with real modifications between different releases of genomic data obtained from Ensembl [4].

Experimental setup: Beginning with a base document of genomic data from chromosome 10 of yeast containing about 400 genes, changes are introduced in a systematic manner. For each type of change and each pairwise combination of changes mentioned in Section 2, new documents are produced with n (n = 4, 20, 40, 60, 80) of the genes subject to each change. The changes are introduced randomly, but conform to the constraints imposed by the well established representation of the data: for example, elements describing a gene, its mRNA, its CDS and its sequence are neighbouring siblings, genes appear in the order they are on the chromosome, and exons appear at the end of the document in the same order as they are on the chromosome. In addition to the updated documents, a change report for each updated document is produced, detailing the changes introduced, gathering corresponding gene-level changes, and presenting them in a manner meaningful to biologists, as illustrated in Figure 3.

Using each of the XML difference algorithms, the base document is compared with each of the updated documents. In a post-processing step, the edit scripts produced by each of the algorithms are processed to: (i) identify the changes reported; (ii) gather changes affecting the same gene; and (iii) reproduce as much as possible of the change report corresponding to each updated document.

A number of observations could be made in the post-processing phase for the majority of edit scripts produced by JxyDiff and 3DM, and for this reason, are summarised here: (i) Edit scripts produced by JXyDiff tend not to be minimal, in that they contain a number of moves of sibling elements within the same parent, an observation reported previously [7]. These move operations are a result of incorrect matching of subtrees (where, for example an updated subtree is matched incorrectly to a different subtree, that is then updated accordingly followed by move operations to restore the order of the siblings). To restore the order of the siblings, large sequences of siblings are split into smaller sequences to improve the performance of the analysis step that seeks to detect moves of nodes within the same parent (see Section 3). As no moves are introduced as changes, the reported move operations are not included in the change report

```
<changeReport>
  <insert_biological_concept concept="gene" name="newGene55">
    <insert> <feature name="gene">
        <qualifier name="gene">newGene55</qualifier>
        <qualifier name="locus_tag">newGene55_YEAST</qualifier>
        <qualifier name="note">Uncharacterized protein newGene55</qualifier>
        <location complement="false" type="single">
          <locationElement complement="false" type="range">
            <basePosition type="simple">95483</basePosition>
            <basePosition type="simple">95483</basePosition>
          </locationElement>
        </location> </feature> </insert>
    <insert> <feature name="mRNA">
        ... </feature> </insert>
    <insert> <feature name="CDS">
        ... </feature> </insert>
    <insert>
      <sequence length="0" type="DNA" version="0.0">agtgaataatttaa...</sequence>
    </insert>
    <insert> <feature name="exon">
        ... </feature> </insert>
  </insert_biological_concept>
  <delete_biological_concept concept="gene" name="YAL011W">
    <delete> <feature name="gene">
        ... </feature> </delete>
    <delete> <feature name="mRNA">
        ... </feature> </delete>
    <delete> <feature name="CDS">
        ... </feature> </delete>
    <delete>
      <sequence length="1878" type="DNA" version="0.0">agtttctgggttt...</sequence>
    </delete>
    <delete> <feature name="exon">
        ... </feature> </delete>
  </delete_biological_concept>
</changeReport>
```

Fig. 3. Example of a change report, developed to present changes in a manner meaningful to biologists

generated from the edit scripts. (ii) Edit scripts produced by 3DM contain the parent element as well as all the child elements that are affected by the change. In cases where the child element is reported with the same type of change as its parent element, i.e., the child element is subsumed by its parent element, the child element is not included in the change report generated from the edit script. If child and parent elements are reported with different types of changes, both are included in the change report. The generated change reports are then evaluated with respect to the quality of the results.

Experiment 1: *Change of genomic sequences.* In this experiment, which requires the fewest atomic element-level changes to the document, the value of the text node `sequence` is updated. This experiment analyses the ability of the algorithms to detect updates to values of text nodes. As the change introduced affects only a single node, this experiment is the least challenging. The number and types of changes reported by each difference algorithm are compared with those in the change report associated with the updated document.

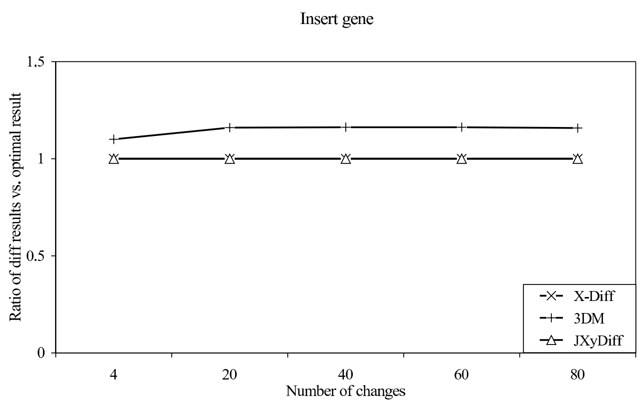

Fig. 4. Experiment 2, Introduction of new genes, relative performance

The following was observed: (i) Both X-Diff and 3DM detect all the introduced changes correctly and as the correct type of change. (ii) JXyDiff, however, detects all the introduced changes correctly, but as delete and insert of the text node `sequence` and not as an update, resulting in twice as many reported changes.

Experiment 2: *Identification of new genes.* In the second experiment, all the elements describing a gene, its mRNA, CDS, sequence and exons are inserted at appropriate positions in the document, following the constraints imposed by the representation of the data.

The following was observed: (i) Both X-Diff and JXyDiff detect all the introduced changes correctly. (ii) 3DM detects all the introduced changes correctly as inserts. In addition to the correctly identified inserts of elements, however, it detects a subset of the inserted sequence elements also as copies and updates of other `sequence` elements. The algorithm does not restrict the number of matches that can be identified between an element in the updated document and elements in the base document. As mentioned in Section 3, 3DM calculates the similarity of nodes using the q-gram string distance measure, and applies a threshold on the similarity to determine which nodes should be matched. As genomic sequence is basically a string of arbitrary length of the alphabet {a, c, t, g}, it is quite likely that two strings of sufficient length will have a sufficient number of q-grams in common to be regarded as similar enough to be copies of each other. As the q-gram string distance doesn't take into account the length of the string, sequences of very different lengths are matched based on their q-gram string distance, and one reported as an updated copy of the other. This results in a higher number of changes reported. The numbers of changes reported by each algorithm are compared with those in the change report associated with the updated document, and the ratios are plotted in Figure 4.

Experiment 3: *Removal of previously predicted genes.* In this experiment, all the elements representing a gene, its mRNA, CDS, sequence and exons are deleted.

The following was observed: all three XML difference algorithms detect the deletion of the subtrees correctly.

Experiment 4: *Change of genomic sequences and identification of new genes / Change of genomic sequences and removal of previously predicted genes.* In this experiment, the changes introduced in Experiment 1 are combined with those introduced in Experiment 2 or 3.

The following was observed: all three algorithms were able to identify the changes, exhibiting the same behaviour as in Experiments 1-3 with the corresponding single changes.

Experiment 5: *Identification of new genes and removal of previously predicted genes.* In this experiment, the changes introduced in Experiment 2 are combined with those introduced in Experiment 3.

The ratios of the numbers of changes reported by each algorithm compared with those in the corresponding change report are plotted in Figure 5. The following can be observed: (i) 3DM detects all the inserted elements and all the deleted elements correctly. As before in Experiment 2, however, a number of the inserted `sequence` elements are also detected as copies of other `sequence` elements combined with subsequent updates, resulting in a slightly higher number of changes reported. In contrast, JXyDiff and X-Diff report a far greater number of changes. (ii) JXyDiff detects inserts and deletes of elements describing the gene, mRNA, sequence and exons correctly, but fails to do so for a number of elements containing information on the CDS. In such cases, the child elements `dbreference, qualifier` and `basePosition` are reported as updates, deletes or inserts, resulting in the high number of changes reported. To restore the order of the elements, a large number of move operations of the siblings describing the elements of genes are reported. The incorrect matching of the inserted and deleted `CDS` elements principally results from the optimisation phase (see Section 3) in which matches of children, e.g., of the `location` elements, are

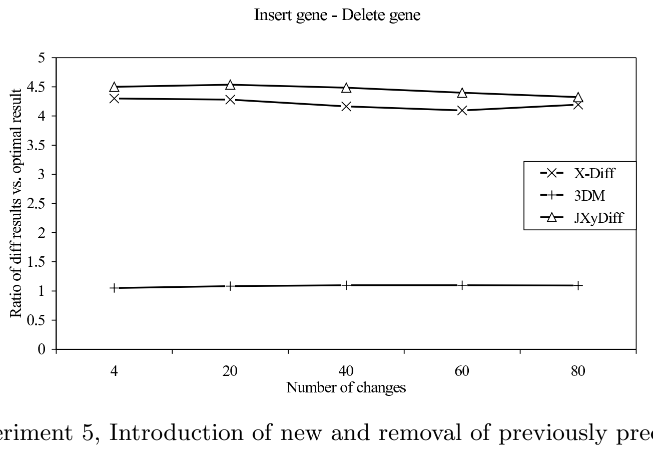

Fig. 5. Experiment 5, Introduction of new and removal of previously predicted genes, relative performance

propagated bottom-up to their parents. As the propagation is controlled by the weight of the matched subtree, this step can result in incorrect matchings of the `CDS` subtrees but not of the subtrees representing the other elements, as the `CDS` subtrees tend to have the larger number of children and therefore tend to be heavier. (iii) X-Diff reports the majority of deleted and inserted elements as updates, inserts or deletes of their corresponding leaf nodes, resulting in a high number of changes. As mentioned in Section 3 and listed in Table 1, X-Diff has the following properties: it regards the XML documents as unordered trees, uses a uniform cost model, creates a match starting with the leaf nodes, and matches nodes of the same type if their ancestors have matching names. These properties result in an almost arbitrary matching of the leaf nodes of the inserted and deleted subtrees representing the elements of the inserted and deleted genes. In the source documents though, order among siblings is significant.

Experiment 6: *Merging of two neighbouring genes.* In this experiment two neighbouring genes are merged. For detailed information on the changes introduced see Figure 2.

The ratios of the number of changes reported by each algorithm compared with those in the corresponding change report are plotted in Figure 6. The following can be observed: (i) X-Diff correctly identifies the deleted elements (representing one of the two neighbouring genes), but fails to match the updated elements correctly, resulting in a number of additional deletes, updates and inserts. For example, elements representing exons or CDS are matched with those representing mRNAs. These incorrect matchings are a result of handling the XML document as an unordered tree. However, no consistent pattern was observed for these incorrect matchings. (ii) JXyDiff identifies only a fraction of the deleted elements and matches very few of the updated elements correctly. This results in a large number of deletes, inserts, updates and moves to compensate for the incorrect matchings. In some cases, elements representing exons are matched with elements representing CDS or genes, however, no consistent

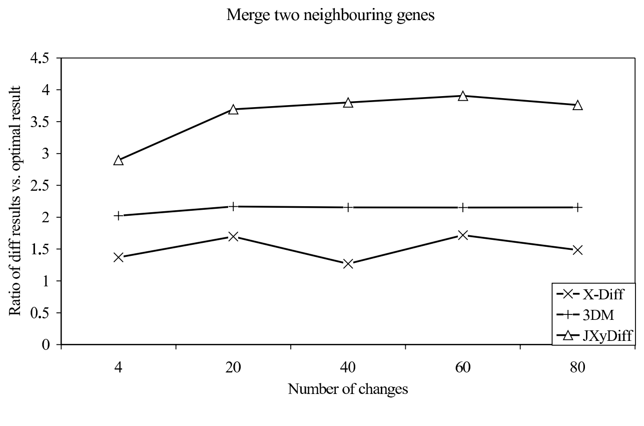

Fig. 6. Experiments 6, Merging of two neighbouring genes, relative performance

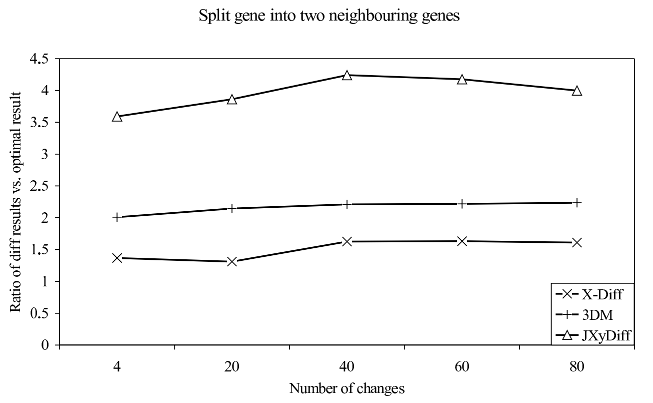

Fig. 7. Experiments 7, Splitting of one gene into two neighbouring genes, relative performance

pattern throughout all changes was observed. (iii) 3DM correctly identifies the majority of deleted elements and matches the updated elements. However, the majority of updates are not reported as such, but as deletes and inserts, resulting in about twice as many changes reported as in a minimal description. Furthermore, a number of elements are reported both as inserted and as updated, a result of the lack of a restriction on the number of matchings between an element in the updated document and elements in the source document.

Experiment 7: *Splitting of one gene into two neighbouring genes.* In this experiment one gene is split into two neighbouring genes. Detailed information on the changes introduced are shown in Figure 2. The ratios of the number of changes reported by each algorithm compared with those in the corresponding change report are plotted in Figure 7. The following was observed: (i) X-Diff identifies all inserted elements correctly and matches the majority of updated elements correctly. However, in some cases elements representing mRNAs are matched with elements representing genes or exons, resulting in detection of additional updates, deletes or inserts. (ii) JXyDiff identifies all the inserted elements correctly, but matches only a few of the updated elements correctly. This results in the identification of a large number of inserts, deletes, updates and additional moves to restore the order of the siblings and compensate for incorrect matchings. (iii) 3DM identifies all the inserted elements correctly but doesn't always match the updated elements correctly, resulting in a significantly increased number of updates, inserts and deletes. In cases where elements are matched correctly, the updates are sometimes reported as inserts and deletes.

5 Evaluation Involving Real Genomic Data

To explore the performance of the XML difference algorithms on detection of real modifications, we compared two different releases (41 and 42) of the genomic data

Table 2. Number of changes detected between different versions of genomic data

Chromosome	X-Diff	JXyDiff	3DM
3	1805	13223	1864
5	1130	18475	679

of yeast chromosomes 3 and 5 obtained from Ensembl. The numbers of changes reported by each of the tools for both chromosomes are shown in Table 2. The following observations were made:

X-Diff: While X-Diff performs reasonably well in the case of singular changes within an element (describing gene, mRNA, CDS, or exon) and detects the majority correctly, the fact that it represents XML documents as unordered trees leads to incorrect reporting of changes in the following cases amongst others: (i) Large number of changes within one element (e.g., inserts, updates, deletes of elements `dbreference` or `qualifier`). In such cases, instead of reporting the various updates, inserts and/or deletes within the element, they may be matched incorrectly with other elements in the document. (ii) Update of the start and end location of an element. As pairs of `basePosition` elements are affected and the order among siblings is not taken into account in X-Diff, updates to the start position can be confused with updates to the end position and vice versa. The order of the start and end positions indicates the direction in which the gene is transcribed.

JXyDiff: For the most part, JXyDiff seems unable to match elements correctly, resulting in the reporting of significantly more changes in comparison to the other two algorithms. The numbers shown for JXyDiff in Table 2 exclude the more than 8,000 and 10,000 moves reported for chromosomes 3 and 5, respectively. In particular `CDS` elements tend to be mismatched, resulting in a large number of updates of all the subelements and attributes. Mismatching of CDS elements was observed earlier in the synthetic experiments (see Experiment 5) and is the result of the optimisation phase in which matches of leaf nodes are propagated bottom-up to their ancestors. The extent of the propagation is controlled by the weight of the subtree.

3DM: 3DM performs reasonably well in detecting changes between the two versions of genomic data. The following cases, amongst others, though, lead to reporting of additional changes: (i) Insert or delete of `dbreference` or `qualifier` elements within one element describing a CDS. In such cases, a newly inserted element might be matched to an existing element, which is reported as updated to represent the new element, and then the original is reported as inserted, therefore increasing the number of changes reported. Similar incorrect matchings can be observed for deleted elements, which aren't reported as deletes, but as a number of updates followed by deletes of different similar elements. This occurs more frequently in cases of a larger number of changes within one element, but also for newly inserted elements describing a gene or mRNA. (ii) Update of sequence elements. In such cases, the updated sequence might be reported as both an update and a copy followed by an update of another sequence element. (iii) Moves of elements, i.e., changes to the order of elements on the chromosome. In this case, not only are the elements affected by the move reported as such, but also a number of neighbouring elements, again resulting in an increase in the number of changes reported. The majority of these mismatches are the result of the q-gram distance measure used to match elements, and a lack of restriction on the number of matches of elements in the updated document with elements in the base document. Both cases have been observed in the experiments with synthetic data.

6 Conclusions

This comparative evaluation has shown that all the algorithms tested contain features that are effective at detecting a subset of changes relevant within the context of genomic data. However, none are useable without significant post-processing of the edit scripts. We have identified the following features of the representation of genomic data and of the algorithms that affect performance.

The task of matching versions of genomic data correctly and detecting changes between them is hindered by the following properties of the fairly flat structure of the data representing genomic sequences and their annotation: (i) Large numbers of siblings. (ii) Related elements are not easily identifiable as such (e.g., elements describing gene, mRNA, CDS, exons and sequence of a gene are only identifiable as related by the values of their qualifier child elements). (iii) Very similar contents of nodes and subtrees. Elements or subtrees sometimes can only be distinguished by values of attribute or descendant text nodes. The following properties of the algorithms affected their performance: (i) Treating XML documents as unordered trees can result in inappropriate matchings. (ii) Fuzzy matching using q-gram string distance measure can result in incorrect matchings of elements, in particular sequence elements. (iii) Propagating matches bottom-up to nodes with the same labels. This, in combination with the similarity of the contents of nodes and subtrees can result in incorrect matchings.

This suggests that selection and use of genome difference algorithms requires: (i) good understanding of the data over which the algorithm is to be used during algorithm selection; and (ii) significant tailoring of results to compensate for the production of non-minimal and challenging-to-interpret edit scripts.

References

1. Curwen, V., Eyras, E., et al.: The Ensembl Automatic Gene Annotation System. Genome Res. 14, 942–950 (2004)
2. Potter, S.C., Clarke, L., et al.: The Ensembl Analysis Pipeline. Genome Res. 14, 934–941 (2004)
3. Dwight, S.S., Balakrishnan, R., et al.: Saccharomyces Genome Database: Underlying Principles and Organisation. Brief Bioinform. 5, 9–22 (2004)

4. Flicek, P., Aken, B.L., et al.: Ensembl 2008. Nucl. Acicds Res. 36, D707–D714 (2008)
5. Kulikova, T., Akhtar, R., et al.: Embl Nucleotide Sequence Database in 2006. Nucl. Acids Res. 35, D16–D20 (2007)
6. Cochrane, G., Akhtar, R., et al.: Priorities for Nucleotide Trace, Sequence and Annotation Data Capture at the Ensembl Trace Archive and the Embl Nucleotide Sequence Database. Nucl. Acids Res. 36, D5–D12 (2008)
7. Wang, Y., DeWitt, D.J., Cai, J.Y.: X-diff: An Effective Change Detection Algorithm for Xml Documents. In: 19th International Conference on Data Engineering (ICDE 2003), pp. 519–530 (2003)
8. Cobena, G., Abiteboul, S., Marian, A.: Detecting Changes in Xml Documents. In: 18th International Conference on Data Engineering (ICDE 2002), pp. 41–52 (2002)
9. Lindholm, T.: A Three-Way Merge for Xml Documents. In: DocEng 2004: 2004 ACM Symposium on Document Engineering, pp. 1–10 (2004)
10. Benson, D.A., Karsch-Mizrachi, I., Lipman, D.J., Ostell, J., Wheeler, D.L.: Genbank. Nucl. Acids Res. 35, D21–D25 (2007)
11. Marian, A., Abiteboul, S., Cobena, G., Mignet, L.: Change-Centric Management of Versions in an Xml Warehouse. In: 27th International Conference on Very Large Data Bases (VLDB 2001), pp. 581–590 (2001)
12. Lindholm, T.: A 3-way Merging Algorithm for Synchronizing Ordered Trees - the 3DM Merging and Differencing Tool for Xml. Master's thesis, Department of Computer Science, Helsinki University of Technology (2001)
13. Maruyama, H., Tamura, K., Uramoto, R.: Digest Values for DOM (Domhash). IBM Research, Tokyo Research Laboratory (2000), `http://www.research.ibm.com/trl/projects/xml/xss4j/docs/rfc2803.html`
14. Tarjan, R.E.: Data Structures and Network Algorithms. CBMS-NSF Regional Conference Series in Applied Mathematics (1983)
15. Zhang, K.: A New Editing Based Distance Between Unordered Labeled Trees. In: Apostolico, A., Crochemore, M., Galil, Z., Manber, U. (eds.) CPM 1993. LNCS, vol. 684, pp. 254–265. Springer, Heidelberg (1993)
16. Ukkonen, E.: Approximate String-Matching with q-grams and Maximal Matches. Theoretical Computer Science 92, 191–211 (1992)