An Edit Script for Taxonomic Classifications

Roderic D. M. Page[1] and Gabriel Valiente[2]

[1] DEEB, IBLS, University of Glasgow, Glasgow G12 8QQ, United Kingdom,
`r.page@bio.gla.ac.uk`
[2] Department of Software, Technical University of Catalonia, E-08034 Barcelona,
`valiente@lsi.upc.edu`

Abstract. Taxonomy provides one of the most powerful ways to navigate sequence data bases but currently, users are forced to formulate queries according to a single taxonomic classification. Given that there is not universal agreement on the classification of organisms, providing a single classification places constraints on the questions biologists can ask. In this paper, we present a solution to the problem of querying sequence data bases using alternative classifications, based on the computation of an edit script that summarises the differences between two classification trees. Our algorithms find the shortest possible edit script based on the identification of all shared subtrees, and only take time quasi linear in the size of the trees because classification trees have unique node labels. These algorithms have been recently implemented, and the software is freely available for download.

Keywords. taxonomic classification, edit script, common subtrees

1 Motivation

Taxonomy provides one of the most powerful ways to navigate the National Center for Biotechnology Information (NCBI) sequence data bases. Every sequence in GenBank is associated with a taxon (which may be unidentified), and each taxon has a unique place in the NCBI taxonomy. Hence, not only can the user retrieve sequences for a given species (such as *Homo sapiens*), but also for a group of species, such as mammals (Mammalia) or animals (Animalia).

The NCBI provides a single classification, assembled from a variety of sources including published literature, a panel of expert advisors, and the taxonomy provided by users when they submit new sequences. Given that there is not universal agreement on the classification of organisms, providing a single classification places constraints on the questions biologists can ask.

To give a concrete example, Fig. 1 shows a simplified classification of animals, based on the current NCBI taxonomy (`http://www.ncbi.nlm.`

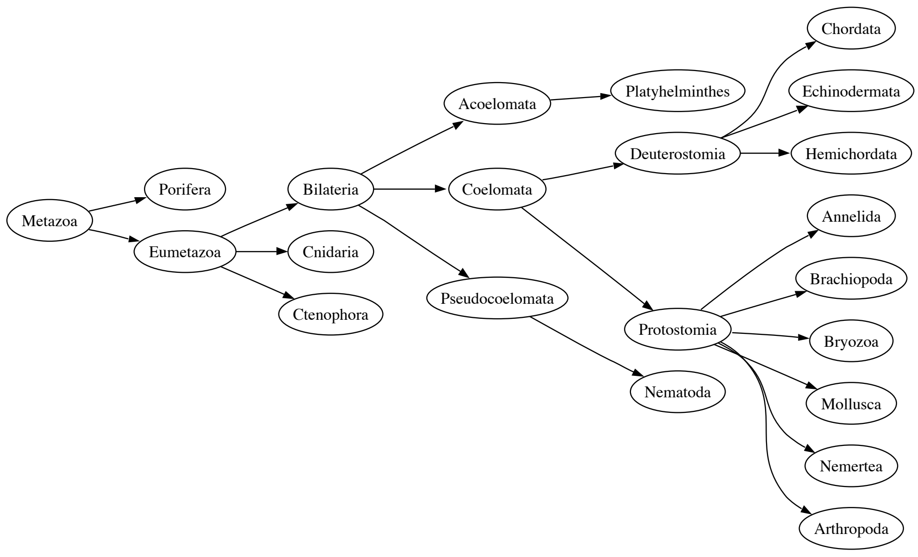

Fig. 1. A "traditional" view of animal relationships, based on the NCBI classification.

`nih.gov/Taxonomy/taxonomyhome.html/`). In this classification, the Bilateria are split into three groups (Acoelomata, Pseudocoelomata, and Coelomata) based on the nature of the internal body cavity (coelom). The Coelomata are themselves split into two groups, the Protostomia and the Deuterostomia, characterised by the fate of the blastopore during development (in the Protostomia this becomes the mouth, in the Deuterostomia it becomes the anus).

An alternative view of animal classification is shown in Fig. 2. The three-fold division based on body cavity disappears, leaving the fundamental split being between the Protostomia and the Deuterostomia. The Protostomia are divided into the Lophotrochozoa and the Ecdysozoa, the latter comprising arthropods, nematodes, and other moulting animals [1]. This classification has implications for comparative genomics. The best known animal genomes are human, *Drosophila* (fly), and *Caenorhabditis elegans* (nematode). Under the classical classification (Fig. 1), the coelomates human and *Drosophila* are more closely related to either other than either is to the aceolomate *C. elegans*, suggesting it would be most productive to compare our genome with that of *Drosophila*, rather than the more distant nematode. However, in the alternative classification (Fig. 2) *Drosphila* and *C. elegans* are more closely related to each other than either is to humans, and we have no (phylogenetic) reason for choosing one over the other as a point of reference for interpreting the human genome.

There is considerable debate about the merits of the two classifications [8, 5, 6]. However, because the NCBI provides only one classification users cannot, for example, easily query GenBank for all ecdysozoan sequences — this taxon simply does not exist in the NCBI database. Instead, users are forced to construct Boolean queries such as (Arthropoda AND Nematoda). While in this simplified example this is not a great hardship, as the trees get larger and the differences more profound, it becomes harder to pose a query that captures the taxa required.

One solution is simply to download the NCBI taxonomy, edit it to reflect the desired alternative classification, then use that to obtain sequences from taxa such as Ecdysozoa. It is reasonably straightforward to store a tree in a relational database an query it using SQL [3]. However, the NCBI taxonomy is continually growing as new organisms are sequenced. Hence, a locally edited classification will quickly become obsolete. Having to download a fresh copy and then manually edit it would quickly become tedious.

Here we suggest a solution based on the notion of an "edit script" that summarises the differences between two trees. Given two trees, T_1 and T_2, a script lists the operations required to convert T_1 into T_2. The script could be constructed manually, but it would be more efficient to generate it automatically. Hence, we imagine the following scenario. A user downloads the NCBI taxonomy tree (or that subtree relevant to their interests), then edits the tree to reflect their preferred classification. Using the algorithm we describe below, the user then computes the edit script that transforms the NCBI tree into their classification. When a new NCBI tree appears on the NCBI ftp site, the user downloads that tree and applies to edit script to regenerate their classification. In this way, the user need only edit the NCBI tree once.

2 Sketch

In this section we sketch out the basic idea. The details of the algorithm follow.

2.1 Taxonomic classifications

Although ideally classifications mirror phylogenetic relationships, it is important to distinguish between classifications and phylogenies. A taxonomic classification can be modelled as a rooted, labelled, unordered

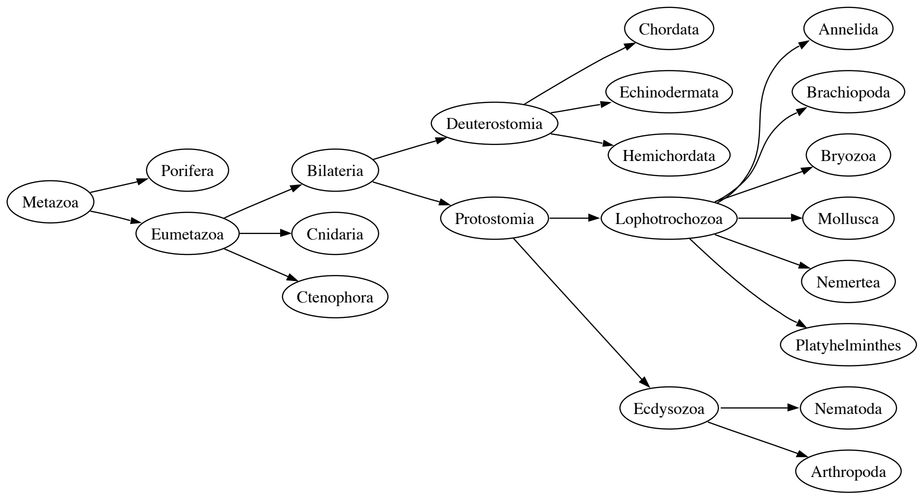

Fig. 2. An alternative tree of animals reflecting the "new animal classification".

tree. Unlike classifications, internal nodes of phylogenetic trees need not be labeled, although the internal nodes of a phylogeny may be decorated with measures of support (such as bootstrap values or Bayesian posterior probabilities).

2.2 Subtree isomorphism

Our approach is to first find subtree isomorphisms between the two trees, T_1 and T_2. A subtree is a connected subgraph of a tree. We distinguish between *top-down* and *bottom-up* subtree isomorphism. A top-down node matching the parent of each node in the matching is itself in the matching (excluding the root which has no parent). In a bottom-up matching, all the children of a node in the matching are also in the matching (Fig. 3).

The algorithm first finds all subtrees, including bottom-up and top-down subtrees, that are common to T_1 and T_2. We find all kinds of subtree because, by themselves the subtrees found by each method can be small (Fig. 4).

2.3 Script

Having identified common subtrees, we then list the operations needed to transform T_1 into T_2. The first step is to delete nodes in T_1 that are not in any of the shared subtrees. The deletion of a node entails deletion of all the edges incident with the deleted node. We then add nodes found

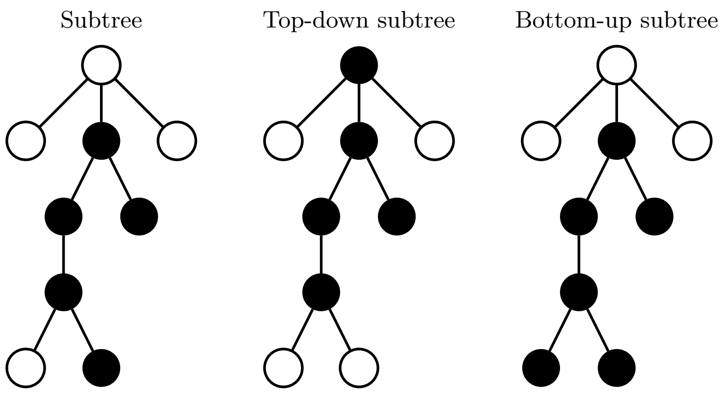

Fig. 3. Connected subgraph and top-down and bottom-up subtrees. In the top-down subtree the parent of any node in the subtree is itself in the subtree. In the bottom-up matching, the children of any node in the matching are also in the matching. Modified from [7].

only in T_2, and the corresponding edges. The size of the script depends on the size of the shared subtrees, hence it is desirable to find the largest such subtrees.

2.4 Complexity

In general, computation of the least number of operations needed to transform T_1 into T_2 is an NP-hard problem [9], even for binary trees with a label alphabet of size two, as long as node and edge deletions, insertions, and label substitutions are allowed. However, in the case of trees with unique node labels, node label substitutions are forbidden because they may generate trees with non-unique node labels [4], and the least number of operations or edit distance becomes a function of the size of shared subtrees [2]. By identifying the largest common subtrees, we obtain the shortest possible edit script.

3 Computing an Edit Script

Taxonomic classifications are modelled as trees with unique node labels, and this fact makes it easier to deal with trees in terms of their sets of node labels and node label pairs, as done for graphs with unique node labels in [4].

Definition 1. *Let $T = (V, E)$ be a tree. The label representation of T, denoted by $R(T)$, is given by $R(T) = (L, C)$, where $L = \{\ell(v) \mid v \in V\}$ and $C = \{(\ell(v), \ell(w)) \mid (v, w) \in E\}$.*

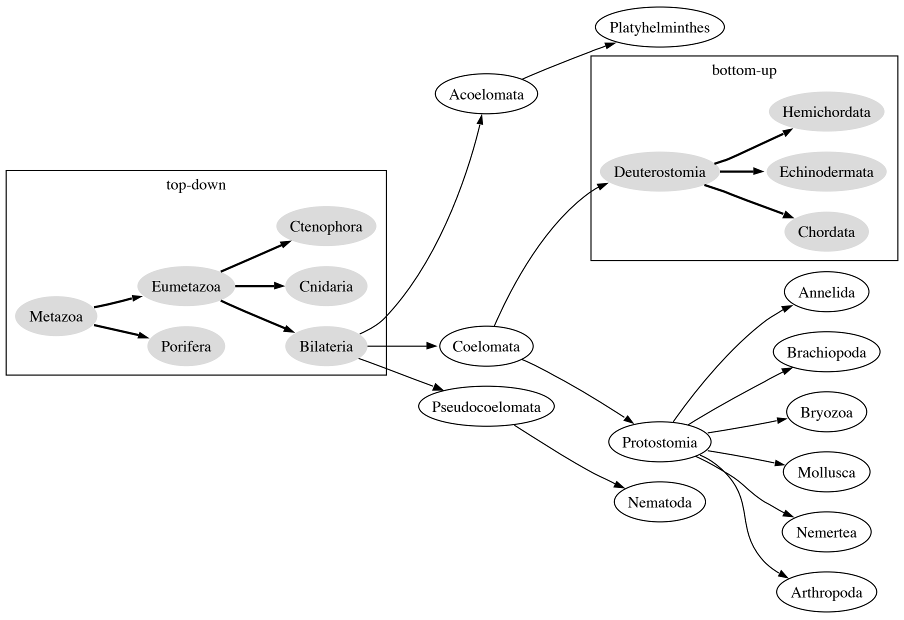

Fig. 4. The top-down and bottom-up subtree isomorphisms between the animal classifications shown in Figs. 1 and 2 (ignoring the trivial bottom-up subtrees that comprise a single leaf).

Thus, the label representation $R(T)$ of a tree T defines the equivalence class of all those trees that are isomorphic to T. The use of label representations simplifies the notation, because isomorphic trees have exactly the same label representation.

The edit operations of node and edge deletion and insertion, allow one to transform any given tree into any other tree. Label substitutions are forbidden because they may generate trees with non-unique node labels [4].

Definition 2. *Let $T_1 = (V_1, E_1)$ and $T_2 = (V_2, E_2)$ be trees, let $R(T_1) = (L_1, C_1)$, and let $R(T_2) = (L_2, C_2)$. Let also $C = L_1 \cup L_2 \cup \{\lambda\}$.*

A node edit operation between T_1 and T_2 is a pair $(a, b) \in C \times C$ with $a \neq \lambda$ or $b \neq \lambda$. A node edit operation of the form (a, λ) establishes deletion of the node $v \in V_1$ with $\ell(v) = a$ together with the edge $(parent(v), v)$, if v is not the root of T_1, and deletion of edge (v, x) for each child x of v in T_1. A node edit operation of the form (λ, b) establishes insertion of the node $w \in V_2$ with $\ell(w) = b$.

An edge edit operation between T_1 and T_2 is a triple $(a, b, c) \in C \times C \times C$ with $b \neq \lambda$ and $a \neq \lambda$ or $c \neq \lambda$. An edge edit operation of the form (a, b, λ) establishes deletion of the edge $(v, x) \in E_1$ with $\ell(v) = a$ and $\ell(x) = b$, and an edge edit operation of the form (λ, b, c) establishes insertion of the edge $(w, y) \in E_2$ with $\ell(w) = b$ and $\ell(y) = c$.

An edit operation is either a node edit operation or an edge edit operation.

An edit script between two trees is just a set of edit operations that, if applied in the right order (essentially, inserting an edge only after having inserted the nodes incident with the inserted edge), allow one to transform one tree into the other.

Definition 3. *An edit script between two trees $T_1 = (V_1, E_1)$ and $T_2 = (V_2, E_2)$ is a set of edit operations that transform $R(T_1)$ into $R(T_2)$.*

Given $R(T_1) = (L_1, C_1)$ and $R(T_2) = (L_2, C_2)$, an edit script between T_1 and T_2 can be easily obtained by sorting the label sets and computing set differences, as follows:

- Delete all nodes with labels in $L_1 \setminus L_2$
- Insert all nodes with labels in $L_2 \setminus L_1$
- Delete all edges with labels in $C_1 \setminus C_2$
- Insert all edges with labels in $C_2 \setminus C_1$

However, such a procedure does not, in general, lead to the shortest possible edit script, because some of the edge deletion operations may be redundant, given that deletion of a node entails deletion of all the edges incident with the deleted node.

The following, alternative procedure is based on the set of common node labels between the two trees, which can be easily obtained as the intersection of the sets of node labels in the label representation of the trees, that is, $C = L_1 \cap L_2 = \{\ell(v) \mid v \in V_1\} \cap \{\ell(w) \mid w \in V_2\}$. The procedure can be sketched as follows:

- Delete all nodes $v \in V_1$ with $\ell(v) \notin C$.
- Insert all nodes $w \in V_2$ with $\ell(w) \notin C$.
- Delete all edges $(v, x) \in E_1$ with $\ell(v), \ell(x) \in C$ and such that the node $w \in V_2$ with $\ell(v) = \ell(w)$ is not the parent in T_2 of the node $y \in V_2$ with $\ell(x) = \ell(y)$.
- Insert all edges $(w, y) \in E_2$ with $\ell(w), \ell(y) \in C$ and such that the node $v \in V_1$ with $\ell(v) = \ell(w)$ is not the parent in T_1 of the node $x \in V_1$ with $\ell(x) = \ell(y)$.

– Insert all edges $(w, y) \in E_2$ such that $\ell(w) \notin C$ or $\ell(y) \notin C$.

The following algorithm is a more detailed description of the previous procedure.

Algorithm 1. *Let C be a set of common node labels of T_1 and T_2. A function call of the form edit script (T_1, T_2, C) returns a set E of elementary edit operations that transform T_1 into T_2.*

```
edit script($T_1, T_2, C$)
$C_1 := \{v \in V_1 \mid \ell(v) \in C\}$
$C_2 := \{w \in V_2 \mid \ell(w) \in C\}$
$E := \emptyset$
foreach node $v \in V_1 \setminus C_1$ do
    $E := E \cup \{(\ell(v), \lambda)\}$
foreach node $w \in V_2 \setminus C_2$ do
    $E := E \cup \{(\lambda, \ell(w))\}$
foreach node $v \in C_1$ do
    let $w$ be the node of $T_2$ with $\ell(v) = \ell(w)$
    foreach child $x$ of $v$ in $T_1$ do
        if $x \in C_1$ then
            let $y$ be the node of $T_2$ with $\ell(x) = \ell(y)$
            if $w$ is not the parent of $y$ in $T_2$ then
                $E := E \cup \{(\ell(v), \ell(x), \lambda)\}$
foreach node $w \in C_2$ do
    let $v$ be the node of $T_1$ with $\ell(v) = \ell(w)$
    foreach child $y$ of $w$ in $T_2$ do
        if $y \in C_2$ then
            let $x$ be the node of $T_1$ with $\ell(x) = \ell(y)$
            if $v$ is not the parent of $x$ in $T_1$ then
                $E := E \cup \{(\ell(w), \ell(y), \lambda)\}$
foreach node $w \in V_2$ do
    foreach child $y$ of $w$ in $T_2$ do
        if $w \notin C_2$ or $y \notin C_2$ then
            $E := E \cup \{(\ell(w), \ell(y), \lambda)\}$
return $E$
```

Correctness of the edit script algorithm is easy to establish.

Theorem 1. *Let T_1 and T_2 be trees, let $C \subseteq \ell(V_1) \cap \ell(V_2)$, let $E = \textit{edit script}\,(T_1, T_2, C)$, and let T_1' be the result of applying the set of edit operations in E to T_1. Then, T_1' and T_2 are isomorphic.*

Proof. It has to be shown that $R(T_1') = R(T_2)$. Let $R(T_1) = (L_1, C_1)$ and $R(T_2) = (L_2, C_2)$. The edit script establishes the deletion of all nodes with labels in $L_1 \setminus C$ and the insertion of all nodes with labels in $L_2 \setminus C$. Thus, $L_1' = L_1 \setminus (L_1 \setminus C) \cup (L_2 \setminus C) = C \cup (L_2 \setminus C) = L_2$.

The edit script also establishes the deletion of all edges with source and target labels in $(C_1 \cap C \times C) \setminus C_2$, the insertion of all edges with source and target labels in $(C_2 \cap C \times C) \setminus C_1$, and the insertion of all edges with source or target label in $L_2 \setminus C$, that is, of all edges in $C_2 \setminus (C_2 \cap C \times C)$. Furthermore, the deletion of all nodes with labels in $L_1 \setminus C$ entails the deletion of all edges with source or target label in $L_1 \setminus C$, that is, of all edges in $C_1 \setminus (C_1 \cap C \times C)$.

Now, $C_1 = (C_1 \setminus C_2) \cup (C_1 \cap C_2) = ((C_1 \cap C \times C) \setminus C_2) \cup ((C_1 \setminus (C_1 \cap C \times C)) \setminus C_2) \cup (C_1 \cap C_2)$. In a similar vein, $C_2 = ((C_2 \cap C \times C) \setminus C_1) \cup ((C_2 \setminus (C_2 \cap C \times C)) \setminus C_1) \cup (C_1 \cap C_2)$.

Thus, $C_1' = C_1 \setminus ((C_1 \cap C \times C) \setminus C_2) \setminus ((C_1 \setminus (C_1 \cap C \times C)) \setminus C_2) \cup ((C_2 \cap C \times C) \setminus C_1) \cup ((C_2 \setminus (C_2 \cap C \times C)) \setminus C_1) = (C_1 \cap C_2) \cup ((C_2 \cap C \times C) \setminus C_1) \cup ((C_2 \setminus (C_2 \cap C \times C)) \setminus C_1) = C_2$ and therefore, $R(T_1') = (L_1', C_1') = (L_2, C_2) = R(T_2)$, that is, T_1' and T_2 are isomorphic. $\square$

The edit script algorithm can be implemented to take time quasi linear in the size of the trees, by using any efficient dictionary data structure to represent the set of common node labels. The same dictionary data structure allows one to compute the set of common node labels within the same time bound and thus, the whole procedure can be implemented to take time quasi linear in the size of the trees. In particular, our `C++` implementation uses the STL associative container `set<string>` as representation of the set of shared node labels.

4 Discussion

4.1 Application

Given the two trees in Figs. 1 and 2, the edit script for these trees is:

```
delete node Pseudocoelemata
delete node Coelomata
delete node Protostomia
delete node Acoelomata
insert node Ecdysozoa
insert node Lophotrochozoa
insert node Protostomia
insert edge Bilateria -> Deuterostomia
insert edge Bilateria -> Protostomia
insert edge Ecdysozoa -> Nematoda
insert edge Ecdysozoa -> Arthropoda
insert edge Lophotrochozoa -> Annelida
insert edge Lophotrochozoa -> Brachiopoda
insert edge Lophotrochozoa -> Bryozoa
insert edge Lophotrochozoa -> Mollusca
insert edge Lophotrochozoa -> Nemertea
insert edge Lophotrochozoa -> Platyhelminthes
insert edge Protostomia -> Lophotrochozoa
insert edge Protostomia -> Ecdysozoa
```

Applying the script to the NCBI tree yields the tree shown in Fig 5. This is identical to the tree shown in Fig. 2.

4.2 Limitations

The size of the edit script will be a function of the size of the input trees, and the degree to which they differ. At the time of writing, there are 83,802 metazoan taxa in GenBank. Given that the disagreement between the Coelomata and Ecdysozoa hypotheses concerns the deep level relationships, we can simplify the task by reducing the subtrees about which there is little disagreement to single nodes. For example, the 36,746 Arthropoda can be represented by a single node. Hence, the tree shown in Fig. 1 is greatly simplified.

4.3 Implementation

We have implemented the edit function in a `C++` program that makes use of the Graph Template Library (GTL) available from `http://infosun.fmi.uni-passau.de/GTL/`. The code has been compiled and tested with the GNU `gcc` compiler on Mac OS X and Linux machines, and is available from `http://darwin.zoology.gla.ac.uk/~rpage/forest/`. The software comprises two programs, `forest` and `script`. The program `forest` takes two trees in GML format (the original tree and the edited tree) and computes an edit script. Given this script and the original tree, `script` generates the edited tree.

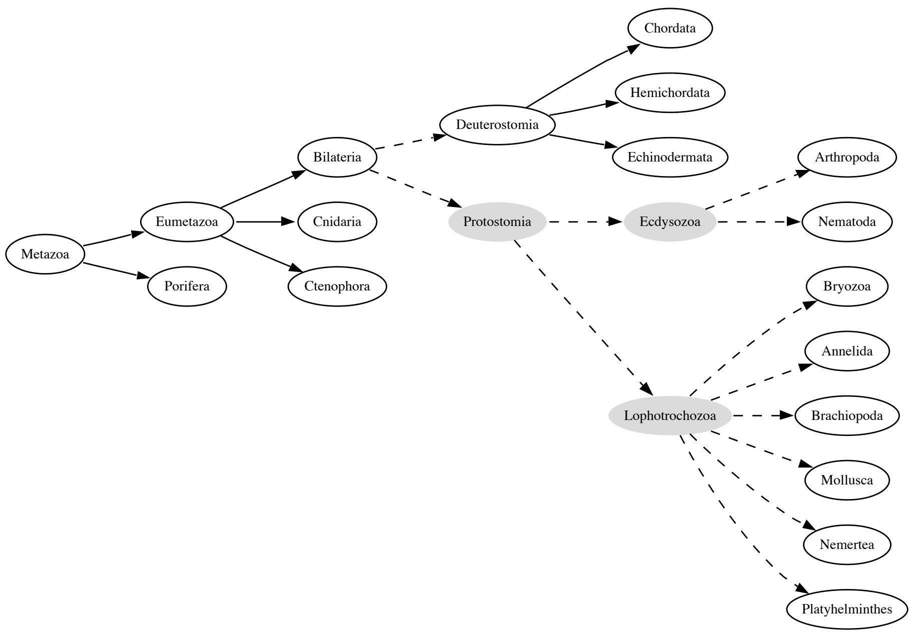

Fig. 5. The result of applying the edit script to the tree in Fig. 1. This tree is the same as that shown in Fig. 2. Nodes which have been inserted into the tree are filled with light grey. A dashed line represents an edge that has been added to the original tree.

5 Acknowledgements

This work was funded by BBSRC grant BB/C004310/1 to RDMP, by the Spanish CICYT, project GRAMMARS (TIN2004-07925-C03-01), and by the Japan Society for the Promotion of Science through Long-term Invitation Fellowship L05511 for visiting JAIST (Japan Advanced Institute of Science and Technology).

References

1. Aguinaldo, A., Turbeville, J., Linford, L., Rivera, M., Garey, J., Raff, R., Lake, J.: Evidence for a clade of nematodes, arthropods and other moulting animals. Nature **387** (1997) 489–493
2. Bunke, H.: On a relation between graph edit distance and maximum common subgraph. Pattern Recogn. Lett. **18** (1997) 689–694
3. Celko, J.: SQL for Smarties: Advanced SQL Programming. 2nd edn. Morgan Kaufmann, San Fransisco, California (1999)
4. Dickinson, P.J., Bunke, H., Dadej, A., Kraetzl, M.: On graphs with unique node labels. In Hancock, E.R., Vento, M., eds.: Proc. 4th IAPR Int. Workshop Graph

Based Representations in Pattern Recognition. Volume 2726 of Lecture Notes in Computer Science., Berlin, Springer-Verlag (2003) 13–23

5. Philip, G., Creevey, C., McInerney, J.: The Opisthokonta and the Ecdysozoa may not be clades: Stronger support for the grouping of plant and animal than for animal and fungi and stronger support for the Coelomata than Ecdysozoa. Mol. Biol. Evol. **22** (2005) 1175–1184
6. Philippe, H., Lartillot, N., Brinkmann, H.: Multigene analyses of bilaterian animals corroborate the monophyly of Ecdysozoa, Lophotrochozoa and Protostomia. Mol. Biol. Evol. **22** (2005) 1246–1253
7. Valiente, G.: Algorithms on Trees and Graphs. Springer-Verlag, Berlin (2002)
8. Wolf, Y., Rogozin, I., Koonin, E.: Coelomata and not Ecdysozoa: evidence from genome-wide phylogenetic analysis. Genome Res. **14** (2004) 29–36
9. Zhang, K., Statman, R., Shasha, D.: On the editing distance between unordered labeled trees. Inform. Process. Lett. **42** (1992) 133–139