Finding K Optimum Edit Scripts between an XML Document and a Regular Tree Grammar

Nobutaka Suzuki

Graduate School of Library, Information and Media Studies
University of Tsukuba
1-2, Kasuga, Tsukuba, Ibaraki 305-8550, Japan
nsuzuki@slis.tsukuba.ac.jp

Abstract. Finding optimum and near optimum edit scripts between an XML document and a schema is essential to correcting invalid XML documents. In this paper, we consider finding K optimum edit scripts between an XML document and a regular tree grammar. We first prove that the problem is NP-hard. We next show a pseudopolynomial-time algorithm for solving the problem.

1 Introduction

An edit script between an (invalid) XML document and a schema is a sequence of edit operations to transform the document into a valid one against the schema. Thus finding such an edit script, especially an optimum one, is essential to correcting invalid XML documents. On the other hand, we often have to find near optimum edit scripts as well as an optimum one, since an optimum edit script is not necessarily the real "best" solution. In this paper, we consider an algorithm for finding K optimum edit scripts between an XML document and a regular tree grammar.

There are three common tree grammars to describe an XML schema; local tree grammar (corresponding to DTD), single-type tree grammar (corresponding to W3C XML Schema), and regular tree grammar (corresponding to RELAX NG) [8]. It is shown that single-type tree grammar is more expressive than local tree grammar, and regular tree grammar is more expressive than single-type tree grammar. Therefore, our algorithm is applicable to a local tree grammar and a single-type tree grammar as well. The expressive power of regular tree grammar is equivalent to that of specialized DTD [10].

An XML document is modeled as a labeled ordered tree, where a text node is labeled by `pcdata` (e.g., Fig. 1(a)). An edit script is a sequence of edit operations, where each edit operation is either *ren* (rename the label of a node), *add* (add a new node), or *del* (delete a node). Each edit operation is associated with a cost. Let t be a tree, G be a regular tree grammar, s be an edit script, and $s(t)$ be the tree obtained by applying s to t. We say that s is an edit script between t and G if $s(t)$ is valid against G. $s_1, s_2, \cdots, s_K$ are called K optimum edit scripts between t and G if for every $1 \leq k \leq K$ s_k is an edit script between t and G such that s_k has the kth least cost among the edit scripts between t and G.

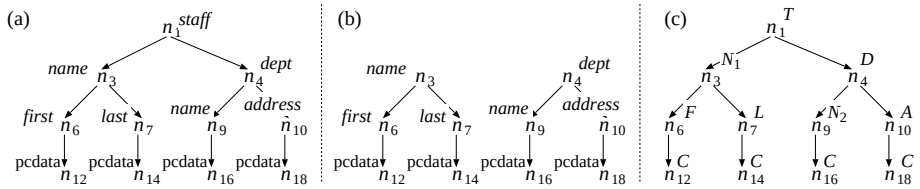

Fig. 1. (a) tree t, (b) subforest t_{n_3,n_4} of t, and (c) the interpretation ν of t against G.

We often have to correct invalid XML documents if we manage a document database, since (i) documents and the schema on a database are changed over time and (ii) documents migrated into the database are not necessarily valid against the schema. Unfortunately, such documents and a schema may be much complicated, and the amount of documents you need to correct is often very large. Therefore, finding manually a correct transformation for each invalid document is surely impractical. On the other hand, our algorithm can present K optimum edit scripts between an XML document and a regular tree grammar, which are reasonable solutions to correcting the document since the real "best" solution is likely to be an optimum or near optimum edit script between the document and the grammar.

Finding an edit script between data has originally been studied in terms of strings [7, 11]. Then there have been studies on finding the minimum edit distance between a string and a language (e.g., [13]). However, these studies consider only strings and deal with neither a tree nor a tree grammar. So far many algorithms that find an optimum edit script between two ordered trees or XML documents have been proposed (e.g., [4, 5, 9, 14]). However, these algorithms consider neither schema nor finding K optimum edit scripts. The algorithm in [1] measures the distance between an unordered XML document (unordered tree) and a DTD, and the algorithm in [3] finds the minimum edit distance between an XML document and a regular hedge grammar. However, these algorithms only measures a distance and do not present any actual edit script. The author proposes an algorithm for finding an optimum edit script between an XML document and a DTD [12], but DTD is strictly less expressive than regular tree grammar and the algorithm finds only one edit script rather than K optimum edit scripts. Since an optimum edit script is not always the best solution, finding K optimum edit scripts is a more practically important problem. For example, let t be the tree in Fig. 2(a) and D be the DTD in Fig. 2(b). Assume that the costs of *ren*, *add*, *del* are 1, 2, 3, respectively. Then the optimum edit script between t and D is to rename n_7 to i, the second optimum one is to add a new node labeled by *ul* between n_3 and n_7, and the third one is to delete n_3 or n_7. It is very likely that the second or the third one is the best solution.

In this paper, we first prove that finding K optimum edit scripts between an XML document and a regular tree grammar is NP-hard, then show a pseudopolynomial-time algorithm for solving the problem.

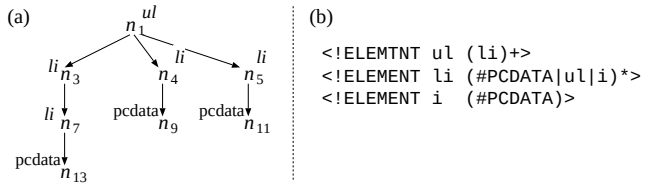

Fig. 2. (a) tree t and (b) DTD D.

2 Definitions

An XML document is modeled as a labeled ordered tree. For simplicity, attributes are not considered. In what follows, we use the term tree when we mean labeled ordered tree. A leaf node represents a text node and an internal node represents an element. Each node n is labeled by a symbol called *terminal*, denoted $\lambda(n)$. For example, $\lambda(n_1) = \mathit{staff}$ and $\lambda(n_3) = \mathit{name}$ in Fig. 1(a). We assume that for any leaf node n, $\lambda(n) = \texttt{pcdata}$.

For the simplicity of the algorithm, we denote each node of a tree t as follows (Fig. 1(a) is an example).

1. The root node of t is denoted n_1.
2. Let n_i be a node in t, and let n_j be the node next to n_i in breath-first order. If n_i and n_j are siblings, then $j = i + 1$, otherwise $j = i + 2$.

No node in t is renumbered even if a new node is added to t or a node in t is deleted. By t_{n_i} we mean the subtree of t rooted at node n_i. Similarly, for consecutive siblings $n_h, \cdots, n_j$ in t, by t_{n_h,n_j} we mean the subforest of t rooted at $n_h, \cdots, n_j$ (Fig. 1(b) is an example). We say that the topmost nodes n_3, n_4 in Fig. 1(b) are the *roots* of t_{n_3,n_4}. By $ch(t, n_i)$, we mean the sequence of the children of a node n_i in a forest t (e.g., $ch(t, n_1) = n_3, n_4$ in Fig. 1(a)).

A *regular tree grammar* is a 4-tuple $G = (N, \Sigma, S, P)$, where N is a set of *non-terminals*, Σ is a set of *terminals*, $S \subseteq N$ is a set of *start symbols*, and P is a set of *productions* of the form $X \rightarrow a(r)$ such that $X \in N$, $a \in \Sigma$, and that r is a regular expression over N. For a production $X \rightarrow a(r)$, we say that X is the *left-hand side*, a is the *right-hand side*, and r is the *content model* of the production. Let t be a forest and ν be a mapping from every node n in t to a non-terminal $\nu(n) \in N$. For a sequence $n_h, \cdots, n_j$ of nodes, we define that $\nu(n_h, \cdots, n_j) = \nu(n_h) \cdots \nu(n_j)$. We say that ν is an *interpretation* of t against G if (i) $\nu(n) \in S$ for every root n of t (if t is a forest, then t may have more than one roots), and (ii) for every node n_i in t, there is a production $X \rightarrow a(r) \in P$ such that $\nu(n_i) = X$, $\lambda(n_i) = a$, and that $\nu(ch(t, n_i)) \in L(r)$, where $L(r)$ is the language represented by r. A forest t is *valid* against G if there is an interpretation of t against G.

Example 1. Let $G = (N, \Sigma, S, P)$ be a regular tree grammar, where $N = \{T, N_1, N_2, F, L, D, A, C\}$, $\Sigma = \{\mathit{staff}, \mathit{name}, \mathit{first}, \mathit{last}, \mathit{dept}, \mathit{address}, \texttt{pcdata}\}$,

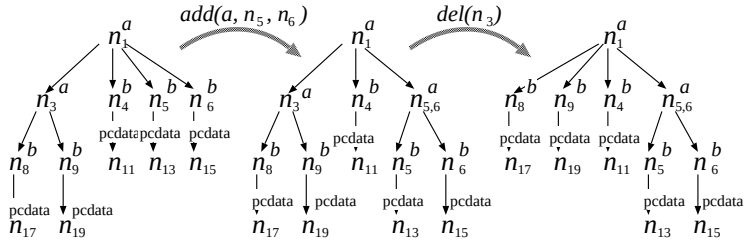

Fig. 3. An example of *add* and *del* operations.

$S = \{T\}$, $P = \{T \rightarrow \mathit{staff}(N_1 D), N_1 \rightarrow \mathit{name}(FL), D \rightarrow \mathit{dept}(N_2 A), F \rightarrow \mathit{first}(C), L \rightarrow \mathit{last}(C), N_2 \rightarrow \mathit{name}(C), A \rightarrow \mathit{address}(C), C \rightarrow \texttt{pcdata}(\epsilon)\}$. Let t be the tree in Fig. 1(a) and ν be the mapping illustrated in Fig. 1(c). Then ν is the interpretation of t against G.

Note that there is no DTD equivalent to G. G states that *name* has two distinct types; the *name* element of a *staff* element must consist of *first* and *last* elements and the *name* element of a *dept* element must consist only of a text string. Such a constraint cannot be represented by any DTD. □

We use the following *edit operations* to transform a tree (Fig. 3 is an example).

- $\mathit{ren}(n_i, a)$: Changes the label of an internal node n_i to $a \in \Sigma \backslash \{\texttt{pcdata}\}$.
- $\mathit{del}(n_i)$: Deletes an internal node n_i. We assume that no leaf node is deleted (i.e., the texts in a document are preserved).
- $\mathit{add}(a, n_h, n_j)$: Adds a new node (denoted $n_{h,j}$) labeled by $a \in \Sigma \backslash \{\texttt{pcdata}\}$ as the parent of siblings $n_h, \cdots, n_j$.
- *nil*: An auxiliary edit operation stating that there is no valid edit operation to transform a tree.

Let t be a forest. An *edit script* for t (w.r.t. a set Σ of terminals) is a sequence of edit operations defined by the following (1)–(4). By $s(t)$ we mean the forest obtained by applying each edit operation in an edit script s (from left to right) to t. If an edit script s contains an edit operation *op*, then we write $\mathit{op} \in s$.

1. ϵ is an edit script for t of length zero, where $\epsilon(t)$ is identical to t.
2. If s' is an edit script for t, n_i is an internal node in t, $\mathit{del}(n_i) \notin s'$, and $a \in \Sigma \backslash \{\texttt{pcdata}\}$, then $s'\mathit{ren}(n_i, a)$ is an edit script for t.
3. If s' is an edit script for t, n_i is an internal node in t, and $\mathit{del}(n_i) \notin s'$, then $s'\mathit{del}(n_i)$ is an edit script for t.
4. If s' is an edit script for t, n_h and n_j are preserved siblings in $s'(t)$, s' contains no duplicate edit operation of $\mathit{add}(a, n_h, n_j)$ (defined below), and $a \in \Sigma \backslash \{\texttt{pcdata}\}$, then $s'\mathit{add}(a, n_h, n_j)$ is an edit script for t.

Due to the construction of the algorithm, we have two restrictions in (4). We say that n_h and n_j are *preserved* siblings if n_h and n_j are siblings in t with $h \leq j$ and n_h and n_j remain siblings in $s'(t)$. We say that $\mathit{add}(a', n_{h'}, n_{j'})$ is a

duplicate edit operation of $add(a, n_h, n_j)$ if $n_{h'} = n_h$ and $n_{j'} = n_j$. (The latter restriction can be eliminated without difficulty, but details are omitted because of space limitation.)

Let $G = (N, \Sigma, S, P)$ be a regular tree grammar, t be a forest, and s be an edit script for t. We say that s is an edit script *between* t and G if $s(t)$ is valid against G. Each edit operation o is associated with a *cost* denoted $\gamma(o)$, where $\gamma(o) \geq 0$. We assume that for any internal node n_i and any $a \in \Sigma \backslash \{\texttt{pcdata}\}$, $\gamma(ren(n_i, a)) = 0$ if $\lambda(n_i) = a$. We also assume that $\gamma(nil) = \infty$. The cost of an edit script s is defined as $\gamma(s) = \sum_{o \in s} \gamma(o)$. For a positive integer K, if the following condition holds, then $s_1, s_2, \cdots, s_K$ are K *optimum edit scripts* between t and G.

- For every $1 \leq k \leq K$, s_k is a *kth optimum edit script* between t and G; that is, s_k is an edit script between t and G such that $s_k(t) \notin S_{k-1}$ and that $\gamma(s_k) \leq \gamma(s)$ for any edit script s between t and G with $s(t) \notin S_{k-1}$, where $S_{k-1} = \{s_1(t), s_2(t), \cdots, s_{k-1}(t)\}$ (we assume that if there is no such s_k, then $s_k = nil$).

In what follows, for simplicity we assume that the root of an XML document cannot be deleted and no new node can be added as the root.

3 NP-Hardness

In this section, we show that finding K optimum edit scripts between a tree and a regular tree grammar is NP-hard. For a tree t and edit scripts s, s' for t, s is *distinct* to s' if $s(t) \neq s'(t)$. The *Kth optimum edit script problem* is to decide, for a tree t, a regular tree grammar G, and positive integers B and K, whether there are K or more distinct edit scripts s between t and G such that $\gamma(s) \leq B$. We have the following theorem.

Theorem 1. *The Kth optimum edit script problem is NP-hard even if only ren operation is allowed.*

Proof(sketch): The theorem can be shown by reducing the Kth largest subset problem to the Kth optimum edit script problem. Details are omitted because of space limitation. □

Thus, it is unlikely that we can find K optimum edit scripts between t and G in time polynomial in the size of t, the size of G, and $\log K$. However, we show in the next section that the problem is solvable in pseudopolynomial time, i.e., in time polynomial in the size of t, the size of G, and K.

4 Overview of the Algorithm

Before presenting the algorithm formally, we show an overview of the algorithm. For simplicity, in this section we assume that $K = 1$. Let t be a tree and G be a regular tree grammar. In short, our algorithm works as follows.

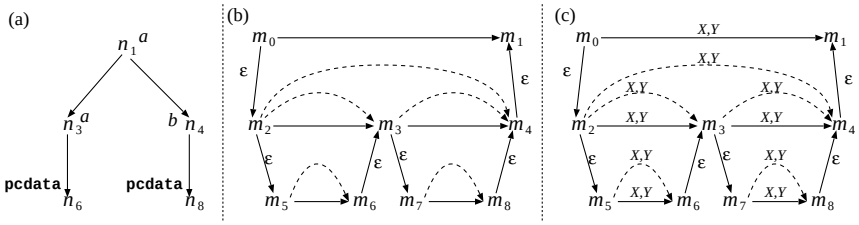

Fig. 4. (a) tree t, (b) graph $H(t)$, and (c) graph $H_1(t, N)$. In (c), $m_i \xrightarrow{X,Y} m_j$ stands for two edges $m_i \xrightarrow{X} m_j$ and $m_i \xrightarrow{Y} m_j$.

1. Any node in t may have various sequences of children according to edit scripts applied to t. We first gather all such sequences into a single graph, denoted $H_1(t, N)$.
2. For each node n_i in t, we find an "optimum" sequence of children of n_i by solving a shortest path problem over $H_1(t, N)$. In fact, finding such a sequence leads to an optimum edit script for t_{n_i}, as explained later.

4.1 Constructing $H_1(t, N)$

A node in a tree may have various sequences of children according to edit scripts applied to the tree. For example, let t be the tree in Fig. 4(a).

- Let $s_1 = \epsilon$. Then $ch(s_1(t), n_1) = n_3, n_4$.
- Let $s_2 = add(a, n_3, n_3)$. Then $ch(s_2(t), n_1) = n_{3,3}, n_4$.
- Let $s_3 = del(n_3)$. Then $ch(s_3(t), n_1) = n_6, n_4$.

Here, let us represent each node n_i ($n_{h,j}$) by an edge $m_{i-1} \rightarrow m_i$ (resp., $m_{h-1} \dashrightarrow m_j$). Then a sequence of children is represented by a path, e.g., $n_{3,3}, n_4$ is represented by path $m_2 \dashrightarrow m_3 \rightarrow m_4$. For every internal node n_i in t with $ch(t, n_i) = n_h, \cdots, n_j$, we also put a pair $(m_{i-1} \xrightarrow{\epsilon} m_{h-1}, m_j \xrightarrow{\epsilon} m_i)$ of edges. This pair is used to visit the children of n_i when n_i is deleted (e.g., $ch(s_3(t), n_1) = n_6, n_4$ is represented by a path $m_2 \xrightarrow{\epsilon} m_5 \rightarrow m_6 \xrightarrow{\epsilon} m_3 \rightarrow m_4$). Let $H(t)$ be a graph consisting of all such edges (Fig. 4(b)). $H(t)$ is "complete"; for any edit script s for t and for any node n_i in t with $ch(t, n_i) = n_h, \cdots, n_j$, $H(t)$ contains a path from m_{h-1} to m_j that represents $ch(s(t), n_i)$.

Let $G = (N, \Sigma, S, P)$ be a regular tree grammar. Since we have to check if a transformed tree is valid against G, we also have to consider the non-terminal associated with each node. Thus we incorporate non-terminals into $H(t)$. This is achieved as follows (assuming $N = \{X, Y\}$): replace each $m_{i-1} \rightarrow m_i$ ($m_{h-1} \dashrightarrow m_j$) in $H(t)$ by two edges $m_{i-1} \xrightarrow{X} m_i$ and $m_{i-1} \xrightarrow{Y} m_i$ (resp., $m_{h-1} \overset{X}{\dashrightarrow} m_j$ and $m_{h-1} \overset{Y}{\dashrightarrow} m_j$) (Fig. 4(c)). For example, path $m_2 \overset{X}{\dashrightarrow} m_3 \xrightarrow{Y} m_4$ represents a sequence $n_{3,3}, n_4$ of children such that X is associated with $n_{3,3}$ and Y is associated with n_4. The obtained graph is denoted $H_1(t, N)$.

4.2 Finding an Optimum Edit Script

Our algorithm is based on the following idea: *we can find an optimum edit script for t_{n_i} provided that we have an optimum edit script for every proper subtree of t_{n_i}*. In our algorithm, each such an edit script is associated with a corresponding edge in $H_1(t, N)$. For example, let $H_1(t, N)$ be the graph in Fig. 4(c). Assume that for each edge $m_{i-1} \xrightarrow{X} m_i$ in $H_1(t, N)$ except $m_0 \xrightarrow{X} m_1$, we have a corresponding optimum edit script between t_{n_i} and $(N, \Sigma, \{X\}, P)$, denoted $\sigma(m_{i-1} \xrightarrow{X} m_i)$ (and that we also have $\sigma(m_{h-1} \overset{X}{\dashrightarrow} m_j)$ for each $m_{h-1} \overset{X}{\dashrightarrow} m_j$).[1] Let us consider finding $\sigma(m_0 \xrightarrow{X} m_1)$, i.e., an optimum edit script between t_{n_1} and $(N, \Sigma, \{X\}, P)$. For simplicity assume that $X \to a(r)$ is the only production whose left hand side is X. Then since n_1 must be labeled by a, $\sigma(m_0 \xrightarrow{X} m_1)$ must consist of (i) $ren(n_1, a)$ and (ii) an optimum edit script s' between t_{n_3,n_4} and (N, Σ, N, P) such that the roots of $s'(t_{n_3,n_4})$ match r. To obtain s', we use the following crucial property of $H_1(t, N)$: for any path p from m_2 to m_4 in $H_1(t, N)$, $\sigma(p)$ is an optimum edit script for t_{n_3,n_4} w.r.t. the nodes represented by the path.[2] For example, if $p = m_2 \overset{X}{\dashrightarrow} m_3 \xrightarrow{Y} m_4$, then $\sigma(p)$ is an optimum edit script between t_{n_3,n_4} and (N, Σ, N, P), assuming that $n_{3,3}, n_4$ are the children of n_1 and are associated with X and Y, respectively. This property and the completeness of $H_1(t, N)$ imply that, in order to find s', it suffices to find a path p from m_2 to m_4 such that

1. the non-terminals on p matches r, and that
2. the cost of $\sigma(p)$ is the smallest of the paths satisfying (1).

Such a path can easily be found by solving a shortest path problem in the "intersection graph" of $H_1(t, N)$ and an NFA representing r.

5 The Algorithm

In this section, we first show some preliminaries, then show the algorithm.

5.1 Preliminary Definitions

We first define $H_K(t, N)$ and show some related definitions. We next show some definitions to check if a path on $H_K(t, N)$, representing a sequence of nodes, satisfies a given regular expression.

---

[1] We also assume that for any pair $(m_{i-1} \xrightarrow{\epsilon} m_{h-1}, m_j \xrightarrow{\epsilon} m_i)$, $\sigma(m_{i-1} \xrightarrow{\epsilon} m_{h-1}) = del(n_i)$ and $\sigma(m_j \xrightarrow{\epsilon} m_i) = \epsilon$.

[2] If $p = e_1 \cdots e_n$, then $\sigma(p) = \sigma(e_1) \cdots \sigma(e_n)$.

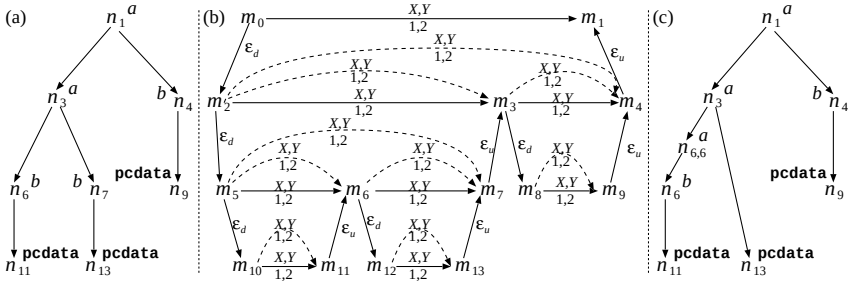

Fig. 5. (a) tree t, (b) graph $H_K(t, N)$, and (c) tree $s(t)$. In (b), $m_i \xrightarrow[1,2]{X,Y} m_j$ in $H_K(t, N)$ stands for four edges $m_i \xrightarrow[1]{X} m_j$, $m_i \xrightarrow[2]{X} m_j$, $m_i \xrightarrow[1]{Y} m_j$, $m_i \xrightarrow[2]{Y} m_j$.

Graph $H_K(t, N)$ Let t be a tree, $G = (N, \Sigma, S, P)$ be a regular tree grammar, and K be a positive integer. Let $z = \max\{i \mid n_i \text{ is a node in } t\}$. $H_K(t, N)$ is defined as a graph consisting of $z + 1$ nodes $m_0, m_1, \cdots, m_z$ and the following edges (Fig. 5(a,b) is an example with $N = \{X, Y\}$ and $K = 2$).

- For every node n_i in t, every $X \in N$, and every $1 \le k \le K$, $H_K(t, N)$ has an edge $m_{i-1} \xrightarrow[k]{X} m_i$, called *node edge (n-edge)*. The non-terminal X means that n_i is associated with X under some mapping ν, i.e., $\nu(n_i) = X$.
- For every pair (n_h, n_j) of siblings in t $(1 < h \le j)$, every $X \in N$, and every $1 \le k \le K$, $H_K(t, N)$ has an edge $m_{h-1} \overset{X}{\underset{k}{\dashrightarrow}} m_j$. This is called *add edge (a-edge)* and means adding a new node $n_{h,j}$ under some mapping ν such that $\nu(n_{h,j}) = X$.
- For every internal node n_i in t with $ch(t, n_i) = n_h, \cdots, n_j$, $H_K(t, N)$ has a *downward ϵ-edge (ϵ_d-edge)* $m_{i-1} \xrightarrow{\epsilon_d} m_{h-1}$ and an *upward ϵ-edge (ϵ_u-edge)* $m_j \xrightarrow{\epsilon_u} m_i$. Pair $(m_{i-1} \xrightarrow{\epsilon_d} m_{h-1}, m_j \xrightarrow{\epsilon_u} m_i)$ is called *ϵ-pair* and represents applying $del(n_i)$.

A path in $H_K(t, N)$ represents a sequence of siblings. Let $p = e_1 e_2 \cdots e_n$ be a path in $H_K(t, N)$. We define $l(p) = l(e_1)l(e_2)\cdots l(e_n)$, where $l(e_i)$ is the label (non-terminal) of edge e_i (ϵ_d and ϵ_u are treated as empty strings). For example, if $p = m_5 \xrightarrow{\epsilon_d} m_{10} \xrightarrow[1]{X} m_{11} \xrightarrow{\epsilon_u} m_6 \overset{Y}{\underset{2}{\dashrightarrow}} m_7$, then $l(p) = XY$. Let q be a sequence of siblings in a tree, ν be a mapping from every node on q to a non-terminal in N, and p be a path in $H_K(t, N)$. Then p *represents* q under ν if (1) $l(p) = \nu(q)$ and (2) q coincides with a sequence of nodes obtained from p by replacing (i) each n-edge $m_{i-1} \xrightarrow[k]{X} m_i$ on p with n_i and (ii) each a-edge $m_{h-1} \overset{X}{\underset{k}{\dashrightarrow}} m_j$ on p with $n_{h,j}$ (any ϵ-edge is skipped).

Example 2. Let us consider Fig. 5(b). Consider $ch(t, n_3) = n_6, n_7$ and suppose that $\nu(n_6) = X$ and that $\nu(n_7) = Y$. Then path $m_5 \xrightarrow[k_1]{X} m_6 \xrightarrow[k_2]{Y} m_7$ represents

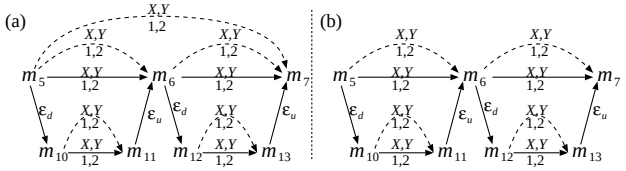

Fig. 6. (a) $H_K(t, N, 6, 7)$ and (b) $H'_K(t, N, 6, 7)$ (w.r.t. Fig. 5(b)).

$ch(t, n_3)$, for any $k_1, k_2 \in \{1, 2\}$. Changes made by *add* and *del* operations are represented by corresponding add and ϵ-edges (a path is not changed by any *ren* operation). For example, suppose that we apply $s = add(a, n_6, n_6)del(n_7)$ to t, then $ch(s(t), n_3) = n_{6,6}, n_{13}$ (Fig. 5(c)). Assuming that $\nu(n_{6,6}) = X$ and that $\nu(n_{13}) = Y$, path $m_5 \overset{X}{\underset{k_3}{\dashrightarrow}} m_6 \overset{\epsilon_d}{\rightarrow} m_{12} \overset{Y}{\underset{k_4}{\rightarrow}} m_{13} \overset{\epsilon_u}{\rightarrow} m_7$ represents $ch(s(t), n_3)$ for any $k_3, k_4 \in \{1, 2\}$. □

In order to compute edit script $\sigma(m_{i-1} \overset{X}{\underset{k}{\rightarrow}} m_i)$ ($\sigma(m_{h-1} \overset{X}{\underset{k}{\dashrightarrow}} m_j)$), We have to examine $\sigma(e)$ for every edge e representing a descendant of n_i (resp., $n_{h,j}$). Let n_h, n_j be siblings in t ($h \leq j$). By $H_K(t, N, h, j)$ we mean a graph consisting of the paths from m_{h-1} to m_j in $H_K(t, N)$ (Fig. 6(a)). For an internal node n_i with $ch(t, n_i) = n_h, \cdots, n_j$, $H_K(t, N, h, j)$ consists of the edges representing the descendants of n_i. We also define $H'_K(t, N, h, j)$, which is identical to $H_K(t, N, h, j)$ except that the a-edges from m_{h-1} to m_j are missing (Fig. 6(b)). $H'_K(t, N, h, j)$ consists of the edges representing the descendants of $n_{h,j}$. It is easy to show that the following lemma holds.

Lemma 1. *$H_K(t, N, h, j)$ and $H'_K(t, N, h, j)$ are "complete"; that is, for any edit script s for t and for any mapping ν from every node in $s(t)$ to a non-terminal in N, the following conditions hold.*

- *For any node n_i in $s(t)$ with $ch(t, n_i) = n_h, \cdots, n_j$, $H_K(t, N, h, j)$ contains a path from m_{h-1} to m_j representing $ch(s(t), n_i)$ under ν.*
- *For any node $n_{h,j}$ in $s(t)$, $H'_K(t, N, h, j)$ contains a path from m_{h-1} to m_j representing $ch(s(t), n_{h,j})$ under ν.*

□

The algorithm computes $\sigma(e)$ for every edge e in $H_K(t, N)$ from lower to higher edges; along a partial order defined as follows. By $m_{i-1} \overset{X}{\rightarrow} m_i$ we mean the *representative* of K n-edges $m_{i-1} \overset{X}{\underset{1}{\rightarrow}} m_i, \cdots, m_{i-1} \overset{X}{\underset{K}{\rightarrow}} m_i$. Similarly, $m_{h-1} \overset{X}{\dashrightarrow} m_j$ is the *representative* of K a-edges $m_{h-1} \overset{X}{\underset{1}{\dashrightarrow}} m_j, \cdots, m_{h-1} \overset{X}{\underset{K}{\dashrightarrow}} m_j$. For edge representatives e_1, e_2 in $H_K(t, N)$, we write "$e_1 \prec e_2$" if e_1 is lower than e_2. Formally, "$\prec$" is a partial order over the edge representatives in $H_K(t, N)$ such that for any $X \in N$,

- $e \prec m_{i-1} \xrightarrow{X} m_i$ if n_i is an internal node in t with $ch(t, n_i) = n_h, \cdots, n_j$ and e is the edge representative of an edge in $H_K(t, N, h, j)$, and that
- $e \prec m_{h-1} \overset{X}{\dashrightarrow} m_j$ if e is the edge representative of an edge in $H'_K(t, N, h, j)$.

For example, in Fig. 5(b) $m_{10} \overset{X}{\dashrightarrow} m_{11} \prec m_5 \xrightarrow{X} m_6$ and $m_5 \overset{X}{\dashrightarrow} m_6 \prec m_5 \overset{X}{\dashrightarrow} m_7$, but $m_5 \overset{Y}{\dashrightarrow} m_6 \not\prec m_5 \overset{X}{\dashrightarrow} m_6$.

Checking the Validity of a Node Let t be a tree, n_i be a node with $ch(t, n_i) = n_h, \cdots, n_j$, and ν be a mapping from every node in $s(t)$ to a non-terminal in N. In order to find edit scripts s such that $\nu(ch(s(t), n_i))$ matches a regular expression r, we find paths p from m_{h-1} to m_j in $H_K(t, N, h, j)$ such that $l(p)$ matches r (if $p = e_1 \cdots e_n$, then $\sigma(e_1) \cdots \sigma(e_n)$ is an edit script we are looking for). Such a path can be found as follows. An NFA is a five-tuple (Q, N, q_s, F, δ), where Q is a set of states, N is a set of non-terminals, $q_s \in Q$ is the start state, $F \subseteq Q$ is a set of final states, and $\delta : Q \times N \to 2^Q$ is a transition function. For a regular expression r over N, by $M(r)$ we mean an ϵ-free NFA such that $L(r) = L(M(r))$, where $L(M(r))$ is the language represented by $M(r)$. Let $M(r) = (Q, N, q_s, F, \delta)$ and let N_H and E_H be the sets of nodes and edges in $H_K(t, N, h, j)$, respectively. We define the *intersection* of $H_K(t, N, h, j)$ and $M(r)$, denoted $H_K(t, N, h, j) \times M(r)$, analogously to the intersection of two regular languages. Formally, $H_K(t, N, h, j) \times M(r)$ is defined as a graph $I = (N_I, E_{node} \cup E_{add} \cup E_{\epsilon_d} \cup E_{\epsilon_u})$, where

$$
\begin{aligned}
N_I &= \{(m_i, q) \mid m_i \in N_H, q \in Q\}, \\
E_{node} &= \{(m_{i-1}, q) \xrightarrow[k]{X} (m_i, q') \mid m_{i-1} \xrightarrow[k]{X} m_i \in E_H, q' \in \delta(q, X)\}, \\
E_{add} &= \{(m_{h-1}, q) \overset{X}{\underset{k}{\dashrightarrow}} (m_j, q') \mid m_{h-1} \overset{X}{\underset{k}{\dashrightarrow}} m_j \in E_H, q' \in \delta(q, X)\}, \\
E_{\epsilon_d} &= \{(m_{i-1}, q) \xrightarrow{\epsilon_d} (m_{j-1}, q) \mid m_{i-1} \xrightarrow{\epsilon_d} m_{j-1} \in E_H, q \in Q\}, \\
E_{\epsilon_u} &= \{(m_j, q) \xrightarrow{\epsilon_u} (m_i, q) \mid m_j \xrightarrow{\epsilon_u} m_i \in E_H, q \in Q\}.
\end{aligned}
$$

($H'_K(t, N, h, j) \times M(r)$ is defined similarly.) Let $p_I = (m_{i_0}, q_{i_0}) \xrightarrow[k_1]{X_1} (m_{i_1}, q_{i_1}) \xrightarrow[k_2]{X_2} \cdots \xrightarrow[k_n]{X_n} (m_{i_n}, q_{i_n})$ be a path in I. Path $p = m_{i_0} \xrightarrow[k_1]{X_1} m_{i_1} \xrightarrow[k_2]{X_2} \cdots \xrightarrow[k_n]{X_n} m_{i_n}$ is *embedded* in p_I. If $(m_{i_0}, q_{i_0}) = (m_{h-1}, q_s)$ and $(m_{i_n}, q_{i_n}) = (m_j, q_f)$ for some $q_f \in F$, then p_I is called *accepting* (recall that m_{h-1} (m_j) is the "leftmost" node (resp., "rightmost" node) in $H_K(t, N, h, j)$). By definition, there is a path p in $H_K(t, N, h, j)$ such that $l(p) \in L(M(r))$ *iff* there is an accepting path p_I embedding p in $H_K(t, N, h, j) \times M(r)$.

Let $p_I = (m_{i_0}, q_{i_0}) \xrightarrow[k_1]{X_1} (m_{i_1}, q_{i_1}) \xrightarrow[k_2]{X_2} \cdots \xrightarrow[k_n]{X_n} (m_{i_n}, q_{i_n})$ be a path in I and p be the path embedded in p_I. We define that $\sigma(p_I) = \sigma(p) = \sigma(m_{i_0} \xrightarrow[k_1]{X_1} m_{i_1}) \cdots \sigma(m_{i_{n-1}} \xrightarrow[k_n]{X_n} m_{i_n})$. For convention, $\sigma(p) = nil$ if $\sigma(m_{i_{j-1}} \xrightarrow[k_j]{X_j} m_{i_j}) = nil$ for some $1 \leq j \leq n$. $\gamma(\sigma(p_I))$ is called the *weight* of p_I.

5.2 The Algorithm

For an edit script s for t_{n_i}, if n_i is the root of $s(t_{n_i})$, then s is called *root preserving*. For an edit script s for t_{n_h,n_j}, if $n_{h,j}$ is the root of $s(t_{n_h,n_j})$, then s is called *root adding*. The algorithm computes

- for every n-edge $m_{i-1} \xrightarrow[k]{X} m_i$, a kth optimum root preserving edit script $\sigma(m_{i-1} \xrightarrow[k]{X} m_i)$ between t_{n_i} and $(N, \Sigma, \{X\}, P)$, and
- for every a-edge $m_{h-1} \overset{X}{\underset{k}{\dashrightarrow}} m_j$, a kth optimum root adding edit script $\sigma(m_{h-1} \overset{X}{\underset{k}{\dashrightarrow}} m_j)$ between t_{n_h,n_j} and $(N, \Sigma, \{X\}, P)$

in a bottom-up manner.

Figure 7 shows the algorithm. If $e_u = m_{i-1} \xrightarrow{X} m_i$ for some leaf n_i, then $\sigma(m_{i-1} \xrightarrow[1]{X} m_i)$ is obtained in lines 9 to 11. If $e_u = m_{i-1} \xrightarrow{X} m_i$ for an internal node n_i, then $\sigma(m_{i-1} \xrightarrow[1]{X} m_i), \cdots, \sigma(m_{i-1} \xrightarrow[K]{X} m_i)$ are obtained in lines 12 to 19. In line 14, $P(X)$ denotes the set of productions in P whose left-hand sides are X. For each production $X \to a(r)$ selected in line 14, lines 15 to 17 find K optimum root preserving edit scripts between t_{n_i} and $(N, \Sigma, \{X\}, P)$ w.r.t. $X \to a(r)$[3] and add them to T. By definition, a root preserving edit script between t_{n_i} and $(N, \Sigma, \{X\}, P)$ w.r.t. $X \to a(r)$ consists of (i) $ren(n_i, a)$ and (ii) an edit script s' between t_{n_h,n_j} and (N, Σ, N, P) such that the non-terminals on the roots of $s'(t_{n_h,n_j})$ match r under some interpretation. Line 16 can be solved by an algorithm for the K shortest paths problem, e.g., [6] (we assume that $\sigma(p_{Ik})$ is set to *nil* if there is no kth accepting shortest path p_{Ik} in I). Thus each $\sigma(p_{Ik})$ in line 17 represents a kth optimum edit script stated in (ii) above. Similarly, if $e_u = m_{h-1} \overset{X}{\dashrightarrow} m_j$, then $\sigma(m_{h-1} \overset{X}{\underset{1}{\dashrightarrow}} m_j), \cdots, \sigma(m_{h-1} \overset{X}{\underset{K}{\dashrightarrow}} m_j)$ are obtained in lines 20 to 26.

We show the correctness of the algorithm.

Theorem 2. *Let t be a tree, $G = (N, \Sigma, S, P)$ be a regular tree grammar, and K be a positive integer. Then the algorithm finds K optimum edit scripts between t and G.*

Proof(sketch): Induction on the "height" of e_u, from lower to higher edge representatives. Since any leaf node is labeled by `pcdata`, by lines 9 to 11 the basis case holds. As for the induction case, assume that for every edge e in $H_K(t, N)$ "lower" than e_u $\sigma(e)$ is correctly obtained. Suppose that e_u is set to $m_{i-1} \xrightarrow{X} m_i$ for some internal node n_i (the case where $e_u = m_{h-1} \overset{X}{\dashrightarrow} m_j$ can

[3] Let s be a root preserving edit script between t_{n_i} and $(N, \Sigma, \{X\}, P)$. For a production $X \to a(r)$, if $\lambda(n_i) = a$ in $s(t_{n_i})$ and $\nu(ch(s(t_{n_i}), n_i)) \in L(r)$ under some interpretation ν, then we say that s is an edit script between t_{n_i} and $(N, \Sigma, \{X\}, P)$ *w.r.t.* $X \to a(r)$.

Input: A tree t, a regular tree grammar $G = (N, \Sigma, S, P)$, and a positive integer K.
Output: K optimum edit scripts between t and G.

1. Construct $M(r)$ for each $X \to a(r) \in P$.
2. Construct $H_K(t, N)$.
3. $\sigma(e) \leftarrow nil$ for every n/a-edge e in $H_K(t, N)$.
4. **for** each ϵ-pair $(m_{i-1} \xrightarrow{\epsilon_d} m_{h-1}, m_j \xrightarrow{\epsilon_u} m_i)$ in $H_K(t, N)$ **do**
5. $\quad \sigma(m_{i-1} \xrightarrow{\epsilon_d} m_{h-1}) \leftarrow del(n_i)$ and $\sigma(m_j \xrightarrow{\epsilon_u} m_i) \leftarrow \epsilon$.
6. Sort the n/a-edge representatives in $H_K(t, N)$ w.r.t. $\prec$ topologically. Let $e_1, e_2, \cdots, e_z$ be the result.
7. **for** $u = 1$ to z **do**
8. $\quad T = \emptyset$;
9. $\quad$ **if** $e_u = m_{i-1} \xrightarrow{X} m_i$ for some leaf node n_i in t **do**
10. $\quad\quad$ **if** P contains a production $X \to \texttt{pcdata}(r)$ such that $\epsilon \in L(r)$ **then**
11. $\quad\quad\quad \sigma(m_{i-1} \xrightarrow[1]{X} m_i) \leftarrow \epsilon$;
12. $\quad$ **else if** $e_u = m_{i-1} \xrightarrow{X} m_i$ for some internal node n_i in t **then**
13. $\quad\quad$ Let $ch(t, n_i) = n_h, \cdots, n_j$.
14. $\quad\quad$ **for** each production $X \to a(r) \in P(X)$ **do**
15. $\quad\quad\quad I \leftarrow H_K(t, N, h, j) \times M(r)$;
16. $\quad\quad\quad$ Find K accepting shortest paths $p_{I1}, p_{I2}, \cdots, p_{IK}$ in I.
17. $\quad\quad\quad T \leftarrow T \cup \{ren(n_i, a)\sigma(p_{Ik}) \mid 1 \le k \le K\}$;
18. $\quad\quad$ Let $s_1, s_2, \cdots, s_K$ be K optimum edit scripts in T.
19. $\quad\quad \sigma(m_{i-1} \xrightarrow[k]{X} m_i) \leftarrow s_k$ for each $1 \le k \le K$.
20. $\quad$ **else if** $e_u = m_{h-1} \overset{X}{\dashrightarrow} m_j$ for some siblings n_h, n_j in t **then**
21. $\quad\quad$ **for** each production $X \to a(r) \in P(X)$ **do**
22. $\quad\quad\quad I' \leftarrow H'_K(t, N, h, j) \times M(r)$;
23. $\quad\quad\quad$ Find K accepting shortest paths $p_{I1}, p_{I2}, \cdots, p_{IK}$ in I'.
24. $\quad\quad\quad T \leftarrow T \cup \{add(a, n_h, n_j)\sigma(p_{Ik}) \mid 1 \le k \le K\}$;
25. $\quad\quad$ Let $s_1, s_2, \cdots, s_K$ be K optimum edit scripts in T.
26. $\quad\quad \sigma(m_{h-1} \xrightarrow[k]{X} m_j) \leftarrow s_k$ for each $1 \le k \le K$.
27. **return** K optimum edit scripts in $\{\sigma(m_0 \xrightarrow[k]{X} m_1) \mid X \in S, 1 \le k \le K\}$.

Fig. 7. An algorithm that finds K optimum edit scripts between a tree and a regular tree grammar.

be shown similarly). It suffices to show that $ren(n_i, a)\sigma(p_{I_k})$ in line 17 is a kth optimum edit script between t_{n_i} and $(N, \Sigma, \{X\}, P)$ w.r.t. $X \to a(r)$. It holds that (i) by Lemma 1 $H_K(t, N, h, j)$ contains, for any edit script s for t and for any mapping ν, a path from m_{h-1} to m_j representing $ch(s(t), n_i)$ under ν and (ii) by the induction hypothesis for every edge e in $H_K(t, N, h, j)$ $\sigma(e)$ is correctly obtained. This implies that $\sigma(p_{I_k})$ found in line 17 is a kth optimum edit script between t_{n_h, n_j} and (N, Σ, N, P) such that the non-terminals on the roots of $\sigma(p_{I_k})(t_{n_h, n_j})$ match r under some interpretation ν. Hence $ren(n_i, a)\sigma(p_{I_k})$ is a kth optimum edit script between t_{n_i} and $(N, \Sigma, \{X\}, P)$ w.r.t. $X \to a(r)$. □

Consider the running time of the algorithm. Assume that t is a k-ary tree for some constant k. Then the algorithm runs in $O(K \cdot |N| \cdot |t|^2 \cdot |P| \cdot R^2 + K \log K)$ time, where $|t|$ is the number of nodes in t, $|P|$ is the number of productions in

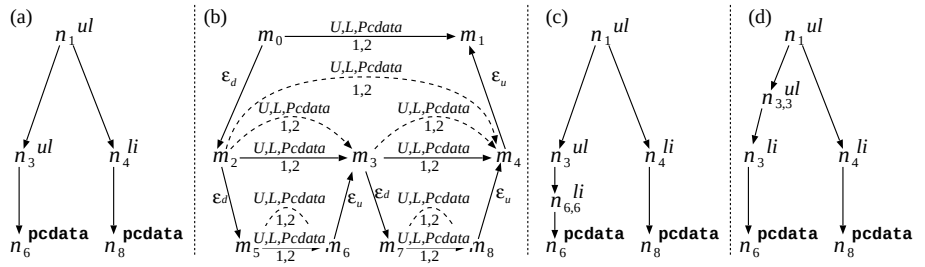

Fig. 8. (a) t, (b) $H_2(t, N)$, (c) $\sigma(m_0 \xrightarrow[1]{U} m_1)(t)$, and (d) $\sigma(m_0 \xrightarrow[2]{U} m_1)(t)$. In (b), $m_i \xrightarrow[1,2]{U,L,Pcdata} m_j$ in $H_2(t, N)$ stands for six edges $m_i \xrightarrow[1]{U} m_j$, $m_i \xrightarrow[2]{U} m_j$, $m_i \xrightarrow[1]{L} m_j$, $m_i \xrightarrow[2]{L} m_j$, $m_i \xrightarrow[1]{Pcdata} m_j$, $m_i \xrightarrow[2]{Pcdata} m_j$.

P, and $R = \max_{X \to a(r) \in P}(|r|)$ ($|r|$ denotes the length of regular expression r). In particular, if the content model of each production is one-unambiguous [2], the algorithm runs in $O(K \cdot |N| \cdot |t|^2 \cdot |P| \cdot R + K \log K)$ time (any content model of DTD and W3C XML Schema must be one-unambiguous).

Example 3. Let t be the tree in Fig. 8(a), $K = 2$, and $G = (N, \Sigma, S, P)$ be a regular tree grammar, where $N = \{U, L, Pcdata\}$, $\Sigma = \{ul, li, \texttt{pcdata}\}$, $S = \{U\}$, $P = \{U \to ul(U?L), L \to li(Pcdata), Pcdata \to \texttt{pcdata}(\epsilon)\}$. We assume for any nodes n_i, n_h, n_j that $\gamma(del(n_i)) = \gamma(add(a, n_h, n_j)) = 2$ and that $\gamma(ren(n_i, a)) = 1$ (except that $\gamma(ren(n_i, a)) = 0$ if $\lambda(n_i) = a$). The σ-values obtained by the algorithm are listed in Table 1 (the edges associated with nil are omitted). For example, consider finding $\sigma(m_0 \xrightarrow[k]{U} m_1)$ for $k = 1, 2$. As listed below, $H_K(t, N, 3, 4)$ contains three paths p from m_2 to m_3 such that $l(p) = U$ and that $\sigma(p) \neq nil$, and contains three paths p from m_3 to m_4 such that $l(p) = L$ and that $\sigma(p) \neq nil$.

path from m_2 to m_3	cost	path from m_3 to m_4	cost
$m_2 \xrightarrow[1]{U} m_3$	2	$m_3 \xrightarrow[1]{L} m_4$	0
$m_2 \dashrightarrow_{1}^{U} m_3$	3	$m_3 \dashrightarrow_{1}^{L} m_4$	4
$m_2 \dashrightarrow_{2}^{U} m_3$	5	$m_3 \xrightarrow{\epsilon_d} m_7 \dashrightarrow_{1}^{L} m_8 \xrightarrow{\epsilon_u} m_4$	3

Thus $H_K(t, N, 3, 4)$ contains $3 \times 3 = 9$ paths p from m_2 to m_4 such that $l(p) \in L(U?L)$ and that $\sigma(p) \neq nil$, where $m_2 \xrightarrow[1]{U} m_3 \xrightarrow[1]{L} m_4$ is the shortest and $m_2 \dashrightarrow_{1}^{U} m_3 \xrightarrow[1]{L} m_4$ is the second shortest. Thus in lines 16 to 19 we obtain $\sigma(m_0 \xrightarrow[1]{U} m_1) = ren(n_1, ul)\sigma(m_2 \xrightarrow[1]{U} m_3 \xrightarrow[1]{L} m_4) = ren(n_1, ul)ren(n_3, ul)add(li, n_6, n_6)ren(n_4, li)$ and $\sigma(m_0 \xrightarrow[2]{U} m_1) =$

Table 1. The edit scripts of the n/a-edges in $H_K(t, N)$.

edge	edit script	cost
$m_5 \overset{Pcdata}{\underset{1}{\rightarrow}} m_6$	ϵ	0
$m_5 \overset{L}{\underset{1}{\dashrightarrow}} m_6$	$add(li, n_6, n_6)\sigma(m_5 \overset{Pcdata}{\underset{1}{\rightarrow}} m_6)$	2
$m_7 \overset{Pcdata}{\underset{1}{\rightarrow}} m_8$	ϵ	0
$m_7 \overset{L}{\underset{1}{\dashrightarrow}} m_8$	$add(li, n_8, n_8)\sigma(m_7 \overset{Pcdata}{\underset{1}{\rightarrow}} m_8)$	2
$m_2 \overset{U}{\underset{1}{\rightarrow}} m_3$	$ren(n_3, ul)\sigma(m_5 \overset{L}{\underset{1}{\dashrightarrow}} m_6)$	2
$m_2 \overset{L}{\underset{1}{\rightarrow}} m_3$	$ren(n_3, li)\sigma(m_5 \overset{Pcdata}{\underset{1}{\rightarrow}} m_6)$	1
$m_2 \overset{U}{\underset{1}{\dashrightarrow}} m_3$	$add(ul, n_3, n_3)\sigma(m_2 \overset{L}{\underset{1}{\rightarrow}} m_3)$	3
$m_2 \overset{U}{\underset{2}{\dashrightarrow}} m_3$	$add(ul, n_3, n_3)\sigma(m_2 \overset{\epsilon_d}{\rightarrow} m_5 \overset{L}{\underset{1}{\dashrightarrow}} m_6 \overset{\epsilon_u}{\rightarrow} m_3)$	5
$m_2 \overset{L}{\underset{1}{\dashrightarrow}} m_3$	$add(li, n_3, n_3)\sigma(m_2 \overset{\epsilon_d}{\rightarrow} m_5 \overset{Pcdata}{\underset{1}{\rightarrow}} m_6 \overset{\epsilon_u}{\rightarrow} m_3)$	4
$m_3 \overset{U}{\underset{1}{\rightarrow}} m_4$	$ren(n_4, ul)\sigma(m_7 \overset{L}{\underset{1}{\dashrightarrow}} m_8)$	2
$m_3 \overset{L}{\underset{1}{\rightarrow}} m_4$	$ren(n_4, li)\sigma(m_7 \overset{Pcdata}{\underset{1}{\rightarrow}} m_8)$	0
$m_3 \overset{U}{\underset{1}{\dashrightarrow}} m_4$	$add(ul, n_4, n_4)\sigma(m_3 \overset{L}{\underset{1}{\rightarrow}} m_4)$	2
$m_3 \overset{U}{\underset{2}{\dashrightarrow}} m_4$	$add(ul, n_4, n_4)\sigma(m_3 \overset{\epsilon_d}{\rightarrow} m_7 \overset{L}{\underset{1}{\dashrightarrow}} m_8 \overset{\epsilon_u}{\rightarrow} m_4)$	5
$m_3 \overset{L}{\underset{1}{\dashrightarrow}} m_4$	$add(li, n_4, n_4)\sigma(m_3 \overset{\epsilon_d}{\rightarrow} m_7 \overset{Pcdata}{\underset{1}{\rightarrow}} m_8 \overset{\epsilon_u}{\rightarrow} m_4)$	4
$m_2 \overset{U}{\underset{1}{\dashrightarrow}} m_4$	$add(ul, n_3, n_4)\sigma(m_2 \overset{U}{\underset{1}{\rightarrow}} m_3 \overset{L}{\underset{1}{\rightarrow}} m_4)$	4
$m_2 \overset{U}{\underset{2}{\dashrightarrow}} m_4$	$add(ul, n_3, n_4)\sigma(m_2 \overset{U}{\underset{1}{\dashrightarrow}} m_3 \overset{L}{\underset{1}{\rightarrow}} m_4)$	5
$m_0 \overset{U}{\underset{1}{\rightarrow}} m_1$	$ren(n_1, ul)\sigma(m_2 \overset{U}{\underset{1}{\rightarrow}} m_3 \overset{L}{\underset{1}{\rightarrow}} m_4)$	2
$m_0 \overset{U}{\underset{2}{\rightarrow}} m_1$	$ren(n_1, ul)\sigma(m_2 \overset{U}{\underset{1}{\dashrightarrow}} m_3 \overset{L}{\underset{1}{\rightarrow}} m_4)$	3

$ren(n_1, ul)\sigma(m_2 \overset{U}{\underset{1}{\dashrightarrow}} m_3 \overset{L}{\underset{1}{\rightarrow}} m_4) = ren(n_1, ul)add(ul, n_3, n_3)ren(n_3, li)ren(n_4, li)$ (see Fig. 8(c,d)). □

6 Conclusion

In this paper, we first showed that finding K optimum edit scripts between an XML document and a regular tree grammar is NP-hard. We next presented a pseudopolynomial-time algorithm for solving the problem. As a future work, we would like to implement the algorithm and making experiments on the algorithm.

Acknowledgement

This work is partially supported by the Grant-in-Aid for Young Scientists (B) #18700019.

References

[1] E. Bertino, G. Guerrini, and M. Mesiti. A matching algorithm for measuring the structural similarity between an xml document and a dtd and its applications. *Information Systems*, 29(1):23–46, 2004.

[2] A. Bruggenmann-Klein and D. Wood. One-unambiguous regular languages. *Information and Computation*, 142(2):182–206, 1998.

[3] E. R. Canfield and G. Xing. Approximate xml document matching. In *Proc. ACM SAC*, pages 787–788, 2005.

[4] S. Chawathe and H. Gracia-Molina. Meaningful change detection in structured data. In *Proc. ACM SIGMOD Conf.*, pages 26–37, 1997.

[5] G. Cobena, S. Abiteboul, and A. Marian. Detecting changes in xml documents. In *Proc. ICDE*, pages 41–52, 2002.

[6] D. Eppstein. Finding the k shortest paths. *SIAM J. Comput.*, 28(2):652–673, 1998.

[7] D. Gusfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, 1997.

[8] M. Murata, D. Lee, M. Mani, and K. Kawaguchi. Taxonomy of xml schema languages using formal language theory. *ACM TOIT*, 5(4):660–704, 2005.

[9] A. Nierman and H. V. Jagadish. Evaluating structural similarity in xml documents. In *Proc. WebDB*, pages 61–66, 2002.

[10] Y. Papakonstantinou and V. Vianu. Dtd inference for view of xml data. In *Proc. PODS*, pages 35–46, 2000.

[11] D. Sankoff and J. Kruskal. *Time Warps, String Edits, and Macromolecules*. Addison-Wesley, 1983.

[12] N. Suzuki. Finding an optimum edit script between an xml document and a dtd. In *Proc. ACM SAC*, pages 647–653 (Journal version to apper in IPSJ Transactions on Databases), 2005.

[13] R. A. Wagner and J. I. Seiferas. Correcting counter-automaton-recognizable languages. *SIAM J. Comput.*, 7:357–375, 1978.

[14] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, 18(6):1245–1262, 1989.