On Inferring K Optimum Transformations of XML Document from Update Script to DTD

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

DTDs are continuously updated according to changes of the real world. Let t be an XML document valid against a DTD D, and suppose that D is updated by an update script s. In general, we cannot uniquely "infer" a transformation of t from s, i.e., we cannot uniquely determine the elements in t that should be deleted and the positions in t that new elements should be inserted into. In this paper, we consider inferring K optimum transformations of t from s so that a user finds the most desirable transformation among them. We first show that the problem of inferring K optimum transformations of an XML document from an update script is NP-hard even if $K = 1$. Then, assuming that an update script is of length one, we show an algorithm for solving the problem, which runs in time polynomial of $|D|$, $|t|$, and K.

1 Introduction

DTDs are continuously updated according to changes of the real world. Suppose that we maintain XML documents valid against a DTD, and that the DTD is updated by some update script. Then the documents may no longer be valid against the DTD, thus we have to transform each of the documents into a valid one. Finding an appropriate transformation for each document manually is surely impractical, thus it is important to construct an algorithm that assists a user to find an appropriate transformation.

In general, for an update script s to a DTD D and an XML document t valid against D, we cannot uniquely "infer" from s (i) the elements in t that should be deleted and (ii) the positions in t that new elements should be inserted into. In other words, we have more than one possible ways to transform t, thus we need to select an appropriate transformation among them.

International Conference on Management of Data
COMAD 2008, Mumbai, India, December 17–19, 2008

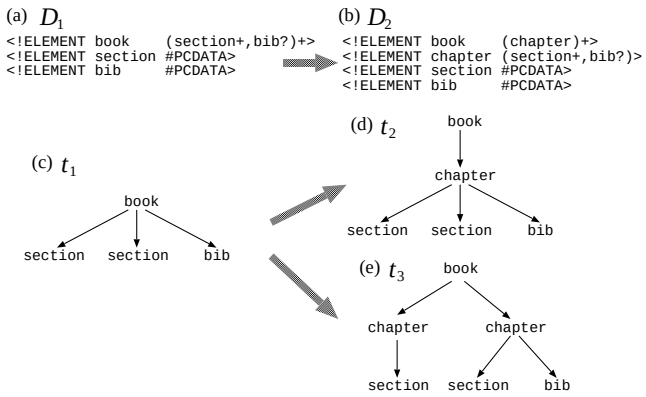

Figure 1: DTDs D_1, D_2 and XML documents t_1, t_2, t_3.

In such a situation, it is useful to compute the list of "top-K" (or K optimum) transformations of t inferred from s so that we can easily find the most desirable transformation among them. In this paper, we consider inferring such K optimum transformations from an update script.

For example, let us consider DTD D_1 (Fig. 1(a)). Suppose that D_1 is updated to D_2 by an update script that aggregates subexpression "(section$^+$,bib?)" of the content model of "book" into "chapter" (Fig. 1(b)). For the tree t_1 in Fig. 1(c), we have two alternatives t_2, t_3 according to the positions at which "chapter" elements should be inserted (Fig. 1(d,e)). Our algorithm can present such a list of transformations, where the listed trees are ordered by the "amount of changes" (the number of insertions/deletions applied to the input tree).

As shown above, when a DTD is updated, more than one transformations of an XML document may be inferred from the update script, and we have to select an appropriate transformation from the list of the transformations. Clearly, listing such transformations in random order is much confusing. Although there is no common established criterion for ordering such transformations, such a list can be reasonable and

easier to understand if its transformations are ordered by the amount of changes; a transformation with less changes is ranked higher. Thus, in this paper the transformation with the least amount of changes is treated as the optimum one.

Let s be an update script to a DTD D, t be an XML document valid against D, and K be a positive integer. Our main results are the following twofold:

- In general, the problem of inferring K optimum transformations of t from s is intractable due to combinatorial explosion. In fact, we show that the problem is NP-hard even if $K = 1$.

- If s is restricted to be of length one, the problem can be solved relatively efficiently. In fact, we construct an algorithm for solving this problem, which runs in time polynomial of $|D|$, $|t|$, and K.

In this paper, we first define update operations to a DTD. We next define a nondeterministic algorithm that infers a transformation of a tree from an update operation. Then, based on this algorithm, we show that the problem of inferring K optimum transformations of a tree from an update script is NP-hard even if $K = 1$. Finally, assuming that an update script s to a DTD D is of length one, we show an algorithm for inferring K optimum transformations of a tree t from s, which runs in time polynomial of $|D|$, $|t|$, and K.

Related Work

Several studies have proposed update operations to schemas and discussed related problems. For a nested relational model, Ref. [9] considered the problem of adapting mappings to schema changes. This assumed that a mapping between schemas is explicitly provided, thus the problem discussed in this paper did not arise.

As for XML, Ref. [4] proposed update operations to represent the "diff" between two DTDs. Ref. [3] proposed update operations to tree grammars to preserve schema's expressive power; for any tree t valid against a tree grammar, there is a supertree of t that is valid against its updated grammar. Refs. [2, 7] proposed update operations for inclusion problem of schemas, which assure that any updated schema includes its original schema. Ref. [6] devised three update operations and an algorithm for generating XSLT scripts that transform XML documents according to a given update operation. Ref. [8] proposed an algorithm for deciding if, for a DTD D and an update script s, a transformation of a tree t inferred from s is unique for any t valid against D. To the best of the author's knowledge, no study considers inferring K optimum transformations of an XML document from an update script.

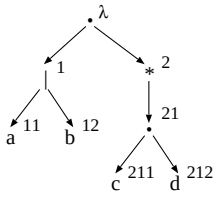

Figure 2: Tree representation of r.

2 Definitions

An XML document is modeled as an ordered labeled tree (attributes are omitted). Each node in a tree represents an element. A text node is omitted, in other words, we assume that each leaf node has a text node implicitly. For a node n in a tree, by $l(n)$ we mean the label (element name) of n. In what follows, we use the term tree when we mean ordered labeled tree.

Let Σ be a set of labels. A *regular expression* over Σ is recursively defined as follows.

- ϵ and a are regular expressions, where $a \in \Sigma$.

- If $r_1, \cdots, r_n$ are regular expressions, then $(r_1, \cdots, r_n)$ and $(r_1|\cdots|r_n)$ are regular expressions $(n \geq 1)$.

- If r_1 is a regular expression, then $(r_1)^*$, $(r_1)?$, and $(r_1)^+$ are regular expressions.

The language specified by a regular expression r is denoted $L(r)$.

In order to define update operations to a DTD concisely, we sometimes represent a regular expression as a term in prefix notation. For example, we may write $\cdot(a, ^*(|(b, c)))$ instead of usual notation $(a, (b|c)^*)$, where '$\cdot$' denotes a "sequence" operator. Let r be a regular expression in prefix notation. The set of *positions* of r, denoted $pos(r)$, is defined as follows.

- If $r = \epsilon$ or $r = a$ for some $a \in \Sigma$, then $pos(r) = \{\lambda\}$, where λ denotes an empty sequence.

- If $r = op(r_1, \cdots, r_n)$ with $op \in \{|, \cdot, ^*, ^+, ?\}$, then $pos(r) = \{\lambda\} \cup \{u \mid u = iv, 1 \leq i \leq n, v \in pos(r_i)\}$. Note that $n = 1$ if $op \in \{^*, ^+, ?\}$.

For example, let $r = ((a|b), (c, d)^*)$. The prefix notation of r is $\cdot(|(a, b), ^*(\cdot(c, d)))$. Figure 2 shows the tree representation of r, in which each node is associated with its corresponding position. Thus $pos(r) = \{\lambda, 1, 11, 12, 2, 21, 211, 212\}$.

Let $u \in pos(r)$. The label at u in r, denoted $l(r, u)$, and the subexpression at u in r, denoted $sub(r, u)$, are recursively defined as follows.

- If $r = \epsilon$ or $r = a$ for some $a \in \Sigma$, then $l(r, \lambda) = r$ and $sub(r, \lambda) = r$.

- If $r = op(r_1, \cdots, r_n)$ with $op \in \{|, \cdot, ^*, ^+, ?\}$, and

– if $u = \lambda$, then $l(r, u) = op$ and $sub(r, u) = r$,

– if $u = jv$ for some $1 \leq j \leq n$ and some $v \in pos(r_j)$, then $l(r, u) = l(r_j, v)$ and $sub(r, u) = sub(r_j, v)$.

For example, in Fig. 2 $l(r, 1) =$ '|', $l(r, 11) = a$, and $sub(r, 21) = \cdot(c, d)$.

Let w be a word over Σ. By $|w|$ we mean the length of w, and by $w[i]$ we mean the ith label of w. We define that $w[i, j] = w[i]w[i + 1] \cdots w[j]$ $(1 \leq i \leq j \leq |w|)$. For example, if $w = kasuga$, then $w[3, 5] = sug$.

Let r be a regular expression. By r' we mean the *subscripted* regular expression resulting from r by subscripting each label in r by its corresponding position. By $sym(r')$ we mean the set of subscripted labels occurring in r'. For example, if $r = ((a|b|c), (d|b)^*)$, then $r' = ((a_{11}|b_{12}|c_{13}), (d_{211}|b_{212})^*)$ and $sym(r') = \{a_{11}, b_{12}, c_{13}, d_{211}, b_{212}\}$. Let a_i be a subscripted label of a. Then by $a_i^\natural$ we mean the label resulting from a_i by dropping the subscript of a_i, that is, $a_i^\natural = a$. Let w_1 be a subscripted word (i.e., a sequence of subscripted labels). We define that $w_1^\natural = w_1[1]^\natural \cdots w_1[|w_1|]^\natural$. For any regular expression r, it holds that $L(r) = L(r')^\natural$, where $L(r')^\natural = \{w_1^\natural \mid w_1 \in L(r')\}$.

A *DTD* is a tuple $D = (d_1, sl)$, where d_1 is a (possibly partial) mapping from Σ to the set of regular expressions over Σ, and $sl \in \Sigma$ is the *start label*. For a label $a \in \Sigma$, $d_1(a)$ is the *content model* of a. A tree t is *valid* against D if (i) the root of t is labeled by sl and (ii) for each node n in t the sequence of labels on the children of n is in $L(d_1(l(n)))$. For labels $a, b \in \Sigma$, b is *reachable* from a if (i) $a = b$ or (ii) b occurs in $d_1(c)$ for some label c reachable from a.

3 Update Operations to DTD

In this section, we define nine update operations to a DTD.

Let $D = (d_1, sl)$ be a DTD. First, the following two operations relate to insertion/deletion of an element in a content model.

- $ins\_elm(a, b, vi)$: Inserts a new label b at position vi in $d_1(a)$, where $vi \in pos(d_1(a))$ with i is a positive integer and $b \in \Sigma$ (Fig. 3(b,c)). This is applicable to D only if $d_1(b)$ is defined, $l(d_1(a), v) \in \{\cdot, |\}$, and $v(i-1) \in pos(d_1(a))$ (i.e., the operator at v has at least $i - 1$ operands).

- $del\_elm(a, vi)$: Deletes the label (possibly ϵ) at vi in $d_1(a)$. More formally, we have two cases according to the operator at v.

  – The case where $l(d_1(a), v) =$ '$\cdot$': $l(d_1(a), vi)$ is deleted from $d_1(a)$ (Fig. 3(a,b)). This is applicable to D only if $vk \in pos(d_1(a))$ for some $k \neq i$ (i.e., the operator at v has more than one children).

  – The case where $l(d_1(a), v) =$ '|': If $l(d_1(a), vi) = l(d_1(a), vk)$ for some $k \neq i$, then $l(d_1(a), vi)$ is deleted from $d_1(a)$ (deleting one of duplicated labels). Otherwise, $l(d_1(a), vi)$ is replaced by ϵ.

Second, the following two operations relate to extraction/aggregation of an element.

- $ext\_elm(a, u)$: "Extracts" label $l(d_1(a), u)$ in $d_1(a)$. Formally, this operation replaces label $l(d_1(a), u)$ in $d_1(a)$ by regular expression $d_1(l(d_1(a), u))$ (Fig. 3(e,f)). This is applicable to D only if $l(d_1(a), u) \in \Sigma$ and $l(d_1(a), u) \neq a$.

- $agg\_elm(a, b, u)$: "Aggregates" subexpression $sub(d_1(a), u)$ into single label b. Formally, this operation (i) sets $d_1(b) = sub(d_1(a), u)$ and (ii) replaces $sub(d_1(a), u)$ by b (Fig. 3(d,e)). This is applicable to D only if $d_1(b)$ is undefined.

The following three operations relate to modifying an operator $(|, \cdot, ^*, ^+, ?)$ in a content model.

- $ins\_opr(a, opr, u, v)$: Inserts a new operator opr as the parent of the siblings at $u, \cdots, v$ in $d_1(a)$, where $opr \in \{\cdot, |, ^*, ^+, ?\}$ (Fig. 3(c,d)). This is applicable to D only if (i) $u = v$ (opr has only one operand) or (ii) $opr \in \{\cdot, |\}$, $i < j$, and $opr = l(d_1(a), w)$, where $u = wi$ and $v = wj$ (nesting the operator at w by opr).

- $del\_opr(a, u)$: Deletes an operator at u in $d_1(a)$ (Fig. 3(f,g)). This is applicable to D only if (i) the operator at u has only one operand and $l(d_1(a), u) \in \{\cdot, |, ^+\}$ or (ii) $l(d_1(a), u) = l(d_1(a), v)$, where $u = vi$ (unnesting the operator at u).

  Note that, in order to delete a '*' operator, it suffices to replace the '*' with '+' by $change\_opr$ (defined below), then delete the '+' by del_opr. '?' can be deleted similarly.

- $change\_opr(a, opr, u)$: Replaces the operator at u (i.e., $l(d_1(a), u)$) by opr. The following four cases are allowed.

$l(d_1(a), u)$	opr
?	*
+	*
*	+
*	?

Finally, the following two operations relate to defining/undefining a content model.

- $def\_cm(a, r)$: Sets $d_1(a) = r$. This is applicable to D only if $d_1(a)$ is undefined.

- $undef\_cm(a)$: Undefine the content model of a. This is applicable to D only if a is unreachable from the start label sl.

Let op be an update operation to a DTD D. By $op(D)$ we mean the DTD obtained by applying op to D. Let $s = op_1 op_2 \cdots op_n$ be a sequence of update operations. We say that s is an *update script* to D if op_i is applicable to $op_{i-1}(op_{i-2}(\cdots op_1(D)\cdots))$ for every $1 \leq i \leq n$. By $|s|$ we mean the *length* of s, that is, $|s| = n$. We say that a DTD D_2 *includes* a DTD D_1 if for any tree t, t is valid against D_2 whenever t is valid against D_1. We have the following lemma.

Lemma 1 *Let $D = (d_1, sl)$ be a DTD and op be an update operation to D. Then $op(D)$ includes D if*

1. *$op = ins\_elm(a, b, vi)$ and $l(d_1(a), v) =$ '|',*
2. *$op = ins\_opr(a, opr, u, v)$,*
3. *$op = del\_opr(a, u)$ and $l(d_1(a), u) \in \{\cdot, |\}$,*
4. *$op = change\_opr(a, opr, u)$ and either (i) $l(d_1(a), u) =$ '?' and $opr =$ '*' or (ii) $l(d_1(a), u) =$ '+' and $opr =$ '*', or*
5. *$op = def\_cm(a, r)$ or $op = undef\_cm(a)$.* □

4 Transformation Algorithm

Let t be a tree valid against a DTD D. If D is updated by an update operation op, we need to transform t according to op. In this section, we show a nondeterministic algorithm that infers, for a DTD D, a tree t valid against D, and an update operation op to D, a transformation of t from op.

The following TRANSOP is the "main" part of the algorithm (TRANS1 to TRANS6 are shown later).

TRANSOP(D, t, op)
Input: a DTD D, a tree t valid against D, and an update operation op to D.
Output: a tree valid against $op(D)$.

1. If t is valid against $op(D)$, return t.
2. Else
   if $op = ins\_elm(a, b, vi)$, return TRANS1(D, t, op),
   if $op = del\_elm(a, vi)$, return TRANS2(D, t, op),
   if $op = ext\_elm(a, u)$, return TRANS3(D, t, op),
   if $op = agg\_elm(a, b, u)$, return TRANS4(D, t, op),
   if $op = del\_opr(a, u)$, return TRANS5(D, t, op),
   if $op = change\_opr(a, opr, u)$, return TRANS6$(D, t, op)$.

Note that if $op = ins\_opr(a, opr, u, v)$, $op = def\_cm()$, or $op = undef\_cm()$, then we do not have to transform t, since t is valid against $op(D)$ by Lemma 1.

To show six TRANS subroutines, we need a definition. Let r be a regular expression, $u \in pos(r)$ be a position in r, and $q = sub(r, u)$ be a subexpression of r. Moreover, let w_1 be a subscripted word

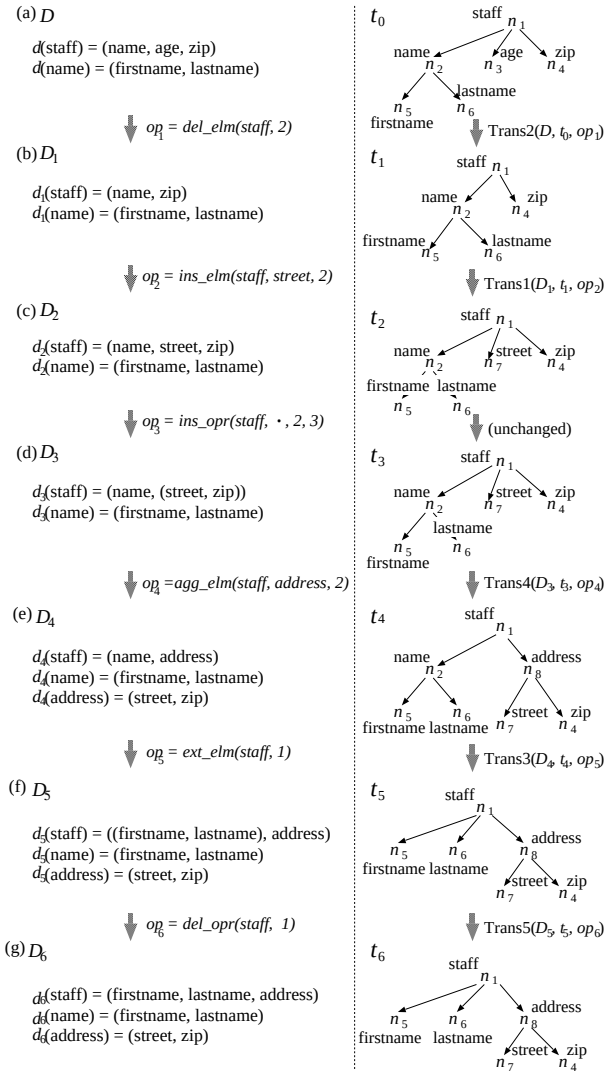

Figure 3: An update script to D (left) and a transformation inferred from the update script (right).

such that $w_1 \in L(r')$. We say that $w_1[i, j]$ *maximally matches* q' if $w_1[i, j] \in L(q')$ and either (i) $i = 1$ and $j = |w_1|$ or (ii) $w_1[i', j'] \notin L(q')$ for any i', j' with $\{i, \cdots, j\} \subset \{i', \cdots, j'\}$. We define that

$$match(w_1, q') = \{(i, j) \mid w_1[i, j] \text{ maximally matches } q'\}.$$

For example, let $r = (a, (b|c))^*$ and $q = sub(r, 12)$. Then $r' = (a_{11}, (b_{121}|c_{122}))^*$ and $q' = (b_{121}|c_{122})$. If $w_1 = a_{11}b_{121}a_{11}c_{122}$, then $match(w_1, q') = \{(2, 2), (4, 4)\}$.

Let us fist show TRANS1. TRANS1(D, t, op) transforms t according to op. In this case, $op = ins\_elm(a, b, vi)$, and by Lemma 1 $l(d_1(a), v) =$ '$\cdot$'. Thus, it suffices to insert new b elements at appropriate positions in t.[1] We need a definition. Let w be a word and b_h be a subscripted label. We say that a

[1] We assume that the text values of such a new element are empty since they can hardly be estimated.

subscripted word w_1 is a *subscripted supersequence* of w w.r.t. b_h if removing every b_h from w_1 yields a word w_2 such that $w_2^\natural = w$. In the following, we denote $D = (d_1, sl)$ and $op(D) = (d_2, sl)$, and assume that each transformation is done in bottom-up manner.

TRANS1(D, t, op)

1. For each node n labeled by a in t, do the following.
   (a) Let $n_1, \cdots, n_m$ be the children of n in t. If $l(n_1) \cdots l(n_m) \notin L(d_2(a))$, do the following.
      i. Find a subscripted supersequence w_1 of $l(n_1) \cdots l(n_m)$ w.r.t. b_h such that $w_1 \in L(d_2(a)')$, where b_h is the subscripted label in $d_2(a)'$ inserted by op.
      ii. For each $(j, j) \in match(w_1, b_h)$, create a new tree t_j valid against DTD (d_2, b) and insert t_j into t as the jth child of n.
2. Return t.

For example, the transformation from t_1 to t_2 in Fig. 4 is done by TRANS1.

Note that in step (1-a-i) above, there may be more than one subscripted supersequences of $l(n_1) \cdots l(n_m)$ w.r.t. b_h matching $d_2(a)'$, and w_1 is selected nondeterministically. Similar behaviors can be found in the other TRANS subroutines.

Let us next show TRANS2. In this case, $op = del\_elm(a, vi)$. Thus, it suffices to delete the elements in t that match the label in $d_1(a)$ deleted by op.

TRANS2(D, t, op)

1. For each node n labeled by a in t, do the following.
   (a) Let $n_1, \cdots, n_m$ be the children of n in t. If $l(n_1) \cdots l(n_m) \notin L(d_2(a))$, do the following.
      i. Find a subscripted word w_1 such that $w_1 \in L(d_1(a)')$ and that $w_1^\natural = l(n_1) \cdots l(n_m)$.
      ii. By definition $sub(d_1(a), vi)'$ is a single subscripted label, say b_h. For each $(j, j) \in match(w_1, b_h)$, delete the subtree rooted at n_j from t.
2. Return t.

The transformation from t_0 to t_1 in Fig. 4 is an example of TRANS2.

We show TRANS3. In this case, $op = ext\_elm(a, u)$. Thus, it suffices to delete the nodes in t that match the label in $d_1(a)$ "extracted" by op.

TRANS3(D, t, op)

1. For each node n labeled by a in t, do the following.
   (a) Let $n_1, \cdots, n_m$ be the children of n in t. If $l(n_1) \cdots l(n_m) \notin L(d_2(a))$, do the following.
      i. Find a subscripted word w_1 such that $w_1 \in L(d_1(a)')$ and that $w_1^\natural = l(n_1) \cdots l(n_m)$.
      ii. By definition $sub(d_1(a), u)'$ is a single subscripted label, say b_h. For each $(j, j) \in match(w_1, b_h)$, delete the jth child n_j of n from t.
2. Return t.

The transformation from t_4 to t_5 in Fig. 4 is an example of TRANS3.

Let us show TRANS4. In this case, $op = agg\_elm(a, b, u)$. Thus, it suffices to insert a new parent node labeled by b into t for each sequence of nodes that matches $sub(d_1(a), u)$.

TRANS4(D, t, op)

1. For each node n labeled by a in t, do the following.
   (a) Let $n_1, \cdots, n_m$ be the children of n in t. If $l(n_1) \cdots l(n_m) \notin L(d_2(a))$, do the following.
      i. Find a subscripted word w_1 such that $w_1 \in L(d_1(a)')$ and that $w_1^\natural = l(n_1) \cdots l(n_m)$.
      ii. For each $(j, k) \in match(w_1, sub(d_1(a), u)')$, insert a new node labeled by b as the parent of $n_j, \cdots, n_k$ into t.
2. Return t.

The transformation from t_3 to t_4 in Fig. 4 is an example of TRANS4.

Let us show TRANS5. We have $op = del\_opr(a, u)$, and $l(d_1(a), u) =$ '+' by Lemma 1. Thus $sub(d_1(a), u) = q^+$ for some subexpression q. Since the '+' of q^+ is deleted by op, we have to "shrink" each sequence of nodes in t that matches q^+ so that the resulting sequence matches q rather than q^+. Let w_1 be a subscripted word such that $w_1^\natural \in L(d_1(a))$. For a subsequence $w_1[i, j]$ such that $(i, j) \in match(w_1, (q^+)')$, we say that $w_1[k, l]$ is a *core* of $w_1[i, j]$ w.r.t. $(q^+)'$ if $w_1[i, k-1] \in L((q^*)')$, $w_1[k, l] \in L(q')$, and $w_1[l+1, j] \in L((q^*)')$. A *shrink* of w_1 w.r.t. $(q^+)'$ is obtained from w_1 by replacing, for each $(i, j) \in match(w_1, (q^+)')$, $w_1[i, j]$ by a core of $w_1[i, j]$.

TRANS5(D, t, op)

1. For each node n in t labeled by a, do the following.
   (a) Let $n_1, \cdots, n_m$ be the children of n in t. If $l(n_1) \cdots l(n_m) \notin L(d_2(a))$, do the following.
      i. Find a subscripted word w_1 such that $w_1 \in L(d_1(a)')$ and that $w_1^\natural = l(n_1) \cdots l(n_m)$.
      ii. Find a shrink w_2 of w_1 w.r.t. q', where $q = sub(d_1(a), u)$. For each $1 \le j \le |w_1|$ such that $w_1[j]$ disappears in w_2, delete the subtree rooted at n_j from t.
2. Return t.

Finally, let us show TRANS6. We have $op = change\_opr(a, opr, u)$, and by Lemma 1 we have either (i) $l(d_1(a), u) =$ '*' and $opr =$ '?' or (ii) $l(d_1(a), u) =$ '*' and $opr =$ '+'. The case of (i) can be handled similarly to TRANS5. In the following, we consider the case of (ii). Then $sub(d_1(a), u) = q^*$ for some regular expression q, and q^* is changed to q^+ by op. Thus, for each position in t matching q^* due to the fact that $\epsilon \in L(q^*)$, it suffices to insert a sequence of elements that matches q. Let $w_1 \in L(d_1(a)')$ be a subscripted word. Then an *extension* of w_1 w.r.t. q' is obtained by, for each "legal" position in w_1, inserting at most one word matching q'. Here, a position p of w_1 is *legal* (w.r.t. q') if p is (i) the head of w_1 with $w_1[1] \notin sym(q')$, (ii) the tail of w_1 with $w_1[|w_1|] \notin sym(q')$, or (iii) between $w_1[i-1]$ and $w_1[i]$ with $w_1[i-1], w_1[i] \notin sym(q')$ $(2 \leq i \leq |w_1|)$.

TRANS6(D, t, op)

1. For each node n in t labeled by a, do the following.
   (a) Let $n_1, \cdots, n_m$ be the children of n in t. If $l(n_1) \cdots l(n_m) \notin L(d_2(a))$, do the following.
      i. Find a subscripted word w_1 such that $w_1 \in L(d_1(a)')$ and that $w_1^\natural = l(n_1) \cdots l(n_m)$.
      ii. Find an extension w_2 of w_1 w.r.t. q' such that $w_2 \in L(d_2(a)')$. For each subscripted label $w_2[i]$ inserted into w_1, create a tree t_i valid against $(d_2(a), w_2[i]^\natural)$ and insert t_i as the ith child of n.
2. Return t.

We write $t_2 \in$ TRANSOP(D, t_1, op) if t_2 can be the result of TRANSOP(D, t_1, op). It is clear that TRANSOP is correct.

Theorem 1 *Let D be a DTD and op be an update operation to D. For any tree t_1 valid against D, every $t_2 \in$ TRANSOP(D, t_1, op) is valid against $op(D)$.* □

By $|t|$ we mean the number of nodes in t and by $|D|$ we mean the size of D. We have the following (its proof is omitted because of space limitation).

Lemma 2 TRANSOP(D, t, op) *runs in $O(|t| \cdot |D|^2)$ time.* □

5 NP-Hardness

In this section, we first formally define the problem of inferring K optimum transformations of an XML document from an update script. Then we show the NP-hardness of the problem.

5.1 Formal Definition of the Problem

Let D be a DTD, t_1 be a tree valid against D, and op be an update operation to D. For a tree $t_2 \in$ TRANSOP(D, t_1, op), the *difference* (or *diff*) between t_1 and t_2, denoted $df(t_1, t_2)$, is defined as follows. We have five cases according to op.

- $df(t_1, t_2)$ is defined as the set of root nodes of the subtrees inserted into t_1 w.r.t. t_2 if
  - $op = ins\_elm(a, b, vi)$, or
  - $op = change\_opr(a, opr, u)$, $l(d_1(a), u) =$ '*', and $opr =$ '+'.
- $df(t_1, t_2)$ is defined as the set of root nodes of the subtrees deleted from t_1 w.r.t. t_2 if
  - $op = del\_elm(a, vi)$,
  - $op = del\_opr(a, u)$ and $l(d_1(a), u) =$ '+', or
  - $op = change\_opr(a, opr, u)$, $l(d_1(a), u) =$ '*', and $opr =$ '?'.
- $df(t_1, t_2)$ is defined as the set of nodes deleted from t_1 w.r.t. t_2 if $op = ext\_elm(a, u)$.
- $df(t_1, t_2)$ is defined as the set of nodes inserted into t_1 w.r.t. t_2 if $op = agg\_elm(a, b, u)$.
- Otherwise, $df(t_1, t_2) = \emptyset$.

Let D be a DTD, $s = op_1 \cdots op_n$ be an update script to D, and t be a tree valid against D. A sequence $TS = t_0, t_1, \cdots, t_n$ of trees is called *transformation sequence* w.r.t. (t, D, s) if $t_0 = t$ and $t_i \in$ TRANSOP(D_{i-1}, t_{i-1}, op_i) for every $1 \leq i \leq n$, where $D_{i-1} = op_{i-1}(\cdots op_1(D) \cdots)$. The cost of a transformation sequence TS, denoted $\gamma(TS)$, is defined as $\gamma(TS) = \sum_{1 \leq i \leq n} |\delta(t_{i-1}, t_i)|$. For a positive integer K, we say that K transformation sequences $TS_1, \cdots, TS_K$ w.r.t. (t, D, s) are K *optimum* transformation sequences w.r.t. (t, D, s) if $\gamma(TS_i) \leq \gamma(TS_{i+1})$ for any $1 \leq i \leq K-1$ and $\gamma(TS_K) \leq \gamma(TS)$ for any transformation sequence TS w.r.t. (t, D, s) such that $TS \notin \{TS_1, \cdots, TS_K\}$. Consequently, our problem is to find, for a DTD D, a tree valid against D, an update script s, and a positive integer K, K optimum transformation sequences w.r.t. (t, D, s).

5.2 Proving NP-Hardness

In this subsection, we show that finding K optimum transformation sequences w.r.t. (t, D, s) is NP-hard even if $K = 1$. We consider the following decision problem, called *transformation decision problem*.

Instance: A DTD D, a tree t valid against D, an update script $s = op_1 op_2 \cdots op_n$ to D, and a positive integer B.

Question: Is there a transformation sequence $TS = t_0, t_1, \cdots, t_n$ w.r.t. (t, D, s) such that $\gamma(TS) \leq B$?

We have the following theorem.

Theorem 2 *The transformation decision problem is NP-hard.*

Proof: We use the following SAT problem.

Instance: A set $X = \{x_1, \cdots, x_n\}$ of variables and a collection $C = \{C_1, \cdots, C_m\}$ of clauses over X.

Question: Is there a satisfying truth assignment for C?

For an instance of the SAT problem, we construct an instance of the transformation decision problem, as follows.

- Tree $t = t_0$ is constructed as shown in Fig. 4(left,top), where T_i and F_i stand for sequences of labels defined as follows ($1 \leq i \leq n$).
  - Let $C_{i_1}, \cdots, C_{i_k}$ be the clauses that contain positive literal x_i. Then $T_i = c_{i_1}, \cdots, c_{i_k}$ (label c_{i_j} corresponds to clause C_{i_j}). That is, T_i consists of the clauses that are satisfied by setting $x_i =$ true.
  - Let $C_{i_1}, \cdots, C_{i_l}$ be the clauses that contain negative literal $\neg x_i$. Then $F_i = c_{i_1}, \cdots, c_{i_l}$. That is, F_i consists of the clauses that are satisfied by setting $x_i =$ false.
- $D = (d, r)$, where $d(r) = (a)^+$, $d(a) = (b)^+$, $d(b) = (T_1|F_1|\cdots|T_n|F_n)$, and $d(c_i) = \epsilon$ for $1 \leq i \leq m$.
- $s = s_1 s_2 s_3$, where

$$
\begin{aligned}
s_1 &= del\_opr(a, \lambda) ext\_elm(a, \lambda) ext\_elm(r, 1), \\
s_2 &= ins\_opr(r, |, \lambda) ins\_subexpr(r, q, 2) \\
&\quad del\_subexpr(r, 1) del\_opr(r, \lambda), \\
s_3 &= ins\_elm(r, c_1, 2) del\_elm(r, 2) \\
&\quad \vdots \\
&\quad ins\_elm(r, c_m, 2) del\_elm(r, 2),
\end{aligned}
$$

and

$$q = ((c_1|\cdots|c_m)^*, (c_1|\cdots|c_m)^*).$$

In s_2, (i) $ins\_subexpr(r, q, 2)$ is a "macro" that inserts q into $d(r)$ at position 2 and (ii) $del\_subexpr(r, 1)$ is a macro that deletes the subexpression at position 1 of $d(r)$. Thus s_2 updates $d(r) = (T_1|F_1|\cdots|T_n|F_n)$ to $((c_1|\cdots|c_m)^*, (c_1|\cdots|c_m)^*)$.

- $B = 3n$.

As shown below, s_1 corresponds to "choosing a truth assignment for $x_1, \cdots, x_n$", and s_3 corresponds to "checking if the truth assignment satisfies C". s_2 is the preliminaries of s_3.

We show that there is a satisfying truth assignment for C *iff* there is a transformation sequence $TS = t_0, t_1, \cdots, t_{|s|}$ w.r.t. (t, D, s) such that $\gamma(TS) \leq B$.

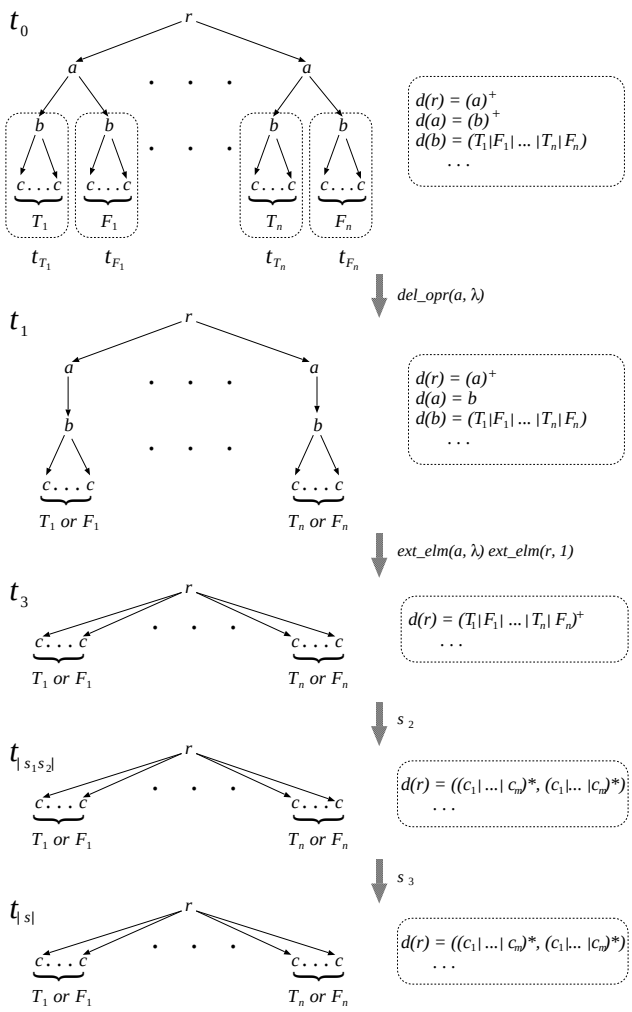

Figure 4: Transformation sequence $t_0, t_1, \cdots, t_{|s|}$.

If part: Assume that there is a transformation sequence $TS = t_0, t_1, \cdots, t_{|s|}$ w.r.t. (t, D, s) such that

$$\gamma(TS) \leq B. \tag{1}$$

Consider first s_1 of s. By $del\_opr(a, \lambda)$ either t_{T_i} or t_{F_i} is deleted from t_0 for every $1 \leq i \leq n$, then by $ext\_elm(a, \lambda)$ n nodes labeled by b are deleted from t_1, and by $ext\_elm(r, 1)$ n nodes labeled by a are deleted from t_2 (Fig. 4). It is easy to see that t_3 is not changed by s_2, i.e., $t_3 = t_4 = \cdots = t_{|s_1 s_2|}$. Thus for the transformation sequence $TS' = t_0, t_1, \cdots, t_{|s_1 s_2|}$ w.r.t. $(t, D, s_1 s_2)$, $\gamma(TS') = 3n = B$. This and (1) imply that by s_3 no node is inserted into $t_{|s_1 s_2|}$ and no node is deleted from $t_{|s_1 s_2|}$. For each $1 \leq i \leq m$, s_3 repeatedly updates $d(r)$ as follows.

1. First, $d(r) = ((c_1|\cdots|c_m)^*, (c_1|\cdots|c_m)^*)$ is updated to $((c_1|\cdots|c_m)^*, c_i, (c_1|\cdots|c_m)^*)$ by $ins\_elm(r, c_i, 2)$,
2. Then $((c_1|\cdots|c_m)^*, c_i, (c_1|\cdots|c_m)^*)$ is updated to $((c_1|\cdots|c_m)^*, (c_1|\cdots|c_m)^*)$ by $del\_elm(r, 2)$.

Since $t_{|s_1s_2|}$ is not changed by s_3, $t_{|s_1s_2|}$ must have a leaf node labeled by c_i for every $1 \leq i \leq m$. Here, consider the following truth assignment ($1 \leq i \leq n$).

$$x_i = \begin{cases} \text{true} & \text{if } t_{F_i} \text{ is deleted by } del\_opr(a,\lambda) \text{ of } s_1, \\ \text{false} & \text{if } t_{T_i} \text{ is deleted by } del\_opr(a,\lambda) \text{ of } s_1. \end{cases}$$

Since $t_{|s_1s_2|}$ has a leaf node labeled by c_i for every $1 \leq i \leq m$, by the definitions of T_i and F_i it is easy to see that the above truth assignment satisfies C.

Only if part: Assume that there is a satisfying truth assignment A for C. Recall that by $del\_opr(a,\lambda)$ of s_1, either t_{T_i} or t_{F_i} is deleted from t_0 for every $1 \leq i \leq n$. Along with the truth assignment A, we can transform t_0 into t_1 so that for every $1 \leq i \leq n$,

- if $x_i = \text{true}$, then t_{F_i} is deleted, and
- if $x_i = \text{false}$, then t_{T_i} is deleted.

Since A is a satisfying truth assignment for C, it is easy to verify that for every $1 \leq i \leq m$, $t_{|s_1s_2|}$ has at least one leaf node labeled by c_i. This implies that $t_{|s_1s_2|}$ is not changed by s_3, i.e., $t_{|s_1s_2|} = t_{|s_1s_2|+1} = \cdots = t_{|s|}$. Here, let $TS = TS_1TS_2$, where $TS_1 = t_0, t_1, \cdots, t_{|s_1s_2|}$ and $TS_2 = t_{|s_1s_2|+1}, \cdots, t_{|s|}$. Then we have $\gamma(TS_1) = 3n$ and $\gamma(TS_2) = 0$. Hence $\gamma(TS) = 3n \leq B$. □

Thus, in general it is unlikely that we can find K optimum transformation sequences efficiently. In the following, we consider finding K optimum transformation sequences assuming that an update script is of length one.

6 Algorithm for Finding K Optimum Transformation Sequences

In this section, we first define the Glushkov automaton [1] of a regular expression, which is required to describe our algorithm. Then we show an algorithm for finding K optimum transformation sequences w.r.t. (t, D, s), assuming that $|s| = 1$.

6.1 Glushkov Automaton

In this subsection, we define the Glushkov automaton of a regular expression. Let r be a regular expression. First, we define the *initial set* I_r and the *final set* F_r, as follows.

- If $r = \epsilon$, then $I_r = F_r = \{E\}$, where E is a new label (I_r and F_r contain E if $\epsilon \in L(r)$).
- If $r = a$ for some $a \in \Sigma$, then $I_r = F_r = \{a_i\}$, where a_i is the subscripted label such that $r' = a_i$.
- If $r = (r_1|\cdots|r_n)$, then $I_r = I_{r_1} \cup \cdots \cup I_{r_n}$ and $F_r = F_{r_1} \cup \cdots \cup F_{r_n}$.
- If $r = (r_1, \cdots, r_n)$, then

$$\begin{aligned} I_r &= (I_{r_1} - \{E\}) \cup \cdots \cup (I_{r_{i-1}} - \{E\}) \cup I_{r_i}, \\ F_r &= F_{r_j} \cup (F_{r_{j+1}} - \{E\}) \cup \cdots \cup (F_{r_n} - \{E\}), \end{aligned}$$

where

$$\begin{aligned} i &= \begin{cases} n & \text{if } E \in I_{r_k} \text{ for every } 1 \leq k \leq n, \\ \min\{k \mid E \notin I_{r_k}, 1 \leq k \leq n\} & \text{otherwise,} \end{cases} \\ j &= \begin{cases} 1 & \text{if } E \in F_{r_k} \text{ for every } 1 \leq k \leq n, \\ \max\{k \mid E \notin F_{r_k}, 1 \leq k \leq n\} & \text{otherwise.} \end{cases} \end{aligned}$$

- If $r = (r_1)^*$ or $r = (r_1)?$, then $I_r = I_{r_1} \cup \{E\}$ and $F_r = F_{r_1} \cup \{E\}$.
- If $r = (r_1)^+$, then $I_r = I_{r_1}$ and $F_r = F_{r_1}$.

Let a_i be a subscripted label occurring in r'. The *set of successors* of a_i in r', denoted $Succ(a_i, r')$, is defined as follows.

- If $r' = a_i$, then $Succ(a_i, r') = \emptyset$.
- If $r' = (r'_1|\cdots|r'_n)$ and a_i occurs in r'_k ($1 \leq k \leq n$), then $Succ(a_i, r') = Succ(a_i, r'_k)$.
- If $r' = (r'_1, \cdots, r'_n)$ and a_i occurs in r'_k ($1 \leq k \leq n$), then

$$Succ(a_i, r') = \begin{cases} Succ(a_i, r'_k) \\ \qquad \text{if } k = n \text{ or } a_i \notin F_{r_k}, \\ Succ(a_i, r'_k) \cup (I_{r_{k+1}} - \{E\}) \cup \\ \cdots \cup (I_{r_j} - \{E\}) \\ \qquad \text{if } k < n \text{ and } a_i \in F_{r_k}, \end{cases}$$

where

$$j = \begin{cases} n & \text{if } E \in I_{r_i} \text{ for every } k+1 \leq i \leq n, \\ \min\{i \mid E \notin I_{r_i}, k+1 \leq i \leq n\} & \text{otherwise.} \end{cases}$$

- If $r' = (r'_1)^*$ or $r' = (r'_1)^+$, then

$$Succ(a_i, r') = \begin{cases} Succ(a_i, r'_1) & \text{if } a_i \notin F_{r_1}, \\ Succ(a_i, r'_1) \cup (I_{r_1} - \{E\}) & \text{otherwise.} \end{cases}$$

- If $r' = (r_1)?$, then $Succ(a_i, r') = Succ(a_i, r'_1)$.

The *Glushkov automaton* of r is a 5-tuple $(Q, \Sigma, \delta, a_I, F)$, where Q is the set of *states*, δ is the *transition function*, a_I is the *initial state*, and F is the set of *final states* defined as follows.

- $Q = sym(r') \cup \{a_I\}$,
- $\delta(a_I, a) = \{a_j \mid a_j \in I_r, a_j^\natural = a\}$ for every $a \in \Sigma$, and $\delta(a_j, a) = \{a_k \mid a_k \in Succ(a_j, r'), a_k^\natural = a\}$,
- $F = \begin{cases} F_r \cup \{a_I\} - \{E\} & \text{if } \epsilon \in L(r), \\ F_r & \text{otherwise.} \end{cases}$

It is easy to show that for any regular expression r, $L(r) = L(G_r)$, where G_r is the Glushkov automaton of r. Figure 5(c) shows the Glushkov automaton of $d_1(a) = (d, (((c)^*, b)|(c, (b)^*)))$.

6.2 Algorithm

In this subsection, we show an algorithm for finding K optimum transformation sequences $TS_1, \cdots, TS_K$ w.r.t. (t, D, op). Because of space limitation, the proof of the correctness and the running time estimation of the algorithm are omitted.

Main Algorithm

Let us first show the "main" algorithm. Let $D = (d_1, sl)$ be a DTD, t be a tree valid against D, n be a node in t, and op be an update operation to D. By t_n we mean the subtree of t rooted at n. Let $D_n = (d_1, l(n))$. We say that $df_1(n), \cdots, df_K(n)$ are K *optimum diffs* w.r.t. (t_n, D_n, op) if for some K optimum transformation sequences $TS_1, \cdots, TS_K$ w.r.t. (t_n, D_n, op), $df_i(n) = df(t_n, t_n^i)$ for every $1 \leq i \leq K$, where $TS_i = t_n, t_n^i$.

The following algorithm computes K optimum diffs $df_1(n), \cdots, df_K(n)$ w.r.t. (t_n, D_n, op) for each node n in bottom-up manner. In the algorithm, for each node n in t, (i) a graph $G(N, E)$ representing the descendants of n and (ii) a "weight" function w assigning a diff to each edge on $G(N, E)$ are obtained in steps 5 to 18, then K optimum diffs $df_1(n), \cdots, df_K(n)$ are computed by finding K "shortest" paths on $G(N, E)$ in step 19. In step 4, $n_1, \cdots, n_m$ denote the children of n and $(d_2, sl) = op(D)$. MkGraph's and FindKDiffs are shown later.

Main(D, t, op, K)
Input: A DTD $D = (d_1, sl)$, a tree t valid against D, an update operation op to D, and a positive integer K.
Output: K optimum diffs w.r.t. (t, D, op).
begin
1. **for** each node n in t (in bottom-up order) **do**
2. **if** n is a leaf node **then**
3. $df_1(n) \leftarrow \emptyset$ and $df_i(n) \leftarrow nil$ for each $2 \leq i \leq K$;
4. **if** n is an internal node with $l(n) = a$ and $l(n_1) \cdots l(n_m) \notin L(d_2(a))$ **then**
5. **if** $op = ins\_elm(a, b, vi)$ **then**
6. $(G(N, E), w) \leftarrow$ MkGraph1(D, t, n, op, K);
7. **else if** $op = del\_elm(a, vi)$ **then**
8. $(G(N, E), w) \leftarrow$ MkGraph2(D, t, n, op, K);
9. **else if** $op = ext\_elm(a, u)$ **then**
10. $(G(N, E), w) \leftarrow$ MkGraph3(D, t, n, op, K);
11. **else if** $op = agg\_elm(a, b, u)$ **then**
12. $(G(N, E), w) \leftarrow$ MkGraph4(D, t, n, op, K);
13. **else if** $op = del\_opr(a, u)$ **then**
14. $(G(N, E), w) \leftarrow$ MkGraph5(D, t, n, op, K);
15. **else if** $op = change\_opr(a, opr, u)$ **then**
16. $(G(N, E), w) \leftarrow$ MkGraph6(D, t, n, op, K);
17. **else** // $l(n) \neq a$ or $l(n_1) \cdots l(n_m) \in L(d_2(a))$
18. $(G(N, E), w) \leftarrow$ MkGraph7(D, t, n, op, K);
19. $(df_1(n), \cdots, df_K(n)) \leftarrow$ FindKDiffs$(G(N, E), w)$;
20. Let n be the root of t. **return** $df_1(n), \cdots, df_K(n)$;
end

Outline of Subroutines

To explain the subroutines, we pick MkGraph2 and FindKDiffs from them (the others are shown later). We first show outlines of MkGraph2 and FindKDiffs, then show their formal definitions.

Assume that $op = del\_elm(a, vi)$, and let n be a node in t labeled by a. We have to find K optimum diffs $df_1(n), \cdots, df_K(n)$. Assuming that $df_1(n_i), \cdots, df_K(n_i)$ have been obtained for each child n_i of n, we find $df_1(n), \cdots, df_K(n)$ as follows.

1. We first make a "child list graph" $CL(N', E')$ of n. Fig. 5(a,b) is an example assuming that $K = 2$. As shown later, each edge $n'_{i-1} \xrightarrow{l} n'_i$ is to hold the lth diff $df_l(n_i)$ of n_i.

2. We make the Glushkov automaton $G_{d_1(a)}$ of $d_1(a)$. For example, Fig. 5(c) shows the Glushkov automaton of $d_1(a) = (d, (((c)^*, b)|(c, (b)^*)))$.

3. We make the "product graph" $G(N, E)$ of $G_{d_1(a)}$ and $CL(N', E')$ as shown in Fig. 5(d), then associate a "weight" (actually, a diff) to each edge in E. $G(N, E)$ has the following properties.

   (a) Any path in $G(N, E)$ from the source to a destination represents the sequence of children that matches $d_1(a)'$. For example, path

   $$(a_I, n'_0) \xrightarrow{l_1} (d_1, n'_1) \xrightarrow{l_2} (c_{221}, n'_2) \xrightarrow{l_3} (b_{2221}, n'_3)$$

   in Fig. 5(d) represents the sequence of children n_1, n_2, n_3 that matches $d_1 c_{221} b_{2221} \in L(d_1(a)')$, for any $l_1, l_2, l_3 \in \{1, 2\}$.

   (b) Each edge $e = (a_{i-1}, n'_{i-1}) \xrightarrow{l} (a_i, n'_i) \in E$ is associated with the lth diff $df_l(n_i)$ of n_i, except that if a_i is the subscripted label deleted from $d_1(a)$ by op, then e is associated with $\{n_i\}$, which represents the root n_i of the subtree deleted by op.

4. Find K "shortest" paths from the source to the destinations. By (a) and (b) above, K optimum diffs on these paths are precisely K optimum diffs $df_1(n), \cdots, df_K(n)$.

Steps 1 to 3 are done by MkGraph2 and step 4 is done by FindKDiffs.

Let us define MkGraph2 formally. Let n be a node in t with children $n_1, \cdots, n_m$ and K be a positive integer. Then the *child list graph* of n (w.r.t. K) is a graph $CL(N', E')$, where

$$\begin{aligned} N' &= \{n'_0, \cdots, n'_m\}, \\ E' &= \{n'_{i-1} \xrightarrow{l} n'_i \mid 1 \leq i \leq m, 1 \leq l \leq K\}, \end{aligned}$$

and $l(n'_0) = a_I$ and $l(n'_i) = l(n_i)$ for $1 \leq i \leq m$. Let $G_r = (Q, \Sigma, \delta, a_I, F)$ be the Glushkov automaton of r. Then the *product* of G_r and $CL(N', E')$ is a graph

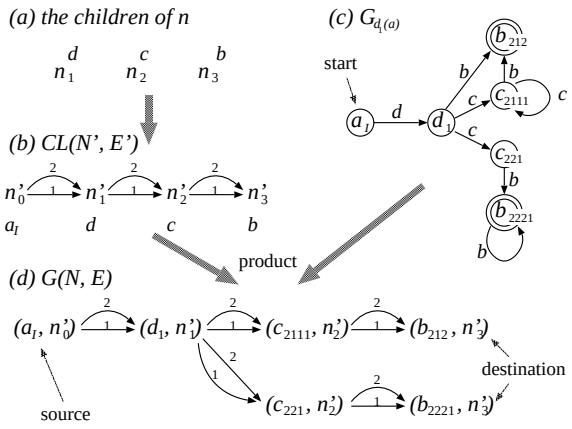

Figure 5: The product $G(N,E)$ of $G_{d_1(a)}$ and $CL(N',E')$.

$G(N,E)$, where

$$\begin{aligned} N &= \{(a_i, n'_j) \mid a_i \in Q, n'_j \in N', a_i^\natural = l(n'_j)\}, \\ E &= \{(a_i, n'_{j-1}) \xrightarrow{l} (a_k, n'_j) \mid \\ &\quad a_k \in \delta(a_i, a_k^\natural), n'_{j-1} \xrightarrow{l} n'_j \in E'\}. \end{aligned}$$

We say that (a_I, n'_0) is the *source* of $G(N,E)$ and (a_h, n'_j) is a *destination* of $G(N,E)$ if $a_h \in F$ and $n'_j = n'_m$. Now MkGraph2 is defined as follows.

MkGraph2(D, t, n, op, K)

Input: A DTD $D = (d_1, sl)$, a tree t valid against D, a node n in t, an update operation $op = del\_elm(a, vi)$, and a positive integer K.
Output: A graph $G(N,E)$ and a function w.
begin
1. Construct the child list graph $CL(N', E')$ of n.
2. Construct the Glushkov automaton $G_{d_1(a)}$ of $d_1(a)$.
3. Construct the product $G(N,E)$ of $G_{d_1(a)}$ and $CL(N',E')$.
4. **for** each $e = (a_i, n'_{j-1}) \xrightarrow{l} (a_k, n'_j) \in E$ **let**
5. $w(e) \leftarrow \begin{cases} \{n_j\} & \text{if } a_k = b_h \text{ and } l = 1, \\ nil & \text{if } a_k = b_h \text{ and } l > 1, \\ df_l(n_j) & \text{if } a_k \neq b_h, \end{cases}$
where b_h is the subscripted label deleted from $d_1(a)'$ by op.
6. **return** $(G(N,E), w)$;
end

We next define FindKDiffs. This algorithm slightly differs from usual algorithms for finding K shortest paths (e.g. [5]), since $G(N,E)$ may have more than one paths having the same diff (among such paths at most one path is required).[2] Let us consider a path

$$p = (a_I, n'_0) \xrightarrow{l_1} (a_1, n'_1) \xrightarrow{l_2} \cdots \xrightarrow{l_j} (a_j, n'_j)$$

on $G(N,E)$ $(1 \leq j \leq m)$ and let $w(p)$ be the diff on p, that is,

$$\begin{aligned} w(p) &= w((a_I, n'_0) \xrightarrow{l_1} (a_1, n'_1)) \cup \cdots \\ &\quad \cup w((a_{j-1}, n'_{j-1}) \xrightarrow{l_j} (a_j, n'_j)). \end{aligned}$$

Then $w(p)$ represents the diffs of n in t assuming that

1. the diffs of $n_{j+1}, \cdots, n_m$ are ignored,
2. n'_i is associated with a_i for every $1 \leq i \leq j$, i.e., $w_1[i] = a_i$ in step (1-a-i) of Trans2, and that
3. under Condition (2) above, t_{n_i} is transformed by the l_ith optimum diff of n_i $(1 \leq i \leq j)$.

Let $\Delta_{(a_j, n'_j)}$ be the collection of K optimum diffs among the diffs on the paths from (a_I, n'_0) to (a_j, n'_j) on $G(N,E)$. FindKDiffs computes $\Delta_{(a_j, n'_j)}$ for every $(a_j, n'_j) \in N$, from the source to the destinations (by steps 3 and 4). In step 3, we write $(a_i, n'_j) \prec (a_h, n'_k)$ if $(a_i, n'_j) \xrightarrow{l} (a_h, n'_k) \in E$. In steps 8 to 10, $\Delta_{(a_j, n'_j)}[k]$ denotes the kth optimum diff in $\Delta_{(a_j, n'_j)}$, and we assume that $|\Delta_{(a_i, n'_j)}[K]| = \infty$ if $\Delta_{(a_i, n'_j)}[K] = nil$.

FindKDiffs$(G(N,E), w)$
Input: A product $G(N,E)$ and a weight function w.
Output: K optimum diffs of n.
begin
1. $\Delta_{(a_i, n'_j)} \leftarrow \{nil, \cdots, nil\}$ (K nil's) for each $(a_i, n'_j) \in N$;
2. $\Delta_{(a_I, n'_0)}[1] \leftarrow \emptyset$;
3. Sort the nodes in N w.r.t. $\prec$ topologically.
Let $(a_{i_1}, n'_{j_1}), \cdots, (a_{i_{|N|}}, n'_{j_{|N|}})$ be the result.
4. **for** $h = 1$ **to** $|N|$ **do**
5. **for** each edge $e \in E$ starting from (a_{i_h}, n'_{j_h}) such that $w(e) \neq nil$ **do**
6. Let $e = (a_{i_h}, n'_{j_h}) \xrightarrow{l} (a_i, n'_j)$.
7. **for** each $k = 1$ to K with $\Delta_{(a_{i_h}, n'_{j_h})}[k] \neq nil$ **do**
8. $df \leftarrow \Delta_{(a_{i_h}, n'_{j_h})}[k] \cup w(e)$;
9. **if** $|df| < |\Delta_{(a_i, n'_j)}[K]|$ and $df \notin \Delta_{(a_i, n'_j)}$ **then**
10. Delete $\Delta_{(a_i, n'_j)}[K]$ from $\Delta_{(a_i, n'_j)}$ and add df to $\Delta_{(a_i, n'_j)}$.
11. $\Delta \leftarrow \bigcup_{(a_i, n'_m) \text{ is a destination}} \Delta_{(a_i, n'_m)}$;
12. **return** K optimum distinct diffs in Δ;
end

The Other Subroutines

In the following, we show the rest of MkGraphs. We assume that, for a node n in t with children $n_1, \cdots, n_m$, $df_1(n_i), \cdots, df_K(n_i)$ have been obtained for each $1 \leq i \leq m$.

First, let us show MkGraph1. We have $op = ins\_elm(a, b, vi)$. Since new elements may be inserted into t by op, we extend the definition of product graph accordingly. Let $D = (d_1, sl)$ be a DTD, b_h be the subscripted label inserted into $d_1(a)'$ by op, $CL(N', E')$ be the child list graph of n, and $G_{d_2(a)} = (Q, \Sigma, \delta, a_I, F)$ be the Glushkov automaton of $d_2(a)$, where $(d_2, sl) =$

[2] The result of FindKDiffs may contain paths having distinct diffs with the same cost.

$op(D)$. Then the *product of* $G_{d_2(a)}$ *and* $CL(N', E')$ *with the insertion of* b_h is a graph $G(N, E)$, where

$$
\begin{aligned}
N &= \{(a_i, n'_j) \mid a_i \in Q, n'_j \in N', a_i^\natural = l(n'_j)\},\\
E &= \{(a_i, n'_{j-1}) \xrightarrow{l} (a_k, n'_j) \mid n'_{j-1} \xrightarrow{l} n'_j \in E', \text{ and,}\\
&\quad \text{(i) } b_h \in \delta(a_i, b) \text{ and } a_k \in \delta(b_h, a_k^\natural) \text{ or}\\
&\quad \text{(ii) } a_k \in \delta(a_i, a_k^\natural)\}.
\end{aligned}
$$

This is defined similarly to the product graph in MkGraph2, except Condition (i) above. This condition states that a node matching b_h is inserted between n'_{j-1} and n'_j. In MkGraph1 below, the weight (diff) of each edge is computed in steps 4 to 12, where Δ_1 in step 7 is the collection of diffs in the case of (ii) and Δ_2 in step 10 is the collection of diffs in the case of (i). $\Delta[l]$ in step 11 denotes the lth optimum diff in Δ.

MkGraph1(D, t, n, op, K)
Input: A DTD D, a tree t valid against D, a node n in t, an update operation $op = ins\_elm(a, b, vi)$, and a positive integer K.
Output: A graph $G(N, E)$ and a function w.
begin
1. Construct the child list graph $CL(N', E')$ of n.
2. Construct the Glushkov automaton $G_{d_2(a)} = (Q, \Sigma, \delta, a_I, F)$ of $d_2(a)$, where $(d_2, sl) = op(D)$.
3. Construct the product $G(N, E)$ of $G_{d_2(a)}$ and $CL(N', E')$ with the insertion of b_h.
4. **for** each edge $(a_i, n'_{j-1}) \xrightarrow{1} (a_k, n'_j) \in E$ **do**
5. $\quad \Delta_1 \leftarrow \emptyset$, $\Delta_2 \leftarrow \emptyset$;
6. $\quad$ **if** $a_k \in \delta(a_i, a_k^\natural)$ **then**
7. $\quad\quad \Delta_1 \leftarrow \{df_l(n_j) \mid 1 \le l \le K\}$;
8. $\quad$ **if** $b_h \in \delta(a_i, b)$ and $a_k \in \delta(b_h, a_k^\natural)$ **then**
9. $\quad\quad$ Create a new node v_j labeled by b.
10. $\quad\quad \Delta_2 \leftarrow \{df_l(n_j) \cup \{v_j\} \mid 1 \le l \le K\}$;
11. $\quad \Delta \leftarrow \Delta_1 \cup \Delta_2$;
12. $\quad w((a_i, n'_{j-1}) \xrightarrow{l} (a_k, n'_j)) \leftarrow \Delta[l]$ for each $1 \le l \le K$.
13. **return** $(G(N, E), w)$;
end

We omit MkGraph3 since it is defined exactly same as MkGraph2.

Let us show MkGraph4. We have $op = agg\_elm(a, b, u)$. Let $G(N, E)$ be the product of $G_{d_1(a)}$ and $CL(N', E')$, as defined in MkGraph2, and let $q = sub(d_1(a), u)$. By op, for each sequence of nodes in t that maximally match q, a node labeled by b is inserted into t as the parent of the nodes. To represent such a node insertion, for each path $(a_{i_0}, n'_{j_0}) \xrightarrow{l_1} \cdots \xrightarrow{l_n} (a_{i_n}, n'_{j_n})$ in $G(N, E)$ that "maximally matches" q, we add new edges from (a_{i_0}, n'_{j_0}) to (a_{i_n}, n'_{j_n}) to $G(N, E)$ that represents a newly inserted node. Formally, we say that a path $(a_{i_0}, n'_{j_0}) \xrightarrow{l_1} \cdots \xrightarrow{l_n} (a_{i_n}, n'_{j_n})$ in $G(N, E)$ *maximally matches* q if

- (i) (a_{i_0}, n'_{j_0}) is the source of $G(N, E)$ or (ii) $a_{i_1} \notin Succ(a_{i_0}, q')$,
- $a_{i_{k+1}} \in Succ(a_{i_k}, q')$ for $1 \le k \le n-1$, and
- (i) (a_{i_n}, n'_{j_n}) is a destination of $G(N, E)$ or (ii) there is an edge $(a_{i_n}, n'_{j_n}) \xrightarrow{l} (a_h, n'_k)$ such that $a_h \notin Succ(a_{i_n}, q')$.

Suppose that there is a path from (a_{i_0}, n'_{j_0}) to (a_{i_n}, n'_{j_n}) maximally matching q. Let $G((a_{i_0}, n'_{j_0}), (a_{i_n}, n'_{j_n}), q)$ be the subgraph of $G(N,E)$ consisting of the paths from (a_{i_0}, n'_{j_0}) to (a_{i_n}, n'_{i_n}) maximally matching q ((a_{i_0}, n'_{j_0}) is the source and (a_{i_n}, n'_{j_n}) is the destination of $G((a_{i_0}, n'_{j_0}), (a_{i_n}, n'_{j_n}), q)$). We create a new edge $e = (a_{i_0}, n'_{j_0}) \xrightarrow{l} (a_{i_n}, n'_{j_n})$ representing a newly inserted node, say v_{j_0,j_n} $(1 \le l \le K)$, and compute $w(e)$ by taking the union of (i) $\{v_{j_0,j_n}\}$ and (ii) the diff on the lth shortest path from (a_{i_0}, n'_{j_0}) to (a_{i_n}, n'_{j_n}) in $G((a_{i_0}, n'_{j_0}), (a_{i_n}, n'_{j_n}), q)$. In step 6 of MkGraph4, E_p is the set of edges representing newly inserted nodes. In steps 7 to 11 the weight of each edge in E_p is obtained. In steps 12,13 the weight of each edge in $G(N,E)$ is defined, and in step 14 E_p is added to $G(N,E)$.

MkGraph4(D, t, n, op, K)
Input: A DTD $D = (d_1, sl)$, a tree t valid against D, a node n in t, an update operation $op = agg\_elm(a,b,u)$, and a positive integer K.
Output: A graph $G(N, E)$ and a function w.
begin
1. Construct the child list graph $CL(N', E')$ of n.
2. Construct the Glushkov automaton $G_{d_1(a)}$ of $d_1(a)$.
3. Construct the product $G(N,E)$ of $G_{d_1(a)}$ and $CL(N', E')$.
4. **for** each $e = (a_i, n'_{j-1}) \xrightarrow{l} (a_k, n'_j) \in E$ **let**
5. $\quad w'(e) \leftarrow df_l(n_j)$;
6. $E_p \leftarrow \{(a_i, n'_j) \xrightarrow{l} (a_h, n'_k) \mid$ there is a path from (a_i, n'_j) to (a_h, n'_k) in $G(N, E)$ that maximally matches q, $1 \le l \le K\}$;
7. **for** each $e = (a_i, n'_j) \xrightarrow{1} (a_h, n'_k) \in E_p$ **do**
8. $\quad$ Construct a graph $G' = G((a_i, n'_j), (a_h, n'_k), q)$.
9. $\quad (df_1, \cdots, df_K) \leftarrow$ FindKDiffs(G', w').
10. $\quad$ Create a new node $v_{j,k}$ labeled by b.
11. $\quad w((a_i, n'_j) \xrightarrow{l} (a_h, n'_k)) \leftarrow df_l \cup \{v_{j,k}\}$ for each $1 \le l \le K$;
12. **for** each $e = (a_i, n'_{j-1}) \xrightarrow{l} (a_k, n'_j) \in E$ **let**
13. $\quad w(e) \leftarrow \begin{cases} nil & \text{if } a_k \in sym(q'),\\ df_l(n_j) & \text{otherwise.} \end{cases}$
14. Add the edges in E_p to $G(N, E)$.
15. **return** $(G(N, E), w)$;
end

We show MkGraph5. We have $op = del\_opr(a, u)$ and $l(d_1(a), u) =$ '+', thereby $sub(d_1(a), u) = q^+$ for some regular expression q, and the the '+' of q^+ is dropped by op. Thus, for each sequence seq of nodes that maximally matches q^+, seq must be "shrunk". The *q-extraction* $d_e(a)$ of $d_1(a)$ is obtained from $d_1(a)$ by replacing q^+ with (q^*, q, q^*). Clearly, $d_e(a)$ is equivalent to $d_1(a)$. MkGraph5 is defined so that the nodes matching the left/right q^* in (q^*, q, q^*) are deleted. In step 5, $sub(d_e(a), u1)$ ($sub(d_e(a), u3)$) is the left (resp., right) q^* in (q^*, q, q^*). $\{n_j\}$ in step 7 denotes the root of an element matching such q^*'s.

MkGraph5(D, t, n, op, K)

Input: A DTD $D = (d_1, sl)$, a tree t valid against D, a node n in t, an update operation $op = del\_opr(a, u)$, and a positive integer K.
Output: A graph $G(N, E)$ and a function w.
begin
1. Construct the child list graph $CL(N', E')$ of n.
2. Construct the $sub(d_1(a), u)$-extraction $d_e(a)$ of $d_1(a)$.
3. Construct the Glushkov automaton $G_{d_e(a)}$ of $d_e(a)$.
4. Construct the product $G(N,E)$ of $G_{d_e(a)}$ and $CL(N',E')$.
5. $LR \leftarrow sym(sub(d_e(a), u1)) \cup sym(sub(d_e(a), u3))$;
6. **for** each $e = (a_i, n'_{j-1}) \xrightarrow{l} (a_k, n'_j) \in E$ **let**
7. $w(e) \leftarrow \begin{cases} \{n_j\} & \text{if } a_k \in LR \text{ and } l = 1, \\ nil & \text{if } a_k \in LR \text{ and } l > 1, \\ df_l(n_j) & \text{otherwise.} \end{cases}$
8. **return** $(G(N, E), w)$;
end

Let us show MkGraph6. In this case, $op = change\_opr(a, opr, u)$, and either (i) $l(d_1(a), u) =$ '*' and $opr =$ '?' or (ii) $l(d_1(a), u) =$ '*' and $opr =$ '+'. The case of (i) can be treated similarly to MkGraph5. In the following, we consider the case of (ii). Let $q = sub(d_1(a), u1)$. Then $sub(d_1(a), u) = q^*$. Since q^* is changed to q^+ by op, for each position matching q^* due to the fact that $\epsilon \in L(q^*)$, we have to insert a sequence of elements matching q. Let $G_{d_1(a)} = (Q, \Sigma, \delta_1, a_I, F)$ be the Glushkov automaton of $d_1(a)$ and $G_{d_2(a)} = (Q, \Sigma, \delta_2, a_I, F)$ be the Glushkov automaton of $d_2(a)$, where $(d_2, sl) = op(D)$. We say that the transition from a_i to a_k is *missing* if $a_i \in \delta_1(a_k, a_k^\natural)$ but $a_i \notin \delta_2(a_k, a_k^\natural)$. If nodes n_i, n_{i+1} match a_i and a_k, respectively, and the transition from a_i to a_k is missing, then for a word $w \in L(q)$, it suffices to insert $|w|$ elements matching w between n_i and n_{i+1} of t. Thus MkGraph6 is defined as follows. In steps 7 and 8, nodes $v_1, \cdots, v_{|w|}$ are the roots of new elements matching w between n_i and n_{i+1}.

MkGraph6(t, n, D, op, K)

Input: A DTD $D = (d_1, sl)$, a tree t valid against D, a node n in t, an update operation $op = change\_opr(a, opr, u)$, and a positive integer K.
Output: A graph $G(N, E)$ and a function w.
begin
1. Construct the child list graph $CL(N', E')$ of n.
2. Construct the Glushkov automaton $G_{d_1(a)}$ of $d_1(a)$.
3. Construct the product $G(N,E)$ of $G_{d_1(a)}$ and $CL(N',E')$.
4. Let w be the shortest word in $L(sub(d_1(a), u1))$.
5. **for** each $e = (a_i, n'_{j-1}) \xrightarrow{l} (a_k, n'_j) \in E$ **do**
6. **if** the transition from a_i to a_k is missing **then**
7. Create new nodes $v_1, \cdots, v_{|w|}$ labeled by $w[1], \cdots, w[|w|]$, respectively.
8. $w(e) \leftarrow \{v_1, \cdots, v_{|w|}\} \cup df_l(n_j)$;
9. **else**
10. $w(e) \leftarrow df_l(n_j)$;
11. **return** $(G(N, E), w)$;
end

Finally, MkGraph7 is defined similarly to MkGraph2. The only difference is that step 5 of MkGraph2 is replaced by the following.

5. $w(e) \leftarrow df_l(n_j)$;

We have the following.

Theorem 3 *Let D be a DTD, t be a tree valid against D, and K be a positive integer. Then* Main(D, t, op, K) *runs in $O(|t| \cdot d(t)^3 \cdot |D|^2 \cdot K^2)$ time, where $d(t)$ is the maximum outdegree in t.* □

7 Conclusion

In this paper, we first showed that the problem of finding K optimum transformation sequences w.r.t. (t, D, s) is NP-hard even if $K = 1$. Then, assuming that $|s| = 1$, we proposed an algorithm for finding K optimum transformation sequences w.r.t. (t, D, s), which runs in time polynomial of $|D|$, $|t|$, and K.

This paper presented no experimental result. As a future work, we need to examine (i) if our algorithm can present appropriate transformations and (ii) the efficiency of our algorithm. Moreover, we used "diff" as the criterion of optimality of transformation. We would like to consider if other appropriate criterion can be employed.

Acknowledgement

This work is partially supported by the Grant-in-Aid for Young Scientists (B) #20700077.

References

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

[2] G. Guerrini, M. Mesiti, and D. Rossi. Impact of xml schema evolution on valid documents. In *Proc. WIDM (in conjunction with ACM CIKM)*, pages 39–44, 2005.

[3] K. Hashimoto, Y. Ishihara, and T. Fujiwara. A proposal of update operations for schema evolution in xml databases and their properties on preservation of schema's expressive power. *IEICE Trans. Inf. & Syst. (Japanese Edition)*, J90-D(4):990–1004, 2007.

[4] E. Leonardi, T. T. Hoai, S. S. Bhowmick, and S. Madria. Dtd-diff: A change detection algorithm for dtds. In *Proc. DASFAA*, pages 817–827, 2006.

[5] E. Martins. K-th shortest path problem. http://www.mat.uc.pt/~eqvm/OPP/KSPP/KSPP.html.

[6] B. V. N. Prashant and P. Sreenivasa Kumar. Managing xml data with evolving schema. In *Proc. COMAD*, 2006.

[7] N. Suzuki. An edit operation-based approach to the inclusion problem for dtds. In *Proc. ACM SAC*, pages 482–488, 2007.

[8] N. Suzuki and Y. Fukushima. An xml document transformation algorithm inferred from an edit script between dtds. In *Proc. the 19th Australasian Database Conference (ADC 2008)*, pages 175–184, 2008.

[9] Y. Velegrakis, R. J. Miller, and L. Popa. Mapping adaption under evolving schemas. In *Proc. VLDB*, pages 584–595, 2003.