경로 매칭 알고리즘을 이용한 구조화된 문서의 변화 탐지

(Change Detection of Structured Documents using Path-Matching Algorithm)

이 경 호 † 변 창 원 †† 최 윤 철 ††† 고 견 ††††

(Kyong Ho Lee) (Chang Won Byun) (Yoon Chul Choy) (Kyun Koh)

요 약 본 논문에서는 SGML/XML 문서의 구 버전과 신 버전간의 차이를 계산할 수 있는 효율적인 알고리즘을 제안한다. 차이는 구 버전의 문서를 신 버전으로 변환하는데 소요되는 편집 스크립트로 간주할 수 있다. 제안된 알고리즘은 상향식과 하향식의 복합적인 접근 방식을 적용한다. 먼저 두 버전을 구성하는 노드간의 대응관계를 상향식으로 생성하며 하향식 너비 우선 탐색을 적용하여 편집 스크립트를 계산한다. 제안된 알고리즘은 모든 노드간의 대응 여부를 모두 조사할 필요가 없기 때문에 대응관계를 보다 빠르게 생성할 수 있다. 또한 삽입, 삭제, 그리고 갱신의 단순한 변화는 물론이고 부트리 이동과 복사의 구조적으로 보다 의미 있는 변화를 탐지할 수 있다.

Abstract This paper presents an efficient algorithm to compute differences between old and new versions of an SGML/XML document. The difference between the two versions can be considered to be an edit script that transforms one document tree into another. The proposed algorithm is based on hybridization of bottom-up and top-down methods: matching relationships between nodes in the two versions are produced in a bottom-up manner and top-down breadth-first search computes an edit script. Because the algorithm does not need to investigate possible existence of matchings for all nodes, faster matching can be achieved. Furthermore, it can detect more structurally meaningful changes such as subtree move and copy as well as simple changes to the node itself like insert, delete, and update.

1. 서 론

SGML(Standard Generalized Markup Language)[1]과 XML(eXtensible Markup Language)[2]은 논리적인 구조 정보를 표현할 수 있으며 이 기종간에 호환이 가능하다는 장점 때문에 CALS(Commerce At Light Speed), EC(Electronic Commerce)/EDI(Electronic Data Interchange), 인터넷 등의 다양한 분야에서 문서처리의 표준 포맷으로 널리 사용되고 있다[3,4,5]. 이에 SGML/XML에 기반한 구조화된 문서[1]의 처리 기법[6,7]에 관한 연구가 활발히 진행 중이다.

특히 구조화된 문서간의 변화(change) 또는 차이(difference)[2]를 탐지하는 방법은 버전 관리(version management)[10,11,12,13,14], 데이타 웨어하우징(data warehousing)[15], 그리고 능동 데이타베이스(active database)[16] 등의 다양한 응용 분야에서 매우 중요하다. 또한 보다 편리한 WWW 환경을 제공하기 위하여 사용자가 자주 방문하는 XML 문서의 변화를 자동으로 탐지할 수 있는 방법이 요구된다.

† 비 회 원 : 미국 국립표준기술연구소 연구원
kyongho@nist.gov
†† 비 회 원 : (주) 프리챌 연구원
ayasiec@freechal.co.kr
††† 종신회원 : 연세대학교 컴퓨터과학과 교수
ycchoy@rainbow.yonsei.ac.kr
†††† 종신회원 : 청주대학교 컴퓨터정보공학과 교수
kyunkoh@chongju.ac.kr
논문접수 : 1999년 9월 21일
심사완료 : 2001년 6월 26일

1) 본 논문에서 "구조화된 문서", "SGML/XML 문서", 그리고 "SGML/XML에 기반한 구조화된 문서"는 모두 동일한 의미로 사용된다.

2) 본 논문에서 변화(change)와 차이(difference)는 동일한 의미를 갖는 용어로 사용된다.

변화 탐지에 관한 기존 연구는 그 대상을 기준으로 문자열[16,17,18,19], 관계형 데이타[21], 그리고 트리나 그래프와 같은 계층적인 데이타[22,23,24,25,26,27,28]에 관한 세가지로 분류할 수 있다. 그러나 문자열과 관계형 데이타를 처리 대상으로 하는 연구의 경우, 계층적인 정보를 반영하지 않기 때문에 구조화된 문서인 SGML/XML 문서를 처리하지 못한다.

SGML/XML 문서는 논리적인 계층 구조를 포함한다. 예를 들어, 과학 기술 논문의 경우, 논문 제목, 저자, 소속, 요약, 키워드, 본문, 장, 절, 단락, 그리고 참고 문헌 등의 논리적인 구성 요소들이 계층적인 구조를 형성한다. 따라서 SGML/XML 문서의 경우, 효과적인 변화 탐지를 위하여 내용에 해당하는 문자열은 물론이고 논리적인 구조 정보를 고려하여야 한다. 예를 들어, 문서를 구성하는 절이 복사 또는 이동된 경우에 이를 탐지할 수 있어야 한다. 일반적으로 사용자는 SGML/XML 문서를 편집하면서 복사 메커니즘을 빈번히 사용한다. 특히 변화에 대한 검색을 지원하는 응용 시스템의 경우, 복사를 단순히 다수의 삽입으로 표현한다면 사용자에게 의미 있는 정보를 제공하지 못할 것이다. 마찬가지로 이동을 다수의 삽입과 삭제로 표현하는 것은 바람직하지 못하다. 그러나 계층적인 데이타를 처리 대상으로 하는 기존 연구의 경우, 트리(tree)를 구성하는 노드(node)의 삽입(insert), 삭제(delete), 그리고 갱신(update)[3]의 단순한 변화만을 추출하거나 속도가 느린 등의 단점을 가지고 있다.

본 논문에서는 SGML/XML에 기반한 구조화된 문서간의 변화를 효율적으로 탐지할 수 있는 알고리즘을 제안한다. 제안된 알고리즘은 구조화된 문서의 구 버전과 신 버전의 차이를 효율적으로 탐지하기 위하여 상향식과 하향식의 복합적인 접근 방식을 적용한다. 먼저 상향식 경로 매칭 알고리즘(path-matching algorithm)을 적용하여 두 버전을 구성하는 노드 간의 대응관계[4]를 생성하고, 이를 기반으로 두 버전에 하향식 너비 우선 탐색(breadth-first search)을 적용하여 변화를 추출한다. 특히 본 논문은 구조화된 문서의 효과적인 표현을 위하여 뿌리 노드(root node)를 갖으며 형제 노드(sibling node)간에 순서가 존재하는 뿌리 순서 트리(rooted ordered tree)에 기반한 문서 모델을 제안한다.

제안된 방법론은 순서 트리를 대상으로 하는 Zhang과 Shasha의 방법[22], Shasha와 Zhang의 방법[23], 그리고 Zhang의 방법[24]과 비교하여 노드의 삽입, 삭제, 그리고 갱신의 단순한 변화는 물론이고 계층적인 변화로써 부트리(subtree)의 이동(move)을 추출한다. 한편 구조화된 문서를 대상으로 하는 Chawathe 등의 방법[27]은 제한적인 가정을 만족하는 문서에 대하여 계층적인 변화로써 부트리의 이동을 추출한다. 반면에 제안된 방법론은 노드간의 일-대-일(one-to-one)의 대응관계는 물론이고 일-대-다(one-to-many)의 대응관계를 지원하기 때문에 부트리 복사(copy)의 탐지가 가능하다. 특히 제안된 경로 매칭 알고리즘은 경로에 기반하기 때문에 노드간의 대응 여부를 모두 조사하는 Chawathe 등의 방법과 비교하여 대응관계를 보다 빠르게 생성한다.

본 논문의 구성은 다음과 같다. 2장에서는 구조화된 문서를 효과적으로 표현할 수 있는 문서 모델을 제안한다. 또한 변화를 효율적으로 표현할 수 있는 변화 연산과 최적의 변화 탐지를 위하여 제안된 비용 모델을 기술한다. 3장에서는 변화 탐지와 관련한 기존의 연구 결과를 간략히 소개하고 이의 문제점을 기술한다. 4장에서는 제안된 알고리즘을 대응관계 생성과 변화 연산 추출의 두 단계로 구분하고 각 단계에 대한 자세한 설명을 제공한다. 5장에서는 제안된 알고리즘의 성능을 관련 연구와 비교 및 분석하고, 마지막으로 6장에서는 결론을 기술한다.

2. 문서 모델, 변화 연산 및 비용 모델

본 절에서는 구조화된 문서를 효과적으로 표현할 수 있는 문서 모델을 제안한다. 또한 구조화된 문서간의 구조적인 변화를 효율적으로 표현할 수 있는 변화 연산을 정의하고, 기존 연구와의 차이점을 기술한다. 마지막으로 최적의 변화 연산을 추출하기 위하여 제안된 비용 모델을 소개한다

2.1 문서 모델

SGML/XML 문서는 선언부(declaration), 문서 유형 정의부(document type definition), 그리고 실제 문서부(document instance)의 세 부분으로 구성된다[8,9]. 본 논문은 동일한 문서 유형 정의부를 따르는 실제 문서부의 구 버전과 신 버전간의 변화 탐지 방법을 제안한다.

그림 1에서 나타나는 바와 같이, 일반적으로 문서 유형 정의부는 문서의 논리적인 구성 요소에 해당하는 엘

3) 기존 연구는 계층적인 데이타 간의 차이를 표현하기 위하여 삽입, 삭제, 갱신, 이동, 그리고 복사 등 다양한 종류의 연산을 사용하였다. 그러나 관련 연구마다 서로 다른 정의의 연산을 사용한다. 본 논문에서 사용하는 연산의 정의 및 기존 연구와의 차이점은 2장에서 자세히 기술한다.

4) 두 버전에서 변하지 않은(즉, 동일한) 노드 또는 유사한 값을 갖는 노드 간의 대응관계를 의미하며 이에 대한 설명은 3~4장에서 자세히 기술한다.

```
<--        Element name      Content model                    -->
<!ELEMENT  MEMO       - -    ((TO & FROM), BODY, CLOSE?) >
<!ELEMENT  TO         - o    (#PCDATA)                    >
<!ELEMENT  FROM       - o    (#PCDATA)                    >
<!ELEMENT  BODY       - o    (PARAGRAPH)*                 >
<!ELEMENT  PARAGRAPH  - o    (#PCDATA | QUOTATION)*       >
<!ELEMENT  QUOTATION  - o    (#PCDATA)                    >
<!ELEMENT  CLOSE      - -    (#PCDATA)                    >
```

그림 1 Memo 문서를 표현하기 위한 문서 유형 정의부의 예

리먼트(element)를 정의하는 엘리먼트 선언문(element declaration)의 집합으로 구성된다[8,9]. 또한 엘리먼트 선언문에 포함되는 내용 모델(content model)은 특정 엘리먼트가 포함할 수 있는 하위 엘리먼트의 종류와 문자열의 포함 여부(그림 1에서 #PCDATA는 엘리먼트가 내용으로 문자열을 포함함을 의미한다.)를 명시한다. 특히 특정 엘리먼트에 포함되는 하위 엘리먼트와 문자열 사이에는 순서가 존재한다. 따라서 본 논문에서는 구조화된 문서를 표현하기 위하여 뿌리 노드를 포함하며 형제 노드간에 순서가 존재하는 순서 트리에 기반한 문서 모델을 제안한다. 제안된 문서 모델에 따라 SGML/XML 문서의 엘리먼트와 문자열은 트리 구조의 노드를 구성한다.

문서 모델을 구성하는 각각의 노드는 레이블(label)과 값(value)의 두 가지 속성값을 포함한다. 본 논문에서 엘리먼트와 문자열에 해당하는 노드는 각각 엘리먼트 노드(element node)와 텍스트 노드(text node)라고 정의된다. 특히 엘리먼트 노드는 해당 엘리먼트 이름을 레이블로 갖는다. 그러나 텍스트 노드는 레이블을 갖지 않는다. 한편 엘리먼트 노드는 값을 갖지 않으며 텍스트 노드는 해당 문자열을 값으로 갖는다.

```
<MEMO>
  <TO>Comrade Napoleon</TO>
  <FROM>Snowball</FROM>
  <BODY>
    <PARAGRAPH>
          In Animal Farm, George Orwell says:
          <QUOTATION>These were large sheets ... </QUOTATION>
          Do you think SGML would be helpful?
    </PARAGRAPH>
  </BODY>
  <CLOSE>Comrade Snowball</CLOSE>
</MEMO>
```

그림 2 그림 1의 문서 유형 정의부를 따르는 실제 문서부의 예

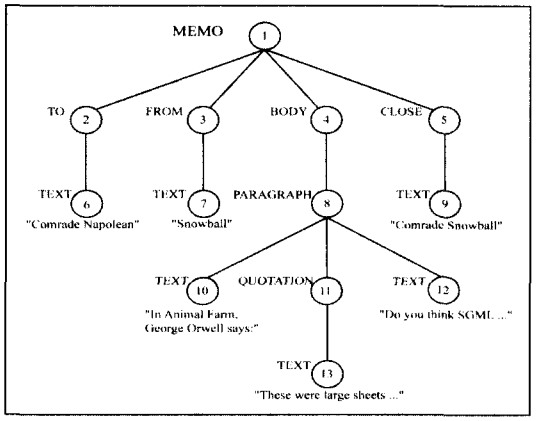

그림 3 실제 문서부를 문서 모델로 표현한 결과

그림 2는 그림 1의 문서 유형 정의부를 따르는 구조화된 문서의 예이며 그림 3은 그림 2의 문서를 제안된 문서 모델로 표현한 결과이다. 그림 3에서 레이블이 *"TEXT"*인 노드는 텍스트 노드에 해당한다(본 논문에서는 편의상 텍스트 노드의 레이블을 *"TEXT"*로 명시하였다.). 특히 텍스트 노드와 엘리먼트 노드는 각각 트리 구조의 단말 노드(leaf node)와 중간 노드(interior node)를 구성한다[5]. 본 논문에서는 중간 노드를 단말 노드를 제외한 나머지 노드(중간 노드의 차수(degree)는 0보다 크다.)로 정의한다.

2.2 변화 연산

기존 연구[28,29,30,31,32,33,34]와 마찬가지로 본 논문은 구조화된 문서간의 차이를 구 버전의 문서를 신 버전의 문서로 변환하는데 필요한 연산(본 논문에서는 이를 변화 연산이라고 정의한다)의 순차적인 집합으로 표현한다. 따라서 제안된 변화 탐지 알고리즘은 두 문서간의 차이로써 구 버전의 문서, T_1, 신 버전의 문서, T_2, 로 변환하는데 필요한 변화 연산의 순차적인 집합을 추출한다. SGML/XML 문서의 경우, 노드의 삽입, 삭제, 갱신의 단순한 변화는 물론이고 이동과 복사의 구조적인 변화를 추출할 수 있어야 한다. 이를 위하여 본 논문에서는 삽입, 삭제, 갱신, 이동, 그리고 복사 연산을 지원한다. 제안된 변화 연산의 정의에 대한 간단한 설명은 표 1과 같다.

한편 계층적인 데이타를 대상으로 변화 탐지를 지원하는 기존 연구는 다양한 종류의 변화 연산을 제공한다. 제안된 변화 연산을 관련 연구와 비교하면 다음과 같다.

5) 본 논문에서 텍스트 노드와 단말 노드는 동일한 의미로 사용되며 엘리먼트 노드는 중간 노드와 동일한 의미로 사용된다.

표 1 변화 연산

종류	기 술
삽입	삽입 연산 $INS((n, l, v), p, c, k)$은 레이블 l과 값 v를 갖는 노드 n을 노드 p의 자식 노드로 삽입함을 표현한다. 여기서 c는 p의 자식노드의 부분 집합을 의미하며 k는 형제 순서를 의미한다. 따라서 삽입 연산의 수행 결과로써 n은 부모 노드 p의 k번째 자식이 되며 c를 자식으로 포함한다.
삭제	삭제 연산 $DEL(n)$은 삽입 연산의 역이며 삭제된 노드의 자식은 부모 노드의 자식이 된다. 특히 문서 엘리먼트(document element)에 해당하는 뿌리 노드는 SGML/XML의 특성상 삭제할 수 없다.
갱신	갱신 연산 $UPD(n, v_1, v_2)$는 노드 n의 값 v_1을 v_2로 수정한다. 특히 구조화된 문서 처리에서 문서의 수정은 엘리먼트 이름의 변경이 아니라 문자열의 수정을 의미하기 때문에 제안된 갱신 연산은 텍스트 노드의 값을 대상으로 한다. 이에 중간 노드에 대한 갱신 연산은 지원하지 않는다.
이동	이동 연산 $MOV(n, p, k)$은 노드 n을 뿌리 노드로 갖는 부트리를 노드 p의 k번째 자식으로 이동한다.
복사	복사 연산 $CPY(n, p, k)$은 노드 n을 뿌리 노드로 갖는 부트리를 노드 p의 k번째 자식으로 복사한다.

삽입과 삭제 연산의 경우, Zhang과 Shasha의 방법과 Shasha와 Zhang의 방법과 마찬가지로 모든 노드의 삽입과 삭제를 표현할 수 있다. 반면에 Chawathe 등의 방법은 중간 노드의 삽입과 삭제 연산을 허용하지 않는다. 이에 중간 노드를 삭제하려면 먼저 해당 자손 노드(descendant)를 모두 이동하거나 삭제하여야 한다는 제약을 갖는다.

갱신 연산의 경우, 구조화된 문서 처리의 특성상 Chawathe 등의 방법과 마찬가지로 문자열에 대한 갱신을 지원한다. 일반적으로 SGML/XML 문서 처리 과정에서 특정 엘리먼트의 변경은 엘리먼트 이름이 아니라 해당 엘리먼트가 포함하는 구조 및 내용의 수정을 의미한다. 따라서 구조화된 문서의 특성을 고려하여 제안된 갱신 연산은 텍스트 노드의 값의 변화를 기술한다. 반면에 Zhang과 Shasha의 방법과 Shasha와 Zhang의 방법의 갱신 연산은 레이블의 변화를 표현한다. 한편 이동과 복사 연산은 각각 Chawathe 등의 방법의 이동 연산과 Chawathe와 Monila의 방법[34]의 복사 연산과 동일한 의미를 갖는다.

2.3 비용 모델

일반적으로 두 문서 T_1과 T_2간의 차이를 표현하는 변화 연산의 경우의 수는 다수가 존재할 수 있다. 이에 본 논문에서는 최적의 변화 연산을 추출하기 위하여 비용 모델을 제안한다. 제안된 비용 모델에서 삽입, 삭제, 갱신, 이동, 그리고 복사 연산의 비용은 각각 c_i, c_d, c_u, c_m, 그리고 c_c로 표현된다.

먼저 본 논문에서는 계산을 쉽게 하기 위하여 c_i, c_d, c_m, 그리고 c_c를 단위 비용으로 정의한다. 전술한 바와 같이, 구조화된 문서의 특성상 중간 노드에 대한 c_u는 존재하지 않는다. 반면에 텍스트 노드의 경우, c_u는 비교함수, $c_u(x, y)$, 에 의하여 계산된다. 제안된 비교함수는 문자열의 유사도 비교에 주로 사용되는 LCS(longest common subsequence)[19,27]에 기반 한다. 두 문자열의 LCS의 비율, $lcsr(x, y)$, 에 기반한 비교함수의 정의는 수식 (1)과 같다. 따라서 c_u는 0과 2 사이의 값을 갖는다.

$$c_u(x, y) = -2 \times lcsr(x, y) + 2 \quad (1)$$

여기서 $lcsr(x, y)$는 수식 (2)와 같이 두 문자열 x와 y간의 LCS의 비율을 의미한다.

$$lcsr(x, y) = \frac{2 \times |lcs(x, y)|}{|x| + |y|} \quad (2)$$

[6]

제안된 비용 모델을 적용하여 변화 연산을 추출하는 예는 다음과 같다. 먼저 두 문서 T_1과 T_2에 텍스트 노드 x와 y가 존재한다고 가정하자. 이때 두 노드의 값과 트리 상의 상대적인 위치가 서로 다를 경우, 두 가지 경우의 변환 과정을 적용하여 두 문서 간의 차이를 표현할 수 있다. 첫 번째 경우, 먼저 노드 x를 노드 y의 위치로 이동한다. 그리고 노드 x의 값을 노드 y의 값으로 변경하여 신 버전을 생성할 수 있다. 두 번째 경우는 노드 x를 삭제한 후에 노드 y를 삽입하여 신 버전을 생성할 수 있다. 두 경우는 각각 이동과 갱신 연산 그리고 삭제와 삽입 연산으로 두 문서간의 차이를 표현하였다.

위의 두 가지 경우 중에서 보다 효율적인 변화 연산을 선택하기 위하여 비교 함수의 값을 고려한다. 만일 두 노드간의 비교함수의 값이 1보다 작으면 이동과 갱신 연산의 전체 비용(< 2)이 삭제와 삽입 연산의 전체 비용(= = 2)보다 작기 때문에 첫 번째 경우의 변화 연산을 추출하는 것이 비용 면에서 효율적이다. 반면에 비교 함수의 값이 1보다 크다면 이동과 갱신 연산의 전체 비용이 2보다 크기 때문에 삭제와 삽입 연산으로 표현되는 두 번째 경우를 선택하는 것이 바람직하다.

3. 관련 연구 및 문제점

전술한 바와 같이 변화 탐지와 관련한 기존 연구는 그 대상을 기준으로 문자열, 관계형 데이타, 그리고 계층적인 데이타에 관한 세 가지로 분류할 수 있다. 본 절에서는 계층적인 데이타 중에서 본 논문과 마찬가지로 순서 트리를 대상으로 하는 기존의 연구 결과를 간략히 소개하고 이의 문제점을 기술한다.

6) |lcs(x, y)|, |x|, 그리고 |y|는 각각 문자열 x와 y간의 LCS의 길이, x의 길이, 그리고 y의 길이를 나타낸다.

Zhang과 Shasha의 연구는 레이블 순서 트리(labeled ordered tree)를 대상으로 하며 두 트리의 차이를 노드의 삽입과 삭제, 그리고 레이블의 갱신 등의 단순한 연산으로 표현한다. 특히 노드간에 일-대-일[7]의 대응관계만을 허용하기 때문에 복사 연산을 지원하지 않는다.

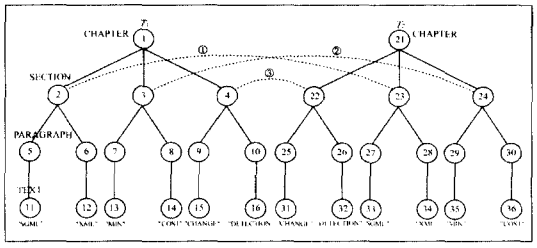

그림 4 형제 순서를 위배하는 대응관계의 예(대응관계는 점선으로 표시하였으며 나머지 노드간의 대응관계는 생략하였음)

또한 대응관계를 형성하는 노드간에 형제 순서(sibling order)[8]를 유지하기 때문에 이동 연산을 추출할 수 없다. 예를 들어, 그림 4에서 형제 순서를 위배하는 대응관계인 ①, ②, 그리고 ③을 생성하지 못한다. 따라서 4번 부트리(4번 노드를 뿌리 노드로 갖는 부트리)가 1번 노드의 첫번째 자식 노드로 이동되었음을 탐지할 수 없다.

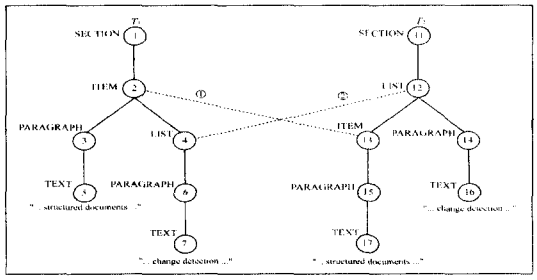

그림 5 조상 순서를 위배하는 대응관계

또한 대응관계를 형성하는 노드 간에 조상 순서(ancestor order)를 유지한다[9]. 예를 들어, 그림 5에서 대응 관계 ②는 조상 순서를 위배하기 때문에 생성될 수 없다. 따라서 Zhang과 Shasha의 방법을 적용하여 생성 가능한 대응관계는 그림 6과 같으며 이로부터 추출 가능한 변화 연산은 세 개의 삭제 연산(4번, 6번, 그리고 7번 노드의 삭제)과 세 개의 삽입 연산(12번, 14번, 그리고 16번 노드의 삽입)에 해당된다.

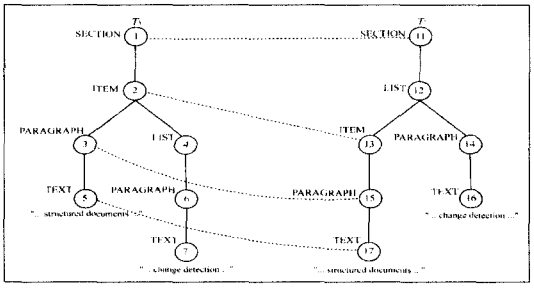

그림 6 조상 순서를 유지하는 대응관계

그러나 그림 5에서 대응관계 ②를 생성하고 4번 부트리를 1번 노드의 두 번째 자식으로 이동한 후에 2번 부트리를 4번 노드의 첫 번째 자식으로 이동한 것으로 두 버전간의 차이를 표현하는 것이 비용면에서 보다 효율적이다. 본 논문에서는 그림 5의 대응관계 ②를 생성하기 위하여 새로운 대응관계 생성 알고리즘을 제안한다. 이에 대한 설명은 "4.1 대응관계의 생성"에서 자세히 기술한다. 두개의 이동 연산을 순차적으로 적용하여 구 버전을 신 버전으로 변환 과정은 그림 7과 같다.

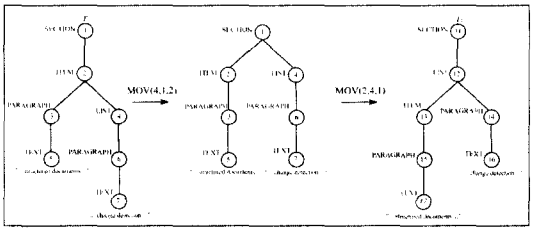

그림 7 구 버전에 두 개의 이동 연산을 단계적으로 적용한 결과

한편 Zhang은 시간 복잡도가 $O(|T_1|\ |T_2|\ \min(\text{depth}(T_1), \text{leaves}(T_1))\min(\text{depth}(T_2), \text{leaves}(T_2)))$인 기존의 알고리즘[22]에 추가적인 제약을 가하여 $O(|T_1||T_2|)$의 알고리즘[24]을 제안하였다. 한편 Chang 등[25]과 Wang 등[26]은 Zhang과 Shasha의 알고리즘을 적용하여 구조화 문서간의 변화 탐지 시스템을 설계 및 구현하였다.

7) 대응관계 [i1, j1], [i2, j2]에 대하여, i1 = = i2 if and only if j1 = = j2.

8) 대응관계 [i1, j1], [i2, j2]에 대하여, 만일 i1이 i2의 왼쪽에 위치한다면 j1또한 j2의 왼쪽에 위치하며 이의 역 또한 만족하여야 한다.

9) 대응관계 [i1, j1], [i2, j2]에 대하여, 만일 i1이 i2의 조상 노드라면 j1은 j2의 조상이며 이의 역 또한 만족하여야 한다.

한편 관련 연구 Chawathe 등의 방법은 본 논문과 마찬가지로 구조화된 문서의 변화 탐지 알고리즘을 제안한다. 제안된 알고리즘은 대응관계의 생성과 이에 기반한 변화 연산 추출의 두 단계로 구성되며 노드의 삽입, 삭제, 그리고 갱신은 물론이고 부트리의 이동을 탐지할 수 있다. 그러나 중간 노드의 삽입과 삭제 연산을 지원하지 않기 때문에 중간 노드의 삭제를 위해서는 먼저 자손 노드를 모두 이동하거나 삭제하여야 하는 제약을 갖는다.

또한 기존 연구와 마찬가지로 일-대-일의 대응관계만을 허용하기 때문에 부트리간의 복사 연산을 추출하지 못한다. 또한 문서를 구성하는 레이블은 순환하지 않는다는 조건(acyclic labels condition)을 가정한다. 따라서 그림 5의 예제와 같이 순환하는 엘리먼트(엘리먼트 *ITEM*과 *LIST*는 각각 *LIST*와 *ITEM*을 포함한다.)를 포함하는 문서를 처리할 수 없다. 또한 문서는 동일하거나 유사한 값의 단말 노드를 두 개 이상 포함하지 않는다고 가정한다. 이와 같이 Chawathe 등의 방법은 제한적인 조건을 만족하는 문서를 대상으로 하기 때문에 Zhang과 Shasha의 방법과 비교하여 처리 속도가 빠르다. 그러나 SGML/XML에 기반한 구조화된 문서의 경우, 일반적으로 순환하는 엘리먼트와 내용이 동일하거나 유사한 문장을 다수 포함하기 때문에 Chawathe 등의 방법은 최적의 변화 연산을 추출하지 못하거나 적용 범위가 제한적이다.

4. 변화 탐지 알고리즘

전술한 바와 같이 제안된 알고리즘은 대응관계 생성과 변화 연산 추출의 두 단계로 구성된다. 첫번째 단계에서는 구 버전과 신 버전에 해당하는 두 문서 T_1과 T_2에 경로 매칭 알고리즘을 적용하여 노드간의 대응관계를 생성한다. 두 번째 단계에서는 대응관계를 기반으로 두 문서에 하향식 너비 우선 탐색 과정을 적용하여 변화 연산을 추출한다. 각 단계에 대한 보다 자세한 설명은 다음과 같다.

4.1 대응관계의 생성

기존 연구 [27,28,29,30,34]는 대응관계의 생성을 위하여 단순히 노드의 레이블만을 고려하였다. 그러나 구조화된 문서의 경우, 이름이 동일하지만 내용이 다른 엘리먼트가 존재할 수 있다. 예를 들어, SGML/XML 문서는 서로 다른 내용의 엘리먼트 *paragraph* 또는 *section* 등을 다수 포함할 수 있다. 따라서 구조화된 문서로부터 의미있는 대응관계를 생성하기 위해서는 엘리먼트의 이름은 물론이고 엘리먼트의 내용에 기반한 방법론이 요구된다.

한편 Chawathe 등의 방법은 본 논문과 마찬가지로 중간 노드의 레이블과 단말 노드의 값을 이용하여 대응관계 생성하였다. 그러나 레이블은 순환하지 않으며 동일하거나 유사한 내용의 단말 노드가 둘 이상 존재하지 않는다는 제한적인 가정에 기반하기 때문에 그 적용 범위가 제한적이다.

본 논문에서 제안된 알고리즘은 구조화된 문서의 내용과 구조 정보를 모두 고려하여 대응관계를 생성한다. 예를 들어, 두 문서를 구성하는 텍스트 노드에 대하여 해당 노드의 값이 서로 동일하거나 유사하더라도 부모 노드의 레이블이 서로 일치하지 않으면 대응관계를 생성하지 않는다. 또한 레이블이 동일한 엘리먼트 노드에 대하여 자손 노드 간에 내용 및 구조에 있어서 유사성이 존재하지 않는다면 대응관계를 생성하지 않는다.

이를 위하여 제안된 알고리즘은 내용에 해당하는 텍스트 노드간의 대응관계를 먼저 생성하고, 이를 기반으로 구조 정보에 해당하는 엘리먼트 노드간의 대응관계를 생성한다. 대응관계 생성에 대한 자세한 설명은 다음과 같다.

대응관계 생성 기준 1: 텍스트 노드, $y \in T_2$, 에 대하여, 부모 노드의 레이블이 서로 동일하며 비교함수의 값이 임계 값 t보다 작은 텍스트 노드, $x_1, \ldots, x_k \in T_1$, 가 존재한다고 하자. 이때 비교함수의 값이 최소에 해당하는 $x_{i1} \ldots x_{im}$, $1 \le i_1 < i_2 < \ldots < i_m \le k$ 을 선택하여 대응관계 $[x_{i1}, y] \ldots [x_{im}, y]$을 생성한다.

먼저 대응관계 생성 기준 1을 적용하여 T_2의 텍스트 노드 각각에 대하여 T_1의 텍스트 노드에 대한 대응관계를 생성한다. 이를 위하여 비교함수를 이용한다. 특히 대응하는 텍스트 노드의 부모 노드의 레이블은 서로 동일하여야 한다. 이는 구조화된 문서에서 최소의 독립 단위는 엘리먼트이기 때문이다. 텍스트 노드에 대한 대응관계의 생성 결과, 구 버전의 텍스트 노드는 대응관계를 갖지 않거나 일-대-일, 일-대-다, 또는 다-대-일(many-to-one)의 대응관계를 갖는다.

특히 다-대-일 관계(구 버전 상의 여러 텍스트 노드가 신 버전에 속하는 한 개의 노드와 대응하는 경우)의 경우, 구 버전 상의 해당 노드 중에서 한 개를 제외한 나머지 노드는 모두 삭제되었음을 의미한다. 삭제된 노드의 경우, 대응관계를 생성할 필요가 없기 때문에 대응관계 생성 기준 2를 적용하여 다-대-일을 일-대-일의 대응관계로 변환한다. 결과적으로 대응관계를 갖는 구 버전의 텍스트 노드는 일-대-일 또는 일-대-다의 대응관계를 형성하며 신 버전의 텍스트 노드는 모두 일-대-일의 대응관계를 갖는다.

정의: 경로(x)란 노드 x의 부모 노드로부터 뿌리 노드까지의 노드의 순차적인 **집합**을 의미한다.

대응관계 생성 기준 2: 텍스트 노드 $y \in T_2$에 대하여, 일-대-다의 대응관계를 일-대-일로 변환한다. 먼저

노드 y와 노드 $x_1, \ldots x_k \in T_1$ 사이에 일-대-다의 대응관계가 존재한다고 가정하자. 최적의 대응관계를 선택하기 위하여 경로 간의 유사도를 이용한다. 이를 위하여 먼저 서로 대응하는 노드 y와 x_i, $1 \leq i \leq k$,로부터 경로의 쌍을 계산하고, 유사도가 가장 높은 경로쌍에 해당하는 대응관계 $[y, x_j]$, $1 \leq j \leq k$,을 선택한다. 유사도가 가장 높은 경로쌍을 결정하기 위하여, 두 경로간의 레이블에 대한 LCS의 크기가 가장 큰 경로쌍을 선택한다. 만일 이를 만족하는 경로쌍이 둘 이상 존재하면 LCS를 구성하는 노드 간의 형제 순서가 가장 많이 일치하는 경로를 선택한다. 만일 유사도가 동일한 경로가 둘 이상 존재한다면 임의의 대응관계를 선택한다.

두 번째로 중간 노드 간의 대응관계를 생성한다. 특히 동일한 레이블의 중간 노드를 모두 비교하는 것은 비효율적이다. 따라서 서로 대응하는 단말 노드를 자손으로 포함하는 중간 노드만을 비교한다. 이를 위하여 제안된 경로 매칭 알고리즘은 대응관계를 형성하는 단말 노드의 경로를 구성하는 중간 노드 간의 대응관계를 생성한다.

전술한 바와 같이, 그림 5의 경우, 보다 효율적인 변화 연산을 추출하기 위하여 조상 순서를 위배하지만 대응관계 ②를 생성하는 것이 바람직하다. 반면에 그림 8은 단순히 4번 노드의 삭제와 12번 노드의 삽입을 통하여 새로운 문서를 생성한 경우이다. 이에 대응관계 ⑤를 생성하지 않는 것이 바람직하다. 한편 그림 5의 대응관계 ②와 그림 8의 대응관계 ⑤는 모두 조상 순서를 위배한다. 따라서 그림 5의 대응관계 ②는 허용하면서 그

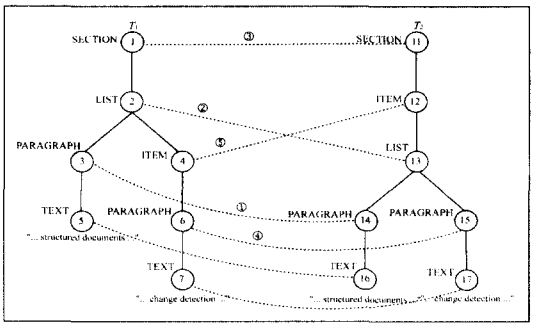

그림 8 조상 순서를 위배하는 대응관계

림 8의 대응관계 ⑤는 허용하지 않는 방법이 요구된다. 이를 위하여 경로 매칭 알고리즘은 서로 대응하는 경로쌍에 대하여 조상 순서를 유지하는 대응관계를 생성한다. 예를 들어, 그림 5의 경우, 텍스트 노드 5번과 17번이 서로 대응하기 때문에 경로 {3, 2, 1}과 {15, 13, 12, 11}로부터 대응관계 [3, 15], ①, 그리고 [1, 11]을 생성한다. 또한 7번과 16번의 경로 {6, 4, 2, 1}과 {14, 12, 11}로부터 대응관계 [6, 14]와 ②를 생성한다. 즉, 대응관계 ①과 ②는 각각 별개의 경로쌍으로부터 생성되었기 때문에 조상 순서를 위배하지만 생성한다. 반면에 그림 8의 경우, 텍스트 노드 7번과 17번의 경로 {6, 4, 2, 1}과 {15, 13, 12, 11}로부터 대응관계 ④, ⑤, ②, 그리고 ③이 생성 가능하다. 그러나 대응관계 ②와 ⑤는 서로 대응하는 경로쌍으로부터 생성되었으며 조상 순서를

```
입력: OldPath[n]  - 구 버전상의 경로, n(∈ 양의 정수)은 노드 수
      NewPath[m]  - 신 버전상의 경로, m(∈ 양의 정수)은 노드 수
      M  - 대응관계
출력: M  - 갱신된 대응관계
함수 정의:
      char* label(x)          ::= 노드 x의 레이블에 해당하는 문자열을 반환한다.
      Boolean IsEqual(s, t)   ::= 문자열 s와 t가 동일하다면 TRUE를 반환하고,
                                  그렇지 않으면 FALSE를 반환한다.
방법:
      int j = 0;          // NewPath[m]의 인덱스 초기화
      for(int i = 0; i < n; i++) {
         for(int k = j; k < m; k++) {
            //대응관계 간의 조상 순서를 유지하기 위한 조건
            if((NewPath[k] is already matched) && (NewPath[k]의 목적노드[10]가 OldPath에 존재한다)) {
               return M; // 경로의 특성상 더 이상 진행할 필요가 없다.
            }
            if(IsEqual(label(OldPath[i]), label(NewPath[k]))) // {
               [OldPath[i], NewPath[j]] →M;  // 새로운 대응관계 [OldPath[i], NewPath[j]] 생성
               j++;
               break;
            }
         }
      }
```

그림 9 경로 매칭 알고리즘

10) 현재 노드에 대응하는 노드

위배한다. 따라서 대응관계 ⑤는 생성하지 않는다. 경로 매칭 알고리즘의 의사 코드는 그림 9와 같다.

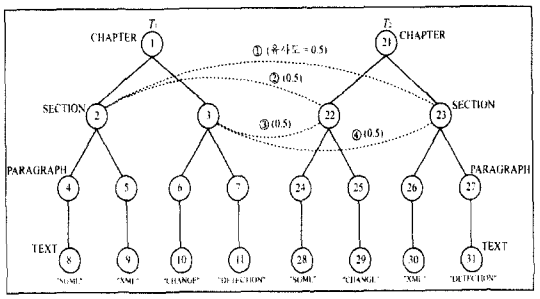

그림 10 중간 노드의 다-대-일 대응관계의 예

한편 경로 매칭 알고리즘을 적용하면 그림 10에서 나타나는 바와 같이 중간 노드간에 다-대-일의 대응관계가 생성될 수 있다. 이에 대응관계 생성 기준 3을 적용하여 다-대-일을 일-대-일의 대응관계로 변환한다.

정의: 대응관계 $[x, y]$의 유사도는 노드 $x \in T_1$와 노드 $y \in T_2$를 뿌리 노드로 갖는 두 부트를 구성하는 자손 노드간의 대응 관계의 비율을 의미하며 이에 대한 정의는 수식 (3)과 같다.

$$\frac{|common(x, y)|}{|descendant(x)|} \tag{3}$$

여기서 함수 $common(x, y)$은 집합 $\{z \mid [w, z] \in M$ and $w \in descendant(x)$, $z \in descendant(y)$, for $label(x) = = label(y)\}$을 반환한다. 또한 $descendant(x)$는 x를 뿌리 노드로 갖는 부트리에서 x를 제외한 나머지 노드의 집합을 반환한다. $|x|$는 집합 x의 개수를 의미한다. 특히 제안된 대응 관계의 유사도는 두 트리 간의 유사한 정도를 의미하지는 않는다. 예를 들어, 대응관계 $[x, y]$와 $[x, z]$의 유사도가 각각 1과 2인 경우, 부 트리 x가 부 트리 y보다 부 트리 z와 더 유사하다는 의미는 아니다.

대응관계 생성 기준 3: 중간 노드 $y \in T_2$에 대하여, 일-대-다의 대응관계를 일-대-일로 변환한다. 먼저 노드 y가 노드 $x_1 \ldots x_k$와 대응관계, $[y, x_1] \ldots [y, x_k]$, 을 형성한다고 하자. 가장 적합한 대응관계 $[y, x_i]$, $1 \le i \le k$,을 선택하기 위하여 유사도가 가장 높은 대응관계를 선택한다. 이를 만족하는 대응관계가 둘 이상 존재할 경우, 대응관계를 형성하는 두 노드의 형제 순서와 경로 간의 유사도(대응관계 생성 기준 2 참조)를 고려한다.

그림 10에서 T_2의 22번과 23번 노드는 일-대-다의 대응관계를 형성한다. 따라서 먼저 대응관계의 유사도를 계산하고 유사도가 높은 대응관계를 선택한다. 노드 22번의 경우, 대응관계 ①과 ②의 유사도는 0.5(=2/4)로써 동일하다. 따라서 형제 순서를 고려하여 대응관계 ②를 선택한다(2번 노드와 22번 노드는 모두 첫 번째 노드이다.). 마찬가지로 노드 23번에 대하여 동일한 방법을 적용하여 대응관계 ④를 선택한다.

4.2 변화 연산의 추출

변화 연산의 추출과정은 전 단계에서 생성된 대응관계를 기반으로 트리를 하향식 너비 우선 탐색하면서 변화 연산을 추출한다. 특히 본 논문에서는 구조적으로 보다 의미있는 변화를 추출하기 위하여 대응관계의 유사도 개념을 제안하며 일-대-일의 대응관계는 물론이고 일-대-다의 대응관계를 지원한다. 대응관계의 종류에 따른 변화 연산의 추출 방법에 대한 보다 자세한 설명은 다음과 같다.

기존 연구와 마찬가지로 신 버전과 구 버전에서 대응관계를 갖지 않는 노드는 각각 삭제 연산과 삽입 연산의 대상이다. 한편 본 절에서 기술하는 변화 연산의 추출 방법은 대응관계를 갖는 구 버전의 노드를 대상으로 한다.

첫째, 일-대-일의 대응관계를 갖는 노드의 경우, 현재 노드가 텍스트 노드이며 서로 대응하는 노드 간의 비교 함수의 값이 0보다 크다면 갱신 연산을 추출한다. 한편 현재 노드가 중간 노드라면 자식 노드의 대응관계로부터 이동 연산을 추출한다. 만일 자식 노드(본 절에서는 특별히 명시하지 않는 한 대응관계를 갖는 노드를 의미한다.)의 수가 한 개라면 부모-외 이동(inter-parent move)의 적용 여부를 조사한다. 자식 노드의 개수가 둘 이상이라면 먼저 부모-내 이동(intra-parent move)의 적용 여부를 조사하고 부모-외 이동 연산을 추출한다.

부모-외 이동 연산의 추출 방법은 다음과 같다. 임의의 대응관계 $[x, y]$와 $[parent(x)^{11}, z]$에 대하여, 만일 노드 z가 노드 y의 부모 노드가 아니라면 노드 x가 노드 y의 위치로 이동된 경우이다. 특히 이동된 노드가 텍스트 노드이며 비교 함수의 값이 1보다 작다면 해당 텍스트 노드는 이동된 후에 갱신된 경우이다. 이동 연산의 추출 과정에 대한 자세한 설명은 4.2.1절에서 자세히 기술한다.

***NIL* 추출 규칙:** 만일 일-대-다의 대응관계 중에서 전 단계에서 이미 이동 연산의 대상으로 식별된 대응관계가 존재한다면 이를 *NIL*의 대상으로 간주한다. 그 밖

11) 본 논문에서 $parent(x)$는 x의 부모 노드를 반환하는 함수로 정의한다

의 경우, 일-대-다의 대응관계, $[x, y_1]$ ... $[x, y_k]$, 중에서 유사도가 가장 큰 대응관계 $[x, y_i]$, $1 \le i \le k$,을 선택한다. 이를 만족하는 대응관계가 둘 이상 존재한다면 부모 노드 간의 대응관계 $[parent(x), parent(y_i)]$, $1 \le i \le k$, 가 *NIL*에 해당하는 대응관계를 선택한다. 또한 이를 만족하는 대응관계가 둘 이상 존재한다면 임의의 대응관계를 선택한다. 특히 x와 y_i의 부모 노드간의 대응관계 $[parent(x), parent(y_i)]$가 *NIL*에 해당하지 않으면 대응관계 $[x, y_i]$로부터 이동 연산을 추출한다.

둘째, 일-대-다의 대응관계를 갖는 노드의 경우, 먼저 *NIL*[12] 추출 규칙을 적용하여 *NIL*에 해당하는 대응관계를 찾고 나머지 대응관계로부터 복사와 삽입 연산을 추출한다. 특히 대응관계의 유사도가 1인 두 부트리는 구조적으로 동일한 경우이다. 이에 *NIL*을 제외한 나머지 대응관계 중에서 유사도가 1인 대응관계로부터 복사 연산을 추출하고 유사도가 1이 아닌 대응관계로부터 삽입 연산을 추출한다. 또한 자식 노드의 대응관계로부터 이동 연산을 추출하며 추출 과정은 일-대-일의 경우와 동일하다. 변화 연산 추출 알고리즘은 그림 11과 같다.

입력: M - 대응관계
출력: 변화 연산의 집합 $S = s_1, \ldots s_m$, 여기서 $s_i \in \{NIL, INS, DEL, UPD, MOV, CPY\}$, $1 \le i \le m$
방법 :
신 버전을 너비 우선 탐색하면서 대응관계를 갖지 않는 노드에 대하여 삽입 연산을 추출한다.
구 버전을 너비 우선 탐색하면서 변화 연산을 추출한다. 대응관계를 갖는 노드에 대하여 *NIL*, 삽입, 복사, 이동 등의 변화 연산을 추출하며 대응관계를 갖지 않는 노드에 대하여 삭제 연산을 추출한다.

그림 11 변화 연산 추출 알고리즘

그림 12는 변화 연산의 추출 결과이다. 구 버전의 문서 T_1을 순회하면서 변화 연산을 추출하는 과정에 대한 자세한 설명은 다음과 같다.

(1) 뿌리 노드인 1번 노드와 21번 노드의 대응관계는 일-대-일이므로 *NIL* 에 해당한다.

(2) 2번 노드는 일-대-다의 대응 관계를 갖는다. 먼저 *NIL* 추출 규칙을 적용하여 *NIL*의 대상으로 대응관계 [2, 23]을 선택하며 나머지 대응관계의 유사도를 이용하여 삽입 연산(*INS*(22), *INS*(24))

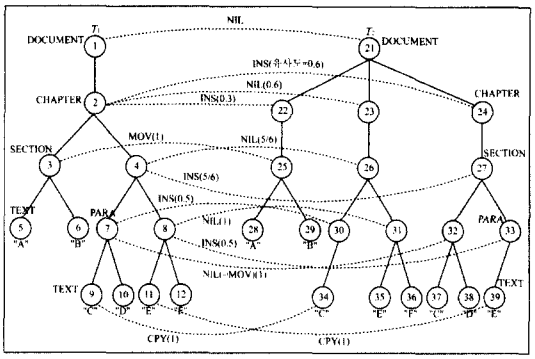

그림 12 변화 연산의 추출 결과

을 추출한다. 또한 자식 노드의 대응관계로부터 이동 연산의 유무를 검사한다. 3번 노드는 23번 노드의 자식 노드에 대응하지 않는다. 이에 대응관계 [3, 25]로부터 이동 연산(*MOV*(3))을 추출한다.

(3) 대응관계 [3, 25]는 이미 이동 연산의 대상으로 식별되었기 때문에 *NIL*에 해당한다.

(4) 대응관계 [4, 26]은 *NIL* 추출 규칙에 의하여 *NIL*에 해당한다. 또한 대응관계 [4, 27]의 유사도는 1이 아니므로 삽입 연산(*INS*(27))에 해당한다. 즉, 27번 노드가 24번 노드의 자식으로 삽입된 경우이다. 한편 5번과 6번 노드는 각각 28번과 29번 노드와 *NIL* 관계를 형성한다.

(5) 7번 노드는 30번과 32번 노드에 대응한다. 이에 유사도를 고려하여 *NIL*의 대상으로 대응관계 [7, 32]를 그리고 삽입 연산(*INS*(30))의 대상으로 [7, 30]을 선택한다. 특히 7번의 부모 노드와 32번의 부모 노드는 *NIL* 관계가 아니기 때문에 대응 관계 [7, 32]로부터 이동 연산(*MOV*(7))을 추출한다.

(6) 대응관계 [8, 31]은 *NIL*에 해당하며 대응관계 [8, 33]로부터 삽입 연산(*INS*(33))을 추출한다.

(7) *NIL* 추출 규칙에 의하여 대응관계 [9, 37]과 [11, 35]는 *NIL*에 해당하며 유사도가 1인 대응관계 [9, 34]와 [11, 39]로부터 복사 연산(*CPY*(9), *CPY*(11))을 추출한다. 한편 대응관계 [10, 38]과 [12, 36]은 *NIL*에 해당한다.

따라서 그림 12의 두 문서 T_1과 T_2의 차이는 변화 연산의 순차적인 집합인 {*INS*((22, *CHAPTER*, NULL_VALUE), 21, NULL_LIST, 1)), *INS*((24, *CHAPTER*, NULL_VALUE), 21, NULL_LIST, 3), *MOV*(3, 22, 1), *INS*((27, *SECTION*, NULL_VALUE), 24,

12) *NIL*은 해당 대응관계에 변화 연산이 적용되지 않았음을 의미한다.

NULL_LIST, 1), *INS*((30, *PARA*, NULL_VALUE), 26, NULL_LIST, 1), *MOV*(7, 27, 1), *INS*((33, *PARA*, NULL_VALUE), 27, NULL_LIST, 2), *CPY*(9, 30, 1), *CPY*(11, 33, 1)}으로 표현된다. 여기서 NULL_VALUE은 엘리먼트 노드가 값을 갖지 않음을 의미하며 NULL_LIST는 노드의 집합이 존재하지 않음을 의미한다.

4.2.1 이동 연산의 추출

전술한 바와 같이 제안된 알고리즘은 자식 노드의 대응관계로부터 부모-내 이동과 부모-외 이동 연산을 추출한다. 먼저 부모-내 이동 연산을 추출하는 방법은 최소 비용의 이동을 추출하기 위하여 Chawathe 등의 방법에서 제안한 LCS에 기반한 방법을 적용한다. 예를 들어, 그림 4의 경우, 대응관계 ①, ②, 그리고 ③은 형제 순서를 위반한다. 이의 해결 방법으로써 두 가지 경우를 고려할 수 있다. 첫 번째 방법는 노드 2번과 3번을 4번 노드의 오른쪽으로 이동하는 것이며 두 번째 방법은 4번 노드를 1번 노드의 왼쪽으로 이동하는 것이다. 따라서 두 번째 방법을 선택하는 것이 비용면에서 효율적이다. 이와 같이 부모-내 이동 연산의 추출 과정은 형제 순서를 유지하기 위하여 형제 노드의 집합을 정렬하는 과정에 해당한다.

부모-외 이동 연산의 추출 방법에 대한 자세한 설명은 다음과 같다. 먼저 노드 $x \in T_1$와 목적 노드 $y \in T_2$의 대응이 *NIL* 연산에 해당하며 $c_1 \ldots c_m$은 x의 자식 노드라고 가정하자. 이때 c_i, $1 \le i \le m$, 가 노드 y의 자식 노드에 대한 대응관계를 갖지 않는다면 c_i는 이동된 경우이다. 만일 c_i가 둘 이상의 대응관계를 갖는다면 유사도가 가장 높은 대응을 이동 연산의 대상으로 선택하며 유사도가 동일한 대응이 둘 이상 존재한다면 임의의 대응을 선택한다.

선택된 대응관계 $[c_i, z]$의 유사도가 1이라면 c_i와 z를 뿌리 노드로 갖는 두 부트리에 속하는 모든 자손 노드 간에는 *NIL* 연산에 해당하는 일-대-일의 대응관계가 존재한다. 만일 대응관계의 유사도가 1보다 작다면 이동되기 전에 자손 노드의 일부가 이동된 경우이다. 예를 들어, 그림 5에서 이동 연산에 해당하는 대응관계 [2, 13]의 유사도는 0.4(=2/5)로써 1보다 작다. 이에 자손 노드의 일부(4번 부트리)가 이미 이동되었음을 알 수 있다. 따라서 변화 연산의 정확한 추출을 위하여 이동 연산의 적용 순서를 고려한다. 한편 만일 유사도가 1보다 크다면 이동 후에 새로운 노드가 삽입 또는 복사되었음을 의미한다.

그림 13은 제안된 변화 탐지 알고리즘을 적용한 결과

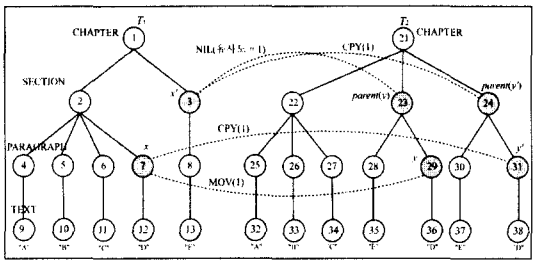

그림 13 변화 연산의 추출 결과

이다. 실제로 7번 부트리를 3번 노드의 두 번째 자식으로 이동한 후에 3번 부트리를 1번 노드의 세 번째 자식으로 복사하였다. 그러나 전술한 알고리즘은 3번 노드의 복사와 7번 노드의 복사와 이동 연산을 추출하였다. 이는 최소 비용의 변화 연산이라고 볼 수 없다.

이를 해결하기 위한 처리 과정은 다음과 같다. 대응관계 $[x, y]$과 $[x, y']$에 각각 이동 연산과 복사 연산이 적용되었다면 *parent*(y)과 *parent*(y')는 임의의 목적 노드 x'에 대응한다. 이때 x'와 *parent*(x)가 서로 다른 노드이며 $[x', \textit{parent}(y')]$에 복사 또는 간접 복사[13] 연산이 적용되었다면 x를 x'로 이동한 후에 x' 또는 x'의 조상 노드가 복사된 경우로써 최소 비용의 변화 연산을 추출한다. 따라서 그림 14에 나타나는 바와 같이, 최소 비용의 변화 연산을 추출할 수 있다.

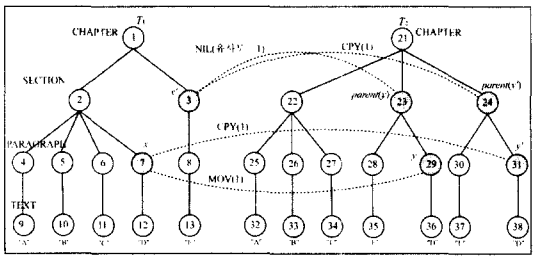

그림 14 최소 비용의 변화 연산 추출

5. 성능 분석

제안된 방법론은 구조화된 문서의 특징을 반영하기 위하여 순서 트리에 기반한 문서 모델을 제안하였다. 또한 대응관계의 효율적인 생성을 위하여 경로 매칭 알고리즘을 개발하였다. 본 절에서는 제안된 알고리즘의 성능을 순서 트리 또는 구조화된 문서를 대상으로 하는

13) 조상 노드의 복사에 의하여 자손 노드가 간접 복사되었을 경우, 이를 간접 복사라고 정의한다.

기존의 연구 결과와 시간 복잡도, 대응관계 생성 기준, 그리고 추출 가능한 변화 연산의 종류 등의 다양한 면에서 비교 및 분석한다.

표 2 관련 연구와 시간 복잡도를 비교한 결과

	Zhang과 Shasha의 방법	Chawathe 등의 방법	본 논문
대응관계 생성	$O(n^2)$[1]	$O(l^2c+ml)$	$O(l^2\frac{c+(m_1)+(m_2}{l_1}\frac{)}{l_2})l_2)$
변화 연산 추출	$O(n^2log^2n)$	$O(nd)$	$O(nd)$

n: the total number of nodes,
l: the total number of leaf nodes
l_1: the total number of leaf nodes of T_1 ,
l_2: the total number of leaf nodes of T_2
m: the total number of internal nodes,
m_1: the total number of internal nodes of T_1
c: the average cost of $c_a(x, y)$ for a pair of leaf nodes x and y.
d: the total number of misaligned nodes

본 논문에서 제안한 방법의 대응관계 생성 과정은 Chawathe 등의 방법과 마찬가지로 단말 노드간의 대응관계를 생성하고 이를 기반으로 중간 노드간의 대응관계를 생성한다. 그러나 제안된 대응관계 생성 알고리즘은 중간 노드간의 대응관계 생성 시, 시간 복잡도가 인 경로 매칭 알고리즘을 적용하기 때문에 $O(ml)$(= $O((m_1+m_2)(l_1+l_2))$)의 Chawathe 등의 방법과 비교하여 처리 속도가 빠르다.

정리 1. 경로 매칭 알고리즘의 시간 복잡도는 이다. 여기서 m_1(또는 m_2)과 l_1(또는 l_2)는 각각 T_1(또는 T_2)을 구성하는 중간 노드와 단말 노드의 개수이다.

증명: 제안된 방법론은 다-대-일의 대응관계를 허용하지 않기 때문에(일-대-일 또는 일-대-다의 대응관계만을 생성한다.) 단말 노드간의 대응관계의 최대 수는 l_2에 해당한다. 한편 T_1과 T_2의 경로를 구성하는 평균적인 노드 수는 각각 m_1/l_1과 m_2/l_2라고 하자. 그렇다면 서로 대응하는 한 쌍의 경로에 대한 대응관계의 생성 시간은 에 비례한다. 따라서 경로 매칭 알고리즘은 의 시간 복잡도를 갖는다.

한편 그림 15는 SGML 문서를 대상으로 대응관계를 생성하는데 소요되는 노드 간의 비교 횟수를 나타낸다. 이를 위하여 1998년 7월에 발행된 특허청 공보 CD-ROM에 수록된 SGML 문서를 실험 대상으로 하였다. 실험에 사용된 문서의 경우, 중간 노드와 단말 노드의 빈도수는 평균적으로 156과 119로서 1.3 : 1의 비율을 보였다.

한편 본 논문의 변화 연산 추출 과정의 시간 복잡도는 Chawathe 등의 방법과 마찬가지로 $O(nd)$로 표현할 수 있다. 표 2는 제안된 방법론을 관련 연구와 처리 속도면에서 비교한 결과이다.

정리 2. 변화 연산 추출 과정의 시간 복잡도는 $O(nd)$이다. 여기서 n은 T_1과 T_2의 노드의 합계이며, d는 부모-내 이동 연산의 대상인 정렬되지 않은 노드의 수이다.

증명: 부모-내 이동 연산을 계산하기 위하여 노드를 정렬하는 과정을 제외하면, 제안된 알고리즘의 너비 우선 탐색 과정은 $O(n)$의 시간에 해결이 가능하다. 먼저 $|x|$를 노드 $x \in T_1$의 자손 노드의 개수라고 하자. 또한 $d(x, y)$는 노드 x와 $y \in T_2$의 자식 노드간의 대응관계 중에서 이동연산의 대상인 정렬되지 않은 자식 노드의 개수라고 하자. 그렇다면 노드 x와 y의 자식 노드를 정렬하는데 필요한 수행 시간은 $O((|x|+|y|)d(x, y))$이다. 따라서 전체 시간 복잡도는 $O(nd)$이다.

대응관계의 생성 기준에 있어서, 본 논문은 Chawathe 등의 방법과 마찬가지로 단말 노드간의 대응관계를 기반으로 중간 노드간의 대응관계를 생성한다. 그러나 Chawathe 등의 방법은 중간 노드가 포함하는 단말 노드의 50% 이상이 서로 대응할 경우에 해당 중간 노드간의

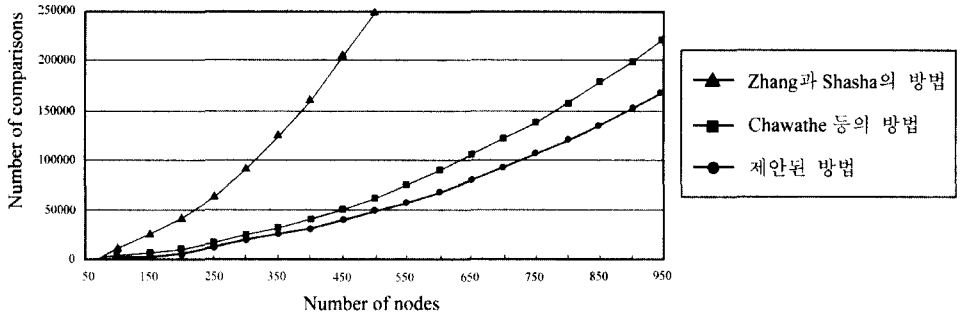

그림 15 대응 관계 생성시 발생하는 노드간의 비교횟수

대응관계를 생성한다. 따라서 그림 16과 같은 경우에 대응관계 ③을 생성할 수 없다. 이에 Chawathe 등의 방법의 대응관계 생성 기준은 최소 비용의 변화 연산을 지원하지 않는다. 또한 Chawathe 등의 방법은 레이블이 비순환하며 동일하거나 유사한 값을 갖는 단말 노드가 중복하지 않는다는 가정에 기반하기 때문에 적용 범위가 제한적이다.

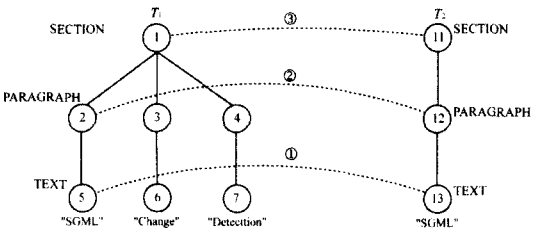

그림 16 대응관계 생성의 예

한편 추출 가능한 변화 연산의 종류에 있어서, 제안된 알고리즘은 삽입, 삭제, 그리고 갱신의 노드 자체에 대한 단순한 변화는 물론이고 구조적으로 보다 의미있는 변화를 탐지하기 위하여 부트리 복사와 이동을 지원한다. 그러나 Zhang과 Shasha의 방법은 일-대-다의 대응관계를 지원하지 않기 때문에 복사 연산을 추출할 수 없다. 또한 형제 순서와 조상 순서를 유지하는 대응관계만을 허용하기 때문에 이동 연산을 추출할 수 없다. 반면에 Chawathe 등의 방법은 부트리의 이동을 탐지할 수 있다. 그러나 전술한 바와 같이 레이블의 비순환 조건에 기반하기 때문에 추출 가능한 이동 연산이 제한적이다. 또한 일-대-일의 대응관계만을 생성하기 때문에 복사 연산을 탐지할 수 없다.

6. 결 론

SGML/XML은 논리적인 구조 정보를 기술할 수 있으며 이 기종간에 호환이 가능하다는 장점 때문에 문서처리의 표준 포맷으로 널리 사용되고 있다. 또한 이외 유연한 모델링 언어적인 특성 때문에 문서 처리는 물론이고 소프트웨어 공학, 소프트웨어 에이전트, 그리고 가상 현실 등의 다양한 분야에서 SGML/XML에 기반한 다양한 응용이 시도되고 있다.

본 논문에서는 SGML/XML에 기반한 구조화된 문서간의 변화를 탐지할 수 있는 효율적인 알고리즘을 제안하였다. 더불어 구조화된 문서를 효율적으로 표현할 수 있는 문서 모델, 문서간의 변화를 효과적으로 기술할 수 있는 변화 연산, 그리고 최적의 변화 연산을 계산하기 위한 비용 모델을 제안하였다. 제안된 알고리즘은 상향식 경로 매칭 알고리즘을 적용하여 노드간의 대응관계를 생성하고, 생성된 대응관계에 하향식 너비 우선 탐색을 적용하여 변화를 추출한다.

특히 제안된 경로 매칭 알고리즘은 경로에 기반하기 때문에 노드간의 대응 여부를 모두 조사하는 기존 연구와 비교하여 의미 있는 대응관계를 보다 빠르게 생성하였다. 특히 본 논문에서는 구조적인 변화를 추출하기 위하여 대응관계의 유사도 개념을 새롭게 제안하였다. 이에 제안된 방법론은 구조화된 문서간의 삽입, 삭제, 그리고 갱신의 단순한 변화는 물론이고 이동과 복사의 구조적으로 보다 의미 있는 변화를 탐지하였다.

한편 제안된 방법은 SGML/XML 문서의 저장과 버전 관리 등의 다양한 분야에 적용이 가능하다. 특히 제안된 알고리즘을 SGML/XML 데이타베이스에 적용한다면 문서를 모두 저장할 필요가 없기 때문에 저장 효율이 증대할 것이다. 또한 각 버전간의 의미 있는 차이를 유지하기 때문에 변화에 대한 검색을 지원할 수 있을 것이다. 한편 데이타베이스로부터 구 버전의 문서를 복구하기 위해서는 추출된 변화 연산에 대한 역 연산을 계산하여야 한다. 향후 본 연구에서는 제안된 변화 연산에 대한 역 연산을 제안하고, 이를 SGML/XML 문서 데이타베이스에 적용할 계획이다.

참 고 문 헌

[1] ISO/IEC 8879, *Information Processing - Text and Office Systems - Standard Generalized Markup Language (SGML)*. International Organization for Standardization, 1986.

[2] W3C Recommendation REC-xml-19980210, *Extensible Markup Language (XML) 1.0*. World Wide Web Consortium, 1998. http://www.x3c.org/TR/1998/REC-xml-19980210.

[3] Mil-M-28001A, *Markup requirements and generic style specification for electronic printed output and exchange of text*. Department of Defense CALS Office, Sep. 1990.

[4] H. Brown, Standards for Structured Documents, *The Computer Journal*, vol. 32, no. 6, pp.505-514, Jun. 1989.

[5] Y. Marcoux and M. Sevingny, Why SGML? Why Not?, *Journal of the American Society for Information Science*, vol. 48, no. 7, pp.584-592, Jul. 1997.

[6] A. Sengupta and A. Dillon, Extending SGML to Accommodate Database Functions: A Methodological Overview, *Journal of American Society For*

Information Science, vol. 48, no. 7, pp.629-637, Jul. 1997.
[7] A. Bruggemannklein and D. Wood, The Validation of SGML Content Models, *Mathematical & Computer Modelling*, vol. 25, no. 4, pp.73-84, 1999.
[8] C. F. Goldfarb, *The SGML Handbook*. Oxford: Clarendon Press, 1990.
[9] C. F. Goldfarb and P. Prescod, *The XML Handbook*. Upper Saddle River, NJ: Prentice Hall, 1998.
[10] A. Haake, CoVer: A Contextual Version Server for Hypertext Applications, *Proc. Fourth ACM Conf. Hypertext*, pp.43-52, Milan, Italy, Nov. 1992.
[11] K. Osterbye, Structural and Cognitive Problems in Providing Version Control for Hypertext, *Proc. Fourth ACM Conf. Hypertext*, pp.33-42, Milan, Italy, Nov. 1992.
[12] H. Moeller, Versioning Structured Technical Document, *Proc. Workshop on Versioning in Hypertext Systems (held in connection with ECHT '94)*, Edinburgh, Sep. 1994.
[13] S. J. Yoo, P. B. Berra, Y. K. Lee, and K. Yoon, Version Management in Structured Document Retrieval System, *Proc. Eighth Int'l Conf. Software Engineering and Knowledge Engineering*, pp.537-544, Lake Tahoe, Nevada, Jun. 1996.
[14] M. A. Noronha, L. G. Golendziner, and C. S. D. Santos, Extending a Structured Document Model with Version Control, *Proc. Int'l Database Engineering & Applications Symposium*, pp.234-243, Jul. 1998.
[15] W. Labio and H. G. Monila, Efficient Snapshot Differential Algorithms for Data Warehousing, *Proc. Twentieth Conf. Very Large Data Bases*, pp.63-74, Bombay, India, Sep. 1996.
[16] J. Widom and S. Ceri, *Active Database Systems: Triggers and Rules for Advanced Database Processing*. San Francisco, CA: Morgan Kaufmann, 1995.
[17] R. Wagner and M. Fischer, The String-to-String Correction Problem, *Journal of the Association of Computing Machinery*, vol. 21, no. 1, pp.168-173, Jan. 1974.
[18] R. Wagner, On the Complexity of the Extended String-to-String Correction Problem, *Proc. Seventh ACM Symposium on the Theory of Computation*, 1975.
[19] E. Myers, An O(ND) Difference Algorithm and Its Variations, *Algorithmica*, vol. 1, no. 2, pp.251-266, 1986.
[20] S. Wu, U. Manber, and C. Myers, An O(NP) Sequence Comparison Algorithm, *Information Processing Letters*, vol. 35, pp.317-323, Sep. 1990.
[21] W. Labio and H. G. Monila, Efficient Snapshot Differential Algorithms for Data Warehousing, *Proc. Twentieth Conf. Very Large Data Bases*, pp.63-74, Bombay, India, Sep. 1996.
[22] K. Zhang and D. Shasha, Simple Fast Algorithms for the Editing Distance between Trees and Related Problems, *SIAM Journal on Computing*, vol. 18, no. 6, pp.1245-1262, 1989.
[23] D. Shasha and K. Zhang, Fast Algorithms for the Unit Cost Editing Distance between Trees, *Journal of Algorithms*, vol. 11, pp.581-621, 1990.
[24] K. Zhang, "Algorithms for the Constrained Editing Distance between Ordered Labeled Trees and Related Problems," *Pattern Recognition*, vol. 28, no. 3, pp.463-474, Mar. 1995.
[25] G. J. S. Chang, G. Patel, L. Relihan, and J. T. J. Wang, "A Graphical Environment for Change Detection in Structured Documents," *Proc. Twenty-First Annual Int'l Computer Software and Applications Conference (COMPSAC'97)*, pp.536-541, Los Alamitos, CA, Aug. 1997.
[26] J. T. L. Wang, D. Shasha, G. J. S. Chang, L. Relihan, K. Zhang, and G. Patel, Structural Matching and Discovery in Document Databases, *Proc. ACM SIGMOD Int'l Conf. Management of Data*, pp.560-563, Tucson, AZ, May 1997.
[27] S. Chawathe, A. Rajaraman, H. G. Molina, and J. Widom, Change Detection in Hierarchically Structured Information, *Proc. ACM SIGMOD Int'l Conf. Management of Data*, pp.493-504, Montreal, Canada, Jun. 1996.
[28] S. Chawathe and H. G. Molina, Meaningful Change Detection in Structured Data, *Proc. ACM SIGMOD Int'l Conf. Management of Data*, pp.26-37, Tucson, AZ, May 1997.

이 경 호
1995년 2월 연세대학교 전산과학과 졸업(이학사). 1997년 2월 연세대학교 컴퓨터과학과 졸업(공학석사). 2001년 2월 연세대학교 컴퓨터과학과 졸업(공학박사). 2001년 4월 ~ 현재 미국 국립표준기술연구소(National Institute of Standards and Technology) 객원연구원. 관심분야는 멀티미디어, XML/SGML, 문서 공학(Document Engineering)

변 창 원

1997년 2월 연세대학교 컴퓨터과학과 졸업(공학사).1999년 8월 연세대학교 컴퓨터과학과 졸업(공학석사). 1999년 9월 ~ 현재 (주)프리챌 근무. 관심분야는 멀티미디어, XML/SGML

최 윤 철

1773년 서울대학교 전자공학과 졸업(공학사). 1975년 6월 Univ. of Pittsburgh (공학석사). 1979년 6월 Univ. of California, Berkeley, Dept. of IE/OR (공학박사). 1979년 8월 ~ 1982년 7월 Lockheed 사 및 RockwellInternational 사 책임연구원. 1982년 9월 ~ 1784년 1월 Univ. of Washington 전산학과 박사과정. 1990년 9월 ~ 1992년 1월 Univ. of Massachusetts 연구교수. 1984년 3월 ~ 현재 연세대학교 컴퓨터과학과 교수. 관심분야는 멀티미디어, 컴퓨터 그래픽스, 가상 현실, 지리정보시스템

고 견

1987년 2월 연세대학교 전산과학과 졸업(이학사). 1989년 2월 연세대학교 수학과 (전산전공) 졸업(이학석사). 1993년 9월 일본 동경대학교 공학부 졸업(공학박사). 1995년 3월 ~ 현재 청주대학교 컴퓨터정보공학과 교수. 관심분야는 컴퓨터비젼, 그래픽스, 가상현실 등.