메시지 다이제스트를 이용한 구조화된 문서의 변화 탐지

김동아[0] 이석균
단국대학교 정보컴퓨터학부
love89@softconsult.co.kr[0] sklee@dankook.ac.kr

Detecting Changes in Structured Documents using Message Digest

Dong-Ah Kim[0] Suk Kyoon Lee
Dept. of Information and Computer Science, Dankook University

요 약

XML/HTML 문서와 같이 트리 구조로 표현되는 데이터의 변화 탐지는 NP-hard의 문제로 이에 대한 효율적인 구현은 매우 중요하다. 본 논문에서는 효율적인 변화 탐지를 위해 트리 구조의 데이터를 X-tree로 표현하고 이에 기초한 휴리스틱 알고리즘을 제안한다. X-tree에서는 모든 서브트리의 루트 노드에 서브트리의 구조와 소속 노드들의 데이터들을 128비트의 해시값으로 표현하여 저장함으로 신·구 버전의 X-tree들에 속한 서브트리들의 비교가 매우 효율적이다. 제시한 변화 탐지 알고리즘에서는 구 버전의 X-tree의 모든 서브트리들에 대해 신 버전의 X-tree에서 동등한 서브트리들을 찾고, 이들에 기초하여 이동 연산이 발생한 서브트리들과 갱신 연산이 발생한 서브트리들을 순차적으로 찾는다. 이때 이동 연산과 갱신 연산으로 대응되는 서브트리는 동등 서브트리로부터 루트 노드로 대응 관계를 확장하는 가운데 발견된다. 이후 깊이 우선으로 검색하면서 나머지 노드들을 대응시킨다. X-tree의 구조적 특징에 기인하여 노드들 간의 비교를 통해 대응 여부를 검사하는 대부분의 기존 연구와는 달리 서브트리의 비교를 통해 대부분의 대응 관계를 결정하므로 효율적인 변화 탐지가 가능하다. 본 알고리즘은 최악의 경우에서도 N을 신·구 버전 문서의 전체 노드 수라 할 때 $O(N)$의 시간 복잡도를 갖는다.

1. 서론

현재 많은 정보시스템들이 웹을 기반으로 구축·활용되고 있어 다양한 정보들이 웹사이트를 통해 제공되고 있다. 이러한 환경 하에서, 웹 사이트에서 제공되는 다양한 정보에 대한 변화를 실시간으로 탐지하고 그 추이를 관찰하는 기술에 대한 요구가 증가되고 있다. 가령, 경쟁사의 기술 동향, 제품 가격 등의 다양한 정보들을 웹 문서를 통해 모니터링 하고자 할 경우 웹 문서의 변화를 적절히 탐지하여 사용자에게 제공하는 기술은 매우 유용하다.

XML/HTML 문서로 표현되는 대부분의 웹 문서는 구조화된 문서(hierarchical structured document)로 이에 대한 변화의 탐지는 NP-hard에 속하는 문제이다[1,2]. 따라서 본 논문에서는 순서 트리로 표현된 XML 문서간의 변화를 효율적으로 탐지할 수 있는 휴리스틱 알고리즘을 제시한다. 단, HTML 문서의 경우 XML 문서로 변환하여 본 알고리즘을 적용한다.

신·구 버전 문서의 변화는 일반적으로 구 버전 문서를 신 버전 문서로 변환하는 일련의 편집 연산인 편집 스크립트로 나타낸다[1,3,4,5,6,7]. 구 버전 문서의 한 노드에 편집 연산이 적용되어 신 버전 문서의 한 노드가 되는 경우, 이들 두 노드는 서로 대응한다고 한다. 구조화된 문서의 변화 탐지는 트리로 표현된 두 문서간에 서로 대응되는 노드들을 결정하고, 이에 따른 편집 연산을 찾아 편집 스크립트를 생성하는 과정으로 두 문서의 노드들간 대응 여부를 결정하는 것이 알고리즘의 핵심이다. 제시된 알고리즘은 효율적인 변화 탐지를 위해 노드 단위의 대응 관계를 찾기에 앞서 트리 단위의 대응 관계를 먼저 찾는다. 이는 신·구 버전 문서간에 존재하는 동등 서브트리(equivalence subtree)를 찾아 동등 서브트리에 속한 노드들을 일괄 대응시키는 방법으로, 모든 노드에 대해 대응 여부 검사가 이루어지는 대부분의 기존 연구들에 비해 빠르고 효율적으로 변화를 탐지한다. 특히, 변화가 적고 자주 바뀌는 웹 문서에서는 더욱 효율적이다.

한편 동등 서브트리를 찾기 위해 서브트리의 구조와 더불어 서브트리에 속한 노드들의 정보를 나타내는 해시값을 제안한다. 이는 메시지 변조 방지(integrity)에 사용되는 해시 알고리즘인 MD4(message digest algorithm 4)[8]를 사용하여 계산된 값이다. MD4는 임의 길이 문자열(메시지)에 대해 128비트의 해시값(MD : message digest 혹은 fingerprint라 함)을 반환하는 해시 함수로, 서로 다른 메시지가 입력으로 주어졌을 때 서로 같은 해시값을 얻을 확률이 극히 적은 (computationally infeasible) 특성을 갖고 있다[9].

본 논문의 구성은 다음과 같다. 2장에서는 문서 변화 탐지에 대한 기존 연구를 살펴보고, 3장에서는 본 논문에서 사용하는 변화 탐지 모델을 설명한다. 4장에서는 본 알고리즘의 처리 단계를 구분하고 각 단계에 대해 설명한다. 마지막 5장에서는 실험 결과 및 결론을 기술한다.

2. 관련 연구

변화 탐지에 대한 기존 연구들은 MH-DIFF[1], ATBE[3], SCD[6] 그리고 BULD Diff[5] 등과 같이 계층적 데이터에 대한 변화 탐지 알고리즘과 관계형 데이터에 대한 변화 탐지 알고리즘[10], 그리고 Unix diff 유틸리티와 같이 문자열에 대한 변화 탐지 알고리즘으로 분류된다.

본 논문은 계층적 데이터에 대한 변화 탐지 알고리즘으로 최근 발표된 BULD Diff 알고리즘을 개선하였다.

BULD Diff 알고리즘은 신·구 버전 문서의 모든 노드를 방문하면서 방문한 노드를 루트 노드로 하는 서브트리를 32비트 해시값으로 나타내고 이를 서브트리의 노드 개수를 나타내는 가중치와 함께 서브트리의 루트 노드에 저장한다. 한편 구 버전 문서에서 계산된 32비트 해시값을 맵 M에 등록하고, 신 버전 문서의 노드를, 가중치가 가장 큰 노드부터 방문하면서 방문한 노드의 서브트리 해시값이 맵 M에 등록되어 있는지를 검사한다. 이때, 동일한 서브트리 해시값을 갖는 구 버전 문서의 노드들을 후보(candidates)라 하고 가중치가 큰 순으로 이들 후보를 정렬한 순서 큐(ordered queue)를 이용하여 최적의 대응을 찾는다. 그러나 이러한 후보들의 정렬 문제로 인해, n을 두 문서의 파일 크기라 할 때 최악의 경우 $O(n*\log(n))$의 비용이 든다. 단, 노드의 개수는 n보다 작거나 같다.

제시된 알고리즘에서는 서론에서 밝힌 바와 같이 MD4의 특성으로 서로 다른 서브트리가 서로 같은 서브트리 해시값을 갖을 확률은 극히 적다. 또한, 동일 트리 내에 동일한 서브트리 해시값이 존재할 경우, 대응이 결정된 부모 노드를 이용하여 대응 여부를 결정하도록 하여 BULD Diff 알고리즘의 후보 정렬 문제를 없앴다. 이로 인해 제시된 알고리즘은 N을 신·구 버전 문서의 전체 노드 수라 할 때 $O(N)$의 시간 복잡도를 갖는다.

3. 변화 탐지 모델

본 절에서는 XML 문서의 데이터를 계층적 구조로 표현하기 위해, 그리고 트리 구조로 표현된 XML 문서의 변화를 탐지하기 위한 알고리즘의 기본 데이터 구조로서 X-tree를 제안한다. 또한 X-tree에서 변화 탐지에 사용되는 핵심 개념인 노드 MD(message digest)와 트리 MD에 대해 소개하고, X-tree를 통해 표현된 XML 문서의 변화를 나타내는 편집 연산과 편집 스크립트를 설명한다.

3.1 X-tree

X-tree는 XML 문서의 특성을 반영한 일종의 순서 트리(ordered tree)이다. X-tree 노드는 그림 1과 같이 8개의 필드들로 구성되며, 이들 중 Label, Type과 Value 필드들은 XML 문서의 표현을 위해, Index 필드는 동일한 Label 필드 값을 갖는 형제 노드들을 구별하기 위해, 그리고 nMD, tMD, nPtr, Op 필드들은 변화 탐지 알고리즘에서 사용된다.

Label	Type	Value	Index	nMD	tMD	nPtr	Op

그림 1 X-tree 노드 데이터의 구성

우선 X-tree를 통한 XML 문서의 표현 방법을 살펴보자. X-tree의 노드는 XML 문서의 엘리먼트(Element)나 문자 데이터(PCData)를 나타내며 Type 필드는 X-tree의 노드가 나타내는 데이터가 엘리먼트인지 문자 데이터인지를 구별한다. Label 필드는 엘리먼트일 경우 엘리먼트 명을, 문자 데이터의 경우에는 #TEXT라는 키워드를 저장한다. Value 필드는 엘리먼트 경우 엘리먼트에 속한 어트리뷰트(attribute)에 대한 정보를, 문자 데이터의 경우에는 문자 데이터의 내용을 저장한다. 각 노드의 Index 필드에는 동일한 Label 필드 값을 갖는 형제 노드들이 존재하는 경우 왼편부터 순서에 따라 1, 2, 3 ... 의 숫자를 부여하며, 이들이 존재하지 않는 경우 디폴트로 1을 부여한다. 이후 설명의 편이를 위해 노드의 필드 명은 함수처럼 사용한다.

X-tree 노드는 Label 필드 값과 Index 필드 값을 통해 표현되며 이를 ILabel(indexed label)이라 하고, 임의 노드 N_p의 ILabel은 Label (N_p)[Index(N_p)]로 표현된다.

한편, 본 알고리즘에서는 X-tree의 각 노드를 구별하기 위해 nID(node identifier)를 제안한다. nID는 X-tree T의 루트 노드 N_r에서 임의 노드 N_p까지의 경로(path) 상의 ILabel의 결합으로, nID(N_p)= .Label[N_r](Index(N_r))/,...,.Label(N_p)[Index(N_p)]로 표현된다. 이때 nID는 경로상의 노드들의 Index와 Label의 결합을 통해 표현하게 되므로, X-tree의 모든 노드는 nID를 통해 유일하게 식별된다.

3.2 노드 MD(Message Digest)와 트리 MD

XML 문서에 대한 변화 탐지는 구 버전의 X-tree T_{old}와 신 버전의 X-tee T_{new} 사이의 변화 탐지를 의미하며, 이들 T_{old}와 T_{new}에 속한 각 노드들을 비교함에 있어 트리 구조를 얼마나 효과적으로 이용하는가는 매우 중요한다. 본 절에서는 이를 위해 노드 MD와 트리 MD 개념을 설명한다.

노드 MD는 노드에 저장되어 있는 정보에 대한 해시값이고 트리 MD는 트리의 구조와 트리에 속한 모든 노드들의 정보를 나타내는 해시값이다. 임의 노드에 대한 노드 MD는 그 노드의 nMD 필드에 저장되며, 임의 트리에 대한 트리 MD는 그 트리의 루트 노드의 tMD 필드에 저장된다.

노드 MD : 노드 N_p의 MD는 Label 필드 값과 Value 필드 값에 대해 MD4 알고리즘을 적용하여 얻은 128비트 해시값을 32바이트 문자열로 표현한 값이다. 이를 nMD를 사용하여 정의하면 다음과 같다.

nMD(N_p) = string(MD4(Lable(N_p) ⊕Value(N_p)))

string(h) : 128비트 해시값 h를 입력받아 16진수로 표기한 32바이트 문자열 반환 함수

⊕ : 문자열 결합 연산자

트리 MD : 트리 T에 대한 MD는 T의 루트 노드 N_r의 노드 MD와 N_r의 자식 노드들을 각각 루트로 하는 서브트리들의 MD를 통해 재귀적으로 정의된다. 이는 tMD를 통해 다음과 같이 정의된다.

tMD(N_r) = string(MD4(nMD(N_r) $\overset{n}{\underset{p=1}{\oplus}}$ tMD(N_p)))

tMD(N_p) : N_r의 p번째 자식 노드 N_p의 tMD 반환 함수

n : 자식 노드의 개수

그림 2는 트리 MD와 노드 MD의 예를 보이고 있다. 그림 2의 tMD(Title[1])은 nMD(Title[1])과 tMD(#TEXT[1])를 기반으로 정의되고 tMD(#TEXT[1])는 더 이상 자식 노드가 없으므로 nMD(#TEXT[1])에 의해 정의됨을 보이고 있다. 이와 같이 트리 MD 즉, 루트 노드의 tMD는 노드 자체의 Label 필드는 물론 Value 필드의 값과 자식 노드들의 tMD들을 포함하여 계산되므로 트리의 구조와 트리에 속한 모든 정보를 반영하게 된다. 따라서 T_{old}와 T_{new}에서 동일한 tMD를 갖는 노드 N_r^{old}과 N_r^{new}이 발견되면, 이들을 루트 노드로 하는 서브트리 ST_{old}와 ST_{new}는 구조는 물론 이에 속한 모든 노드들의 정보도 같게 된다. 이때 트리 ST_{old}와 ST_{new}는 동등(equivalence)하다고 한다.

3.3 편집 연산(Edit Operation)과 편집 스크립트(Edit Script)

트리 T에 대한 편집 연산은 노드에 대한 삭제, 삽입, 갱신 연산과 서브트리에 대한 이동 연산으로 구성된다. 각각의 편집 연산은 다음과 같다.

$DEL(nID(N_p))$: 트리 T의 노드 N_p을 삭제한다. 단, XML 문서의 특성을 고려하여 X-tree의 루트 노드는 삭제할 수 없다고 가정한다.

$INS(l, v, nID(M_q), i)$: Label이 l이고 Value가 v인 노드를, 노드 M_q의 i번째 자식 노드로 삽입한다.

$UPD(nID(N_p), v_1, v_2)$: 트리 T에 속한 노드 N_p의 $Value(N_p)$을 v_1에서 v_2로 바꾼다.

$MOV(nID(N_p), nID(M_q), i)$: 트리 T에서 노드 N_p을 루트 노드로 하는 서브트리를 노드 M_q의 j번째 자식 노드로 이동시킨다.

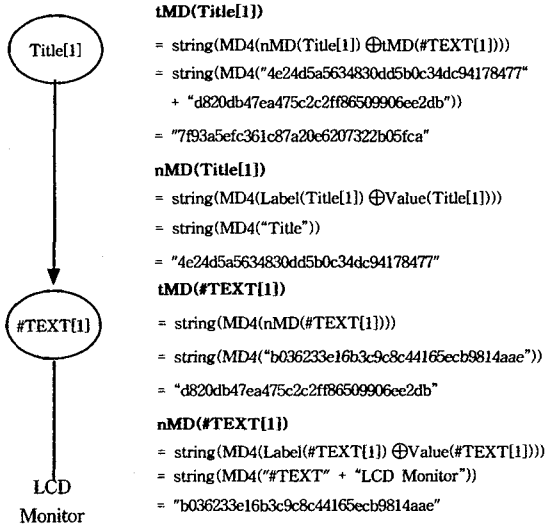

그림 2 nMD와 tMD

편집 연산은 아니지만 노드에 아무런 변화가 일어나지 않았음을 표현하기 위해 NOP 연산을 사용한다. 한편, X-tree T_{i-1}에 하나의 편집 연산 e_i가 적용된 결과를 T_i라 하면, T_{i-1}이 편집 연산 e_i에 의해 T_i가 되었다고 하고, 이를 $T_{i-1} \xrightarrow{e_i} T_i$로 표현한다.

편집 스크립트는 구 버전의 문서를 신 버전의 문서로 변환하는데 필요한 일련의 편집 연산이다. 즉, X-tree $T_0, T_1, \ldots, T_k$와 일련의 편집 연산(단, $T_{i-1} \xrightarrow{e_i} T_i$)이 주어지면, 일련의 편집 연산 $e_1, e_2, \ldots, e_k$를 편집 스크립트라 하고 $T_0 \xrightarrow{\varepsilon} T_k$ (단, $\varepsilon = e_1, e_2, \ldots, e_k$)로 표현한다. 변화 탐지의 시각에서는 T_0는 T_{old}로, T_k는 T_{new}로 볼 수 있으며 $T_0, T_1, \ldots, T_k$들은 변화 과정 중의 임시 트리들로 이해할 수 있다.

4. 변화 탐지 알고리즘

XML 문서의 변화 탐지는 트리로 표현된 구 버전 문서 T_{old}와 신 버전 문서 T_{new}에서, T_{old}의 임의 노드와 대응되는 노드를 T_{new}에서 찾고 이에 적용할 편집 연산을 결정하는 과정을 통해 이루어진다. 한편 T_{old}의 임의 노드에 대해 대응되는 T_{new}의 노드에 대해서 변화 여부 또는 그 성격에 따라 NOP, UPD나 MOV 연산이, T_{new}의 노드에 대해 대응되는 T_{old}의 노드가 없을 경우는 INS 연산이, 반대의 경우는 DEL 연산이 적용된다. 이때 X-tree 노드의 Op 필드에는 적용 연산이, 그리고 nPtr 필드에는 대응되는 노드의 포인터가 저장된다.

그러나, T_{new}의 하나의 노드에 대해 대응 가능한 T_{old}의 노드들이 다수 존재할 수 있으며, 이때 어떤 노드의 쌍을 대응하는 것으로 결정하느냐에 따라, 그리고 이때 어떤 연산을 적용하는가에 따라 다양한 편집 스크립트가 생성될 수 있다. 본 논문에서는 X-tree T_{old}와 T_{new} 사이의 최소 비용의 편집 스크립트를 찾지 않는다. 이는 비용이 클 뿐 아니라 변화 탐지라는 본 논문의 목적에 부합하지 않는다. 따라서, 본 논문에서는 XML 문서의 신·구 버전들 간에 변화가 발생했는지 여부를 확인하고, 변화 발생 시 어느 노드에서 발생했는지를 빠르게 찾고자 함을 목적으로 한다.

본 알고리즘은 T_{old}와 T_{new}에서 모든 동등 서브트리의 쌍들을 찾는 것으로부터 시작된다. 이를 위한 준비 단계로 두 트리의 모든 노드들에 대해 nMD와 tMD를 계산한다. 이는 각 노드의 데이터 비교는 물론 그 노드를 루트로 하는 서브트리들의 비교를 가능하게 한다.

그림 3은 본 알고리즘의 단계별 처리 노드를 나타내고 있다.

1 단계(동등 서브트리의 발견) : T_{old}와 T_{new}에서 모든 동등 서브트리의 쌍 (ST_{old_i}, ST_{new_j})을 찾아 각 노드들을 NOP로 대응시킨다. 이때 동등 서브트리의 쌍에 속한 모든 노드들은 대응 확정되었다고 한다. 이 과정에서는 T_{new}의 모든 노드들의 tMD 값을 키로 해시테이블에 등록하고 T_{old}를 깊이 우선 탐색(Depth first Search)으로 검색하며 해시테

이를에 해당 tMD가 존재하는지 여부를 확인한다. 이때 tMD의 정의에 따라 동등 트리의 발견시 그의 서브트리에 대한 탐색은 계속하지 않는다. 그림 3에서 노드 T_{old}와 T_{new}의 노드 c[1]을 루트 노드로 하는 서브트리는 동일한 tMD를 갖게 되며, 이에 속한 모든 노드들은 NOP로 대응 확정된다. 한편, T_{new}의 노드들을 해시테이블에 등록할 때 tMD 값이 중복되는 엔트리들은 모두 삭제한다. 이를 동일 tMD 노드의 배제 원칙이라고 하며 이는 대응 관계의 결정이 불분명한 노드들을 일단 고려 대상에서 제외시킴을 의미한다. 그림 3에서 볼 수 있듯이 T_{new}의 .c[1].h[1]과 .c[1].b[1].h[1]은 동일 tMD 노드의 배제 원칙에 의해 대응 확정을 유보한다.

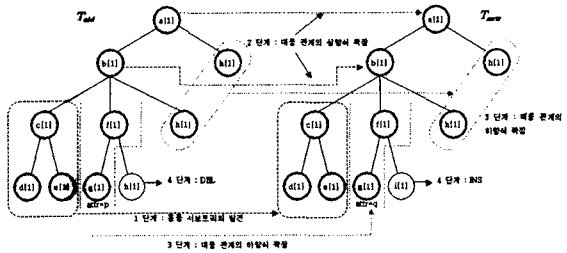

그림 3 단계별 처리 노드

2 단계(대응 관계의 상향식 확장) : 현재까지 NOP나 UPD로 대응이 확장된 노드 쌍으로부터 루트 노드로의 경로 상에 있는 노드들에 대해 점진적으로 대응 여부를 결정하는 단계이다. 우선 1단계에서 발견된 동등 서브트리 쌍의 각 루트 노드의 부모 노드들의 대응 여부를 고려한다. 두 루트 노드들을 각각 A, B라하고 이들의 부모 노드들을 각각 pA, pB라 할 때, a) 두 부모 노드들(pA, pB)의 nMD 값(Label과 Value 값)이 같으면 그 부모 노드들(pA, pB)을 NOP로, b) 두 부모 노드들의 nMD 값은 서로 다르지만 Label 값이 같은 경우는 그 부모 노드들(pA, pB)을 UPD로 대응시킨다. 한편, c) 두 부모 노드들(pA, pB)의 nMD 값과 Label 값이 다른 경우에는 A를 루트로 하는 서브트리가 pB로 MOV됨을 의미한다. 이때 노드 A와 B를 MOV로 대응시킨다.

본 단계는 1단계에서 발견된 동등 서브트리 쌍의 루트로부터 시작되었지만 위의 a)와 b)의 경우에서 대응이 확정된 노드들의 쌍에 대해 반복 적용된다. 본 단계를 통해 그림 3의 T_{old}와 T_{new}의 노드 b[1]과 c[1]이 NOP로 대응 확정된다.

3 단계(대응 관계의 하향식 확장) : 대응이 확정된 노드의 자식 노드들의 대응 여부를 결정하는 단계로 루트로부터 하향식으로 진행된다. T_{old}에서 대응이 확정된 임의 노드를 A, A의 T_{new}의 대응 노드를 B, 그리고 A에는 대응이 결정되지 않은 자식 노드들(cA[1...s])이 있다고 가정하자. B의 대응이 결정되지 않은 자식노드들을 cB[1...t]라고 할 때, 그림 4의 알고리즘을 수행한다.

```
For j in [1..t]  /* 각 cB[j]에 대해 */
 if cA[1..s]에 tMD(cB[j])와 같은 tMD 값의 노드 cA[i]가 있다면,
    cA[i]와 cB[j]를 루트 노드로 하는 서브트리에 속한
    모든 노드를 각각 NOP으로 대응시킨다.
 else if
 cA[1..s]에 ILabel(cB[j])와 같은 ILabel 값의 노드 cA[i]가 있다면,
    if cA[i]와 cB[j]의 nMD가 같다면,
       cA[i]와 cB[j]를 NOP으로 대응시킨다.
    else
       cA[i]와 cB[j]를 UPD로 대응시킨다.
```

그림 4 대응 관계의 하향식 확장 알고리즘

그림 4의 첫 if 절은 1단계에서 동일 tMD 노드의 배제 원칙에 의해 대응이 유보되었던 노드들에 대한 처리를 의미한다. else if 절은 cB[j]와 같은 Label과 Index를 갖는 A의 자식 노드를 cB[j]의 대응 노드로 선택한다는 의미이다. cA[1...s]의 tMD와 ILabel은 모두 해시테이블에 등록하고 이를 비교시 사용한다. 그림 4의 알고리즘은 T_{old}의 루트 노드로부터 깊이 우선 탐색하면서 적용된다. 이때 T_{old}와 T_{new}의 루트 노드는 XML 문서의 특성에 의해 항상 대응되는 것으로 가정한다. 따라서 본 단계는 T_{old}의 루트 노드로부터 점진적으로 대응 관계를 확장해 나간다고 볼 수 있다.

4 단계(DEL, INS 연산의 확정과 편집 스크립트 생성) :
본 단계는 1~3 단계 과정을 통해 대응된, 혹은 대응되지 않은 노드들로부터 편집 연산을 추출하여 편집 스크립트를 생성한다.
(1) 깊이 우선 검색을 통해 T_{new}의 모든 노드를 방문하면서 INS, MOV, UPD 연산을 생성한다. 이때 대응이 확정되지 않은 노드로부터 INS 연산을, Op가 MOV와 UPD인 노드로부터 각각 MOV, UPD 연산을 생성한다. (2) T_{old}을 깊이 우선 검색을 하면서 대응이 확정되지 않은 노드로부터 DEL 연산을 생성한다. (3) 편집 연산 생성 순서에 따라 편집 스크립트를 생성하고 파일에 저장한다.

4. 실험 결과 및 결론

본 논문에서는 트리로 표현된 XML 문서간의 변화를 탐지할 수 있는 효율적인 알고리즘과 문서의 변화를 효율적으로 표현할 수 있는 편집 연산과 편집 스크립트를 보였다. 한편, 본 논문에서는 구체적으로 비용을 명시하지 않았으나 적절한 자료구조의 활용으로 신·구 버전 문서의 전체 노드 개수를 N이라 할 때 $O(N)$의 시간 복잡도를 갖는다. 본 논문에서 제시된 알고리즘은 현재 윈도우 2000 환경에서 VC++로 구현, WIDS(Web Document Intrusion Detecting System)에 활용되고 있다. 그림 5는 WIDS에서 국내 및 국외의 신문, 방송사 인터넷 페이지 60여 개를 대상으로 실험한 예를 보이고 있다.

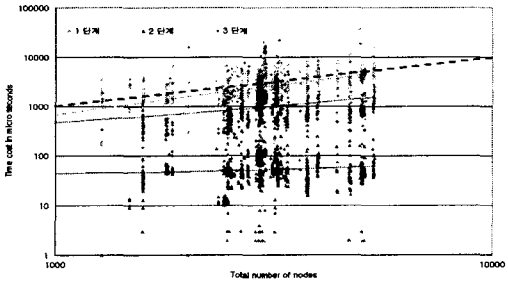

그림 5 실험 결과 예

그림 5에서 볼 수 있듯이 노드 개수에 따른 각 단계별 시간 비용의 추세는 노드 개수의 증가에 따라 단조 증가함을 보이고 있어 본 알고리즘의 시간 복잡도가 $O(N)$임을 실험적으로 증명하고 있다.

현재 WIDS에서 사이트 별로 변화에 대한 패턴을 모으고 있으며 향후 변화에 대한 패턴을 이용하여 변화 탐지에 적용할 수 있는 방안을 연구할 계획이다.

참고 문헌

[1] S. S. Chawathe and H. Garcia-Molina. Meaningful Change Detection in Structured Data. In *Proc. Of SIGMOD '97*, pp.26-37, 1997.

[2] K. Zhang, J. T. L. Wang, and D. Shasha, On the editing distance between undirected acyclic graphs and related problems. In *Proc. of the 6th Annual Symposium on Combinatorial Pattern Matching*, pp.395-407, 1995.

[3] J. T. Wang and K. Zhang, A system for Approximate Tree Matching. *IEEE Transactions on Knowledge and Data Engineering*, 6(4), pp.559-570, August 1994.

[4] J. T. Wang, D. Shasha, G. J. S. Chang, L Relihan, L. Zhang, and G. Patel, Structural Matching and Discovery in Document Databases. In *Proc. Of SIGMOD '97*, pp.560-563, 1997.

[5] S. Abiteboul and A. Marian, Detecting Changes in XML Documents, In *Int'l Conf. on Data Engineering*, 2002.

[6] S. J. Lim and Y. K. Ng, An Automated Change-Detection Algorithm for HTML Documents Based on Semantic Hierarchies, In *Int'l Conf. on Data Engineering*, pp.303-312, 2001.

[7] S. S. Chawathe, Comparing Hierarchical Data in External Memory, In *Proc. Of the 25th VLDB Conf.*, pp.90-101, 1999.

[8] Rivest, R. , The MD4 Message Digest Algorithm, RFC 1320, MIT and RSA Data Security, Inc., April 1992.

[9] N. Doraswamy and D. Harkins, IPSec : The New Security Standard for the Indernet, Intranets, and Virtual Private Networks, Prentice Hall PTR, 1999.

[10] W. Labio and H. G. Monila, Efficient Snapshot Differential Algorithms for Data Warehousing, In *Proc. Of the 20th VLDB Conf.*, pp. 63-74, 1996.