유사성 기반 XML 문서 분석 기법
(XML Document Analysis based on Similarity)

이 정 원 † 이 기 호 ††
(Jung-Won Lee) (Kiho Lee)

요 약 XML 문서가 가지고 있는 태그의 자유로운 정의와 내포된 구조 정보는 정보 검색 및 문서 관리 분야에 많은 이점을 제공할 수 있다. 본 논문은 XML 요소(element)의 의미와 구조 정보를 반영한 문서간의 유사성을 검사할 수 있는 XML 문서 분석 기법을 제시하고자 한다. 도출된 문서간 유사성은 많은 정보 검색 및 마이닝 등의 기초 자료로 사용될 수 있다. 먼저 XML 요소를 시소러스를 이용하여 유사어와 합성어로 구성된 확장-요소 벡터로 확장하고 유사 행렬을 구축하여 요소간 유사성을 판별한다. 또한 오토마타(NFA(Nondeterministic Finite Automata)와 DFA(Deterministic Finite Automata)(를 이용하여 XML 문서의 내포된 구조를 발견하고 최소화 한다. 요소간의 유사 행렬과 최소화된 XML 구조를 이용하여 구조간의 유사성을 판별한다. 본 논문의 XML의 의미를 반영한 유사성 분석 기법은 온라인 서점의 실제 문서의 카테고리를 인식하는 데 있어 100% 정확도를 보였다.

키워드 : XML, 유사성, 오토마타, 순차패턴, 클러스터링

Abstract XML allows users to define elements using arbitrary words and organize them in a nested structure. These features of XML offer both challenges and opportunities in information retrieval and document management. In this paper, we propose a new methodology for computing similarity considering XML semantics - meanings of the elements and nested structures of XML documents. We generate extended-element vectors, using thesaurus, to normalize synonyms, compound words, and abbreviations and build similarity matrix using them. And then we compute similarity between XML elements. We also discover and minimize XML structure using automata(NFA(Nondeterministic Finite Automata) and DFA(Deterministic Finite automata). We compute similarity between XML structures using similarity matrix between elements and minimized XML structures. Our methodology considering XML semantics shows 100% accuracy in identifying the category of real documents from on-line bookstore.

Key workds : XML, similarity, automata, sequential patterns, clustering

1. 서 론

XML은 웹상에서 데이타 표현과 교환의 수단으로 급속히 퍼져 나가고 있다. XML은 반구조적인 데이타(semistructured data)를 명세화하는 언어로 XML 데이타 수집시 스키마 역할을 하는 Document Type Descriptor (DTD)를 갖는다. 그러나 모든 XML 문서는 DTD 존재여부와 상관없이 문서 자체에 암시적인 구조를 내포하고 있다. 이러한 구조를 명시적으로 발견하려는 연구도 활발히 진행되고 있다[1, 2, 3]. XML은 정보검색, 문서관리, 그리고 데이타/텍스트 마이닝 등의 분야에 몇 가지 이점을 제공한다. 예를 들면, XML의 요소(element)는 실제 문서상에서 태그로 나타나고 이러한 요소로 구성된 암시적 구조를 갖는다. 요소는 바로 다음에 오는 컨텐트에 대한 설명이자 타입이 된다. 또한 사용자는 정해진 요소의 집합이 아니라 요소에서 사용될 수 있는 용어를 마음대로 정의할 수도 있다. 이렇게 정의된 XML의 요소는 비구조적인(flat, non-structured) 문서에 나타나는 단어들과는 다른 의미를 갖는다. 즉 XML의 요소는 정보 검색을 위한 인덱스를 구축하거나 웹 데이타를 데이타베이스로 옮기는 스키마의 역할을 할 수 있는 중요한 단서가 되는 것이다.

† 학생회원 : 이화여자대학교 컴퓨터학과
jungwony@ewha.ac.kr
†† 종신회원 : 이화여자대학교 컴퓨터학과
khlee@ewha.ac.kr
논문접수 : 2001년 8월 24일
심사완료 : 2002년 4월 23일

따라서 XML의 요소가 갖는 의미의 중요도와 그 구조를 반영하여 문서를 분석할 수 있는 기술이 필요하다. 이를 반영한 문서 분석을 위해서는 다음과 같은 점을 고려해야 한다. 첫째, 단어가 위치에 따라 그 중요도가 달라진다. 즉, 요소에서 사용된 단어는 컨텐트에 나타난 단어보다 그 문서를 설명하는 데 있어 더 많은 의미를 갖는다. 둘째, 요소를 유사어, 합성어, 약어를 이용하여 정의할 수도 있다. XML 문서 작성자는 자유로이 요소를 정의할 수 있기 때문에 같은 의미로 사용된 단어들을 구별할 수 있어야 한다. 셋째, 단어의 실제 의미가 구조와 관계가 있다. 넷째, 요소들은 하나의 구조를 이루고 이 구조는 문서를 대표할 수 있다. 문서 전체를 살펴 보지 않더라도 요소만으로도 그 문서를 분류하는 데 도움을 줄 수 있다.

본 논문은 XML 문서에 나타나는 단어의 의미를 차별화 하고 구조 정보를 추출하여 유사성을 비교할 수 있는 방법을 제안하고자 한다. XML의 요소와 구조간의 유사성을 기반으로 문서를 분석하여 XML을 사용하는 응용 분야의 기초를 제공할 수 있다. 순차 패턴(sequential patterns) 마이닝 알고리즘을 이용하여 문서 구조간의 유사성을 검사한다. 최소화된 문서 구조를 추출하기 위하여 오토마타 즉, 비결정적 오토마타(NFA: Nondeterministic Finite automata)와 결정적 오토마타(DFA:Deterministic Finite Automata)를 이용한다. 그리고 구조를 이루는 각각의 요소들은 유사어, 합성어 그리고 약어로 구성된 벡터로 확장된다. 생성된 확장-요소 벡터간의 유사 행렬을 구축하여 요소간의 유사성을 판별한다.

2. 관련 연구

반구조적인 데이타로부터 구조를 발견하려는 몇몇 연구가 있다. [1]은 RDBMS에서 반구조적인 데이타를 질의, 저장, 관리하기 위해 스키마를 생성하는 알고리즘을 제안하고 있다. 이는 주어진 반구조적인 데이타 인스턴스를 관계형 모델로 매핑하기 위해 두 가지의 비용 함수-저장과 질의 비용-를 고려한 최적화 문제에 집중되어 있다. 그러나 본 논문에서는 RDB로의 매핑이 아닌 XML 요소의 유사성을 기반으로 문서의 구조를 발견할 필요가 있다. [2]는 반구조적인 데이타의 근사 타이핑 (*approximate typing*)을 위한 알고리즘을 제시하고 있다. 타이핑 문제는 결국 문서의 구조를 발견하는 일을 기반으로 한다. 최적의 타이핑을 찾는 일은 NP-hard 문제이므로 최소의 구조 정보만을 추출하고 나머지 손실 정보를 분리하여 저장한다는 아이디어이다. 그러나 XML 문서간의 유사성을 검사하기 위한 구조 발견은 여러 문서들이 반드시 하나의 구조로 타이핑 될 필요가 없고 각각의 문서에 대해 대표할 수 있는 구조를 손실없이 추출해야 한다. [3]은 연관 규칙을 사용하여 빈도수가 높은 문서 트리의 경로를 찾는 것이다. 그러나 이 연구는 한 문서내의 구조정보를 다룬 것이 아니라 여러 문서를 계층적으로 링크하고 있는 링크 정보를 대상으로 한 것이다. [4]는 훈련을 통하여 반자동으로 구조를 발견하는 방법을 제안하고 있다. 사용자가 먼저 파일을 흥미 있는 영역으로 분할하면 분할 정보를 파싱하여 구조를 발견한다. 그러나 사용자가 분할 정보를 사전에 입력하게 하여 이를 훈련한다는 것은 큰 제약이 된다.

발견된 구조에서 매칭되는 패턴을 찾는 문제는 컴파일러의 최적화 과정에서 트리 패턴 매칭 문제로 오랫동안 연구 되어 왔다[5]. 그리고 반구조적인 데이타의 스키마를 발견하는 연구에서도 연관 규칙을 사용하여 빈번히 발생하는 경로를 추출하는 방법이 있다[3]. 그러나 여러 문서들 사이의 구조를 비교하기 위해서는 두 가지 연구들을 그대로 이용하기 어렵다. 트리 패턴 매칭 알고리즘은 트리의 일부라고 하더라도 하부 표현 요소(subexpression)의 교체(replacement)를 목적으로 하기 때문에 완전 매칭(exact matching)이어야 한다. 즉 a-b-c-d와 같은 경로에서 a-b, b-c, c-d와 같은 부분 매칭은 가능하나 a-b-d와 같은 매칭은 불가능하다. 또한 연관 규칙을 이용한 반구조적인 데이타의 빈발 경로를 찾는 문제는 한 문서 내에서의 구조의 중복된 패턴을 찾는 방법이다. 따라서 서로 다른 문서들 구조간 유사성 검사를 위해서 새로운 방법이 필요하다.

한편 프로그램의 중복된 코드를 찾는 문제는 소프트웨어 유지 보수에 드는 비용을 줄이기 위해서 혹은 프로그램의 표절을 찾기 위하여 등의 목적으로 여러 분야에서 연구되어 왔다[6]. 프로그램의 대상은 프로그래밍 언어로 최근 연구들은 구문 트리(abstract syntax tree)를 이용하여 부분 트리의 중복된 코드를 찾는 방법을 선택하고 있다[6][7]. 그러나 각 언어의 파서에 의존적이고 여러 언어에 동시에 적용되기는 어렵다고 알려져 있다[7][8]. 또한 부분 트리의 중복성을 검사하기 위한 알고리즘의 시간 복잡도가 트리의 노드의 개수를 N이라고 했을 때, $O(N^2)$~$O(N^3)$에 이른다[5][7][17]($O(N^2)$은 [6]이 제시한 해쉬 함수를 이용하여 비교 대상을 줄임으로써 이룰 수 있다.). 이는 프로그램이 아닌 방대한 웹 문서인 XML을 대상으로 한다면 실제적으로 적용되기 어려운 복잡도이다. 또한 본 논문에서 대상으로 하는 XML은 일반 프로그래밍 언어와는 달리 프로그램의 변

수 개념도 존재하지 않고 반드시 따라야 하는 특정 구문도 존재하지 않는 특징을 갖는 마크업 언어이다. 따라서 XML만이 가지는 의미를 고려하여 요소간의 유사성을 구별하고 문서의 계층적인 구조 최소화하여 이를 시퀀스화 함으로써 최대 유사한 경로를 찾는 알고리즘의 시간 복잡도를 선형(linear) 시간으로 줄일 수 있다 (본 논문에서 사용하는 순차 패턴 마이닝 알고리즘[9]이 선형 시간을 갖는다.).

3. XML 요소간 유사성

XML 문서는 자유롭게 정의되는 요소로 인하여 XML 문서에 내재된 의미를 자동으로 해석하기 어렵다. 서로 다른 단어를 사용하여 정의된 요소를 식별해 내기 위해 본 장에서는 XML의 요소를 추출하고 WordNet[10] 시소러스와 사용자-정의 용어 사전을 이용하여 요소의 유사어와 합성어, 약어를 가진 확장-요소 벡터를 생성한다. 생성된 확장-요소 벡터를 기반으로 유사 행렬을 구축하여 요소간 유사성을 도출한다. 요소간 유사성은 발견된 구조 정보와 연결되어 XML 문서간의 유사성을 검사하는 데 기초가 된다.

3.1 확장-요소 벡터의 생성

XML의 확장-요소 벡터를 생성하는 단계는 다음과 같다.

- 먼저 XML 요소를 추출하기 위하여 문서를 파싱하여 DOM[11] 트리를 생성한다.
- 추출된 요소는 의미 있는 토큰으로 재생성 되기 위해 공백(space), 하이픈(hyphen), 밑줄(underscore)을 구분자로 걸러진다.
- 걸러진 요소의 불용어를 제거한다.
- 어간 추출 과정을 통해 어간 혹은 원형을 추출한다.
- 최종적으로, 생성된 요소 정보는 WordNet 시소러스와 사용자-정의 용어 사전과 같은 개념 지식을 통해 유사어, 합성어, 약어로 확장한다.

WordNet은 자연어 처리를 위한 사전으로 요소의 유사어를 찾기 위해 사용된다. 그리고 사용자-정의 용어 사전은 WordNet의 유사어 이외의 새로운 유사어, 합성어, 약어를 포함한다. 예를 들어 'book'의 확장-요소 벡터는 book → (volume, record, recordbook, script, playscript, ledger, leger, book_info, textbook, booklist, bib) 이다. 이러한 확장-요소 벡터를 형식화하면 다음 정의 1과 같다.

(정의 1) 확장-요소 벡터

// SynVect_UDL : 사용자-정의 라이브러리의 유사어, 생략어, 합성어
// SynVect_WN : WordNet으로부터 도출된 유사어
// MaxSize : 확장-요소 벡터의 최대 길이
// <T>는 XML 요소, t는 확장-요소 벡터의 용어 (term)

$Doc_i = (<T^i_1>, <T^i_2> \cdots, <T^i_m>), \quad T^i_m = (t^i_{m1}, t^i_{m2}, \cdots, t^i_{ma})$

$Doc_j = (<T^j_1>, <T^j_2> \cdots, <T^j_n>), \quad T^j_n = (t^j_{n1}, t^j_{n2}, \cdots, t^j_{nb})$

when $a, b \leq$ MaxSize, t^i_{ma} and $t^j_{nb} \in$ SynVect_UDL or SynVect_WN

하나의 XML 문서는 T^i_1, T^i_2 …, T^i_m 와 같은 요소를 이루는 용어(term)로 표현되고 각각의 요소 T^i_m은 t^i_{m1}, t^i_{m2}, …, t^i_{ma}와 같이 유사어, 합성어 등으로 확장된 용어를 포함하게 된다. XML 문서는 확장-요소 벡터 생성과정을 통해 각각의 요소들이 내재적인 유사어들을 가지게 되고 이를 통하여 다른 문서에서 사용된 유사한 요소들을 구별해 낼 수 있는 것이다.

3.2 요소간 유사성 척도

생성된 확장-요소 벡터간의 유사성은 서로 얼마나 매치되는가 하는 정도에 따라 계산할 수 있다. 한 문서의 요소가 다른 문서의 요소와 완전히 혹은 부분 매치되는가 그리고 확장-요소 벡터의 단어와 매치되는가에 따라 유사도는 달라진다. 따라서 매칭되는 정도에 따라 요소의 유사성에 대한 척도를 다음 7가지 레벨로 정의한다.

(정의 2) 요소간 유사성

- Level 6 (ori-ori exact) : 본래 두 요소가 완전 일치할 때. (완전 매칭)
- Level 5 (ori-syn exact) : 한 요소가 다른 요소의 확장-요소 벡터내의 용어와 완전 일치할 때.
- Level 4 (syn-syn exact) : 확장-요소 벡터내의 용어들이 일치할 때.
- Level 3 (ori-ori sub) : 두 요소가 부분적으로 일치할 때, 즉 부분 문자열인 경우.
- Level 2 (ori-syn sub) : 한 요소가 다른 요소의 확장-요소 벡터내의 용어와 부분 일치할 때.
- Level 1 (syn-syn sub) : 두 요소의 확장-요소 벡터내의 용어들이 부분 일치할 때.
- Level 0 (except) : 위의 어느 관계도 성립하지 않을 때.

3.3 요소간 유사성 측정을 위한 유사 행렬

두 문서의 요소를 행과 열로 하는 유사 행렬을 구축한다. 그림 1은 book_catalogue.xml과 books.xml의 유사 행렬을 보여준다.

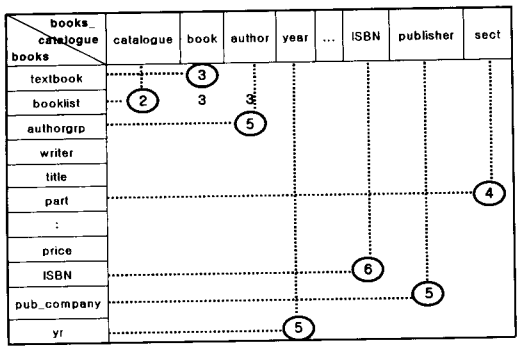


그림 1 요소간 유사 행렬

catalogue, book, author와 같은 요소들은 각자 확장-요소 벡터를 내재하고 있다. 공란은 두 요소간에 아무 관계도 없음을 의미한다. 그리고 2, 3, 5와 같은 값은 요소 유사도를 의미한다. 만약 요소에 매칭되는 값이 여럿일 경우 유사성이 높은 것을 선택한다. 확장-요소 벡터를 내재한 요소들의 유사성은 정의 2의 요소간 유사성 척도를 기반으로 계산된다.

기준이 되는 문서의 요소를 행으로 놓고 비교 하고자 하는 문서의 요소를 열로 하여 유사 행렬을 구축한다. 즉 행렬은 정의 1에서처럼 Doc_i의 $<T^i_1>$, $<T^i_2>$ ..., $<T^i_m>$ 과 Doc_j의 $<T^j_1>$, $<T^j_2>$ ..., $<T^j_n>$ 로 구성된다. 그리고 정의 2의 레벨 값을 유사성의 실제 값으로 지정한다. 이 값은 얼마든지 늘이거나 줄일 수 있다. 그리고 두 문서 Doc_i와 Doc_j의 요소쌍의 유사성을 WX(Weight of Maximum)이라고 하자. WX는 요소간 유사성의 최종 값으로 문서의 유사성을 구하기 위해 직접 적용되는 가중치라고 할 수 있다. 임의의 Doc_i의 요소 $<T^i_c>$와 매칭되는 Doc_j의 $<T^j_k>$ 요소와의 가중치 WX는 Max(Level($<T^i_c>$,$<T^j_1>$), Level($<T^i_c>$,$<T^j_2>$), ..., Level($<T^i_c>$,$<T^j_n>$))로부터 구할 수 있다. 즉 0보다 큰 레벨 값이 여러 개 존재할 때 WX값은 최대값을 선택한다. 이 때 Level($<T^i_c>$,$<T^j_k>$)은 본래의 요소뿐만 아니라 내재된 확장-요소 벡터의 단어들 T^j_n((t^j_{n1}, t^j_{n2}, ..., t^j_{nb})과도 비교한 값이다.

그림 1에서 book_catalogue.xml의 <author> 요소는 books.xml의 <authorgrp>와 <writer> 두 요소로부터 WX(<author>, <writer>)은 Max(Level(<author>, <authorgrp>), Level(<author>, <writer>))를 통하여 5값을 얻게 된다. 이러한 WX 값을 기반으로 요소간의 유사성을 인식할 수 있다. 이러한 전체적인 알고리즘을 다음 그림 2에 기술하였다.

```
Procedure ComputeElementSim ( T^i : original element set of i document,
                              t^i_v : extended-element vector sets of each element of i document,
                              T^j : original element set of j document,
                              t^j_v : extended-element vector sets of each element of j document )
returns  Similarity matrix between XML Elements
begin
    setup parameter level 1 ~ level 6;
    for (k=1; k <= m; k++) do
    begin
        if T^i_k is completely matched with any element of T^j set
        then sim(T^i_k) := level6; // 본래의 두 요소가 완전 일치할 때
        elseif T^i_k is contained in t^j_v of any element of T^j set
        then sim(T^i_k) := level5;  // 한 요소가 다른 요소의 확장-요소 벡터의 용어와 완전 일치할 때
        elseif t^i_k is completely matched with any t^j_v of any element of T^j set
        then sim(T^i_k) := level4; // 확장-요소 벡터내의 용어들이 일치할 때
        elseif T^i_k is substring of any element of T^j set
        then sim(T^i_k) := level3; // 두 요소가 부분적으로 일치할 때
        elseif T^i_k is substring of any term of t^j set
        then sim(T^i_k) := level2;  // 한 요소가 다른 요소의 확장-요소 벡터의 용어와 부분 일치할 때.
        elseif t^i_k is substring of any term of t^j set
        then sim(T^i_k) := level1;  // 두 요소의 확장-요소 벡터 용어들이 부분 일치할 때
    end
    for (k=1; k <= m; k++) do
    begin
        if there are multiple similarity values of T^i_k
        then begin
            WX is selected maximum value among them;
            T^j_c := original element of document j, which has maximum value; // Similarity pair of WX(T^i_k )
        end
    end
    return WX(T^i_k , T^j_c);   // similarity between elements
```

그림 2 요소간 유사성 계산 알고리즘

4. XML 구조 발견

데이타베이스 분야에서 질의 처리 등을 위하여 스키마를 사용하여 왔다. 이제 웹 데이타 처리를 위해서도 문서의 구조를 발견할 필요가 있으며 질의 최적화, 여러 데이타 소스 통합, 인덱스를 구축하는 등의 목적으로 사용된다[12]. XML 구조간의 유사성을 판별해 내기 위하여 유한 오토마타를 이용하여 손실 정보 없이 구조를 추출하고 단순화하는 방법을 제시한다.

4.1 오토마타를 이용한 XML 구조 형식화

반구조적인 데이타의 구조를 형식화 하는 방법에는 여러 가지가 있다[12]. 본 논문에서는, XML 문서 요소의 계층 구조는 어느 한 상태에서 입력 심벌인 요소를 보고 갈 수 있는 다음 상태가 하나 이상 존재하므로 NFA(Nondeterministic Finite Automata)를 이용하여 형식화 한다[13]. NFA는 같은 구조를 인식할 수 있는

DFA(Deterministic Finite Automata)로 변환할 수 있고 상태 최소화 알고리즘을 이용하여 구조의 경로를 최소화 할 수 있다[13]. 다음 그림 3은 서적에 관한 XML 문서의 예로 <bib>라는 루트 요소로부터 <book>, <article> 등의 요소가 중복되어 있다.

```
<bib>
  <book year="1995">
    <title>An Introduction to Database Systems</title>
    <author><lastname>Date</lastname></author>
    <publisher><name>Addison-Wesley</name></publisher>
  </book>
  <book year="1999">
    <title>Data on the Web: from Relations to Semistructured Data &amp;amp; XML</title>
    <author><firstname>Serge</firstname><lastname>Abiteboul</lastname></author>
    <author><firstname>Peter</firstname><lastname>Buneman</lastname></author>
    <author><firstname>Dan</firstname><lastname>Suciu</lastname></author>
    <publisher><name>Morgan-Kaufman</name></publisher>
  </book>
  <article year="1999" type="inproceedings" month="June">
    <author><firstname>Mary</firstname><lastname>Fernandez</lastname></author>
    <author><firstname>Alin</firstname><lastname>Deutsch</lastname></author>
    <author><firstname>Dan</firstname><lastname>Suciu</lastname></author>
    <title>Storing Semi-structured Data Using STORED</title>
    <booktitle>ACM SIGMOD</booktitle>
  </article>
  <article year="1995" type="inproceedings" month="Jan">
    <author><firstname>Norman</firstname><lastname>Ramsey</lastname></author>
    <author><firstname>Mary</firstname><lastname>Fernandez</lastname></author>
    <title>The New Jersey Machine-Code Toolkit</title>
    <booktitle>USENIX</booktitle>
  </article>
</bib>
```

그림 3 bib.xml

이 문서를 NFA-XML로 표현하면 다음 그림 4와 같다. 'An Introduction to Database System'이라는 데이타로 가는 경로는 <bib>라는 태그를 루트로 시작하여 '1'상태, <book>, '2', <title>에 도달하면 얻을 수 있다.

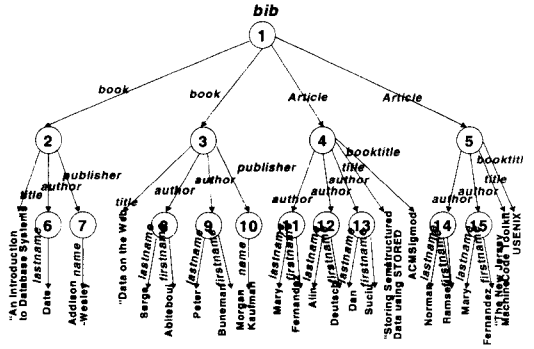


그림 4 bib.xml 의 NFA-XML

XML 문서 표현을 위한 NFA-XML을 형식화 하기 위해 다음과 같이 정의한다.

(정의 3) NFA-XML M = (Q, D, Σ_N, Σ_T, δ, q0, F)

Q : 유한 상태 집합. 요소가 컨텐트를 유도하지 않으면 새로운 상태가 생성된다.

D : 유한 데이타 집합(컨텐트)

Σ_N : 컨텐트를 유도하지 않고 새로운 상태를 생성시키는 유한 요소 집합.

Σ_T : 직접 컨텐트 D를 유도하는 유한 요소 집합.

δ : 전이 함수. 만약 어느 요소 $a \in \Sigma_N$이면 $\delta(q,a) = p_1, p_2, ..., p_n$(p는 새로운 상태)이고 $a \in \Sigma_T$이면 $\delta(q,a) = D_1, D_2, ..., D_n$이다.

q_0 : 시작 상태. 루트 요소는 시작 상태로의 전이를 일으킨다.

F : 종결 상태. 컨텐트를 유도하는 요소 ($\in \Sigma_T$) 만이 종결상태로의 전이를 일으킨다.

4.2 XML 구조 최소화

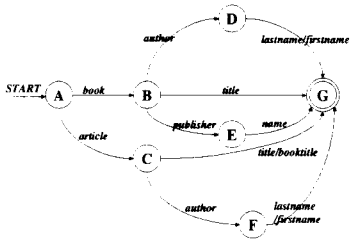
일반적으로 NFA는 문서의 구조를 쉽게 표현할 수 있으나 구조의 단순화를 위해 요소의 중복성을 제거해야 한다. 따라서 한 상태에서 요소를 보고 갈 수 있는 다음 상태가 하나로 결정되는 DFA로 변환한다.

δ	book	article	title	author	publisher	booktitle	lastname	name	firstname
[1]	[2,3]	[4,5]	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
[2,3]	ϕ	ϕ	–	[6,8,9]	[7,10]	ϕ	ϕ	ϕ	ϕ
[4,5]	ϕ	ϕ	–	[11,12,13,14,15]	ϕ	⊥	ϕ	ϕ	ϕ
[6,8,9]	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	⊥	ϕ	⊥
[7,10]	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	⊥	ϕ
[11,12,13,14,15]	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	⊥	ϕ	–

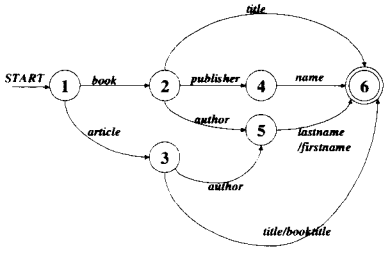
⊥ : atomic data

[1] ◊ A. [2,3] ◊ B. [4,5] ◊ C.
[6,8,9] ◊ D. [7,10] ◊ E. [11,12,13,14,15] ◊ F

(a) 상태 전이 테이블



(b) DFA-XML



(c) 최소화된 DFA-XML

그림 5 생성된 bib.xml의 최소화된 DFA-XML

변환된 DFA와 이를 유도한 상태 전이표는 다음 그림 5(a)와 같다. 루트 요소로부터 유도된 상태 [1]에서 시작하여 요소들 ($\Sigma_N \cup \Sigma_T$)을 보고 다음 상태로 전이된다. 모두 데이타 D를 유도할 때까지 진행된다. '⊥'는 데이타 D를 유도하는 Σ_T 요소를 만난 경우로 종결 노드로 전이된다. 이렇게 요소를 입력 심벌로 보고 전이된 상태들의 동치류가 새로운 상태를 만들면서 구조가 단순화 된다. 새로 생성된 상태를 A~F로, 종결 상태를 G로 보면 그림 5(b)의 DFA-XML을 얻을 수 있다. 또한 생성된 DFA-XML의 상태를 더욱 최소화하기 위해 동치 관계를 다음 그림 6의 구조 최소화 알고리즘을 이용하여 상태들을 합친다. 이렇게 상태 수를 최소화시킨 DFA-XML은 그림 5(c)와 같다. 각 상태 이름은 구별을 위해 임의로 변경하였다.

```
Procedure MinimizeStructure (D: document)
returns  minimized structure of D;
begin
    convert D to NFA-XML; // formalize (node : state number)
                                (transition : element)
    transform NFA-XML to DFA-XML; // reduce duplicate
                                elements
    MinimizeDFA(DFA-XML); // convert DFA-XML to minimum
                            state DFA
                          // by finding all sets of equivalence
                            states
                          // and merging them to form a
                            single state
end
Procedure MinimizeDFA (M : DFA-XML of a document)
returns  minimized M;
begin
    initialize all entries as unmarked;
    partition all pairs of final (F) and non-final state (NF); //
    partition two sets
    for all pairs of unmarked state p, q do
    begin
                    // p and q are contained in different partition
        if ( state p ∈ F and q ∈ NF ) or ( state p ∈ NF and
          q ∈ F )
        then entry d[p, q] := true;
    end
    for each all pairs where d[p, q] is false do // p and q are
        contained in same partition
    begin
        let r = ((p, a) and s = ((q, a); // with all inputs
        if  d[r, s]  is true // p and q are contained in different
                              partition
        then d[p, q] := true;
        repeat until no change ;
    end
    merge all state pairs for which d[p, q] is false; // merge
    equivalence class
end
```

그림 6 구조 최소화 알고리즘

4.3 경로 표현

그림 5(c)의 DFA-XML의 경로를 표현하기 위해서 요소의 전이를 중심으로 추출하면 (bib, book, title, ⊥), (bib, book, publisher, name, ⊥), (bib, book, author, lastname, ⊥), (bib, book, author, firstname, (), (bib, article, author, lastname, (), (bib, article, author, firstname, (), (bib, article, booktitle, (), (bib, article, title, () 이다. 이렇게 추출된 경로는 여러 XML 문서의 유사성을 추론하기 위한 경로로 사용되는데 두 문서가 한 경로에서 같은 스트링을 많이 생성할수록 유사하다고 판단할 수 있다. 구체적으로 순차 패턴 마이닝 (sequential pattern mining) 알고리즘[9]을 이용하여 하나의 요소(element)를 트랜잭션으로 보고 최소 지지도 개념을 도입하여 최대 빈발 패턴을 구한다. 순차 패턴 마이닝 알고리즘은 연관 규칙과는 달리 트랜잭션의 발생 횟수만이 아닌 발생 순서(sequence)를 고려한다는 점에서 구조를 이루는 경로상에서 요소들의 순차를 고려 할 수 있다.

5. XML 구조간 유사성 계산

요소간 유사성과 발견된 구조를 기반으로 XML 구조간의 유사성을 분석한다. 구조간 유사성을 얻기 위한 방법으로는 XML의 구조를 트리로 보고 서로 다른 문서들이 트리의 경로를 얼마나 공유하고 있는지를 파악할 수 있다. 그리고 공유정도에 따라 구조간 유사성을 판별할 수 있다. 이때 동일 요소가 나타나는 트리의 레벨, 한 요소에 속한 요소들의 가짓수 등에 상관 없이 경로의 포함 관계를 고려할 수 있어야 한다. 본 논문에서는 포함 관계를 고려할 수 있는 변형된 순차 패턴 마이닝 알고리즘을 사용한다. 순차 패턴 마이닝 알고리즘은 고객의 트랜잭션을 하나의 시퀀스로 보고 모든 고객의 시퀀스에서 사용자가 명시한 최소 지지도를 만족하는 최대 시퀀스를 찾는 알고리즘으로 N을 고객 트랜잭션 수로 보았을 때, 선형 시간 복잡도를 갖는다[9]. 다음 표 1은 주어진 고객의 시퀀스에서 순차 패턴 마이닝 알고리즘을 사용하여 최대 빈발 시퀀스-밑줄 친 시퀀스-를 찾아내는 과정을 보인다.

표 1 순차 패턴 마이닝 알고리즘(최소 지지도 40%)

Customer Sequences	Large 1-Sequences & Support		Large-2 Sequences & Support		Large-3 Sequences & Support		Large-4 Sequences & Support	
			<1 2>	2				
			<1 3>	4				
<(1 5) (2) (3) (4)>	<1>	4	<1 4>	3	<1 2 3>	2		
<(1) (3) (4) (3 5)>	<2>	2	<1 5>	3	<1 2 4>	2		
<(1) (2) (3) (4)>	<3>	4	<2 3>	2	<1 3 4>	3	<1 2 3 4>	2
<(1 3 5)>	<4>	4	<2 4>	2	<1 3 5>	2		
<(4 5)>	<5>	4	<3 4>	3	<2 3 4>	2		
			<3 5>	2				
			<4 5>	2				

추출된 구조의 경로에서 같은 시퀀스를 많이 생성할수록 두 문서는 유사하다고 볼 수 있다. 순차 패턴 마이닝 알고리즘을 이용하여 하나의 XML 요소를 트랜잭션으로 보고 최대 빈발 패턴을 구한다. 연관 규칙[14]과는 달리 트랜잭션의 발생 횟수만이 아닌 발생 순서, 즉 시퀀스를 고려한다는 점에서 구조를 이루는 경로상에서 요소들의 순차를 고려할 수 있다.

5.1 XML 구조간 유사성 측정을 위한 순차 패턴 마이닝 알고리즘의 적용

XML 구조간의 유사한 시퀀스를 찾아내기 위해서 순차 패턴 마이닝 알고리즘[9]을 변형한다. 본래의 순차 패턴 마이닝 알고리즘은 총 5단계로 나누어 순서 개념을 갖는 빈발 항목을 찾는다. 변형된 개념을 다음 표 2에서 요약하였다.

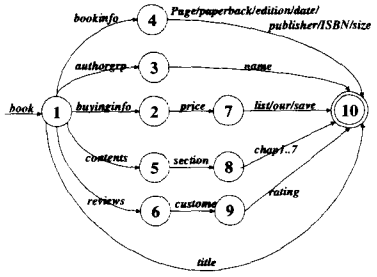
표 2 XML을 위해 변형된 순차 패턴 마이닝 알고리즘

단계	본래의 순차 패턴	변형 순차 패턴
정렬 (Sort)	데이타베이스는 고객 ID와 트랜잭션 시간으로 정렬된다. 본래의 트랜잭션 데이타베이스는 고객 시퀀스로 변경된다.	XML 문서는 최소화 된 구조(DFA-XML)로 변경된다. 이로부터 경로 표현(path exprcssions)을 추출하면 경로를 구성하는 요소는 각 레벨로 정렬된다.
빈발항목찾기 (Litemset :Large Itemsets)	모든 길이 1의 빈발 시퀀스 패턴이 발견된다.	모든 길이 1(경로상의 하나의 요소)의 빈발 시퀀스 패턴을 발견한다. 즉 두 문서간 유사 요소들이 식별된다.
변형 (Transformation)	고객 시퀀스는 빈발 시퀀스들이 고객 시퀀스에 포함되는지를 빠르게 결정하기 위해 표현 형식이 변형된다.	모든 추출된 경로 표현을 구성하는 요소들은 정수 표현으로 변경한다. 이때 유사한 요소들은 같은 정수로 지정한다.
시퀀스 (Sequence)	길이 1, 2, …, n의 빈발 시퀀스를 최소 지지도에 입각하여 찾는다.	길이 1, 2, …, n의 빈발 경로 표현을 지지도에 입각하여 찾는다. (여기에서 최소 지지도는 두 문서에 모두 있다는 가정으로 100%가 된다.)
최대항목찾기 (Maximal)	최대 빈발 시퀀스를 찾는다.	최대 유사 경로를 찾는다.
유사성 계산 (Computing)		XML 구조간 유사성은 기준 문서의 전체 경로에 대해 비교 문서의 최대 유사 경로의 비율로 계산된다.

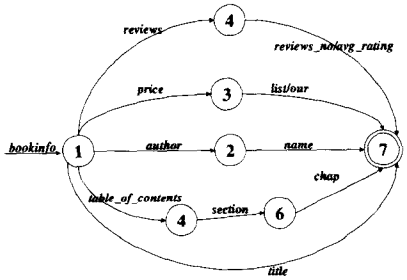
먼저 문서의 DFA-XML로부터 경로의 중복을 제거한 최소화된 구조를 추출한다. 데이타베이스에서 트랜잭션을 정렬하듯이 경로의 각 요소들은 구조의 트리의 레벨로 정렬화 한다. 다음으로 루트에서 단말 노드에 이르는 경로들을 탐색함으로써 두 문서에 모두 나타나는 1-시퀀스, 2-시퀀스 순으로 최대 n-시퀀스를 찾아 낸다. 이때 시퀀스를 구성하는 요소간의 유사 행렬을 기반으로 유사한 트랜잭션을 인식할 수 있다. 빈발의 기준인 최소 지지도는 두 문서에 모두 나타나는 경로이어야 빈발 경로라고 말할 수 있으므로 항상 100%가 된다. 포함관계가 있는 시퀀스를 제거하여 최대 시퀀스를 구하고 생성된 최대 시퀀스들이 경로상에 차지하는 비율은 두 문서 구조 사이의 유사성이 된다.

< Example >

책에 관한 정보를 표현하기 위한 두 XML 문서의 구조를 추출한 DFA-XML은 다음 그림 7과 같다. 예제로 사용한 문서는 온라인 책 서점인 아마존(www.amazon.com)과 반즈앤노블스(www.barnes&nobles.com)에서 "professional WAP"책을 검색하여 나온 결과 페이지의 내용을 참조하여 XML로 문서화 한 것이다.



(a) amazon.xml



(b) b&n.xml

그림 7 DFA-XML의 예

각각의 DFA-XML을 참조하여 경로를 추출한다. 적용된 순차패턴 마이닝 알고리즘에서 변형(transformation) 단계는 확장된-요소 벡터의 유사성을 참조하여 같은 의미를 갖는 요소는 같은 트랜잭션으로 처리하고 단순화 하는 절차이다. 다음 표 3은 이러한 요소 유사정보, 즉 (<author>, <authorgrp>), (<contents>, <tableofcontents>), (<rating>, <avg_rating>)를 이

용하여 두 문서의 경로 표현을 변형한 것이다.

표 3 amazon.xml 및 b&n.xml 문서구조의 변형 경로

문서	본래의 경로 표현	변형된 경로
amazon.xml	book.title.⊥	<{1} {2}>
	book.bookinfo.page.⊥	<{1} {3} {4}>
	book.bookinfo.paperback.⊥	<{1} {3} {5}>
	book.bookinfo.edition.⊥	<{1} {3} {6}>
	book.bookinfo.date.⊥	<{1} {3} {7}>
	book.bookinfo.publisher.⊥	<{1} {3} {8}>
	book.bookinfo.ISBN.⊥	<{1} {3} {9}>
	book.bookinfo.size.⊥	<{1} {3} {10}>
	book.buyinginfo.price.list.⊥	<{1} {11} {12} {13}>
	book.buyinginfo.price.our.⊥	<{1} {11} {12} {14}>
	book.buyinginfo.price.save.⊥	<{1} {11} {12} {15}>
	book.contents.section.chap.⊥	<{1} {16} {17} {18}>
	book.reviews.customer.rating.⊥	<{1} {19} {20} {21}>
	book.authorgrp.name.⊥	<{1} {22} {23} >
b&n.xml	bookinfo.title.⊥	<{3} {2}>
	bookinfo.price.list.⊥	<{3} {12} {13}>
	bookinfo.price.our.⊥	<{3} {12} {14}>
	bookinfo.author.name.⊥	<{3} {22} {23}>
	bookinfo.tableogcontents.section.chap.⊥	<{3} {16} {17} {18}>
	bookinfo.reviews.reviews_no. ⊥	<{3} {19} {24}>
	bookinfo.reviews.avg_rating.⊥	<{3} {19} {21}>

그리고 표 4는 두 변형된 경로에서 빈발 항목인 유사 경로를 구한 것이다. 표에서 밑줄 친 <{2}>, <{3}>, <{12} {13}>, <{12} {14}>, <{19} {21}>, <{22} {23}>, <{16} {17} {18}>가 최대 유사 경로(maximal similar path)이다. 최대 유사 경로를 가지고 DFA-XML을 따라가 보면 b&n.xml의 거의 모든 경로가 amazon.xml의 경로와 유사하다는 것을 알 수 있다.

표 4 유사 경로

Large 1-sequence	Mapping Element	Large 2-sequence	Large 3-sequence
<{2}>	<bookinfo>		
<{3}>	<title>		
<{12}>	<price>	<{12} {13}>	
<{13}>	<list>	<{12} {14}>	
<{14}>	<our>	<{16} {17}>	
<{16}>	<contents(=tableofcontents)>	<{16} {18}>	<{16} {17} {18}>
<{17}>	<section>	<{17} {18}>	
<{18}>	<chap>	<{19} {21}>	
<{19}>	<reviews>	<{22} {23}>	
<{21}>	<rating(=avg_rating)>		
<{22}>	<author(=authorgrp)>		
<{23}>	<name>		

5.2 구조간 유사성 척도

앞 단계에서 얻어진 최대 유사 경로를 이용하여 기준 문서와 비교 문서를 설정하여 실질적인 유사 값을 계산한다. 예로 5.1절의 문서 amazon.xml을 기준 문서로 보고 b&n.xml을 비교하여 유사성을 계산한다. 기준 문서 구조의 경로 표현을 PE_B (path expression of base document)라 하고 비교 문서 구조의 경로 표현을 $PE_{Q1..n}$ (path expression of query document 1..n) 라 하자. 기준 문서의 내포관계를 표현한 경로를 NPE_B (nested path expression of base document)라 하면 다음 그림 8의 알고리즘을 통해 구할 수 있다.

```
Procedure GenerateNPE (Minimized DFA-XML of a
          base document)
begin
    NPE_B = {{R: root_element}, {CreateNode(child_node 1
          of R)}, {CreateNode(child_node 2of R)},···,
          {CreateNode(child_node n of R)}};
end
Procedure CreateNode(child_node n);
begin
    for each child_node n do
      begin
        state := child_node n;
        if state has more than two child nodes
        then append (state, CreateNode(child_node 1 of
             state), ..., CreateNode(child_node m of
             state))
        elseif state has only one child node
        then append (state, child_nodeof state);
             // make child node to sibling;
      end
end
```

그림 8 NPE 생성 알고리즘

예를 들어 기준 문서 amazon.xml의 NPE_B 는 { {1}, {2}, {3, (4, 5, 6, 7, 8, 9, 10)}, {11, 12, (13, 14, 15)}, {16, 17, 18}, {19, 20, 21 }, {22, 23}}로 표현된다. PE_B 는 표 3의 변형 경로로 표현되고 그리고 표 4에서 도출된 비교 문서 경로의 최대 유사 경로는 다음 NPE_B에 밑줄 친 시퀀스와 같다({{1}, **{2}**, {**3**, (4, 5, 6, 7, 8, 9, 10)}, {11, **12**, (**13**, **14**, 15)}, {**16, 17, 18**}, {**19**, 20, **21**}, {**22, 23**}}. 두 문서간의 유사성 계산은 기준 문서의 전체 경로에 대한 비교 문서의 최대 유사 경로의 비율로 다음 표 5와 같이 계산한다.

표 5 PE^B 과 PE^{Q1} 유사성 계산

NPE_B	Paths	Ratio
NPE_B^1	<1>	1/7*(1/1*(0))
NPE_B^2	<2>	1/7*(1/1*(1))
NPE_B^3	<3, (4, 5, 6, 7, 8, 9, 10)>	1/7*(1/2*(1+1/7*(0+0+0+0+0+0+0)))
NPE_B^4	<11, 12, (13, 14, 15)>	1/7*(1/3*(0+1+1/3*(1+1+0)))
NPE_B^5	<16, 17, 18>	1/7*(1/3*(1+1+1))
NPE_B^6	<19, 20, 21>	1/7*(1/3*(1+0+1))
NPE_B^7	<22, 23>	1/7*(1/2*(1+1))
$\sum_{k=1}^{length(NPE_B)}$ the ratio of NPE_B^k		0.833

amazon.xml과 b&n.xml간 유사성은 83.3%로 이와 같은 유사성 계산을 공식화 하면 다음과 같다.

$$\sum_{i=1}^{L(NEP)} \frac{1}{S(n)} \text{ X } V(E_k)$$

S(n) is a number of siblings of 1st level,
L( ) is length function

$$V(E_k) = \begin{cases} 1, \text{ if } E_k \text{ is a atomic and selected} \\ 0, \text{ if } E_k \text{ is a atomic and not selected} \\ \frac{1}{L(Ek)} \sum_{i=1}^{L(Ek)} V(E_i), \text{ if } E_k \text{ is a list} \end{cases}$$

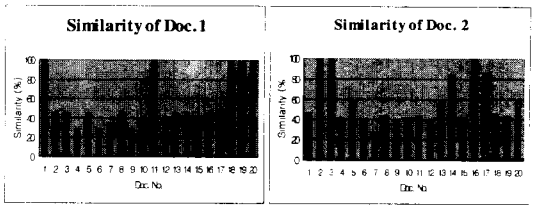
그림 9 유사성 계산식

6. 실 험

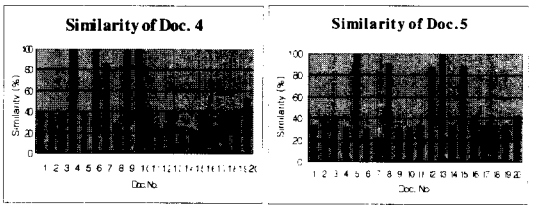
본 논문에서 제시한 XML을 위한 텍스트 준비 방법을 적용하기 위해서는 매우 다양한 구조의 XML 문서가 필요하다. 그러나 실제 웹상에서 찾을 수 있는 XML 문서들은 동일한 DTD, 즉 획일적인 구조를 가진 문서들이 대부분으로 방대한 크기의 다양한 구조를 가진 XML dataset을 구하기 어렵다. 따라서 본 논문에서 제시한 XML 문서간의 유사성 분석을 위해서 HTML문서를 XML문서로 직접 변형하여 테스트하였다. 실험은 본 논문에서 제안한 알고리즘에 입각하여 수동으로 수행하였다. 실험은 실제 문서를 대상으로 온라인 서점인 아마존 사이트의 music, book, electronics 과 videos를 판매하는 페이지들을 수집하여 유사성을 기반으로 문서들의 카테고리를 자동 식별할 수 있는지를 보인다. 수집된 문서에 파일의 번호를 무작위로 붙이고(표 6) <title>, <price>, <buying_info>, <recording_company> 등과 같은 태그를 갖는 XML 문서로 변형하였다.

표 6 카테고리별 XML 문서

Category	File Name	File No.	Category	File Name	File No.
Books	0025107755.xml	14	Music	B0000000BU.xml	19
	0060175400.xml	2		B000000QQY.xml	1
	0060175869.xml	16		B000000ZML.xml	20
	0060193395.xml	3		B000001ER3.xml	18
	0060193646.xml	17		B000001HU9.xml	11
Videos	0767819586.xml	15	Electronics	B00000J07K.xml	10
	0780628799.xml	8		B00000J0AR.xml	6
	0783113943.xml	5		B00000J1EP.xml	9
	0783216084.xml	12		B00000J1FV.xml	4
	0783240503.xml	13		B00000J1SS.xml	7



(a) 기준 문서 1(music) (b) 기준 문서 2(book)



(c) 기준 문서 4(electronics) (d) 기준 문서 5(video)

그림 10 각 카테고리별 유사성

이를 각 카테고리별 대표 문서를 1(music), 2(book), 4(electronics), 5(video)로 보고 유사성을 비교해 보면 다음 그림 10과 같다.

각 문서의 상위 4개의 유사한 문서는 문서 1 (category : music)은 {문서 11(100%), 18(100%), 20(100%), 19(98.21%)}, 문서 2는 (category: book) : {3(100%), 16(100%), 17(85.71%), 14(84.82%)}. 문서 4는 (category - electronics) {10(100%), 6(98.96%), 9(98.96%), 7(86.46%)}, 그리고 문서 5는 (category - video) : {13(99.24%), 8(90.15%), 15(88.26%), 12 (86.74%)}. 이다. 이는 표 6과 비교해 보았을 때, 각 문서들의 유사성을 토대로 정확히 각 문서의 카테고리를 식별할 수 있음을 보여 준다.

그러나 각 문서간 유사성 결과는 기준 문서가 무엇이냐에 따라 그 결과가 달라진다. 즉 기준 문서가 무엇이냐에 따라 최대 유사 경로의 비율이 달라지기 때문에 유사성 결과가 대칭적(symmetric)이지 않다. 예를 들어 10(b)에서 기준 문서 2와 13은 58%의 유사성을 보인다. 그리고 10(d)의 5와 13은 99.24%의 유사성을 보인다. 그렇다면 2와 5사이의 유사성은 만약 유사성 결과가 대칭적이라면 58%정도 유사해야 한다. 하지만 2와 5사이의 유사성은 2가 기준 문서일 경우 58%, 5가 기준 문서일 경우는 41.67%로 달라지게 된다. 현재는 XML 문서간 유사성을 문서 자동 분류를 위한 카테고리 인식에만 테스트를 해 보았지만 본 논문의 최종 목표인 XML 클러스터링에 적용한다면 이러한 비대칭적 성질에 관한 좀 더 자세한 고찰이 필요 할 것이다.

7. 결론 및 향후 연구

본 논문에서 XML 문서간의 유사성 계산을 위하여 XML의 의미를 반영한 문서 분석 기법을 제안하였다. 확장-요소 벡터간의 유사 행렬을 구축함으로써 요소간의 유사성을 도출하였다. 그리고 XML의 구조를 최소화하기 위하여 DFA-XML을 이용하여 형식화 하였다. 또한 발견된 구조로 부터 XML을 위해 변형된 순차 패턴 마이닝 알고리즘을 통해 문서간의 최대 유사 시퀀스를 추출할 수 있었다. XML 요소와 구조의 의미를 반영한 유사성 계산 방법으로 기존의 벡터-공간 모델과 비교하여 봤을 때, 더 정확한 유사성 결과를 보였다. 또한 실제 온라인 서점의 문서를 분류한 결과 100%의 정확도를 보였다. 확장-요소 벡터 개념과 구조 및 유사 시퀀스 발견 방법은 XML 마이닝 이외에도 XML을 위한 인덱스 구축 시, EDMS에서 XML 문서관리, XML문서 병합 등의 분야에서 활용될 수 있다.

현재는 컨텐트 정보를 제거하고 XML 요소와 구조 정보, 즉 경로상의 유사성만을 반영하였다. 앞으로 컨텐트와 혼합하여 본 논문에서 제시한 XML의 특성을 반영한 문서 분석 방법을 텍스트 마이닝에 이용하여 실질적인 문서의 클러스터링을 수행하고자 한다.

참 고 문 헌

[1] Deutsch, Fernandez and Suciu. "Storing Semi-structured Data with STORED," *In Proc. of SIGMOD*, pages 431-442, 1999

[2] Nestorov, Abiteboul, Motwani. "Extracting Schema from Semistructured Data," *In Proc. of SIGMOD*, pages 295-306, 1998

[3] Ke Wang and Huiqing Liu. "Discovering Typical Structures of Documents: a Road Map Approach," In *the Proc. of SIGIR*, pages 146-154, 1998

[4] Brad Adelberg. "NoDoSE - A Tool for Semi-Automatically Extracting Structured and Semi-structured Data from Text Documents," In *Proc. of SIGMOD*, pages 283-294, 1998

[5] Christoph M. Hoffmann and Michael J. O'Donnell. "Pattern Matching in Trees," *Journal of ACM 29(1)*, pages 68 - 95, Jan. 1982.

[6] Ira D.Baxter, Andrew Yahin, Leonardo Moura, Marcelo Sant'Anna, and Lorraine Bier. "Clone Detection using Abstract Syntax Tree," In *Proc. of the ICSM'98*, Nov. 1998.

[7] 장성순, 서선애, 이광근, "프로그램 유사성 검사기", 제 28회 한국정보과학회 추계학술대회 논문집, pages 334-336, 2001.

[8] S. Ducasse, ,M.Reiger, S.Demeyer. "A Language Independed Approach for Detecting Duplicated Code," In *Proc. of the ICSM'99*, pages 109-118, Sep. 1999.

[9] R. Srikant and R. Agrawal. "Mining Sequential Patterns:Generalizations and Performance Improvements," In *Proc. of the Fifth Int'l Conf. on Extending Database Technology(EDBT)*, Avignon, France, March 1996.

[10] C.Fellbaum. *WordNet : An Electronic Lexical Database*, Cambridge:MIT Press. 1998.

[11] http://www.w3.org/DOM/

[12] Abiteboul, Buneman, and Suciu. *Data on the web : from relations to semistructured data and XML*, Morgan-Kaufmann, 2000

[13] A.V.Aho, R.Sethi and J.D.Ullman. *Compilers: Principles, Techniques, and Tools*, Addison Wesley, 1986.

[14] R. Agrawal, R. Srikant. "Fast Algorithms for Mining Association Rules," In *Proc. of the 20th Int'l Conference on Very Large Databases*, Santiago, Chile, Sept. 1994.

이 정 원

1989sus 3월 ~ 1993년 2월 이화여자대학교 전자계산학과 학사. 1993년 3월 ~ 1995년 2월 이화여자대학교 전자계산학과석사. 1995년 2월 ~ 1997년 8월 LG 종합기술원 주임연구원. 2000년 6월 ~ 2000년 8월 IBM Almaden Research Center 인턴쉽. 1997년 9월 ~ 현재 이화여자대학교 컴퓨터학과 박사과정. 관심분야는 데니타미이닝, XML, 프로그래밍 언어

이 기 호

1961년 이화여자대학교 수학과 학사, 석사. 1972년 텍사스 대학교 전산학 석사, 박사 수료. 1981년 서울대학교 전산학 박사. 캘리포니아대 연구교수, 이화여대 공대학장, 대학원장 역임. 1973년 ~ 현재 이화여대 컴퓨터학과 교수. 관심분야는 인터넷언어, XML기반 전자상거래