태그 간 동의어 집합을 통한 XML 문서 유사도 측정

: XSC(XML Similarity Calculation)

이강석° 송인상 김응모
성균관대학교 정보통신공학부 컴퓨터공학과
gangseok°@hotmail.com, insang@ece.skku.ac.kr, umkim@ece.skku.ac.kr

Similarity Measurement for XML Documents Using Tag Synonyms

Gangseok Lee° Insang Song Ungmo Kim
Dept. of Computer Engineering, Sungkyunkwan University

요 약

월드와이드웹에서의 정보를 재사용, 공유할 수 있도록 기준을 제시한 XML은 많은 곳에서 사용 중에 있으며, 널리 확산되고 있다. 사용자정의태그를 이용한 XML 의 특징은, 같은 도메인의 문서라도 사람의 인식이 아닌 컴퓨터와 같은 기계적으로는 다르게 인식될 수 있다는 문제점을 드러내기도 한다.

본 논문에서는 이러한 문제점을 해결하고자 시소러스와 온톨로지 등을 이용해 XML 문서간 유사도를 측정하는 방법을 제시하며 이를 바탕으로 제작한 프로그램인 'XML Similarity Calculation' 를 이용하여 제시한 방법이 타당하다는 것을 증명하게 된다. 또한 주어진 예시자료를 가지고 이 프로그램의 성능평가를 통해 정확성과 효율성을 평가하고 앞으로의 연구방향을 제시한다.

1. 서 론

웹상의 문서에 담긴 정보를 재사용하기 위한 방안을 제시한 것 중 하나가 XML 이다. W3C 에 의해 제안된 XML 은 사용자정의태그를 지원하고 기존 HTML와는 다르게 문서의 구조를 정의할 수 있기에 정보의 재사용, 공유가 가능해졌다.[1]

XML 문서를 분석(파싱)하여 문서에 담긴 데이터를 이용하도록 한 방식은 XML을 웹상 데이터베이스로 의미부여가 가능도록 했다. 또한 더 나아가 Ajax 와 같은 기술방식에도 응용됨으로써 인터넷 정보표현 방식의 기준으로 자리 잡고 있다. 그러나 위에서 말한 사용자정의태그는 장점만큼이나 부작용이 존재한다. 단순히 문서를 재사용하는데 그치지 않고 문서들 간이 유사성, 즉 동일분류로 취급되어야할 문서들이 서로 다른 태그지정으로 어플리케이션에서 다른 분류로 인식하는 문제점을 드러냈다. 이를 해결하기 위해 동의어, 반의어, 상위어 등을 정의한 시소러스와 같은 사전식 도구를 이용한 연구를 하였으나 심층적인 단어 정의를 하지 못한 도메인에 대해서는 제대로 된 유사성을 검사하지 못하는 한계를 드러내었다.

본 논문에서는 이를 극복하고자 사용자가 직접 단어들 간의 관계에 대해 정의해놓은 사용자정의사전(온톨로지)를 이용하고자 한다.

2 장에서 관련연구로 시맨틱 웹(Semantic Web)의 정의와 XML 의 문제점에 대해 파악해보고 3 장에서 XML 문서의 유사성을 측정하기 위해 제시된 알고리즘을 이용한 프로그램의 계산과정을 설명한다. 그리고 4장에서는 제작된 프로그램의 성능을 평가하고 5장에서 결론을 내리며 앞으로의 연구방향을 제시한다.

2. 관련 연구

2.1 Semantic Web

웹상에 있는 HTML 데이터 혹은 문서들은 작성자에 의해 제공이 될 때는 정보로서 가치가 있지만 공유 가능하는, 즉 기계적인 판독이 가능 하는 데는 문제가 있다. 재사용이 가능하려면 태그와 데이터를 분리할 수 있어야 한다. 태그와 데이터가 혼재되어 있으면 구분이 어려워지며, 정확한 데이터추출이 불가능하다. 또한 같은 의미의 문서라도 표현하는 사람에 따라서 천차만별로 달라질 수 있으며 전혀 다른 별개의 정보로 취급될 수 있다. 그리고 다른 형식으로 표현이 되는 문서만큼이나 정보해석 어플리케이션도 필요하게 되므로 효율성도 극히 떨어진다. 이에 대한 기준을 마련한 것이 바로 시맨틱 웹이다. W3C(World Wide Web Consortium)에 의해 주도되고 있는 시맨틱 웹은 웹의 진화를 꾀함으로써 공유가능하고

신뢰성 높은 웹문서를 제공할 수 있다.

RDF(Resource Description Framework) 는 시맨틱 웹에서 데이터 프레임워크를 제공한다. RDF 는 구조적, 의미적으로 데이터를 기술하는데 쓰이는 단어들에 대한 메타데이터(스키마)를 제공한다. 즉, RDF 는 특정 도메인에서 쓰이는 단어에 대한 규칙성을 정의함으로써 해당 도메인에서의 어플리케이션이 RDF 를 기반으로 한 문서를 손쉽게 해석, 재사용할 수 있다.

OWL(Ontology Web Language)는 웹에서 쓰이는 언어들에 대한 특정 클래스를 정의한다. 웹문서에 존재하는 여러 언어들 간의 관계에 대해 정의하고 결론적으로 여러 가지의 언어적 표현을 RDF 메타데이터와 연결한다.

XML(eXtensible Markup Language) 는 웹상에서 표현형식을 제공한다. XML 은 사용자정의태그를 사용해 웹문서가 표현하는 정보에 대한 구조적인 틀을 마련한 것이다.

XML로 표현된 문서를 OWL 을 통해 RDF 에서 제시한 메타데이터로 해석해서 어플리케이션에서 재사용하는 이러한 과정이 바로 시맨틱 웹이다.[2]

2.2 XML의 문제점

XML은 데이터를 재사용하는데 필요한 표현방식을 제공하고 있지만 일련의 문제점을 가지고 있다. 바로 사용자정의태그 분류의 모호성이다. 자유롭게 사용자가 태그를 정의할 수 있기 때문에 실질적으로 같은 의미의 태그일지라도 완전히 다른 의미의 태그로 분류될 수 있다는 것이다. 예를 들어 물품가격을 나타내는데 <price>, <cost>, <productprice>, <goodscost> 등으로 표현할 수 있지만 여기서 productprice 나 goodscost 는 인위적으로 price나 cost 와 동일하다고 인식 가능함에 비해 기계적인 인식은 불가능하다. 이러한 문제를 해결할 수 있다면 XML 문서간의 유사성을 검사하는데 보다 더 높은 정확성을 보장할 수 있다.

3. 유사도 계산

본 장에서는 2장에서 언급한 문제점을 극복하고자 제안된 사용자정의사전을 이용하고, 사용자주요단어 등록을 통한 XML 유사도 측정 정확성을 높일 수 있는 방법에 대해 구체적으로 설명하고 이를 이용해 구현한 프로그램의 구조를 그림과 함께 기술하였다.

3.1 시스템 구조

본 논문에서 제안하는 시스템 구조는 네 단계로 나뉜다.

첫 번째, XML문서들을 읽어 파싱한 다음 태그 이름으로 DOM 트리를 구성한다.[3]

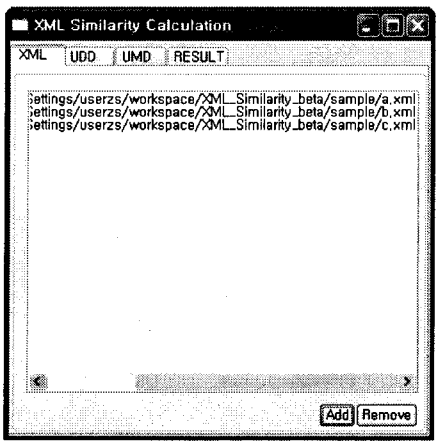
두 번째, 구성된 DOM 트리내의 태그들을 WordNet을 이용해 동의어를 추출한다.

세 번째, 사용자정의사전을 읽고, 사용자가 정의한 주요 단어들을 입력받는다.

네 번째, 파싱한 태그들과 추출한 동의어, 사용자정의사전, 사용자주요사전을 이용해 유사도를 측정한다.[4]

3.2 XML문서 파싱

비교하고픈 XML 파일을 선택하여 자바에서 지원하는 DocumentBuilder 클래스를 이용해 각 파일의 태그를 파싱하게 된다. 읽어 들여진 파일정보는 다음과 같은 멤버로 구성된 데이터객체에 저장된다.



< 그림 1> XML 문서 정보 리스트

< 표 1 > XML 데이터 저장 객체

멤버명	설명
filename	파일 이름
tags	파일 내 태그에 대한 정보배열 (다음에 언급하는 태그데이터객체를 배열형태로 담는다.)
similarityRate	평균 유사도 값(각 태그별 유사도 값을 평균 낸 값이다.
maxDepth	태그 노드의 최대깊이(모든 태그를 검색한 뒤 저장된 tags 배열객체를 바탕으로 가장 깊은 Depth 값을 추출한다. 깊이별 평균 유사도 값을 구분하기위해 쓰인다.)
root_node	루트노드(GUI 트리표현에 쓰인다.)

DocumentBuilder 에 의해 파싱된 파일은 첫 번째 루트노드부터 DFS 방식으로 탐색을 하며, 태그이름은 중복처리하지 않는다. 참고로 태그이름은 띄어쓰기, 특수문자, 구분기호를 제거하고 소문자로 변경된다.

각 태그에 대한 정보는 다음과 같은 멤버로 구성된 데이터객체에 저장되며 위에서 언급한 tags 배열객체를 구성한다.

< 표 2 > XML 태그 데이터 저장 객체

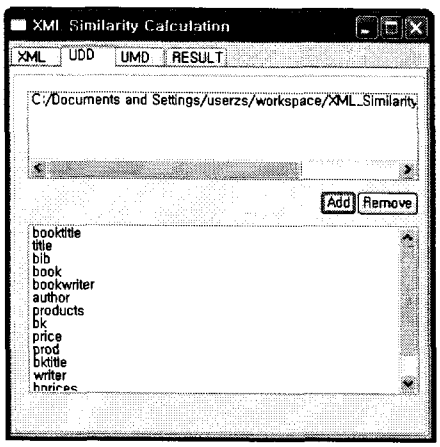
멤버명	설명
name	태그 이름
synonym	가장 유사도 값이 높은 단어(주어진 알고리즘에 의해 유사성이 가장 높은 단어를 저장한다.)
depth	노드깊이(태그의 노드깊이를 저장한다.)
s_rate	유사성값(synonym 에 저장된 단어의 위치에 따른 유사성값을 저장한다.)

3.3 WordNet을 이용한 동의어 추출

XML 파일에서 추출한 태그들의 동의어벡터를 구하기 위해 가장 널리 쓰이는 시소러스인 WordNet을 사용하였다. 단어에 대한 유의어, 동의어, 반의어 등의 정보를 가지고 있는 WordNet을 사용함으로써 추출한 태그들 간의 유사도를 측정하는 하나의 기준을 제시한다. WordNet을 사용해 동의어명사만을 중복되지 않도록 추출하였다.[5]

3.4 사용자정의사전(User Defined Dictionary)

온톨로지 제작 프로그램인 Protege(v3.3) 을 이용해 작성된 온톨로지(OWL) 파일을 읽으며, 동시에 여러 개의 온톨로지 파일을 처리할 수 있다. 직접 작성된 온톨로지를 추가적으로 시소러스에 포함시킴으로서 현재 널리 쓰이는 WordNet 의 한계를 어느 정도 극복할 수 있다. 온톨로지 파일은 Protege 에서 주어진 자바 API 를 이용해 읽어 들이게 된다.[6]

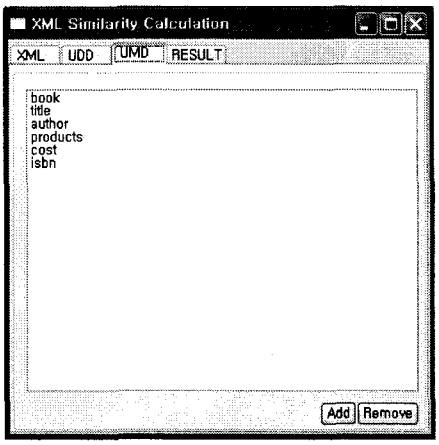


< 그림 2 > UDD 정보 리스트

읽어 들인 온톨로지 파일내의 단어들은 하단 리스트 창에 나열되고, 위의 태그처럼 중복 처리되지 않으며, 띄어쓰기, 특수문자, 구분기호 제거, 소문자로 변경된다. 온톨로지 파일이 리스트에 추가, 리스트에서 삭제될 때마다 하단의 단어 리스트 창은 갱신된다.

3.5 사용자정의주요사전 (User-defined Main Dictionary)

사용자가 정의한 온톨로지 이외에도 탐색의 대상이 되는 파일들의 담고 있는 정보의 분야에서 가장 많이 쓰이는 단어를 선정하여 직접 입력함으로써 유사성 계산의 정확도를 높일 수 있다. 예를 들어, 도서 분야에 대한 유사단어들을 WordNet 와 UDD 를 통해 추출한다고 해도 ISBN 과 같은 축약어는 유사성 정확도가 높음에도 불구하고 비교대상에서 제외될 수 있다. 또한 최근 들어 새로 등장한 단어를 포함하고 있지 않다면 사용자가 직접 정의해 주어야 한다.



< 그림 3 > UMD 입력 리스트

이와 같이 UMD 를 지정함으로써 UDD 를 좀 더 보완할 수 있다.[4]

3.6 유사도 계산

비교할 XML 문서와 UDD, UMD 를 지정한 후에 Calculate 버튼을 누르면 다음과 같이 처리가 진행된다.

(1) 선택된 XML 문서로부터 태그를 추출한다.
(2) 추출한 태그의 유사단어를 WordNet으로부터 추출한다.
(3) 추출한 단어들로 주어진 알고리즘에 의해 유사도가 계산된다.

유사도는 0.0 부터 1.0 까지 6단계로 이루어지며 다음과 같이 계산된다.[4][7]

LEVEL1 = 1.0f; : UMD 와 완전일치하면 가장 높은 값인 1.0
LEVEL2 = 0.8f; : WordNet 으로 추출한 동의어와 완전일치하면 0.8
LEVEL3 = 0.6f; : UDD내 단어와 완전일치하면 0.6
LEVEL4 = 0.4f; : WordNet 으로 추출한 동의어와 부분일치하면 0.4
LEVEL5 = 0.2f; : UDD내 단어와 부분일치하면 0.2
LEVEL6 = 0.0f; : 일치성이 없으면 0.0

```
public void calSimilarityRates() {

  // 각 파일별로 진행
for(int i=0;i<XMLInfoList.size();i++) {

 float sum = 0.0f;

      //파일당 각 태그별로 진행
 for(int j=0;j<XMLInfoList.get(i).getTags().size();j++) {

  if(matchUMD(i,j)) {}  //UMD 와의 일치여부
    else if(matchSynonym(i,j)) {}  //동의어와 일치여부
    else if(matchUDD(i,j)) {}  //UDD단어와 일치여부
    else if(submatchSynonym(i,j)) {} //동의어와 부분일치여부
    else if(submatchUDD(i,j)) {}  //UDD단어와 부분일치여부
    else
        XMLInfoList.get(i).getTags().get(j).setS_rate(LEVEL6);
         //일치하지않으면 가장 낮은 0.0으로 계산

  sum += XMLInfoList.get(i).getTags().get(j).getS_rate();
  }

XMLInfoList.get(i).setSimilarityRate(sum/XMLInfoList.get(i).getTags(
).size());
    }
}
```

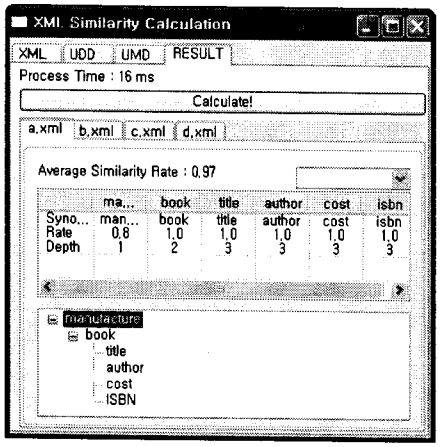
< 그림 4 > 유사도 측정 함수

하나의 태그를 가지고 가장 높은 유사도부터 판별하게 된다. 즉, UMD 와 완전 일치가 되어 1.0으로 계산이 되었을 경우하위 레벨계산은 생략하게 된다. 전체적으로 파일의 유사도가 높아질수록 성능이 향상된다.
일치여부를 처리하는 함수내용은 다음과 같다.

```
private boolean matchUMD(int fileNum, int tagNum) {

  Tag tag = XMLInfoList.get(fileNum).getTags().get(tagNum);
  String tagName = tag.getName();

  for(int i = 0; i < UMDList.size(); i++) {
    if(tagName.equals(UMDList.get(i))) {
      tag.setSynonym(UMDList.get(i));
      tag.setS_rate(LEVEL1);
      return true;
    }
  }
  return false;
}

private boolean matchSynonym(int fileNum, int tagNum) {
  Tag tag = XMLInfoList.get(fileNum).getTags().get(tagNum);
  String tagName = tag.getName();

  for(int i = 0; i < synonym_set.size(); i++) {
    if(tagName.equals(synonym_set.get(i))) {
      tag.setSynonym(synonym_set.get(i));
      tag.setS_rate(LEVEL2);
      return true;
    }
  }
  return false;
}

private boolean matchUDD(int fileNum, int tagNum) {
  Tag tag = XMLInfoList.get(fileNum).getTags().get(tagNum);
  String tagName = tag.getName();

  for(int i = 0; i < UDD_set.size(); i++) {
    if(tagName.equals(UDD_set.get(i))) {
      tag.setSynonym(UDD_set.get(i));
      tag.setS_rate(LEVEL3);
      return true;
    }
  }
  return false;
}

private boolean submatchSynonym(int fileNum, int tagNum) {
  Tag tag = XMLInfoList.get(fileNum).getTags().get(tagNum);
  String tagName = tag.getName();

  for(int i = 0; i < synonym_set.size(); i++) {
    String synonym = synonym_set.get(i);
    if(synonym.contains(tagName)) {
      tag.setSynonym(synonym);
      tag.setS_rate(LEVEL4);
      return true;
    }
  }
  return false;
}

private boolean submatchUDD(int fileNum, int tagNum) {

  Tag tag = XMLInfoList.get(fileNum).getTags().get(tagNum);
  String tagName = tag.getName();

  for(int i = 0; i < UDD_set.size(); i++) {
    String udd = UDD_set.get(i);
    if(udd.contains(tagName)) {
      tag.setSynonym(udd);
      tag.setS_rate(LEVEL5);
      return true;
    }
  }
  return false;
}
```

< 그림 5 > 유사도 측정 하위 함수

처리가 진행되면 결과가 다음과 같이 표시된다.



< 그림 6 > 결과 화면

각 문서들의 이름이 탭 형식으로 표시되며 가장 관심도가 높은 평균 유사성 값이 맨 처음 표시된다.
평균 유사성 값은 다음과 같은 공식으로 계산된다.[4]

$$\frac{\sum_{k=1}^{n} R_k}{n}$$ (n : 총 태그 수, R_k : 태그의 유사도 값)

각 태그별로 가장 유사한 단어와 해당되는 값, 문서 내에서의 노드 깊이 값으로 이루어진 테이블화면이다.[4] Synonym 으로 표시된 행은 가장 유사도가 높은 동의어를 말하며, Rate 는 그 동의어가 몇 단계에서 일치되었는가를 말한다. 그리고 Depth 는 각 태그의 노드 위치(깊이)를 말한다.
우측에 해당노드깊이이하의 태그들의 평균값을 새로 측정할 수 있게 콤보박스를 배치해 두었는데, 이를 도입한 이유는, 비교에 주가 되는 title, cost, ISBN, author 등의 태그의 유사성은 높은데 비해 그 태그들을 감싸는 상위태그는 전혀 관계없는 정의어가 등장할 수 있기 때문이다. 이에 대한 예는 3장 마지막 XML 문서의 결과분석에 언급하도록 한다. 태그의 깊이를 트리형태로도 확인할 수 있다. 태그는 중복 표시 되지 않으며, 트리형태로 표시되어있어 손쉽게 구조를 파악할 수 있다.
네 개의 문서를 가지고 실행결과를 표시하면 다음과 같다.

< 표 3 > A.XML 유사도 결과

manufacture	book	title	author	cost	isbn
manufacture	book	title	author	cost	isbn
0.8	1.0	1.0	1.0	1.0	1.0
1	2	3	3	3	3

< 표 4 > B.XML 유사도 결과

goods	bibliography	booktitle	writer	bprice	isbn
goods	bibliography	booktitle	writer	bprices	isbn
0.8	0.8	0.6	0.8	0.2	1.0
1	2	3	3	3	3

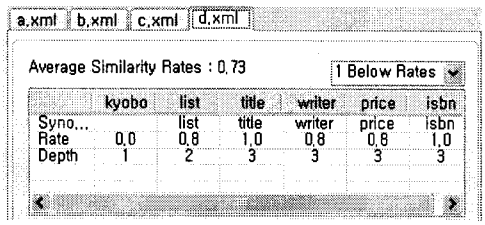
< 표 5 > C.XML 유사도 결과

products	bib	bktitle	bookw	bookc	isbn
products	bib	bktitle	bookwriter	bookc	isbn
1.0	0.8	0.6	0.2	0.6	1.0
1	2	3	3	3	3

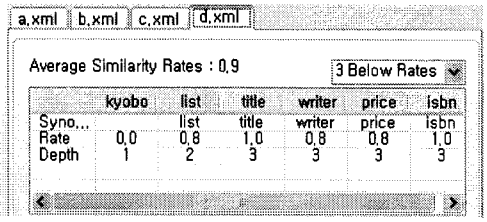
< 표 6 > D.XML 유사도 결과

kyobo	list	title	writer	price	isbn
	list	title	writer	price	isbn
0.0	0.8	1.0	0.8	0.8	1.0
1	2	3	3	3	3

첫 번째 문서는 97%, 두 번째는 70%, 세 번째는 70%, 네 번째는 73%의 유사도가 측정되었다.
이러한 결과 값은 실제 사람이 인위적으로 문서를 분석했을 때와 동일한 결과를 나타냄을 알 수 있다.
여기서 네 번째는 깊이별 차이 값을 위해 루트노드태그와 하위태그의 이름을 인위적으로 변경하였다.



< 그림 7 > D.XML 1~3 Depth 결과

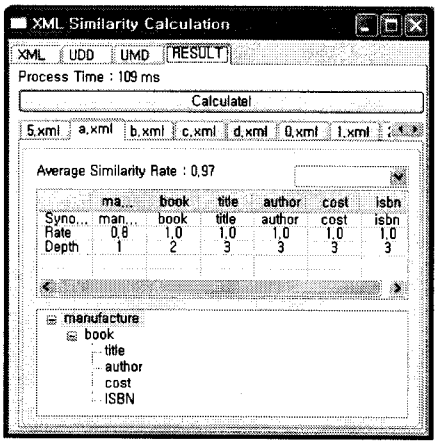


< 그림 8 > D.XML 3 Depth 결과

위와 같이 깊이 1과 2의 태그들의 이름의 유사성이 0이 나와 전체평균유사도는 73% 가 나오지만 실제적으로 정보가 담고 있는 내용은 도서에 관한 것이므로 가장 낮은 깊이의 태그들만 계산할 경우 90% 가 나오게 된다. 이렇게 부분적 유사도가 가장 높게 나오며 이러한 계산이 현실적으로 타당한 것임을 알 수 있다.

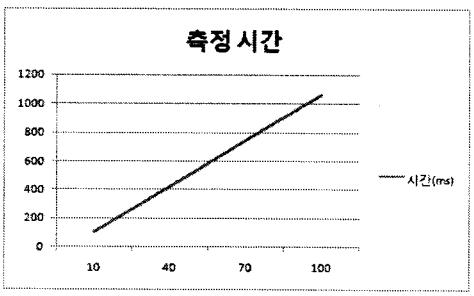
4 성능평가

3장에서는 유사도를 측정하는 프로그램의 구조를 설명하였다. 설명에 사용된 예제는 4개의 XML문서이며
이 장에서는 유사도를 계산할 문서의 개수에 따른 결과 도출시간이 어떠한지 변화도를 측정하고 그에 따른 결론을 내리고자 한다.
다음은 10개의 문서를 측정한 결과이다.



< 그림 8 > 10개 문서 측정 결과

10개의 문서를 계산한 시간은 평균적으로 100ms 의 시간이 소비되었다. 결과로 나타난 측정시간은 UDD 를 읽는 시간을 제외한 것이다. UDD 를 읽는 시간은 파일 개수에 관계없으므로 상수시간으로 계산할 수 있다.
즉, 측정시간은 주어진 XML 문서를 읽어서 파싱한 다음 주어진 태그들을 WordNet으로 부터 동의어를 추출하고 동의어, UDD, UMD와의 일치여부를 계산하는 시간을 말한다.
더욱 많은 문서 처리시간을 알아보기 위해 30개 단위로 시간을 측정하였고 그래프로 표시하면 다음과 같다.



< 그림 9 > 문서 개수별 측정 시간 결과

그래프와 같이 문서의 개수에 따라 처리시간은 O(n) 이다. 이는 실용적으로 만족할 수준의 속도임을 알 수 있다.

5. 결론, 향후연구

본 논문에서는 XML 문서들 간의 유사성 검사를 하는데 있어서 기존의 두개의 문서를 가지고 비교하는데 걸리는 많은 시간과 복잡한 방법들을 해결하기 위해 일정한 분야에서의 많은 XML문서들 간의 유사성을 검사하는 방법을 제시하였다. 이러한 방식은 빠른 시간과 XML문서의 유사성 검사의 분류가 확실해 지는 편리함을 제공해주고 있다. 하지만 주어진 정의사전은 특정 분야에 대한 정보만을 담고 있기 때문에 여러 분야에 적용 가능한 포괄적인 정의사전이 요구된다. 그리고 단순히 단어비교만이 아닌 트리구조에 대한 기계적인 유사도 분석도 필요하다.

Acknowledgement
본 연구는 21세기 프론티어 연구개발사업의 일환으로 추진되고 있는 정보통신부의 유비쿼터스컴퓨팅 및 네트워크 원천 기반기술 개발사업의 지원에 의한 것임

참고문헌

[1] "eXtensible Markup Language(XML)", http://www.w3.org/XML
[2] "Semantic Web", http://www.w3.org/2001/sw
[3] "Document Object Model(DOM)", http://www.w3.org/DOM
[4] In-sang Song, Ju-ryun Park, Ung-mo Kim,, "Semantic-based Similarity Computation for XML Document", IEEE CS(MUE2007)
[5] "WordNet 2.1", http://wordnet.princeton.edu
[6] "Protege 3.3", http://protege.stanford.edu
[7] 이정원, 이기호, "유사성 기반 XML 문서 분석 기법", 정보과학회논문지, 2002년
[8] Joe Tekli, Richard Chbeir, Kokou Yetongnon, "Semantic and Structure Based XML Similarity: The XS^3 Prototype", COMAD 2006