# 내포된 공노드를 포함하는 RDF 문서의 변경 탐지 기법

# (A Change Detection Technique Supporting Nested Blank Nodes of RDF Documents)

이 동 희 † 임 동 혁 † 김 형 주 ††

(Dong-Hee Lee) (Dong-Hyuk Im) (Hyoung-Joo Kim)

**요 약** RDF 문서들은 빈번히 갱신이 발생하므로 RDF 문서간의 변경부분을 찾아내는 것은 중요한 관심사가 된다. RDF 문서 내에 공노드가 존재할 경우 변경부분을 탐지해내려면 공노드간의 매칭을 지원하는 기법이 필요하다. RDF 문서에서 공노드는 내포된 형태로 존재하며 실제 사용되는 RDF 문서 대부분이 공노드를 포함하고 있다. RDF 문서를 그래프로 모델링하면 하나의 문서는 여러 개의 트리로 나누어진다. 따라서 문서간의 변경탐지는 동일한 루트를 가지는 트리간의 최소 비용 매칭 문제로 생각할 수 있다. 본 논문에서는 공노드에 대한 레이블링 기법을 기용하여 내포된 공노드를 포함한 RDF문서의 변경탐지 기법을 제안한다. 또한 공노드가 아닌 일반 트리플들의 비교에 있어서도 효율성을 높이는 술어 그룹화와 분할 기법을 제안한다. 실험을 통해 제안한 기법이 기존의 방법보다 더 정확하며 효율적임을 보였다.

**키워드** : RDF, 변경 탐지, 문서 갱신

***Abstract*** It is an important issue to find out the difference between RDF documents, because RDF documents are changed frequently. When RDF documents contain blank nodes, we need a matching technique for blank nodes in the change detection. Blank nodes have a nested form and they are used in most RDF documents. A RDF document can be modeled as a graph and it will contain many subtrees. We can consider a change detection problem as a minimum cost tree matching problem. In this paper, we propose a change detection technique for RDF documents using the labeling scheme for blank nodes. We also propose a method for improving the efficiency of general triple matching, which used predicate grouping and partitioning. In experiments, we showed that our approach was more accurate and faster than the previous approaches.

**Key words** : RDF, change detection, document update

## 1. 서 론

RDF(Resource Description Framework)는 웹 상의 정보를 표현하기 위한 언어로써 W3C(World Wide Web Consortium)에서 제정한 언어이다. 웹 상의 자원들을 에이전트가 자동화하여 처리 할 수 있는 형태로 설명하는 일반적인 기술규격이다[1]. 시맨틱 웹을 지향함에 따라 많은 웹 사이트들이 RDF 형태로 데이타를 제공하기 시작했고 RDF를 위한 저장소와 질의 언어들도 개발되고 있다. 대표되는 저장소로는 Sesame[2], Jena[3]가 있다. 이러한 RDF는 현재 다양한 분야에서 사용되고 있으며, 특히 생물정보학에서도 유전정보를 저장하기 위해 RDF 형식으로 표현한 데이타를 많이 사용하고 있다.

웹 상의 데이타들은 빈번히 갱신이 일어나기 때문에

· 본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT 연구센터 육성 지원 사업(IITA-2007-C1090-0701-0031)의 연구결과로 수행되었음

† 학생회원 : 서울대학교 전기컴퓨터공학부
inurisis@naver.com
dhim@idb.snu.ac.kr
†† 종신회원 : 서울대학교 전기컴퓨터공학부 교수
hjk@snu.ac.kr

논문접수 : 2006년 12월 28일
심사완료 : 2007년 9월 29일



정보과학회논문지: 데이타베이스 제34권 제6호(2007.12)

갱신이 일어난 부분을 찾아내는 변경 탐지 툴이 필요하다. 일반적으로 갱신된 부분이 전체 내용에서 차지하는 비중은 보통 5%이내로 작다. 따라서 새로운 데이타를 이용해서 처음부터 다시 작업을 하는 것보다는 변경 내용을 찾은 후 기존 데이타에 변경내용을 적용하는 것이 좀 더 효율적일 것이다. 특히 전체 데이타 크기가 매우 클 경우에는 더욱 그러하다. 빈번히 갱신되는 데이타의 예로, Gene Ontology[4]는 매일 갱신이 일어나고 있으며 Uniprot 데이타[5]는 2주 단위로 갱신되고 있다. 하지만 현존하는 많은 웹 데이타의 배포형태는 갱신된 부분에 대한 별도의 정보 제공 없이 단순히 새로운 데이타만을 제공하고 있는 경우가 많다.

따라서 변경 탐지 툴을 이용하면 다음과 같은 장점이 있다[6].

• 문서의 버전관리 및 관련질의

문서를 사용하다 보면 문서를 버전 단위로 관리할 필요성이 생기게 된다. 역사를 기록하듯 계속적으로 변화되고 있는 문서의 각 버전들을 기록하는 것이다. RDF는 온톨로지와 대용량의 RDF 인스턴스 데이타 제작과 같은 다수의 인원의 협력을 통해 수행되는 작업에 많이 사용되고 있다. 따라서 각각의 버전들을 기록해둘 필요가 있게 된다. 차후에 기록해둔 이전의 데이타에 질의를 하여 필요한 정보를 얻거나 기록들간의 차이를 구하고자 하는 등 여러 가지로 필요가 생길 때 유용하게 사용할 수 있다. 이때 저장과 여러 버전에 대한 질의를 할 때 변경내용을 찾아서 이용한다면 각 버전마다 개별적으로 새로이 저장된 것을 이용하는 것보다 훨씬 효율적이다.

• 변경내용 파악을 통한 이점

텍스트 문서를 비교하는 Diff의 경우 공동작업의 프로젝트에서 버전관리의 일부로 많이 사용되고 있다. 즉 동일한 부분을 여러 사용자가 수정하는 경우가 생기기 마련이다. 이때 각자가 작업한 내용을 모아 하나로 모으려고 하면 충돌이 발생한다. 이 때 변경 탐지 툴을 사용하게 되면 바뀐 부분만 확인할 수 있고 그 내용만을 파악하여 그에 맞게 수정할 수 있게 된다. 꼭 충돌이 일어나는 내용이 아니어도 각각의 정확한 변경내용만을 찾아서 저장관리하면 버전 갱신에 따라 일어난 새로운 버그 파악 등 추후 여러 가지로 도움이 될 것이다.

• 점진적 데이타 갱신

RDF 문서를 작성하는 측면이 아닌 그 배포 파일을 받아서 사용하는 측면에서도 활용할 수 있다. 많은 문서가 일일, 월간으로 주기적으로 갱신이 일어나지만 변경내용을 별도로 제공하지 않는 경우가 있다. 하지만 전체 데이타에서 변경되는 데이타의 양의 비중은 일반적으로 크지 않기 때문에 전체 데이타의 크기가 매우 클 경우에 저장소에서 기존 내용을 지우고 새로운 데이타로 저장하는 것 보다 변경내용을 찾아서 이를 이용해 갱신하는 것이 더 효율적이다.

RDF 문서는 주어(subject), 술어(predicate), 목적어(object)의 트리플 구조를 갖는 일련의 문장들로 이루어져 있으며, 이 트리플의 집합은 RDF 그래프로 불리며 그림 1과 같이 그래프로 표현된다. 이때 주어로 URI, 술어로 URI, 목적어로 URI 또는 그림과 같이 문자열(Literal)을 보통 사용한다. 이런 URI, Literal로만 트리플의 구성하게 되면 텍스트 비교를 통해서 갱신 전후 문서의 두 트리플 집합의 변경내용을 쉽게 구할 수 있다. 하지만 그림 1의 노드 _:1과 같은 공노드(blank node)를 가지게 되면 노드는 고유한 값을 가지지 않으므로 파싱 시 시스템이 자동으로 부여하는 아이디를 갖는다. 따라서 같은 공노드일지라도 아이디가 다를 수 있으므로 공노드 매칭이 어렵게 된다.

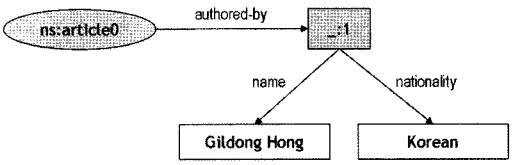



그림 1 RDF 모델의 예

본 논문에서는 RDF 문서의 변경탐지를 어렵게 하는 공노드 매칭 문제를 해결하기 위해 실제 RDF 문서에서 공노드의 사용 패턴을 분석하여 이를 이용한 RDF 문서 변경 탐지 기법을 제안한다. 추가적으로 문장에서 주어, 술어, 목적어의 사용 패턴을 이용한 변경 탐지 기법도 함께 제안한다. 제안하는 기법에서 실재 RDF 문서에 쓰이는 공노드는 최초 URI 값을 가지는 주어에 내포된 형태로 기술된다. 따라서 공노드와 연결된 서브그래프는 이 최초 주어를 루트로 하는 트리 형태로 표현이 가능하며 갱신 전후 문서에서 동일한 루트를 가지는 두 트리를 레이블링 기법을 사용하여 비교함으로써 공노드간의 최소비용 매칭 결과를 이용하여 변경 내용을 찾는다. 또한 RDF 문서에서 사용되는 문장에서 주어, 목적어에 비해서 사용되는 술어의 가지 수가 매우 적음을 확인하고 단순히 두 문서의 트리플 집합의 차이를 구하는 것이 아니라 술어 그룹화와 분할을 통해 두 문서의 차이를 구하는 연산을 수행하도록 한다.

본 논문의 구성은 다음과 같다. 2장에서는 변경탐지 기법과 관련된 선행 연구들에 대해 살펴본다. 3장에서는 효율적인 변경탐지를 위해 우선 RDF 문서의 특성을 살펴본다. 4장에서는 본 논문에서 제안하는 기법의 구체적 해결 절차에 대해 설명한다. 5장에서는 실험을

통해 효율성을 보이고, 6장에서는 결론을 내는 것으로 마무리한다.

## 2. 관련연구

문서의 변경 탐지 방법은 두 텍스트 파일간의 변경 내용을 찾는 것뿐만 아니라 HTML, XML 등 여러 형태의 문서에 대해서 다양한 방법이 소개되고 있다. 그 중 LCS(Longest Common Subsequence) 알고리즘을 이용하는 GNU diff[7]가 많이 쓰여지고 있다. 하지만 단순 텍스트가 아닌 구조화된 문서 형태의 경우에는 적용될 수 없기 때문에 XML를 비롯한 각 문서의 구조적 특징을 이용한 다양한 변경 탐지 기법이 연구되고 있으며 RDF에서도 관련 연구가 진행되고 있다.

웹 상의 문서교환 형태로 많이 사용되고 있는 XML 문서의 경우에도 변경 탐지 기법에 대해 많은 연구가 진행되고 있다[8]. XML문서는 트리로 표현할 수 있으므로 부모, 자식 간의 관계 등 트리의 특성을 이용하여 XML문서의 변경내용을 찾는 방법이 사용된다. 하지만 RDF 문서는 그래프이지만 트리로 표현할 수 없기 때문에 XML에서 사용되는 기존 방법을 단순히 적용하기에는 어려움이 있다. 본 논문에서는 [8]에서 사용한 시그니처와 매칭 기법을 RDF 문서의 공노드에 맞게 적용함으로써 공노드 매칭에 사용하였다.

Delta[9]에서는 RDF 문서가 갱신되었을 때 어떤 내용에 갱신이 일어났는지 갱신 정보를 표현하는 방법을 제안하고 있다. 이를 통해 변경된 내용을 표현한 패치 파일을 생성함으로써 갱신 정보의 배포, 동기화 등에 활용하자는 것이다. 더불어 고유한 ID를 가지지 않는 공노드에 대한 매칭에 대해서 제한적인 방법에 대해 언급하고 있다. 즉 RDF 문서에 사용되는 술어가 *owl:FunctionalProperty, owl:InverseFunctionalProperty*를 만족하는 경우에 한해서 지원한다. 하지만 이러한 제한요소를 사용하는 문서는 제한되어 있으므로 실재 사용되는 많은 RDF 문서에는 사용에 어려움이 있다.

Signing RDF[10]에서는 암호화를 위해서 동일한 RDF 그래프는 유일한 하나의 형태로 표시되어야 하기 때문에 유일한 표준형으로 변환하는 방법을 설명하고 있다. 우선 각 트리플을 사전 순으로 정렬한다. 변경 탐지를 어렵게 하는 공노드 매칭을 위해서는 기존의 ID를 무시하고 정렬한 후 새로이 ID를 부여하여 두 문서에서 동일한 ID를 가진 공노드를 동일한 것으로 여긴다. 단순히 두 RDF 그래프를 텍스트 Diff를 이용해 비교하는 것보다 이 방법을 이용해서 두 RDF 그래프를 표준형으로 변환한 다음 두 표준형을 비교하면 더 정확하게 차이를 구할 수 있다[8]. 하지만 문서가 갱신이 되면서 새롭게 트리플의 추가, 삭제로 인해 공노드에 부여되는 ID가 달라지게 되면 실제보다 더 많은 차이를 찾게 되는 문제점을 가지고 있다.

Jena[3] 라이브러리에서는 변경 내용을 구할 때 공노드를 고려하지 않고 단순히 차집합을 구하는 방법을 사용한다. 즉, 동일한 공노드일지라도 파싱시에 부여되는 ID값이 매번 달라지기 때문에 같은 노드인지 확인하는데 어려움이 있어 이점을 고려하지 않는다. 그래서 실재 변경된 내용이외에 공노드를 포함한 모든 트리플은 변경된 것으로 찾게 되는 문제점을 가지게 된다.

## 3. 실제 사용되는 RDF 문서의 특성

### 3.1 실제 사용되는 RDF에서의 공노드의 형태

공노드는 실제하지 않거나 실제하지만 지금 당장은 URI를 부여할 정도는 아닐 때 사용하며 RDF 문서 작성시 자원의 제한요소(restrictions)을 기술할 때 문서 기술 표현상 사용할 수 밖에 없다[11]. 공노드는 URI는 없지만 RDF 저장소에서 다른 자원과 구별하기 위해서 자동으로 별도의 아이디를 부여 받는다. 하지만 URI와 같이 모든 문서에서 유일한 아이디를 가지는 것이 아닌 저장소에서 단지 다른 자원과의 구별을 목적으로 부여하는 아이디이므로 같은 RDF문서의 공노드라 하더라도 부여되는 아이디가 달라질 수 있다. 따라서 두 공노드를 비교할 때 단순히 그 아이디를 비교하는 것은 아무런 의미가 없다. 이를 해결하는 변경 탐지 방법을 찾기 위해 실제 문서에서 어떻게 사용되고 있는 지 살펴보고자 한다.

그림 2와 같이 RDF 문서는 RDF/XML의 형태로 많이 기술되고 있고 그 그래프 표현은 그림 3과 같은 형태로 많이 표현된다. 앞서 언급하였듯이 RDF 문서는 그래프이지만 "John ns:brother Jack"과 같은 트리플은 문장은 모두 URI로 구성되어 있기 때문에 텍스트 비교를 통해 쉽게 변경되었는지 아닌지를 쉽게 확인할 수 있다.

변경 탐지를 어렵게 하는 공노드의 사용 패턴을 살펴보면 그림 3에서 노드 _:1, _:2, _:3, _:4와 같이 ns:pet의 목적어이면서 ns:mother, ns:size의 주어로 사용되고 있고 URI와 같이 모든 문서 내에서 고유성을 가지는 ID값이 아닌 저장소의 필요에 의해 부여된 임시 아이디가 부여되어 있다. 또 각 공노드는 나가는 간선은 여러 개지만 들어오는 간선은 단지 하나가 있음을 알 수 있다. 왜냐하면 그림 3에서 보듯이 공노드가 생성되는 부분은 문서 전체에서 단 한번 밖에 사용되지 않는 내포된 형태로 기술되고 있기 때문이다. 즉 _:1의 공노드가 생성되는 지점을 자세히 살펴보면 그림 4(a)와 같이 공노드를 생성하게 하는 <*rdf:Description*>이 문서 기술시에 문서에서 한번밖에 사용될 수밖에 없는 별도의 ID

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:ns="http://example.org/ns/"
  xml:base="http://example.org/housepet/">
  <rdf:Description rdf:about="John">
    <ns:brother rdf:resource="Jack"/>
    <ns:pet>
      <rdf:Description>
        <ns:mother>Kate</ns:mother>
        <ns:size>small</ns:size>
      </rdf:Description>
    </ns:pet>
    <ns:pet>
      <rdf:Description>
        <ns:mother>Kitty</ns:mother>
        <ns:size>small</ns:size>
      </rdf:Description>
    </ns:pet>
  </rdf:Description>
  <rdf:Description rdf:about="Jack">
    <ns:brother rdf:resource="John"/>
    <ns:pet>
      <rdf:Description>
        <ns:mother>Kate</ns:mother>
        <ns:size>small</ns:size>
      </rdf:Description>
    </ns:pet>
    <ns:pet>
      <rdf:Description>
        <ns:mother>Kitty</ns:mother>
        <ns:size>large</ns:size>
      </rdf:Description>
    </ns:pet>
  </rdf:Description>
</rdf:RDF>
```

그림 2 애완동물에 관한 RDF/XML 문서

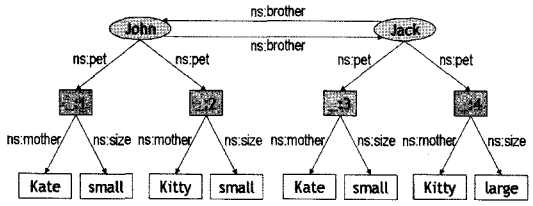



그림 3 그림 2의 그래프 표현

가 없는 내포된 형태로 기술되기 때문이다. 이 때문에 그림 3에서 공노드 _:1과 같이 들어오는 간선이 하나 밖에 없는 형태의 그래프로 표현됨을 알 수 있다.

하지만 RDF/XML에서는 *rdf:nodeID*를 이용해서 문서 작성시에 사용자가 직접 공노드에 ID를 부여함으로써 하나의 공노드가 여러 번 참조될 수 있게, 즉 목적어로 여러 번 사용될 수 있으므로 그래프로 표현될 때 들어오는 간선이 하나 이상이 될 수 있게 할 수 있다. 물론 문서 작성시 그림 4(b)와 같이 공노드를 다른 곳에서 참조하기 위해 ID를 부여하였지만, 실제 저장소에서 파싱할 경우에는 이 ID값에 고려하지 않고 별도의 ID가 새로이 부여될 수 있다. 그래서 이 경우도 저장할 때 마다 공노드의 ID는 달라 질 수 있다. 이 방법은 RDF 모델의 일부가 아니라 RDF/XML에서만 유일한 형태이고 또한 *rdf:nodeID*를 사용하는 여러 문서를 합병할 경우

```
<rdf:Description rdf:about="John">
    <ns:pet>
      <rdf:Description>
        <ns:mother>Kate</ns:mother>
        <ns:size>small</ns:size>
      </rdf:Description>
    </ns:pet>
</rdf:Description>
```

(a) 내포된 공노드(Nested blank node)

```
<rdf:Description rdf:about="John">
    <ns:pet rdf:nodeID="pet1"/>
</rdf:Description>
<rdf:Description rdf:nodeID="pet1">
    <ns:mother>Kate</ns:mother>
    <ns:size>small</ns:size>
</rdf:Description>
```

(b) rdf:nodeID를 이용한 공노드 사용

그림 4 공노드의 표현 예

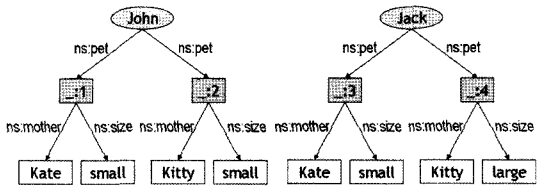
에 서로간의 충돌이 발생할 수 있다. 이렇게 공노드를 여러 곳에서 사용할 경우에 공식적인 URI를 사용하는 것이 더 옳은 방법이 된다[11]. 그래서 일반적인 RDF 문서 작성시에는 대부분 내포된 공노드의 형태로 표현하고 있다.

따라서 RDF 문서의 효과적인 변경 탐지를 위해서 실제 문서 작성시에 사용되고 있는 내포된 공노드의 특징을 이용할 필요가 있다. 내포된 공노드는 들어오는 간선이 하나 밖에 없으므로 공노드를 포함하고 있는 주어와 술어가 중요하다. 또한 공노드를 포함하고 있는 서브그래프는 결국 공노드를 싸고 있는 최초 URI 노드를 루트로 하고 단말노드는 URI, Literal로 하는 하나의 작은 트리형태가 됨을 알 수 있다. 따라서 그림 3의 RDF 그래프를 변경 탐지가 쉬운 트리플을 제외하고 공노드를 포함한 작은 서브 그래프들로 나누면 각 서브그래프는 그림 5(a)와 같이 트리로 표현할 수 있다.

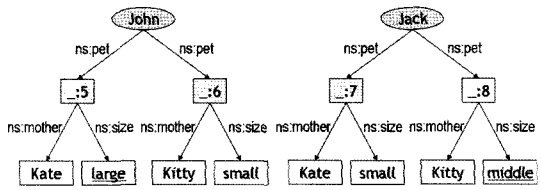
이때 그림 5의 (a)에서 (b)로 문서가 갱신된 경우 내포된 공노드의 특징을 고려하지 않고 공노드의 매칭을 할 경우 공노드의 세부 기술 내용, 즉 트리에서 자식들을 비교하는 방법을 통해 공노드를 매칭하게 되면 불필요하게 변경 전 모든 공노드를 변경 후 모든 공노드와 일일이 비교해야 한다. 하지만 내포된 공노드의 특성을 이용하게 되면 동일한 루트를 가지는 각각의 트리 내에서만 비교하게 되어 더 빠르게 공노드 매칭을 할 수 있게 되고 결국 더 빠르게 변경내용을 찾을 수 있게 된다.

### 3.2 RDF 트리플 집합의 특징

하나의 RDF 문서 내에는 수많은 웹 자원들을 기술하고 있고 그에 따라 한 문서 내에 서로 다른 URI, Literal 값을 가진 다양한 자원들이 있다. 하지만 하나의 RDF 문서는 특정 주제에 대한 자원들의 정보를 기술하




(a) 그림 3의 공노드 관점의 트리 표현




(b) 그림 3의 갱신된 트리 표현

그림 5 공노드를 포함하는 RDF 문서의 트리 표현

고 있기 때문에 기술하고자 하는 자원이 늘어나더라도 그 기술하는 술어의 종류는 일정한 규칙을 따르는 몇 가지에 한정되게 된다. 즉, 문서 전체를 놓고 보면 주어, 목적어로 쓰이는 자원들의 서로 다른 유일한 자원들이 많지만 술어의 경우에는 문서가 아무리 커지더라도 실제 사용되는 그 수가 제한되어 있다.

실제로 많이 사용되는 Gene Ontology(약 23MB)와 Uniprot Taxonomy(약 108MB) 문서에서 사용되는 주어, 술어, 목적어로 사용되는 유일한 자원 수를 비교해 보면 그림 6과 같다. RDF 문서에서 주어, 목적어로 사용되는 자원의 종류는 그림에서 보듯이 각각 26만, 40만 가지이지만 술어로 사용되는 용어의 수를 보면 각 12, 13개로 그 종류가 매우 적음을 알 수 있다. 따라서 술어를 통한 그룹화 방법을 이용한다면 비교대상이 되는 트리플의 수가 줄어들기 때문에 좀 더 효율적인 RDF 문서의 변경 부분을 찾아 낼 수 있다.

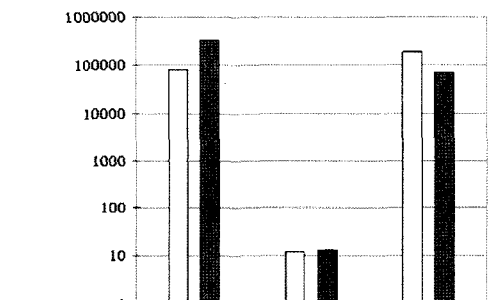



그림 6 GO, Uniprot Taxonomy RDF문서의 트리플의 자원 수 비교

## 4. 변경 탐지 기법

### 4.1 탐지 기법 개요

두 RDF 문서가 주어졌을 때, 두 문서가 서로 다른지 확인하고 다를 경우 그 변경 내용을 확인하여 패치를 만든다. 이때 공노드가 포함된 트리플은 다른 문서의 공노드가 포함된 트리플 집합과 비교하며 주어, 술어, 목적어가 모두 이름이 있는 URI, Literal로 구성된 트리플은 나머지 문서의 이름 있는 트리플 집합과 비교를 한다. 탐지기법의 개요는 다음과 같다.

**1. 파싱과 레이블링**

먼저 두 RDF 문서를 각각 RDF 파서를 이용하여 파싱한다. 파싱 과정에서 주어, 술어, 목적어가 모두 이름이 있는 URI, Literal 일 경우에는 우선 술어 값으로 그룹화한 후 주어, 목적어를 이용해 해쉬 분할한다. 목적어에 공노드가 있을 경우에는 RDF의 특성을 이용하여 공노드가 속해 있는 서브그래프를 트리로 간주하여 주어와 술어 값을 이용해 공노드를 레이블링하고 별도로 저장한다.

**2. 공노드 매칭**

공노드가 포함된 트리플에서는 나머지 문서에 있는 같은 공노드와 비교하여 매칭하는 것이 중요하다. 공노드를 포함한 작은 트리로 표현했을 때 동일 루트내에서 같은 레이블의 공노드끼리만 비교하여 매칭한다. 사용되는 매칭 기법에 대해서는 4.3절에서 구체적으로 설명을 하고자 한다.

**3. 나머지 트리플 간의 차집합 계산**

공노드가 포함되지 않은 일반 트리플간의 비교는 간단히 차집합 계산을 통해 이루어진다. 하지만 좀더 효과적인 방법으로 계산하기 위해 미리 파싱시 술어를 통한 그룹화, 주어, 목적어를 이용한 해쉬 분할을 하였다. 이를 이용하여 각 문서에서 동일한 술어 그룹, 동일한 해쉬 분할내에 있는 트리플끼리만 서로 비교한다. 비교 대상이 되는 트리플 집합의 크기가 줄었기 때문에 분할내 트리플들을 정렬 후 합병정렬과 유사한 방식으로 변경 내용을 구한다.

### 4.2 파싱과 레이블링

4.2.1 공노드 레이블링

실제 RDF 문서의 사용형태에서 보았듯이 내포된 형태의 공노드에 대해서는 해당 공노드의 선조 중에서 최초 URI값을 가지는 주어를 루트로 하는 트리구조로 볼 수 있다. 따라서 공노드를 비교하는 것은 각각의 문서에서 동일한 URI값을 가지는 두 트리를 비교함으로써 공노드를 비교할 수 있다. 이렇게 함으로써 다른 문서의 전체 공노드와 비교하는 불필요한 연산을 줄일 수 있다. 두 트리에 포함된 공노드가 동일한 URI 값을 가지는

루트에 속해있는지 비교를 위해서 우선 RDF 문서에서 각 공노드 레이블링을 하는 방법을 정의하고자 한다.

**정의 4.1:** 트리 T에 자손으로 연결된 공노드 x가 있다고 하자.

Label(x) = Name($x_1$).Name($x_2$). ···. Name($x_n$)

이때 $x_1$는 공노드를 내포하고 있는 선조 중에 최초 URI 값을 가지는 주어인 트리 T의 루트노드이고 ($x_2$, $x_3$, ···, $x_n$)은 루트에서 공노드 x까지 오는 경로상에 있는 URI 값을 가지는 술어이다.

정의된 공노드 레이블링 기법은 [8]에서 사용한 시그니처 기법을 RDF 문서의 공노드에 맞게 변형한 것이다. 공노드는 URI 값을 가지는 자원에 대해서 기술하기 위해서 목적어로 사용되는 것이기 때문에 반드시 URI 값을 지니는 루트노드가 존재하게 되고 공노드를 기술하기 위해 다시 공노드를 목적어로 사용할 수 있지만 결국엔 URI, Literal로 기술할 수밖에 없으므로 트리의 단말노드는 공노드가 아닌 노드가 된다.

정의 4.1을 그림 5(b)의 공노드에 적용을 하면 그림 7과 같으며 정의 4.1의 레이블은 공노드가 루트가 같은 트리에 내포되어 있는 공노드인지를 비교할 수 있게 해준다. 그림 7의 공노드 _:1의 레이블은 "John.ns:pet", 공노드 _3:의 레이블은 "Jack.ns:pet"이 된다. 공노드로 연결된 트리의 레벨이 증가 할 경우는 연결된 술어를 계속 추가 해 주면 된다. 공노드 _:1와 연결된 노드 "Kate"가 Literal이 아니고 공노드라고 가정하면 "Kate"의 레이블은 "John.ns:pet.ns:mother"이 될 것이다.

4.2.2 공노드가 없는 트리플의 술어 그룹화와 분할

공노드가 없는 트리플에 대해서는 두 문서간의 차집합을 쉽게 구할 수 있다. 하지만 RDF 인스턴스 문서의 경우 파일 하나의 크기가 대용량화 되고 있고 앞으로 더 큰 용량의 문서가 나올 것이다. 그래서 좀더 효과적인 비교를 위해 앞서 말한 RDF문서에서의 술어 사용의 특징을 이용하여 먼저 술어를 이용해 그룹화하고 다음으로 주어, 목적어를 이용해 분할한다.

일반적으로 많이 사용되는 차집합 계산법을 이용한다고 하자. 자바 라이브러리에 구현된 HashSet, SortedSet 방법을 이용하면 HashSet에서는 각 트리플을 해쉬함수를 통해 구한 해쉬값을 이용해 빠르게 그 차이를 구할 수 있고, SortedSet을 이용할 경우 우선 정렬한 후 비교를 통해 그 차이를 구할 수도 있다. 하지만 HashSet에서는 해쉬 시간, 해쉬값의 충돌, SortedSet에서 정렬시간 그리고 문서를 트리플 단위로 모두 메모리에 가지고 있어야 한다는 어려움이 있다.

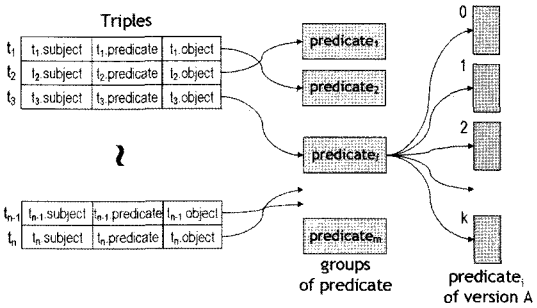



그림 8 RDF 트리플 집합의 술어 그룹화와 분할

하지만 RDF 문서에서 사용되는 술어의 가지수는 10개에서 20개 내외로 매우 적다는 것을 이용하여 그림 8과 같은 술어 그룹화와 나머지 주어, 목적어를 통해 해쉬를 하게 되면 술어의 분리를 통해 메모리 저장 비용을 줄일 수 있고 파티션 내에서만의 정렬을 하면 되므로 정렬 비용 또한 줄일 수 있다. 따라서 앞으로 RDF 문서의 용량 더욱 커지더라도 빠른 속도로 그 차이를 구할 수 있게 된다.

### 4.3 공노드 매칭

공노드를 포함한 RDF 문서에서 변경 내용을 찾기 위해서는 우선 변경 전후의 공노드의 매칭을 통해 변경 전 공노드가 변경 후의 어떤 공노드와 같은 것인지를 먼저 확인해야 한다. 이를 확인해야만 공노드와 연결된 나머지 값의 비교를 통해 정확한 변경 내용을 찾을 수 있게 된다. 앞에서 RDF 문서를 공노드를 포함한 개별 서브그래프로 나누고 이 서브그래프는 트리로 표현되므로 이는 곧 두 트리 간의 매칭 문제가 된다.

곧, 자식으로 공노드를 포함하며 같은 URI를 루트하

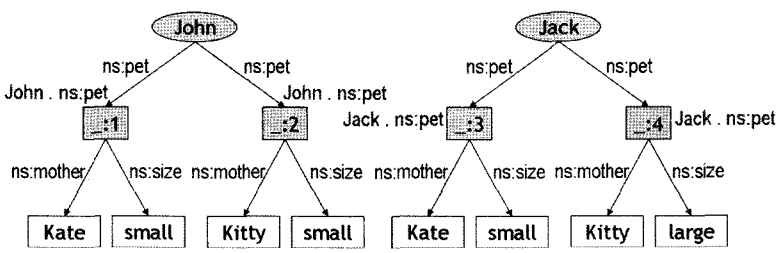



그림 7 그림 5(b)의 공노드에 대한 레이블링

는 두 트리 $T_1$, $T_2$ 간에 최소 비용 매칭을 구하는 것이다. 결국 문서 안에 있는 여러 개의 작은 트리들 중에서 루트값이 같은 두 트리를 비교하는 작업을 반복하는 것으로 볼 수 있다. 따라서 변경 비용 산출 방법을 RDF 문서에 맞게 새로이 정의하고 적절히 수정을 하면 XML 문서 비교시 쓰는 방법을 유사하게 적용할 수 있다. RDF 문장의 비용 연산에 있어서 갱신은 고려하지 않고 오직 트리플의 삭제와 삽입만 있는 것으로 생각한다[12].

**정의 4.2:** label(x)=label(y)가 같은 x, y 두 공노드가 있다고 하자. 이때 x, y는 각 문서 $D_1$, $D_2$에 속한다. 즉 $x \in D_1$, $y \in D_2$이다. Triple(sub, pred, obj)는 주어가 sub, 술어가 pred, 목적어가 obj인 트리플을 뜻한다고 하자. 이때 $D_1$에서 $D_2$로 문서가 변경되었다고 할 때 삭제되는 트리플의 집합 D와 삽입된 트리플의 집합 I는 각각

$$D = \{(pred, obj) \mid Triple(x, pred, obj) \in D_1 \text{ and } Triple(y, pred, obj) \notin D_2\}$$

$$I = \{(pred, obj) \mid Triple(x, pred, obj) \notin D_1 \text{ and } Triple(y, pred, obj) \in D_2\}$$

이고 두 공노드 x, y간의 변경 비용은 다음과 같다.

$$Cost = Dist(x, y) = n(I) + n(D)$$

정의 4.2에서 정의한 비용 연산을 그림 5에 적용을 하면 두 RDF 문서간의 변경 비용은 다음과 같다. 우선 정의 4.1을 적용한 레이블이 같은 공노드 _:1과 공노드 _:5를 매칭하고 마찬가지로 공노드 _:2와 공노드 _:6을 매칭하면 계산된 변경 비용은 2("small"이 삭제되고 "large"가 삽입)가 된다. 나머지 공노드들에 대해서도 같은 방식으로 계산하면 공노드 _:4의 "large"가 삭제되고 공노드 _:8의 "middle"이 삽입되었으므로 변경 비용은 2이고 최종 변경 비용은 4가 된다. 이러한 비용 산출 기법에 기반하여 두 문서 내에서 같은 URI를 루트로 하는 두 공노드의 변경비용 Dist(x,y)을 계산한다. 만약 공노드의 부모가 역시 공노드일 경우에는 동적 프로그래밍(Dynamic programming)기법을 이용한다. 최하단 공노드간의 변경비용을 계산하여 Distance Table에 값을 저장하고 부모 공노드로 이동하면서 저장된 값을 이용하여 부모 공노드간의 변경 비용을 계산하는 방식으로 두 공노드간의 비용을 산출한다. 동일한 레이블을 갖는 공노드가 여러 개일 경우는 공노드간의 최소 비용 매칭 기법, 즉 minimum-cost bipartite mapping을 구하기 위해 minimum-cost maximum flow 알고리즘을 이용한다. 이것을 구하는 방법이 여러 가지가 있지만 본 논문에서 구현할 때 Hungarian method[13-15]를 사용하였다. 두 트리 사이의 변경내용을 minimum-cost bipartite mapping을 이용하여 찾는 방법은 항상 최적결과를 가져오는 것은 XML 상의 트리간 변경 매칭[8]에서 이미 증명되어 사용되고 있으므로 별도의 증명은 필요치 않다.

그림 9는 두 RDF 문서에서 공노드 간의 최소 변경 비용을 구하는 알고리즘을 보여준다.

```
Bx, By : RDF 문서 X, Y의 공노드 집합
L : 두 문서의 레이블 집합
Delta : 최소 변경 비용 (삽입과 삭제된 트리플의 수)

Input : the set of blank nodes Bx and By, the set of all labels L
Output : a minimum-cost Delta
Initialize : Set the Distance Table DIST = { }

DO {
    For every label in L
        For every blank nodes x in Bx
        For every blank nodes y in By
            If (label == label(x) && label == label(y))
                /*두 공노드간의 distance를 계산*/
                Compute Distance (x,y) in DIST
                /*부모가 공노드일 때 사용하기 위해 값을 저장한다*/
                Save matchingChildrenOf(x,y) with Distance(x,y) in DIST
            else
                /* 매칭되는 공노드가 없는 경우 공노드에 연결된 트리플을 비용에 추가*/
                Add the number of triples in x to Delta
                Add the number of triples in y to Delta

    If DIST ! = 0
    /* Distance 테이블에서 Hungarian 알고리즘으로 최소 비용을 계산하여 비용에 추가*/
        Compute the minimum cost in DIST
        Add the minimum cost of to Delta
    } While (L is not empty)
```

그림 9 공노드를 고려한 최소비용 변경 탐지 기법

### 4.4 나머지 트리플 간의 차집합 계산

이제 URI, Literal 로만 이루어진 트리플 집합간의 차집합을 구한다. 파싱시 술어 그룹화와 주어, 목적어를 이용한 분할을 통해 각각의 트리플을 이미 개별 파티션에 할당했다. 이 파티션을 이용해 동일한 영역에 속한 파티션간에 비교를 한다. 두 문서에서 동일 술어의 동일 파티션에 속하는 각 파티션을 개별단위로 그림 10과 같이 정렬하고 위에서 아래로 따라가면서 병합정렬(Merge sort)과 같은 방법으로 삽입, 삭제된 트리플을 각각 구한다.

그러면 결국 그림에서 술어는 그룹이름이며 삭제된 부분은 (bbb, ccc), (bbb, ddd), 추가된 부분은 (aaa, aab), (aaa, ccd)의 주어, 술어를 구할 수 있다.

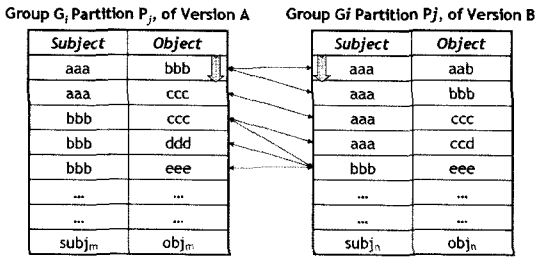



그림 10 파티션간의 차집합 계산

## 5. 실 험

본 실험은 Linux(Fedora Core 3) 환경의 2GByte 메인 메모리를 가진 Pentium 4-3.2Hz에서 수행하였다. 프로그램 구현은 Java SDK 1.5.0으로 구현하였으며 RDF 파서로는 Sesame 1.2.4에 포함되어 있는 Rio 1.0.8을 이용하였다. 공노드 매칭 기법 성능 비교에서는 공노드 매칭에 대한 정확도를 평가하였으며 이 때의 정확도는 4장에서 정의한 변경 비용을 사용하였다. 또한 그룹화와 분할을 이용한 방법의 성능 비교를 위한 실험에서는 트리플 비교에 대한 실행 시간으로 평가하였다.

실험에 쓰인 데이타로는 우선 공노드 매칭 기법을 위한 데이타로는 Gene Ontology Term DB 2006년 5월부터 9월까지의 RDF버전의 월 배포 문서(약 23MByte)와 KEGG Pathway data[16]의 버전 0.4~0.6.1까지의 대장균인 eco(Escherichia coli K-12 MG1655) 데이타(약 9MByte)의 RDF 버전[17]을 이용하였다. 트리플의 술어 그룹화와 분할을 통한 변경 탐지를 위해서는 트리플 비교에 대한 수행 시간이 중요하므로 공노드가 없는 데이타인 UniProt Taxonomy 2006년 10월 2,16일자 RDF버전 데이타(약 108MByte)[18]를 이용하였다.

### 5.1 공노드 매칭 실험

실험 방법은 Jena의 방식과 본 논문에서 제시하는 방법을 비교하였다. Jena는 앞에서 언급했듯이 변경 내용을 찾는 데 있어서 공노드는 파싱할 때 시스템에서 부여되는 ID가 달라질 수 있다는 문제 때문에 변경 전후의 공노드는 모두 다른 것으로 생각한다. 따라서 완전히 동일한 내용이라 하더라도 공노드가 포함된 부분은 변경되었다고 생각하는, 즉 공노드를 고려하지 않고 단순히 이름있는 트리플 집합 간의 차집합을 구하는 연산만을 수행한다. 하지만 본 논문에서 제시하는 방법은 공노드를 기술하고 있는 술어와 목적어, 즉 기술내용을 비교하여 변경 전후의 공노드가 동일한 노드인지 비교, 판별함으로써 공노드를 고려하여 보다 정확한 변경 내용을 탐지 한다. 트리간 최소 변경 비용 매칭이 항상 최적임이 이미 증명되어[8] 사용되고 있고 RDF 그래프에서 트리로 표현될 수 있는 개별 서브그래프에 이를 반복하여 적용한 본 논문의 방법은 항상 최적을 결과를 찾음을 앞에서 이미 밝힌바 있다. 실험결과인 그림 11에서 보듯이 동일한 변경 문서에 대해서 Jena에서는 공노드를 고려하지 않아 실재보다 매우 많은 변경 결과를 찾았지만, 본 논문에서 제시하는 방법은 그림과 같이 최소 변경 비용의 최적의 결과를 찾음을 알 수 있다.

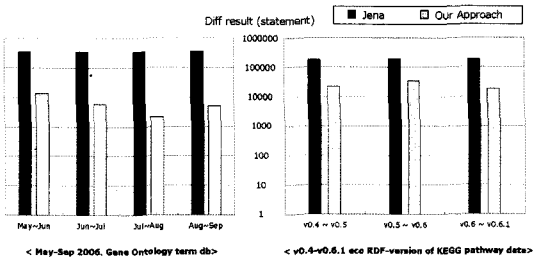



그림 11 공노드 매칭을 이용한 변경 탐지 결과

### 5.2 공노드가 없는 데이타 실험

다음 실험에서는 트리플 집합에서 일반적인 차집합 계산에서 많이 사용되고 있는 정렬을 통한 계산방법, 해쉬를 통한 계산방법과 본 논문에서 제시하는 RDF 트리플 집합의 술어 그룹화와 분할을 통한 변경탐지 방법을 비교하였다. 실험에 사용한 Uniprot Taxonomy 데이타는 공노드가 없는 데이타로써 이를 그래도 사용할 경우 정렬을 통한 계산방법에서는 데이타의 특성이 정렬시간에 영향을 미칠 수 있으므로 전체 트리플 집합을 랜덤하게 섞은 데이타를 가지고 실험하였다. 또 단일의 데이타를 사용할 경우에 사용한 데이타 특성이 해쉬된 데이타의 분포에 영향을 미쳐 해쉬를 통한 계산 방법에 영향을 끼치므로 Uniprot Taxonomy 데이타에 사용된 URI의 네임스페이스를 랜덤하게 새로이 부여하는 방법을 통해 실험 데이타만의 특성으로 인해 실험에 미치는 영향을 최소화 하였다.

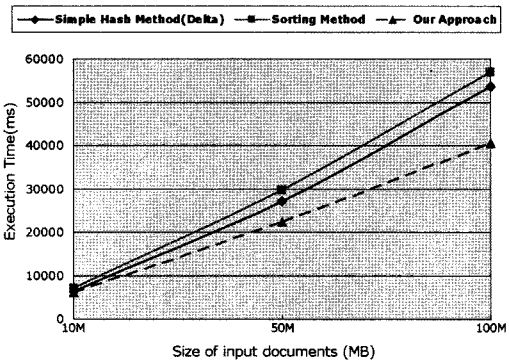



그림 12 공노드 없이 그룹-분할을 통한 변경 탐지 결과

그림 12에서 보듯이 10MByte이하의 두 문서를 비교하였을 때는 세가지 방법상에 거의 차이가 없다는 것을 알 수 있다. 그룹화와 분할의 두 단계를 거치는 작업을 처리하는 것이 오히려 더 느려질 수 있음을 보여주고 있다. 하지만 10MByte의 이상의 데이타를 비교하는데 있어서는 그룹화와 분할을 통해 변경내용을 탐지하는 방법이 더 효율적임을 알 수 있고 문서의 용량이 커질수록 속도 증가 기울기가 조금씩 줄어들어 그 차이가 점점 커지는 것을 볼 수 있다. RDF 인스턴스 데이타의 경우 하나의 파일에 관련 정보를 모두 묶어서 관리, 배포하는 경우가 많기 때문에 앞으로 RDF 문서가 더욱 대용량이 되면 본 논문에서 제시한 방법의 효용 가치는 더 커질 것으로 평가할 수 있다.

## 6. 결 론

우리는 본 연구에서 RDF 문서의 빈번한 갱신으로 인해 변경탐지툴이 필요함을 알고 변경 내용을 효과적으로 탐지하는 방법을 제안하였다. 변경 탐지를 하기 위해 우선 RDF 문서의 특징을 살펴보고 이름이 없는 공노드 사용될 경우 변경탐지에 어려움이 생기는 것을 알게 되었다. 이를 해결하기 위해 실재 RDF 문서에서 공노드를 사용할 때 그 사용 패턴을 분석하여 실재 RDF 문서에서 공노드는 내포된 형태로 사용됨을 알게 되었다. 이런 특징으로 인해 공노드가 사용된 서브그래프 부분은 하나의 URI 노드를 루트로 하는 트리 모습을 띄게 된다. 그러므로 두 문서간에 공노드를 비교할 때는 각각의 작은 서브그래프로 나누고 서브그래프를 URI값을 루트로 하는 트리로 표현하여 두 문서에 속한 두 트리를 최소비용 노드 매칭 기법을 사용하여 비교하므로써 공노드를 매칭하게 된다. 그 다음 매칭된 결과를 이용해 공노드가 속한 트리플 집합을 비교함으로써 문서의 변경 내용을 찾게 된다.

더불어 공노드가 없는 경우에도 문서에 사용되는 술어의 수가 매우 적음을 이용하여 술어를 통한 그룹화와 분할을 통해 변경내용 탐지의 성능이 되었음을 실험 결과를 통해 확인하였다.

## 참 고 문 헌

[1] Ora Lassila, Ralph R. Swick, eds. Resource Description Framework (RDF) Model and Syntax Specification. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/

[2] Broekstra, J. Kampman, A. van Harmelen, F., "Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema," In Proceedings of the International Semantic Web Conference, 2002.

[3] Wilkinson, K. Sayers, C. Kuno, H. Reynolds, D., "Efficient RDF storage and retrieval in Jena2," In First International Workshop on Semantic Web and Databases, 2003.

[4] Gene Ontology Consortium, http://www.geneontology.org

[5] UniProt, Uniprot: The Universal Protein Resource, http://www.pir.uniprot.org/

[6] Cobena, G., Abiteboul, S., and Marian, A., "Detecting changes in XML documents," In Proceedings of the International Conference on Data Engineering, 2002.

[7] GNU Diff, http://www.gnu.org/software/diffutils/

[8] Wang, Y., DeWitt, D. J., Cai, J.-Y., "X-Diff: An effective change detection algorithm for XML documents," In 19th International Conference on Data Engineering, 2003.

[9] Berners-Lee T., Connolly, D., "Delta: An Ontology for the Distribution of Differences Between RDF Graphs," http://www.w3.org/DesignIssues/Diff

[10] Carroll, J. J., "Signing RDF Graphs," In Proceedings of the International Semantic Web Conference, 2003.

[11] Shelley Powers, Practical RDF, 1st Ed., p.43, O'Reilly & Associates, 2003.

[12] Ognyanov, D., Kirakov, A., "Tracking Changes in RDF(S) Repositories," In the Proceedings of 13th International Conference on Knowledge Engineering and Knowledge Management, 2002.

[13] Harold W. Kuhn, "The Hungarian Method for the assignment problem," Naval Research Logistic Quarterly, Vol. 2, pp. 83-97, 1955.

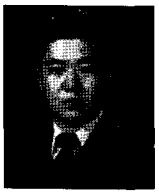
[14] James Munkres, "Algorithms for the Assignment and Transportation Problems," Journal of the Society of Industrial and Applied Mathematics, Vol. 5, No. 1, pp. 32-38, 1957.

[15] Hungarian algorithm, http://en.wikipedia.org/wiki/Hungarian_algorithm#Algorithm

[16] Kanehisa, M., Goto, S., "KEGG: Kyoto Encyclopedia of Genes and Genomes," Nucleic Acids Rese-

arch, 2000, Vol. 28, No. 1, pp. 27-30, http://www.genome.jp/kegg/
[17] W3C; Eric Prud'hommeaux, KEGG RDF Mapping, http://www.w3.org/2005/02/13-KEGG/
[18] UniProt RDF, http://expasy3.isb-sib.ch/~ejain/rdf/

이 동 희
2005년 경북대학교 컴퓨터공학과(학사) 2007년 서울대학교 전기컴퓨터공학부(석사). 2007년~현재 한국오라클 재직 중 관심분야는 데이타베이스, 시맨틱웹, 바이오인포메틱스 등



임 동 혁
2003년 고려대학교 컴퓨터교육과(학사) 2005년 서울대학교 컴퓨터공학부(석사) 2005년~현재 서울대학교 컴퓨터공학부 박사과정 재학 중. 관심분야는 데이타베이스, 시맨틱웹, 바이오인포메틱스 등

김 형 주
정보과학회논문지 : 데이타베이스
제 34 권 제 5 호 참조