상이한 구조의 XML 문서들에서 경로 질의 처리를 위한 RDBMS 기반 역 인덱스 기법

(An RDBMS-based Inverted Index Technique for Path Queries Processing on XML Documents with Different Structures)

민 경 섭 † 김 형 주 ††
(Kyung-Sub Min) (Hyoung-Joo Kim)

요 약 XML은 웹을 비롯한 모든 문서들을 표현할 수 있는 데이타 중심 표준 언어이다. XML 기반의 여러 문서 개발 도구들의 등장과 이를 이용한 사유 XML 문서의 증가, XML 문서로의 기존 데이타 변환 가속화로 인해, 우리는 대량의 서로 상이한 구조로 표현된 XML 문서들을 가지게 되었으며, 이러한 문서 집합으로부터 원하는 정보를 담은 문서를 추출해 내는 것이 중요해 졌다.

하지만, 기존의 XML 문서에 대한 연구들은 한 개의 대규모 XML 문서나 동일한 구조를 가진 문서들에 대한 저장, 검색에 초점이 맞춰져 있거나, 상이한 구조를 지원하더라도, 구조적인 정보를 이용한 빠른 검색을 지원하지 못하는 단점을 가지고 있었다. 이에, 본 논문에서는 상이한 구조를 가진 문서들에 대해서도 빠른 경로 질의를 제공할 수 있도록 지원하기 위한 새로운 기법으로, 관계형 데이타베이스 시스템을 이용한 새로운 역 인덱스를 제안하였다. 우리는 제안된 기법이 기존의 방법에 비해 높은 성능을 보임을 실험을 통해 확인하였다. 특히, 간접 포함 관계를 포함한 모든 질의에서 높은 성능을 제공함으로서, 상이한 구조를 가진 문서들에 대해 적합한 인덱스 구조임을 보여 주었다.

키워드 : XML, 데이타베이스, 역 인덱스 테이블

Abstract XML is a data-oriented language to represent all types of documents including web documents. By means of the advent of XML-based document generation tools and grow of proprietary XML documents using those tools and translation from legacy data to XML documents at an accelerating pace, we have been gotten a large amount of differently-structured XML documents. Therefore, it is more and more important to retrieve the right documents from the document set.

But, previous works on XML have mainly focused on the storage and retrieval methods for a large XML document or XML documents had a same DTD. And, researches that supported the structural difference did not efficiently process path queries on the document set. To resolve the problem, we suggested a new inverted index mechanism using RDBMS and proved it outperformed the previous works. And especially, as it showed the higher efficiency in indirect containment relationship, we argues that the index structure is fit for the differently-structured XML document set.

Key words : XML, Database, Inverted Index Table

1. 서 론

XML[1]은 웹 환경에서 운영되는 모든 데이타들을 표현할 수 있는 주도적인 데이타 표현 언어이다. XML 형식의 데이타를 생성, 변형, 조작하기 위한 여러 표준들[2,3]과 툴들[4,5]이 등장하면서 많은 사유(Proprietary) XML 문서들이 생성되고 있다. 또한, 비즈니스에서 필요로 하는 많은 문서들과 웹 사이트들의 내부 데이타도 XML 형식으로 변화하고 있다.

· 본 연구 결과는 서울대 Bk-21 정보기술 사업단과 정보통신부 ITRC (e-Biz 기술 연구센터)에 의해 지원되었음
† 비 회 원 : 서울대학교 컴퓨터공학과
ksmin@oopsla.snu.ac.kr
†† 종신회원 : 서울대학교 컴퓨터공학과 교수
hjk@oopsla.snu.ac.kr
논문접수 : 2002년 10월 15일
심사완료 : 2003년 3월 27일

이러한 상황에서, 유사한 정보를 가진 대규모의 XML 문서들은 서로 상이한 구조를 가지며, 이들로부터 필요한 정보를 빠르게 추출하는 것은 매우 중요한 이슈가 되고 있다. 예를 들어, 웹으로 자동차 관련 정보를 제공하는 많은 사이트들은 서로 유사한 정보를 제공하지만, 내부적으로는 서로 상이한 구조의 XML 문서들을 가질 수 있다. 한편, 지식 관리 시스템을 사용하는 여러 구성원들은 동일한 주제를 가진 여러 상이한 구조의 문서들을 등록할 수 있다.

XML 문서에 대한 효율적인 경로 검색 지원을 위해 여러 XML 인덱스들이 제안되었다[6-8]. 이들은 한 개의 대규모 XML 문서나 동일한 구조를 가진 여러 XML 문서들에 대한 효율적인 경로 검색 지원을 목표로 하고 있다. 그러므로, 구조가 다른 여러 XML 문서들로부터 원하는 경로를 찾기 위해 이러한 인덱스들을 사용하는 경우, 우리는 각 XML 문서들에 대해 따로 인덱스를 구성하고, 모든 인덱스들을 검사해야 한다. 게다가, 찾는 경로가 조상 자손(ancestor descendant)관계로 표현되는 경우, 구축된 인덱스들의 모든 노드들을 방문해야만 원하는 경로를 찾을 수 있어, 검색 성능이 급격히 떨어지게 된다. 이를 보완하기 위한 인덱스가 제안되기도 하였으나, 루트가 아닌 경로 중간에 조상 자손 관계가 나타나는 경우, 성능이 떨어지는 단점을 가진다[9].

한편, 구조가 다른 XML 문서들을 함께 검색할 수 있는 통합 인덱스들[10,11,12,13,14,15]도 제안되었는데, 이들은 정보 검색 분야에서 사용한 용어 기반 역 인덱스에 XML 의 구조적 성질을 반영시켜 확장하는 방식을 취하였다. 또한, 인덱스내의 모든 노드 정보들을 데이타베이스내의 테이블에 저장하여 안정적으로 관리될 수 있도록 구성하였으며, 데이타베이스 시스템에서 제공하는 효율적인 질의 처리 지원 기능들을 경로 검색에 이용함으로써 경로 검색을 위한 추가적인 구현을 막고 효율적인 검색을 지원하였다. 하지만, 제안된 인덱스들에서 [10,11,12,13]은 문서의 수가 증가함에 따라 비교/검색해야 할 인덱스 내의 데이타가 증가하게 되어 검색 성능이 저하되며, [14,15]은 구조가 다른 문서들이 늘어날수록 성능이 떨어지는 단점을 보인다. 이러한 특징은 대량의 XML 문서들에 대한 경로 검색을 필요로 하는 상황에서, 이들을 적용하게 어렵게 만드는 요인으로 작용한다.

이에, 본 연구에서는 구조가 다른 문서들이 대량으로 발생하는 상황에서도 빠른 경로 검색 성능을 보장하는 새로운 데이타베이스 기반 역 인덱스 기법을 제안하였으며, 실험을 통해 제안한 기법이 기존의 인덱스들에 비해 높은 검색 성능을 제공함을 증명하였다.

본 논문의 구성은 다음과 같다. 2장에서는 본 연구와 관련된 여러 선행 연구들을 살펴보고, 3장에서는 상이한 구조의 XML 문서들에 대한 문서 모델을 소개한다. 4장에서는 3장의 문서 모델을 기반으로 역 인덱스 구조와 질의 변환 방식을 소개한다. 5장에서는 성능 평가를 통해, 기존의 연구와의 분석 결과를 보인다. 6장에서는 결론과 향후 연구 방향에 대해 기술한다.

2. 관련 연구

XML 인덱스에 대한 연구는 활발히 진행되어 왔다[6,7,8,9,10,11,12,13,14,15]. 본 장에서는 이러한 인덱스들 중, 상이한 구조의 XML 문서들에 대한 통합 검색을 지원하는 역 인덱스 기반 인덱스들에 대해 중점적으로 살펴본다.

[13]은 여러 가지 테이블 기반 역 인덱스 방식을 소개하였다. 먼저, XML 문서 내의 엘리먼트와 애트리뷰트 발생 정보를 추출하여, Edge 라는 테이블에 저장하는 방식을 제안하였다. Edge 테이블은 부모 엘리먼트 번호, 자식 엘리먼트 이름, 자식 엘리먼트의 순서 번호, 엘리먼트가 포함하고 있는 값의 타입, 그리고 대응되는 값(혹은 참조)으로 구성된다. Edge 테이블로부터 임의의 경로를 검색하기 위해, 우리는 찾고자 하는 경로에 포함된 엘리먼트와 애트리뷰트의 수만큼 Edge 테이블을 복제하여 자기 조인(self-join)을 수행해야 한다. 이 때, 복제된 Edge 테이블들은 조인에 참여하지 않는 불필요한 데이타들을 포함하며, 모든 데이타에 대한 비교 연산을 수행하므로, 많은 비용을 초래한다. 이러한 단점을 해소하고자, [13]은 각 엘리먼트와 애트리뷰트 단위로 테이블을 생성하는 방식인 이진(Binary) 접근 방식을 함께 제안하였다. 이 방식에서 경로 검색은 조인에 참여하는 엘리먼트와 애트리뷰트의 발생 정보만을 참조하므로, Edge 테이블 방식에 비해 비용이 절약된다. 하지만, 경로를 검색하기 전에, 경로를 구성하는 각 엘리먼트와 애트리뷰트에 대응되는 테이블이 존재하는 지를 파악하기 위해 추가 질의들을 수행해야 하며, 상이한 구조를 가진 문서들에 대해 매우 많은 분산된 테이블을 관리해야 하는 번거로움을 가진다. 또한, 각 엘리먼트와 애트리뷰트의 발생 정보 간의 비교 연산을 통해 경로 검색을 수행하므로, 문서의 수가 증가함에 따라, 더욱 많은 검색 시간을 필요로 한다는 단점이 있다. [13]은 위 두 방식과 함께, 각 엘리먼트나 애트리뷰트가 포함하고 있는 값을 따로 테이블에 관리하는 방식과 함께 관리하는

방식을 제안하였다. 이는 성능 면에서 약간의 차이를 가지고 있으나, 앞서 기술한 문제점을 해결하지 못한다.

[10]은 XML 문서에 나타나는 모든 단어들에 대해 순서대로 번호를 부여하고, 이를 이용하여 발생 정보를 구성하는 방식을 사용하였다. 인덱스는 엘리먼트와 애트리뷰트의 발생 정보를 포함하는 Element 테이블과 값의 발생 정보를 포함하는 Term 테이블로 구성된다. Element 테이블은 각 엘리먼트나 애트리뷰트가 나타난 문서의 번호, 시작 태그 위치 번호, 종료 태그 위치 번호, 경로 상의 레벨 번호로 구성된다. Term 테이블은 값이 나타난 문서의 번호, 값의 발생 위치 번호, 경로 상의 레벨 번호로 구성된다. 이 방식은 앞서 살펴본 [13]과 인덱스를 구성하는 발생 정보의 형태가 서로 다르나, 모든 엘리먼트와 애트리뷰트의 발생 정보를 이용하여 경로를 검색하므로, [13]과 마찬가지로 많은 검색 비용을 필요로 한다. 이에, [10]은 이러한 문제점을 완화시키기 위해 다중 술어 머지 조인(MPMGJN : Multi Predicate Merge Join)이라는 새로운 조인 방식을 제안하였다. 이 조인 기법은 경로의 존재 여부를 파악하기 위한 여러 술어(Predicate)들을 함께 처리할 수 있는 조인 방식으로, 기존의 조인 알고리즘들(예:Merge-Join, Nested-Loop Join)에 비해 높은 검색 성능을 보인다. 최근에는, [10]의 MPMGJN을 개선한 연구들도 진행되었다[11,12]. 하지만, MPMGJN을 포함한 이들 조인 기법들은 여전히 엘리먼트와 애트리뷰트의 발생 정보를 기반으로 경로를 검색하므로, 문서의 수와 크기가 증가함에 따라, 검색 비용 또한 비례적으로 증가하게 되어, 대용량의 문서 검색에서는 적용하기 어렵다.

위의 두 방식이 발생 정보를 기반으로 경로를 검색하는 방식을 취하는 데 반해, [14,15]는 경로 문자열을 기반으로 경로 검색을 지원한다. 두 방식은 모두 [10]과 유사한 번호 부여 방식을 사용하여 각 단어에 일련번호를 할당한다. 그리고, 문서를 순회하면서, 루트 엘리먼트로부터 시작하는 모든 경로에 대해 경로 문자열을 생성한다. 예를 들어, <author> <name> <lastname> Min </lastname> </name> </author>과 같은 XML 문서가 있다고 할 때, "#/author", "#/author#/name", "#/author#/name#/lastname"과 같이 3개의 경로 문자열이 생성된다. 그리고, 만들어진 경로 문자열들은 경로 식별자와 함께 Path 테이블에 저장된다. Path 테이블은 관리되는 모든 XML 문서들에서 서로 다른 경로들만을 포함하고 있다. 예를 들어, 문서 1에서 "#/author" 문자열이 생성되고, 문서 2에서 동일한 경로 문자열이 생성되더라도, 경로 문자열은 한 개만 Path 테이블에 저장된다. 그러므로 Path 테이블은 모든 문서들에 대한 구조적 요약 정보를 나타낸다고 볼 수 있다. Path 테이블과 함께, [14,15]는 각 경로들과 연관된 발생 정보들을 가진 테이블들을 정의하였는데, 이들에 대해 두 방식은 다른 구조를 선택하였다. 먼저 [14]는, Path 테이블의 각 경로에 부여된 경로 식별자를 중심으로 해당 경로의 모든 발생 정보를 담은 Element 테이블과 Attribute 테이블을 정의하였다. Element 테이블은 경로가 발생된 문서 번호, 경로의 마지막 위치에 나타나는 엘리먼트의 시작 위치 번호와 종료 위치 번호, 형제(sibling) 엘리먼트들과의 순서 등으로 구성되며, Attribute 테이블은 경로가 발생된 문서 번호, 경로의 마지막 위치에 나타나는 애트리뷰트의 시작 위치 번호와 종료 위치 번호, 그리고 애트리뷰트가 소유한 값으로 구성된다. 추가로, Element 테이블의 발생 경로들이 포함하는 값들을 Text 라는 테이블에 따로 관리하였다. 한편, [15]는 [14]의 Element 테이블에서 형제 엘리먼트간의 관계는 표현하지 않았으며, Attribute 테이블과 Element 테이블을 하나로 통합한 PathIndex 테이블을 정의하였다. 그리고, [14]의 Text 테이블을 Term 테이블과 TermIndex 테이블로 구분하여 정의하였다. Term 테이블은 문서에서 발생하는 모든 값들에 대해, 서로 다른 값들만을 추출하여 각 값에 식별자를 부여한 것이며, TermIndex 테이블은 Term 테이블에 저장된 값들의 모든 발생 정보를 나타낸 것이다. 예를 들어, <title> Database theory </title> 이 있을 때, "Database"와 "theory"는 Term 테이블에 저장되며, TermIndex 는 "Database"가 어떤 문서의 몇 번째 단어인지에 대한 발생 정보를 저장한다. 만약, "Database"가 여러 번 나타나면, 그 때마다 TermIndex 테이블에 해당 발생 정보가 저장된다.

이처럼, [14]와 [15]는 모두 경로 문자열 정보를 기반으로 한 인덱스를 정의하였으나, 발생 정보를 관리하는 부분을 다르게 정의하였다. 두 인덱스를 이용하여 경로를 검색하는 방식은 단순하다. 찾고자 하는 경로를 Path 테이블에 표현된 경로 문자열과 동일한 형식으로 변환한 후, Path 테이블 내의 경로 문자열들과 문자열 비교 연산을 수행함으로써 대응되는 경로를 찾을 수 있다. 이 때 B-Tree 인덱스를 Path 테이블에 정의하면, 더욱 빨리 대응되는 경로를 찾을 수 있다. Path 테이블의 검색 결과 대응되는 경로가 존재하면, 각 경로와 연관된 모든 발생 정보는 경로 식별자를 통해 발생 정보 테이블들로부터 바로 추출할 수 있다. 그러므로, 두 방식은 [10]과 [13] 같은 발생 정보를 기반으로 한 경로 검색 방식에 비해, 더 효율적으로 경로를 검색할 수 있다. 또한, 검색

대상이 되는 XML 문서들이 동일한 구조를 가지고 있는 상황에서, 문서 수의 증가에 무관하게 빠른 검색을 지원한다. 하지만, 조상-자손 관계를 포함한 경로를 찾는 경우, [14,15]는 B Tree 인덱스의 존재 여부와 관계없이 Path 테이블 내의 모든 경로를 비교해야만 대응되는 경로가 존재하는 지를 파악할 수 있다. 그러므로, 만약 상이한 구조의 문서들의 수가 증가하게 되면 Path 테이블의 크기는 점점 커지게 되며, 이에 따른 검색 비용은 비례하여 증가된다. 이러한 단점은 상이한 문서들이 많이 발생하는 환경에 [14]와 [15]를 적용하기 어렵게 만든다.

이에 본 연구에서는, [14,15]와 같이 문서들의 구조적 요약 정보를 이용한 경로 검색을 지원하고, 상이한 문서들이 급격히 증가하는 상황에서도 안정적인 검색 성능을 보장해 주는 새로운 인덱스를 제안하였다.

3. XML 문서 모델

우리는 상이한 구조를 가진 XML 문서들에 대해 적용할 수 있는 문서 모델을 정의하였다.

XML 문서는 유효한(valid) 문서이며, 순서가 있는 트리로 가정한다. XML 문서는 문서의 구조를 나타내는 태그인 엘리먼트(Element)와 엘리먼트의 특징을 기술하는 애트리뷰트(Attribute), 엘리먼트나 애트리뷰트에 포함된 값(또는 문자열)인 용어(Term)로 구성된다. XML에서 두 엘리먼트 간의 참조 관계를 표현하는 애트리뷰트인 ID 와 IDRef는 XML 문서를 그래프로 표현할 수 있도록 해준다. 하지만, 본 연구에서는 XML 문서를 트리로 가정하므로, 두 애트리뷰트는 단순히 값을 포함하는 애트리뷰트로 취급한다. 또한, 우리는 애트리뷰트와 엘리먼트를 구분하지 않고, 동일하게 엘리먼트로 간주한다. 그 이유는 엘리먼트와 애트리뷰트를 사용하는 방식에 대한 명확한 구분이 없어 혼용되는 경향이 있기 때문이다. 한편, 애트리뷰트는 엘리먼트와 같이 시작 태그와 종료 태그를 갖지 않으므로, 애트리뷰트 이름을 시작 태그로, 애트리뷰트의 값이 나온 뒤 바로 가상의 종료 태그가 있다고 간주하여 모델에 표현한다. 엘리먼트와 애트리뷰트의 시작 태그와 종료 태그, 용어는 문서에서 나타나는 순서대로 번호를 부여받는다. 이에 따라, 각 엘리먼트와 애트리뷰트는 일정 범위를 가지게 되며, 용어는 발생 위치를 갖게 된다.

그림 1은 두 개의 XML 문서 예이다. 두 문서는 모두 Movie XML 문서를 나타내며, 각각 영화와 관련된 여러 정보들을 엘리먼트, 애트리뷰트, 용어들로 가진다. 두 문서는 유사한 내용을 가지며, 서로 상이한 구조를 가지고 있다. 둘 다 Movie 엘리먼트를 루트로 가지지만, a)는 Actor의 애트리뷰트로 BirthYear를 가지며, b)는 Actor의 자식 엘리먼트로 BirthYear를 가진다. a)는 두 개의 Genre를 가지고 있으며, Genres 엘리먼트가 이를 포함하고 있지만, b)는 하나의 Genre 만을 가지고 있다. Actor의 Name이나 Director도 상이한 구조로 표현되어 있다.

그림 2는 그림 1의 a)를 표현한 것이다. 그림에서 보듯이, ID와 BirthYear는 원래 애트리뷰트이지만 모두엘리먼트로 표현되었으며, 각 노드는 시작 태그와 닫는 태그 사이의 범위를 가지고 있다. 각 용어는 용어마다 일련번호를 부여받으며, 용어를 포함한 노드의 범위에 포함된다. 그림 1의 b)도 동일한 규칙에 의해 표현 가능하다.

```
<Movie ID="1">
 <Title>Rashomon</Title>
 <Year>1950</Year>
 <Director>Akira Kurosawa</Director>
 <Genres>
  <Genre>Drama</Genre>
  <Genre>Mystery</Genre>
 </Genres>
 <Actors>
  <Actor BirthYear="1950">
   <Name> Toshir Mifune </Name>
   <Company> Su-Man. </Company>
  </Actor>
  <Actor>
   <Name>Machiko Ky</Name>
   <Company>TaeHung</Company>
  </Actor>
  <Actor>
   <Name>Masayuki Mori</Name>
   <Company>TaeHung</Company>
  </Actor>
 </Actors>
</Movie>
```

a) Movie XML 예-1

```
<Movie ID="1" >
 <Title>Rashomon</Title>
 <Year>1950</Year>
 <Directed_By>
  <Director>Akira Kurosawa</Director>
 </Directed_By>
 <Genre>Drama</Genre>
 <Cast>
  <Actor>
   <FirstName>Toshir</FirstName>
   <LastName>Mifune</LastName>
   <BirthYear>1950</BirthYear>
  </Actor>
  <Actor>
   <FirstName>Machiko</FirstName>
   <LastName>Ky</LastName>
  </Actor>
  <Actor>
   <FirstName>Masayuki</FirstName>
   <LastName>Mori (I)</LastName>
  </Actor>
 </Cast>
</Movie>
```

b) Movie XML 예-2

그림 1 서로 다른 Movie XML 문서 예

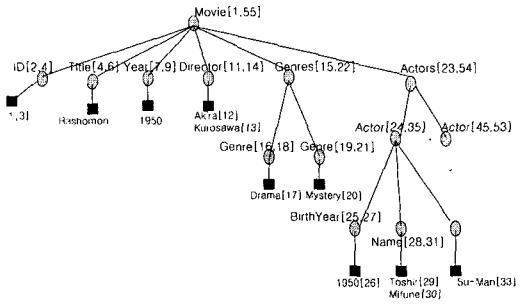

그림 2 XML 데이타 모델 예

4. 역 인덱스 구조와 질의 처리 기법

이 장은 3 장의 문서 모델을 기반으로 새로운 역 인덱스의 구조를 정의하고, 이를 이용한 검색 질의 구성

방식에 대해 설명한다.

4.1 역 인덱스 구조

본 논문에서 제안하는 역 인덱스는 데이타베이스 시스템을 사용하며 네 개의 테이블로 구성된다. 그림 3은 각 테이블의 구조를 보여준다. 먼저, TermTbl은 XML 문서들에서 사용된 모든 용어들을 저장한 테이블로서, 용어(Term)와 용어 식별자(TermID)로 구성된다. TermTbl에는 서로 다른 용어들만이 저장된다. 그러므로, 용어의 유무에 대한 검사는 TermTbl을 통해 바로 수행될 수 있다. TermOccurrenceTbl은 TermTbl에 있는 각 용어들에 대한 발생 정보를 저장하는 테이블로서, 각 용어를 대표하는 용어 식별자(TermID), 용어가 나타난 문서의 식별자(docID), 용어가 직접적으로 포함되는 경로의 식별자(pathID), 용어 발생 위치(position)로 구성된다. 다음, UniqPathElmTbl은 전체 XML 문서들에 포함된 모든 발생 경로들로부터 추출된 서로 다른 경로들을 기반으로 생성된다. 전체 XML 문서들로부터 추출된 서로 다른 경로들은 전체 문서들에 대한 구조적 요약 정보이다. 이 경로들로부터, 우리는 각 경로를 구성하는 엘리먼트들에 대한 정보를 추출한다. 추출되는 정보는 각 엘리먼트의 이름(ename), 엘리먼트가 속해 있던 경로의 식별자(pathID), 엘리먼트가 경로에 나타난 위치(level)이다. 이 때, level은 0부터 시작한다. 그리고, 경로의 마지막 위치에 나타나는 엘리먼트에는 '#'을 붙여서 경로의 마지막 엘리먼트임을 표현한다. 예를 들어, "/Movie/Genres/Genre"라는 경로가 경로 식별자 3을 가지고 있다고 가정할 때, 우리는 이 경로를 <"Movie",3,0>,<"Genres",3,1>,<"Genre#",3,2>와 같이 세 개의 정보로 분할한다. 이렇게 분할된 정보들은 UniqPathElmTbl 테이블에 튜플들로 저장된다. PathOccurrenceTbl 은 모든 발생 경로에 대한 정보를 담은 테이블로서, UniqPathElmTbl 에 분할되어 저장된 경로들의 경로 식별자(pathID), 경로가 나타난 문서 식별자(docID), 경로의 범위(startPos, endPos)로 구성된다. 경로의 범위는 3장에서 제안한 문서 모델에서의 범위와 동일하다.

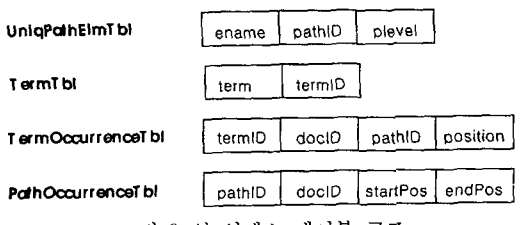

그림 3 역 인덱스 테이블 구조

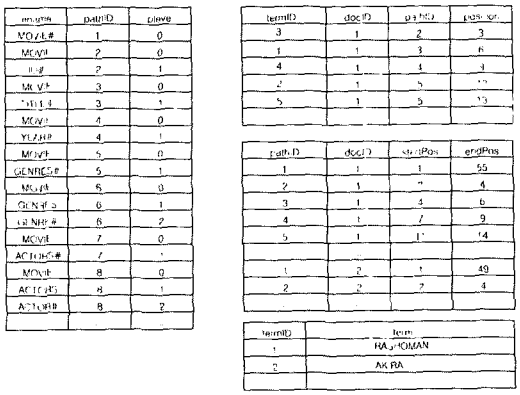

그림 4 역 인덱스 테이블 구조

그림 4는 그림 1의 XML 문서들을 위에 정의한 인덱스 테이블로 대응시킨 결과이다. 보는 바와 같이, "Movie" 엘리먼트는 "Movie"와 "Movie#"으로 나뉘어 UniqPathElmTbl에 저장되어 있다. 그 이유는 루트 엘리먼트인 "Movie"로만 표현된 경로와 "Movie"로 시작하는 경로들이 있기 때문이다. "Movie#"에 대한 발생 정보를 살펴보면 다음과 같다. "Movie#"은 경로 식별자 1을 가지고 있으며, 이 경로는 1,2번 문서에서 발생했으며, 각 문서에서 [1,55]와 [1,49]의 범위를 가진다. "Movie#" 경로는 어떠한 용어도 직접 포함하고 있지 않다. 이처럼, UniqPathElmTbl을 기반으로 하여, 특정 경로와 이에 대응되는 모든 발생 정보를 파악할 수 있다.

제안된 역 인덱스 구조는 기존의 연구들[10,13,14,15]에 비해 다음과 같은 차별화된 특징을 가지고 있다. [10,13]은 경로 검색을 위해, 경로를 구성하는 모든 엘리먼트나 애트리뷰트들의 발생 정보들의 포함 관계를 비교한다. 문서의 수가 증가됨에 따라 인덱스에 저장되는 발생 정보의 수는 매우 커지게 되며, 이들을 통한 비교 연산과 조인 비용은 급격히 증가하게 된다. 반면, 본 연구에서 제안한 인덱스를 사용하는 경우, UniqPathElmTbl에 저장된 전체 문서들에 대한 경로 요약 정보를 이용하여 경로의 존재 여부를 파악한다. UniqPathElmTbl에 저장된 요약 정보는 전체 발생 정보에 비해 극히 적은 양이므로, [10,13]에 비해 매우 적은 비교 연산만을 필요로 한다. 한편, 경로 문자열들을 이용하는 [14,15]의 경우, 조상-자손 관계를 포함한 경로를 찾기 위해 PathTbl 테이블 내의 모든 경로들을 순차 비교해야 한다. 이러한 상황에서, 만약 상이한 문서의 수가 많아지게 되면, PathTbl의 크기는 커지게 된다. 그리고, 테이블 내의 모든 경로는 순차 검색되어야

하므로, 경로 검색 시간은 테이블의 크기에 비례하여 점점 길어지게 된다. 반면, 본 연구에서 제안한 UniqPathElmTbl은 찾는 경로에 포함된 엘리먼트 정보들만 추출하여 비교 연산을 수행하므로, 경로의 유무를 빠르게 알아낼 수 있다. 이처럼, 본 논문에서 제안한 인덱스 구조는 높은 선택도(Selectivity)를 제공하고 임시 테이블의 크기를 줄여줌으로써, 기존 방식에 비해 높은 검색 성능을 보장한다.

4.2 질의 처리

이 장에서는 제안한 인덱스 구조를 이용하여 경로를 검색하기 위한 질의 구성 방식에 대해 소개한다.

4.2.1 질의 형식

XML 문서에 대한 질의 수행을 위해 여러 XML 질의 언어들[16,17]이 제안되어 왔다. 이러한 질의 언어들은 동일한 구조를 가진 XML 문서들에 대한 질의를 가정하고 있으며, 매우 정교한 질의 구성이 가능하다. 하지만, 상이한 구조를 가진 XML 문서들에 대해서 이러한 질의 언어를 적용하기는 매우 어렵다. 이에, 본 논문에서는 상이한 구조를 가진 문서들에도 적용 가능하며, XML 질의 언어들에 공통적으로 포함되어 있는, 포함 질의 형식을 이용한다. 포함 질의는 [10]에서 정형화하였으며, XML의 구성 요소인 엘리먼트, 애트리뷰트, 용어 간의 포함 관계에 따른 질의 구성 형식을 말한다. 포함 질의에 대한 정형화된 정의는 [10]에 잘 나타나 있으므로, 본 논문에서는 각 포함 질의 형식을 예로서 설명하고자 한다.

간접 포함 질의 : 엘리먼트, 애트리뷰트, 용어 간의 조상 자손 관계를 표현한 질의 형식이며, '//' 으로 표현된다. 예를 들어, /Movie/Actors//FirstName에서 Actors와 FirstName은 조상-자손의 관계를 가진 것으로, 그 사이에 임의의 엘리먼트가 올 수 있다.

직접 포함 질의 : 엘리먼트, 애트리뷰트, 용어 간의 부모 자식 관계를 표현한 질의 형식이며, '/' 으로 표현된다. 앞선 예제에서, Movie와 Actors는 부모-자식 관계를 가진 것으로, Movie와 Actors 사이에 어떠한 엘리먼트도 올 수 없음을 말한다.

완전 포함 질의 : 엘리먼트, 애트리뷰트, 용어 간의 유일한 포함 관계로만 이루어진 질의 형식으로, 예를 들어, /Movie/Title='성냥팔이소녀의재림'에서 Title이 '성냥팔이소녀의재림'만을 포함하고 있는 경우가 이에 해당된다. Movie와 Title 관계에서는 Movie가 Title 외에 Actors와 같은 다른 엘리먼트를 포함하므로, 완전 포함 관계가 아니다.

근접 포함 질의 : 용어들 간의 위치 근접도에 기반한 질의 형식으로, Distance와 같은 함수를 사용한다. 예를 들어, Distance('환타지', '영화') <= 3는 '환타지'와 '영화'라는 용어의 용어상 거리가 3이하인 것을 찾으라는 의미이다.

위의 포함 질의 형식은 서로 간의 결합을 통해, 복잡한 포함 질의를 구성할 수 있다. 예를 들어, //Movie[Title='성냥팔이소녀의재림']//Actor/FirstName='나라'은 //Movie/Title='성냥팔이소녀의재림'과 //Movie//Actor/FirstName='나라'의 두 경로식으로 구분되며, 두 경로식이 같은 Movie에 포함되어 있다는 것을 의미한다. 그러므로 이 질의는 총 세 개의 경로식에 대한 포함 관계를 나타낸 것이다.

4.2.2 SQL로의 질의 변환

4.2.1에서 보았듯이, 포함 질의는 여러 경로식으로 구성될 수 있다. 또한, 각 경로식은 여러 종류의 포함 질의 형식을 결합한 형태이다. 본 논문에서 제안한 역 인덱스 구조는 모든 포함 질의의 관계를 표현할 수 있도록 구성되었으며, SQL 문으로의 변환이 모두 가능하다.

포함 질의를 SQL 문으로 변환하는 과정은 다음과 같다. 복잡한 포함 질의는 단순 포함 질의들로 분리된다. 단순 포함 질의는 직접, 간접, 완전, 근접 포함 형태의 조합으로 표현된 질의이다. 단순 포함 질의는 경로식과 용어에 대한 부분으로 분리된다. 경로식은 UniqPathElmTbl에 나타난 경로 구성 엘리먼트들 간의 조인 연산으로 표현한다. 그리고, 발생 정보를 추출하기 위해 PathOccurrenceTbl과 조인하도록 표현한다. 용어에 대한 질의 부분은 TermTbl과 TermOccurrenceTbl 간의 조인으로 구성한다. 경로식과 용어 간의 포함 관계를 필요로 하는 경우는 PathOccurrenceTbl과 TermOccurrenceTbl 간의 조인 연산으로 표현한다. 복잡한 포함 질의는 단순 포함 질의들의 수행 결과에 대한 조인 연산으로 표현한다. 즉, 단순 포함 질의들을 서브 SQL 문으로 표현하고, 이들 간의 조인 연산으로 표현함으로써 SQL로 변환할 수 있다. 구성된 SQL 문장은 경로식이 복잡해짐에 따라 UniqPathElmTbl의 자기 조인 연산이 많아지는 것을 제외하고는 테이블 간의 조인 수가 늘어나지 않는다. 한편, UniqPathElmTbl에서 추출되는 데이타는 조인에 참여하는 엘리먼트들의 정보들뿐이므로, 그들 간의 조인 연산은 매우 빠르게 수행된다. 또한, 경로식의 마지막 엘리먼트는 '#'을 포함하고 있으므로, 테이블 내에서 추출되는 데이타를 줄여 주어 검색 성능을 향상시킨다. 그림 5는 단순 포함 질의를 구성하는 직접, 간접, 완전, 근접 포함 질의의 변환 예를 보여주고 있다. 단순 포함 질의는 이들의 조합으로 구성된다.

```
Q : Actors//FirstName (간접포함질의)
SQL :
    From UniqPathElmTbl Actors, UniqPathElmTbl FirstName
    Where Actors.ename='Actors' and FirstName.ename='FirstName#'
      and Actors.pathID=FirstName.pathID and Actors.plevel < FirstName.plevel

Q : Movie/Title (직접포함질의)
SQL :
    From UniqPathElmTbl Movie, UniqPathElmTbl Title
    Where Movie.ename= Movie' and Title.ename='Title#'
      and Movie.pathID=Title.pathID and Movie.plevel + 1 = Title.plevel

Q : //Title = '성냥팔이소녀의재림'
SQL :
    From UniqPathElmTbl Title, PathOccurrenceTbl po,
         TermTbl tt, TermOccurrenceTbl to
    Where Title.ename='Title#' and Title.pathID = po.pathID
      and tt.term='성냥팔이소녀의재림' and tt.termID = to.termID
      and po.startPos = to.position - 1 and po.endPos = to.position + 1

Q : Distance('환타지', '영화') <= 3
SQL :
    From TermTbl tt1, TermOccurrenceTbl to1,
         TermTbl tt2, TermOccurrenceTbl to2,
    Where tt1.term = '환타지' and tt2.term = '영화'
      and tt1.termID = to1.termID and tt2.termID = to2.termID
      and to1.position - to2.position <= 3
```

그림 5 포함 질의 변환 예

5. 성능 평가

이 장에서 우리는 제안한 역 인덱스 구조를 이용한 경로 검색의 성능을 검증하기 위한 실험과 실험 결과에 대해 소개한다.

5.1 실험 구성

우리는 실험을 위해, Pentium4 1.2G CPU에 256 Memory를 가진 PC와 상용 데이타베이스 시스템을 사용하였다. 역 인덱스 생성과 질의 처리 모듈의 구현은 .NET 환경 하에서 제공되는 기본 클래스 라이브러리를 이용하였다. 질의는 SQL 변환하여 데이타베이스 시스템으로 수행한 뒤, 수행 결과를 받는 형식으로 수행하였다.

실험을 위한 데이타는 Wisconsin 에서 제공하는 문서들[18]을 사용하였다. 문서들은 DTD 기반으로 구성되어 있으며, 한 DTD에 대해 여러 문서들로 구성되었다.

본 연구의 성능 평가를 위해서, 우리는 [14]와 [15]의 연구를 비교 대상으로 선정하였다. 발생 정보를 기반으로 한 [10]과 [13]은 비교 대상에서 제외하였다. 그 이유는 다음과 같다. [10]의 경우, [15]의 연구 결과에서 [15]에 비해 검색 성능이 떨어짐이 확인되었으며, [13]의 경우, 경로 검색을 수행하기 전에 경로를 구성하는 테이블의 존재 여부를 묻는 질의들을 수행해야 하며, 일반조인 방식을 사용하므로, [10]에 비해 나은 성능을 보이기 어렵기 때문이다. 본 연구에서는 [14]와 [15]에서 제안한 역 인덱스를 자체적으로 구현하였으며, 성능 향상을 위해 모든 테이블에 완전 클러스터 인덱스(Full Clustered Index)를 제공하였다. 성능 평가에 사용된 질의의 변환은 빠른 실험을 위해 수작업으로 진행하였다.

실험 질의는 길이가 다른 직접 경로 질의, 간접 경로 질의, 완전 포함 질의, 근접 질의를 모두 구성하였으며, 단순한 포함 관계와 복잡한 포함 관계에 대한 질의로 구성하였다. 그림 6은 실험에서 사용한 질의들이다.

Query	Query Type
Q1 : /Movie/Title	Direct Containment Query(short)
Q2 : /Movie/Actors/Actor/FirstName	Direct Containment Query(long)
Q3 : //Movie	Indirect Containment Query(short)
Q4 : //Actor//Movie//Year	Indirect Containment Query(long)
Q5 : /Movie/Year ='1994'	Tight Direct Containment Query
Q6 : //Movie//Genre ='Romance'	Tight Indirect Containment Query
Q7 : /Movie [Year ='2000']//Actor	Complex Containment Query(short)
Q8 : //Actor[Name/FirstName ='Richard']//Movie	Complex Containment Query(long)
Q9 : Distance(XML , DB) <= 3	Proximity Containment Query

그림 6 실험 질의 구성

5.2 실험 결과

5.2.1 저장 공간

그림 7은 실험을 위해 사용된 문서들에 대한 크기와 각 역 인덱스 테이블의 데이타베이스 저장 공간의 크기를 보여준다. 그림에서 보듯이, 문서의 크기에 비해 역 인덱스 정보의 크기가 매우 크다는 것을 알 수 있다. 역 인덱스 테이블은 문서의 수가 증가함에 따라 매우 급격히 증가하게 되며 지속적이고 안정적으로 유지되어야 하므로, 데이타베이스를 이용한 저장은 바람직하다 할 수 있다. 역 인덱스 테이블의 전체 크기는 [14]의 방식이 다른 방식에 비해 5배 정도 더 큼을 볼 수 있다. 이는 [14]가 문서의 용어들을 분리하여 저장하지 않고, 엘리먼트 내의 텍스트 전체를 하나의 필드로 저장함에 따라, 해당 테이블의 크기가 매우 커지기 때문이다. 가변 필드를 이용하여 테이블의 크기를 줄일 수 있으나, 성능 향상을 위해 테이블에 생성한 클러스터 인덱스는 각 필드의 최대 값으로 구성하도록 되어 있어, 많은 저장 공간을 차지하게 된다. 이에 따라, 문서의 수와 크기가 증가함에 따라, 이 테이블의 크기는 매우 커지게 되며, 매우 큰 저장 공간을 필요로 한다. 반면, 본 연구와 [15]에서 제안한 테이블들은 문서 내의 용어들을 분리하여 저

Relation	Size
UniqPathElmTbl	0.2M(182K)
UniqTermTbl	8.7M
PathOccurrenceTbl	8.2M
TermOccurrenceTbl	27.3M
Total	44.2M

a) KSMIN

Relation	Size
PathTbl	0.1M(75K)
TermTbl	8.7M
PathIndex	8.2M
TermIndex	27.3M
Total	44.2M

b) ChiYoung

Relation	Size
PathY	0.1M(75K)
AttributeY	0.1M(79K)
ElementY	8.0M
TextY	193.5M
Total	202.0M

c) Yoshikawa

d) Document information

# of Documents : 371
Total Size (MB) : 10.3

그림 7 저장 공간

장하도록 함으로써 필요한 저장 공간이 상대적으로 매우 적게 필요하여, 문서의 수와 크기가 증가하더라도, 큰 부담을 주지 않아, 적용하기에 적합하다.

한편, 각 연구들이 관리하는 경로 테이블의 크기는 모두 1M 미만으로 전체 역 인덱스 테이블 크기에 비해, 극히 작음을 알 수 있다. [14,15]의 테이블 크기가 본 연구에서 제안한 테이블의 크기보다 작지만, 전체 저장 공간의 측면에서 보았을 때, 그 차이는 경미하다고 할 수 있다.

이로서 우리는 본 연구에서 제안한 역 인덱스 테이블 구조의 저장 공간 효율성이 [15]의 저장 공간 효율성에 비해 떨어지지 않음을 확인할 수 있으며, [14]에 비해서는 매우 높은 저장 효율성을 보임을 알 수 있다.

5.2.2 질의 성능

그림 8은 그림 6의 질의들을 수행한 결과를 보여준다. 몇몇 질의에 대해, [14,15]의 응답 시간이 매우 오래 걸려, 그림에서는 수행 성능을 로그 스케일로 표현하였다.

직접 질의인 Q1, Q2의 경우, [14]와 [15]는 제안한 방법에 비해 약간 나은 성능을 보였다. 이는 [14,15]가 경로 테이블에 저장된 모든 경로에 대해 인덱스를 유지함으로써, 루트로부터 시작하는 경로에 대해 빠르게 해당 경로를 찾을 수 있기 때문이다. 반면, 제안한 방법은 질의에 나타난 경로를 구성하는 엘리먼트를 찾아 조인하는 방식을 취하므로 약간 느려지게 된다. 하지만, 대응되는 튜플의 수가 매우 적으므로, 빠른 조인을 통해 성능 상에 별 차이를 보이지 않는다. Q3, Q4와 같은 간접 질의의 경우, [14,15]는 본 연구에서 제안한 방법에 비해, 10배 이상의 응답 시간을 필요로 하였다. 이는 간접 포함 질의의 특성 상, 질의 시작 엘리먼트 앞에 임의의 엘리먼트가 나타날 수 있어, 인덱스를 통한 검색이 불가능하기 때문이다. 그러므로 [14,15]에서는 경로 테이블 전체를 순차 검색하여 대응되는 경로들을 찾을 수밖에 없게 되어, 많은 수행 시간을 필요로 한다. 한편, 본 연구에서 제안한 방법은 질의에 나타나는 엘리먼트에 대응되는 튜플만을 찾아 경로를 추출하는 방식을 사용하므로, 전체 테이블 검색이 필요 없으며, 적은 수의 튜플만을 추출하므로, 상대적으로 매우 적은 시간에 응답을 제공할 수 있다. 그러므로, 경로 테이블의 크기가 커질수록, 본 연구에서 제안한 방법은 [14,15]에 비해, 매우 큰 성능 향상을 제공한다. Q5, Q6은 용어를 포함한 질의로서, 제안한 방법은 [15]와는 1.5배로 근소한 차이를 보였으나, [14]의 연구와는 20배 이상의 큰 차이를 보였다. 이는 [14]가 용어 정보를 저장하는 방식이 많은 저장 비용을 필요로 함에 따라, 질의에 나타난 용어를 찾기 위한 검색 비용을 많이 필요로 하기 때문이다. 또한, Q7과 Q8과 같이 복잡한 질의의 경우에도, 제안된 방법은 기존 연구에 비해 매우 빠른 성능을 제공하였는데, 적게는 3배에서, 많게는 10배 이상 빠른 성능을 보였다. 마지막으로, Q9의 경우, [14]는 근접도를 묻는 질의를 구성하기 어려워 실험에서 배제하였다. 제안된 기법과 [15]의 수행 결과는 30배 이상으로 매우 큰 차이를 보였다. 하지만, 이 결과는 두 방법이 동일한 용어 테이블 구조를 가진 점으로 볼 때, 질의최적화기가 비용이 큰 질의 계획을 선택한 것으로 판단된다.

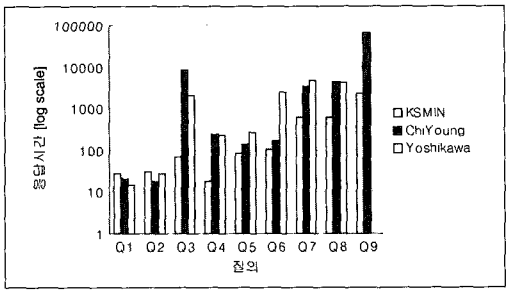

그림 8 실험 결과

이로써 우리는, 본 연구에서 제안한 역 인덱스 구조가 기존의 방식들에 비해 매우 높은 성능을 제공함을 증명하였다. 특히, 경로의 수가 많아질수록, 상이한 구조를 가진 문서의 수가 많아질수록, 본 연구에서 제안한 방법은 다른 연구 결과에 비해 더욱 높은 성능을 보임을 알 수 있다.

6. 결론 및 향후 연구 계획

XML 형식의 문서가 빠르게 늘어남에 따라, 상이한 문서 구조를 가진 XML 문서들로부터 사용자가 원하는 문서를 정확히 추출하는 것이 매우 중요해지고 있다.

본 연구에서는 이러한 환경을 고려하여, 상이한 구조의 문서들이 많아지는 상황에도 사용자의 질의를 빠르고 정확하게 수행할 수 있도록 하기 위한 새로운 역 인덱스 구조를 제안하였다. 또한, 역 인덱스를 데이타베이스 테이블로 저장, 관리하고, 사용자 질의를 SQL 로 수행하는 방식을 제공함으로써, 기존의 성숙된 데이타베이스 기술을 적절히 이용할 수 있도록 하였다.

실험을 통해, 우리는 제안된 기법이 기존의 연구에 비해 월등히 높은 성능을 보임을 증명하였다. 특히, 상이한 구조의 문서가 많아질수록 더 높은 성능을 보임을 알 수 있었다.

향후, XML 문서 검색을 위한 효율적인 질의 변환 방

식과 인덱스 성능 향상을 위한 연구들을 진행할 것이다.

참 고 문 헌

[1] Neil Bradley, The XML companion second edition, Addison Wesley, 2000.

[2] XSL Transformations (XSLT) Version 1.0 W3C Recommendation, http://www.w3.org/TR/xslt, 1999.

[3] XML Schema, http://www.w3.org/XML/Schema#dev.

[4] XMLSpy, http://www.xmlspy.com/.

[5] Xeena, http://www.alphaworks.ibm.com/tech/xeena.

[6] Roy Goldman and Jennifer Widom, DataGuides : Enabling Query Formulation and Optimization in Semistructured Databases, VLDB, pp 436-445, August 1997.

[7] Tova Milo and Dan Suciu, Index Structures for Path Expressions, ICDT, pp 277-295, January 1999.

[8] Brian F. Cooper, Neal Samfple, Michael J. Franklin, Gisli R. Hjaltason, and Moshe Shadmon, A fast index for semistructured data, VLDB, pp 341-350, January 2001.

[9] Chin-Wan Chung, Jun-Ki Min, Kyuseok Shim, APEX : An Adaptive Path Index for XML Data, SIGMOD, pp 121-132, June 2002.

[10] Chun Zhang, Jeffery Nahgton, David DeWitt, Qiong Luo, and Guy Lohman, On Supporting Containment Queries in Relational Database Management Systems, SIGMOD, pp 425 436, May 2001.

[11] Shurug Al-Khalifa, H. V. Jagadish, Nick Koudas, Jignesh M. Patel, Divesh Srivastava, and Yuqing Wu, Structural Joins : A Primitive for Efficient XML Query Pattern Matching, ICDE, pp 141-153, February 2002.

[12] Shu-Yao Chien, Zografoula Vagena, Donghui Zhang, Vassilis J. Tsotras, and Carlo Zaniolo, Efficient Structural Joins on Indexed XML Documents, VLDB, pp 263-274, August 2002.

[13] Daniela Florescu and Donald Kossmann, Storing and Querying XML Data using an RDBMS, Bulletin of Data Engineering, pp 27-34, September 1999.

[14] Masatoshi Yoshikawa and Toshiyuki Amagasa, XRel : A Path-Based Approach to Storage and Retrieval of XML Documents Using Relational Databases, ACM TOIT, Volume 1, Number 1, pp 110-141, August 2001.

[15] Chiyoung Seo, Sang-won Lee, and Hyoung-Joo Kim. An Efficient Inverted Index Technique for XML Documents using RDBMS. Information and Software Technology (Elsevier Science), Volume 45, Issue 1, pp 11-22, January 2003.

[16] XQuery 1.0: An XML Query Language W3C Working Draft, http://www.w3.org/TR/xquery/, 2002.

[17] XML Path Language(XPath) Version 1.0 W3C Recommendation, http://www.w3.org/TR/xpath, 1999.

[18] Wisconsin XML Data Set, http://www.cs.wisc.edu/niagara/data.html

민 경 섭

1995년 2월 한국항공대학교 컴퓨터공학과 학사. 1997년 2월 서울대학교 컴퓨터공학과 석사. 1997년~현재 서울대학교 인지과학 박사과정. 관심분야는 XML IR, Database

김 형 주

정보과학회논문지 : 데이타베이스 제 30 권 제 1 호 참조