www.calsec.or.kr
http://dx.doi.org/10.7838/jsebs.2013.18.2.153

효율적인 온톨로지 검색과 추론을 위한 인덱스

Index for Efficient Ontology Retrieval and Inference

송승재(Seungjae Song)*, 김인성(Insung Kim)**, 전종훈(Jonghoon Chun)***

초 록

근래에 들어와서 각광받고 있는 시맨틱 웹과 관련기술의 부상으로 온톨로지에 대한 관심이 증대되었으며, 그중에서도 고난이도의 추론을 요구하는 의미기반 시맨틱 검색을 위해서 온톨로지를 효율적으로 저장하고 검색하는 다양한 기법들이 활발히 연구되어왔다. W3C에서의 표준권고안은 RDFS, OWL을 활용하도록 하고 있다. 하지만 메모리 기반으로 구현되어 있는 에디터나 추론엔진들, 온톨로지의 원형을 그대로 유지하여 저장하는 트리플 저장소를 이용하여 대용량 온톨로지를 처리하기에는 성능상의 한계가 있다. 따라서 이를 해결하기 위해 관계형 데이터베이스 엔진을 이용하여, 온톨로지를 저장하고 효율적으로 활용하기 위한 다양한 방식의 추론엔진과 질의처리 알고리즘들이 제안되었으나, 온톨로지 프로퍼티의 다섯 가지 핵심특성에 따른 추론 결과를 완전하게 획득하지는 못하고 있는 실정이다. 본 논문에서는 하이퍼 큐브 인덱스(Hyper Cube Index)를 제안함으로서 관계형 데이터베이스에 저장한 온톨로지를 효율적으로 검색할 수 있는 환경을 제공하는 것은 물론, 온톨로지 프로퍼티의 핵심특성을 빠짐없이 투영하여 숨겨진 추론 결과를 획득할 수 있는 방안을 제시한다.

ABSTRACT

The ontology has been gaining increasing interests by recent arise of the semantic web and related technologies. The focus is mostly on inference query processing that requires high-level techniques for storage and searching ontologies efficiently, and it has been actively studied in the area of semantic-based searching. W3C's recommendation is to use RDFS and OWL for representing ontologies. However memory-based editors, inference engines, and triple storages all store ontology as a simple set of triplets. Naturally the performance is limited, especially when a large-scale ontology needs to be processed. A variety of researches on proposing algorithms for efficient inference query processing has been conducted, and many of them are based on using proven relational database technology. However, none of them had been successful in obtaining the complete set of inference results which reflects the five characteristics of the ontology properties. In this paper, we propose a new index structure called hyper cube index to efficiently process inference queries. Our approach is based on an intuition that an index can speed up the query processing when extensive inferencing is required.

키워드 : 하이퍼 큐브 인덱스, 비트 큐브, 비트 매트릭스, 온톨로지, 추론
Hyper Cube Index, Bit Cube, Bit Matrix, Ontology, Inference

* Associate research engineer, Ubiquitous R&D center, Daeyang eT&C, Co., Ltd.
** Research engineer, R&D center, Prompt Technology, Co., Ltd.
*** Corresponding Author, Professor, Department of computer engineering, Myongji University(This paper is a part of research results conducted by the author while visiting Telecom SubParis, Evry Cedex, France) (E-mail : jchun@mju.ac.kr)
2013년 01월 21일 접수, 2013년 04월 30일 심사완료 후 2013년 05월 14일 게재확정.

1. 서 론

월드와이드웹(World Wide Web)은 정보 표현과 전달에 혁신적인 방법을 제공하여 사람들의 의사소통 방법을 변화시키면서 세계를 지식 자본주의와 정보사회로 이끄는 혁명의 중심이 되었다. 이러한 혁명은 과거 수치계산을 위해 사용되었던 컴퓨터를 정보처리에 활용하도록 컴퓨터에 대한 사람들의 인식을 바꾸게 되었고 웹상에 존재하는 정보자원의 의미를 컴퓨터가 처리할 수 있도록 지원하는 차세대 웹 기술인 시맨틱 웹(Semantic Web)이 대두되는 계기를 마련하였다[2, 8, 20, 22].

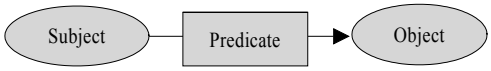
시맨틱 웹의 핵심은 컴퓨터가 처리할 수 있는 형태의 온톨로지(Ontology)[25, 30]로 콘텐츠를 표현하는 것이다. 온톨로지는 누구나 동의하는 사물과 사건들의 개념을 컴퓨터가 처리 가능한 형태로 정형화한 것이다. 컴퓨터가 이러한 온톨로지에 정의된 규칙과 관계를 기반으로 추론을 수행하면 사람이 사고하는 것과 유사한 결론을 도출할 수 있다. 온톨로지는 트리플(Triple) 형태로 개념과 개념들 그리고 그들 사이에 존재하는 프로퍼티(property)를 정의함으로써 표현되는데, 월드와이드웹을 위한 표준을 개발하고 장려하는 국제 협력기구인 W3C(World Wide Web Consortium)[27]에서 온톨로지를 XML(Extensible Markup Language)[7]을 기반으로 한 OWL(Web Ontology Language)[23]로 정의하도록 권고하고 있다. 근래에 들어와서 시맨틱 웹과 관련기술의 부상으로 온톨로지에 대한 관심이 증대되었으며, 온톨로지의 활용을 극대화하기 위해서 고난이도의 추론이 가능하도록 온톨로지를 효율적으로 저장하고 검색하는 다양한 기법들이 활발히 연구되고 있다. 최근 대용량 데이터에 대한 관심이 커지면서 저장소의 확장성이 중요시되고 저렴한 비용의 데이터 관리가 필요하게 되면서 관계형 데이터베이스와 추론 엔진 기능을 결합하는 방법이 각광을 받게 되었다[4, 11, 15, 16, 19, 29]. 2011년에 시행된 데이터베이스 소프트웨어별 세계 시장 점유율 조사[3]에 따르면 관계형 데이터베이스가 90% 이상의 점유율을 보이고 있다. 따라서 트리플 데이터 저장을 위해 별도의 데이터베이스 시스템을 도입하는 것보다 관계형 데이터베이스에 저장된 기반 데이터를 토대로 추론질의 처리가 가능하다면, 기존 관계형 데이터베이스 시스템을 트리플 레파지토리로 활용하는 방향이 효용가치가 높을 것이다. 또한 관계형 데이터베이스를 이용하면 시맨틱 스키마 설계가 가능하고 스키마의 업데이트와 확장이 용이하다는 장점을 갖고 있다. 특히 관계형 데이터베이스 엔진들은 질의의 속성과 검색 조건에 부합하는 다양한 형태의 인덱스를 제공하며 있으며, 이는 대용량 데이터 검색에 우수한 성능을 발휘한다. 하지만 온톨로지 기반의 추론을 요구하는 질의를 처리하는데 적합한 인덱스는 존재하지 않기 때문에 대용량 온톨로지 데이터가 저장된 관계형 데이터베이스에 추론을 요구하는 질의를 효과적으로 처리하기 위해 하이퍼 큐브 인덱스(Hyper Cube Index)를 제안한다.

2. 온톨로지

2.1 온톨로지 개념

온톨로지는 개념과 개념 간의 프로퍼티를 특정 도메인에 맞춰 정의하여 지식을 표현하는

방법으로 시맨틱 웹, 지식관리 등의 목적으로 다양한 분야에서 널리 사용되고 있다. 온톨로지는 T-Box(Terminological compon ent)와 A-Box(Assertion component)로 나누어 정의한다. T-Box에서는 클래스의 형태로 개념을 정의하고, 프로퍼티를 통해 클래스와 클래스 사이에 존재하는 관계를 정의한다. A-Box에서는 T-Box에서 정의된 각각의 클래스에 속하는 인스턴스를 생성한다. T-Box와 A-Box에서 정의되는 온톨로지는 주어(Subject)-술어(Predicate)-목적어(Object) 형태의 트리플로 기술되며 <Figure 1>과 같이 방향성을 가지는 그래프 형태로 트리플을 표현할 수 있다.



〈Figure 1〉 Triple

W3C에서는 RDF(Resource Description Framework)[10], RDFS(RDF Schema)[3] 및 OWL(Web Ontology Language)[23]를 통해 온톨로지를 정의할 것을 표준 권고안으로 채택하였다. RDF는 리소스를 정의하며, RDFS는 T-Box에서 정의되는 클래스와 프로퍼티를 정의하며, 클래스간의 계층구조 및 프로퍼티간의 계층구조를 정의한다. 그러나 RDFS가 지닌 표현력의 한계로 인해, W3C에서는 OWL을 사용하여 T-Box를 정의하는 것을 권고하고 있다. OWL은 RDFS의 상위 집합의 개념으로 완벽하게 호환을 이루고 있으며, RDFS보다 표현력이 풍부하여 폭 넓게 지식을 표현할 수 있다. OWL에서는 오브젝트 프로퍼티가 가질 수 있는 다섯 가지의 특성을 정의하고 있으며, 이 다섯 가지 특성을 활용하면 명시적으로 기술된 온톨로지 이외의 새로운 정보를 추출해 내는 추론이 가능하게 된다.

2.2 OWL 프로퍼티 특성

OWL에서는 오브젝트 프로퍼티에 이행적(Transitive), 대칭적(Symmetric), 도치적(InverseOf), 함수적(Functional) 및 역함수적(Inverse Functional)의 다섯 가지 특성을 부여할 수 있다. 각각의 특성에 대해 다음과 같이 정의된다[23].

〈Table 1〉 Owl Property Definition

	Definition
Transitive	P(x, y), P(y, z) → P(x, z)
Symmetric	P(x, y) → P(y, z)
InverseOf	P(x, y) → P'(y, x)
Functional	P(x, y), P(x, z) → y = z
Inverse Functional	P(y, x), P(z, x) → y = z

정의된 온톨로지의 오브젝트 프로퍼티가 위 다섯 가지 특성 중의 한 가지 이상을 만족한다면, 각각의 특성에 따라 초기 온톨로지에서 명시적으로 주어지지 않았던 정보, 즉 오브젝트나 인스턴스 간에 프로퍼티가 추가적으로 성립할 수 있음을 추론해 낼 수 있다.

이행적 특성에서는 P(x, y), P(y, z)로부터 P(x, z)를 추론하며, 대칭적 특성에서는 P(x, y)로부터 P(y, x)를 추론한다. P와 P'이 도치적 관계가 있을 때 P(x, y)로부터 P'(y, x)를 추론할 수 있다. 함수적 특성에서는 P(x, y), P(x, z)로부터 y와 z가 동일함을 추론하며, 역함수적 특성에서는 P(y, x), P(z, x)로부터 y와 z가 동일함을 추론한다.

2.3 추론엔진

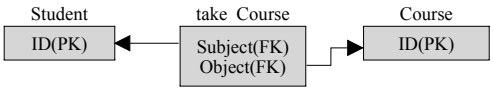
RDFS와 OWL을 통해 정의된 온톨로지에서 추론을 지원하기 위한 추론 엔진[4, 15, 16, 19, 26]들이 제안되어 왔다. 그러나 XML 원형을 그대로 유지하는 형태로 대용량 온톨로지의 추론을 처리하고자 할 때 메모리 기반 접근 방식의 부적합함을 지적하고 있다[11]. 따라서 이를 극복하기 위하여 관계형 데이터베이스를 이용하여 온톨로지를 저장하여 온톨로지에 대해 추론을 수행하는 엔진[4, 16, 19]들이 제안되고 있다. Jena[16]에서는 인스턴스들을 저장하기 위해 수직적 구조의 스키마를 사용하여 트리플의 형태로 저장하는 방식을 사용한다. 하지만 추론을 위해 반복적으로 셀프 조인을 수행해야 하므로 검색 비용이 증가하는 문제점이 발생한다. Sesame[4]에서는 T-Box에 정의되는 클래스와 프로퍼티 별로 개별적 테이블을 생성하여 온톨로지를 저장하여 트리플에서 주어와 술어가 중복 저장 되는 것을 방지하도록 한다. Sesame에서는 RDFS에 기반한 추론을 수행하기 위한 스키마를 제공하지만, OWL의 5가지 프로퍼티(<Table 1>)에 의한 추론은 수행하지 못한다는 단점이 있다.

DLDB[19]에서는 수직적 구조(e.g. Jena)와 바이너리 구조(e.g. Sesame)의 스키마 구조를 혼합하여 온톨로지를 저장하는 하이브리드 형태의 스키마를 제안한다. 하이브리드 스키마는 OWL 클래스를 관계형 데이터베이스의 테이블로 표현하고 클래스와 클래스 사이의 프로퍼티에 대한 관계 또한 테이블로 표현하여 저장하며 OWL에서 클래스의 데이터타입 프로퍼티는 클래스 테이블의 속성으로 정의한다. 하지만 추론을 수행하게 될 경우 OWL 프로퍼티 특성을 고려하지 못하기 때문에 일부 추론 결과만을 획득한다는 단점이 있다.

3. 효율적인 온톨로지 추론과 검색을 위한 인덱스

3.1 하이퍼 큐브 인덱스

하이퍼 큐브 인덱스는 관계형 데이터베이스를 기반으로 온톨로지의 효율적인 검색과 추론을 위한 다차원 비트 형태로 이루어진 인덱스이다. 온톨로지는 관계형 데이터베이스에 클래스별로 테이블을 생성하여 저장하고 클래스간의 프로퍼티는 관계(relationship) 테이블로 저장한다.



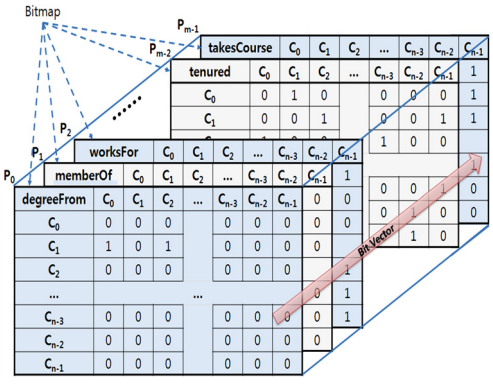
〈Figure 2〉 Ontology in Relational Database

하이퍼 큐브 인덱스는 3차원의 비트 큐브(Bit Cube)와 2차원의 비트 매트릭스(Bit Matrix)가 결합된 형태로 구성되어 다차원의 형태로 표현되며, 이를 이용하여 프로퍼티의 특성에 따른 추론을 수행하는 것이 가능해진다. 티-박스에서 정의되는 온톨로지는 비트 큐브로 표현하며, 티-박스의 정의를 통해 생성되는 에이-박스 영역의 인스턴스들에 대한 추론 결과를 비트 매트릭스로 표현한다.

비트 큐브(Bit Cube)는 온톨로지의 티-박스에서 정의된 클래스 간의 프로퍼티 속성을 비

트를 이용하여 <Figure 3>과 같이 클래스, 클래스, 프로퍼티 축으로 구성된 3차원 공간으로 표현한다. 각각의 단면은 비트맵(Bitmap)이고, 하나의 비트맵은 특정 프로퍼티 프로퍼트 특성에 따른 클래스 사이에 프로퍼트 특성이 존재하면 1, 없으면 0을 값으로 갖는다. 비트 큐브는 프로퍼트 특성에 대한 관계를 나타내는 비트맵의 집합이며, 비트 큐브에서 특정 클래스와 클래스 사이의 모든 프로퍼트 특성에 대한 관계를 비트 벡터(Bit Vector)로 표현된다. 비트맵은 클래스 수 N에 대해 N×N 2차원 행렬로 클래스와 클래스 사이의 관계를 정의하며, 프로퍼티 개수 M만큼 비트맵이 존재하고, 비트 벡터는 M개의 비트 배열로 표현된다. 만약 비트 벡터의 첫 번째 비트가 memberOf 프로퍼티이고 (C_0, C_1)에 해당하는 비트 벡터는 11000이라 한다면, 클래스 C_0와 클래스 C_1에 memberOf 관계가 있음을 확인할 수 있다.

비트 매트릭스(Bit Matrix)는 티-박스의 정의를 통해 생성되는 에이-박스 영역의 인스턴스들 간에 존재하는 프로퍼티와 프로퍼티의 특성에 따른 추론결과를 저장하는 2차원 배열의 인덱스 구조이다. 비트 매트릭스는



〈Figure 3〉 Bit Cube

<Figure 4>와 같이 각 축은 인스턴스로 구성되며, 비트 큐브와 같이 관계의 유무에 따라 0 또는 1의 값을 갖는다. 비트 매트릭스는 비트 큐브에서 정의된 클래스 간 이행적, 대칭적, 도치적, 함수적, 역함수적 프로퍼트 특성에 따라 해당 클래스의 인스턴스들 간의 추론 결과를 저장한다. 따라서 특정 인스턴스에 대해 5가지 프로퍼트 특성에 대해 관계를 갖는 인스턴스 검색에 대해 특정 인스턴스와 동일한 행과 열에 접근하여 관계를 갖는 모든 인스턴스들을 추론할 수 있다. <Figure 8>의 예를 살펴보면, degreeFrom 프로퍼티는 hasAlumnus 프로퍼티와 도치적(inverseOf) 속성을 가지고 있으므로 해당 추론 결과를 저장하고 있는 비트 매트릭스에서 인스턴스 I_1과 관계를 갖는 모든 인스턴스 검색은 I_1이 속한 행에 접근하여 관계가 있는 인스턴스 I_0, I_1, … I_{n-1}을 추론할 수 있다.

degreeFrom (InverseOf hasAlumnus)	I_0	I_1	I_2	…	I_{n-3}	I_{n-2}	I_{n-1}
I_0	0	1	0		0	0	1
I_1	1	1	1		1	1	1
I_2	0	0	0		0	0	0
…	…						
I_{n-3}	0	0	1		0	0	0
I_{n-2}	0	0	0		0	1	0
I_{n-1}	1	0	1		0	1	0

〈Figure 4〉 Bit Matrix

3.2 OWL 프로퍼티 추론 알고리즘

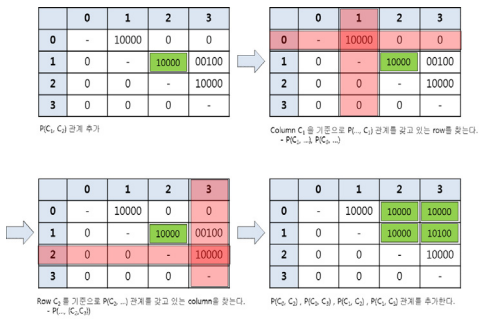
하이퍼 큐브 인덱스는 이행적, 도치적, 함수적, 역함수적 특성을 지닌 프로퍼티에 대한 추론 결과를 저장한다. T-Box에 클래스와 클래스 사이에 프로퍼티가 추가로 정의 될 경우, 프로퍼티의 특성에 따라 추론이 수행되

어야 하며 해당 클래스의 정의에 따른 인스턴스들에 대해서도 추론이 수행 되어야 한다. T-Box에서의 추론은 비트 큐브를 통해 수행된다. 추론은 비트 매트릭스를 이용하여 추론의 대상이 되는 비트 벡터와 프로퍼티의 고유 벡터와의 논리적 연산을 통해 추론 결과를 저장한다.

특정 클래스 C_i에 대해 $Px(C_i, C_y)$를 만족하는(C와 P는 각각 클래스와 프로퍼티를 의미하며 i, x, y는 각각의 식별자를 의미) P_x와 C_y들은 비트 큐브의 i번째 행을 조회하여 확인된다. 또한 C_i에 대해 $P_x(C_y, C_i)$를 만족하는 P_x와 C_y들은 비트 큐브의 i번째 열을 조회하여 확인된다. 또한, 클래스와 클래스 사이에 기존에 존재하는 프로퍼티 관계가 추가 될 때는 비트 벡터와 프로퍼티의 고유 벡터와의 OR 연산의 결과 값을 저장함으로써 비트 큐브에 반영한다. A-Box에서의 추론은 비트 매트릭스를 이용하여 추론을 수행하며, 비트 매트릭스로부터 인스턴스를 검색하는 방법은 비트 큐브를 통한 방법과 유사하다. 새로운 프로퍼티 관계가 추가 되었을 경우 비트 큐브를 이용하여 추론을 수행하는 방법은 다음과 같다.

3.2.1 이행적 프로퍼티의 추론

T-Box에서 이행적 특성의 프로퍼티 P에 대해 <Figure 5>와 같이 $P(C_i, C_j)$가 추가 되었을 경우 $P(C_x, C_i)$와 $P(C_j, C_y)$에 해당하는 C_x와 C_y들 사이에도 P관계가 있음이 추론 되어져야 한다. C_x는 P에 대한 비트맵에서 i열을 조회하여 검색할 수 있다. C_y 또한 유사한 방법으로 P에 대한 비트맵의 j행을 조회하여 검색할 수 있다. 이행적 특성의 프로퍼티에 대한 추론 결과는 검색 된 C_x와 C_y 사이에 P 관계가 있음을 정의함으로써 이뤄진다. 이행적 특성의 프로퍼티에 대한 추론 방법은 <Algorithm 1>에 기술되어 있다. 또한 해당 프로퍼티에 대한 인스턴스들에 대한 A-Box 추론은 비트 매트릭스를 이용하여 수행되며 인스턴스들 간에 이행적 특성에 따른 추론 결과를 저장하며 추론 방법은 T-Box 추론과 유사하다.



〈Figure 5〉 Inference of Transitive Property

```
Algorithm1. AddTransitiveProperty
Input : subject class Cs, object class Co, property P
Output : Inference Bit Cube
subject class list S ← NULL
object class list O ← NULL
S ← Cs + FindClassesFromColumn(Cs, P)
O ← Co + FindClassesFromRow(Co, P)
for(i = 0; i < S.length; i++){
        for(j = 0; j < O.length; j++){
                Add Inference Biot Cube(Si, P, Oi)
        }
}
```

〈Algorithm 1〉 Inference of Transitive Property

3.2.2 대칭적 프로퍼티의 추론

대칭적 특성의 프로퍼티 P에 대해 $P(C_i, C_j)$가 추가 되었을 경우 $P(C_j, C_i)$에 대해서 추론이

이루어져야 하며, 기존에 정의 또는 추론되어져 있는 $P(C_i, C_x)$, $P(C_j, C_y)$에 대해서도 $P(C_x, C_y)$, $P(C_y, C_x)$관계가 추론 되어야 한다. $P(C_j, C_i)$는 비트맵의 j번째 행과 i번재 열의 bit 값을 1로 설정하여 추론 결과를 저장할 수 있다. 또한 $P(C_y, C_x)$에 대한 추론은 비트맵의 i행과 j열을 검색하여 C_x와 C_y를 검색 후 y행과 x열에 P의 비트 벡터를 OR 연산하여 그 결과를 저장한다. 추론 결과를 저장하는 방법은 <Algorithm 2>에 기술되어 있다. 또한 해당 프로퍼티에 대한 인스턴스들에 대한 A-Box 추론은 비트 매트릭스를 이용하여 수행되며 인스턴스들 간에 대칭적 속성에 따른 추론 결과를 저장하며 추론 방법은 T-Box 추론과 유사하다.

```
Algorithm2. AddSymmetricProperty
Input : subject class Cs, object class Co, property P
Output : Inference Bit Cube
Add Inference Bit Matrix (Co, P, Cs)
subject class list S ← NULL
object class list O ← NULL
S ← Cs + FindClassesFromRow(Cs, P)
O ← Co + FindClassesFromRow(Co, P)
for(i=0; i < S.length; i++){
        for(j=0; j<O.length; j++){
                Add Inference Biot Cube(Si, P, Oi)
                Add Inference Biot Cube(Oi, P, Si)
        }
}
```

〈Algorithm 2〉 Inference of Symmetric Property

3.2.3 도치적 관계에 있는 프로퍼티의 추론

프로퍼티 P와 다른 프로퍼티 P'에 대해 서로 도치적 관계가 정의 되었을 경우 $P(C_i, C_x)$에 대해 $P'(C_x, C_i)$가 추론 되어져야 한다. C_x는 P에 대한 비트맵의 i행을 검색하여 확인할 수 있으며, P'에 대한 비트 벡터를 x행과 i열에 OR 연산하여 그 결과를 저장한다. 추론 결과를 저장하는 방법은 <Algorithm 3>에 기술되어 있다. 또한 해당 프로퍼티에 대한 인스턴스들에 대한 A-Box 추론은 프로퍼티 P와 P'에 대해 각각의 비트 매트릭스를 생성하여 수행된다. 프로퍼티 P에 대해 $P(I_i, I_j)$가 정의 되었을 경우(I는 인스턴스를 의미하며, i와 j는 각 인스턴스들에 대한 인덱스를 의미한다.) P에 해당하는 비트 매트릭스에서 i번째 행과 j번째 열에 해당하는 값을 1로 설정한다. 또한 P'에 해당하는 비트 매트릭스의 j번째 행과 i번째 열에 해당하는 값을 1로 설정하여 추론 결과를 저장한다.

```
Algorithm3. AddInverseOfProperty
Input : property P, inverse property P'
Output : Inference Bit Matrix
Subject, Object Class set {Cs1, Co1} ← FindClassesSet(P)
Subject, Object Class set {Cs2, Co2} ← FindClassesSet(P')
for(i=0; i<{Cs1, Co1}.length; i++({
            Add Inference Bit Cube(Co1, P', Cs1);
}
for(i=0; i<{Cs2, Co2}.length; i++({
          Add Inference Bit Cube(Co2, P, Cs2);
}
```

〈Algorithm 3〉 Inference of Inverse of Property

3.2.4 함수적 프로퍼티의 추론

프로퍼티 P가 함수적 특성을 지니고 있을 경우 인스턴스 사이에서 $P(I_i, I_j)$, $P(I_i, I_k)$의 관계로부터 I_j와 I_k가 동일한 인스턴스임을 추론 되어야 한다. P에 대한 인스턴스들의 비트 매트릭스를 생성하는 시점에는 비트매트릭스의 (i, j)와 (i, k)의 값을 1로 설정한다. 추론 질의 처리 시에는 단순히 i번째 행을 조회하여 1의 값으로 설정된 위치에 해당하는 인스턴스들이 모두 동치 관계에 있음을 추론할 수 있다.

3.2.5 역함수적 프로퍼티의 추론

프로퍼티 P가 역함수적 특성을 지니고 있을 경우 인스턴스 사이에서 $P(I_j, I_i)$, $P(I_k, I_i)$의 관계로부터 I_j와 I_k가 동일한 인스턴스임을 추론 되어야 한다. P에 대한 인스턴스들의 비트 매트릭스를 생성하는 시점에는 생성하는 시점에는 비트매트릭스의 (j, i)와 (k, i)의 값을 1로 설정한다. 추론 질의 처리 시에는 단순히 i번째 열을 조회하여 1의 값으로 설정된 위치에 해당하는 인스턴스들이 모두 동치 관계에 있음을 추론할 수 있다.

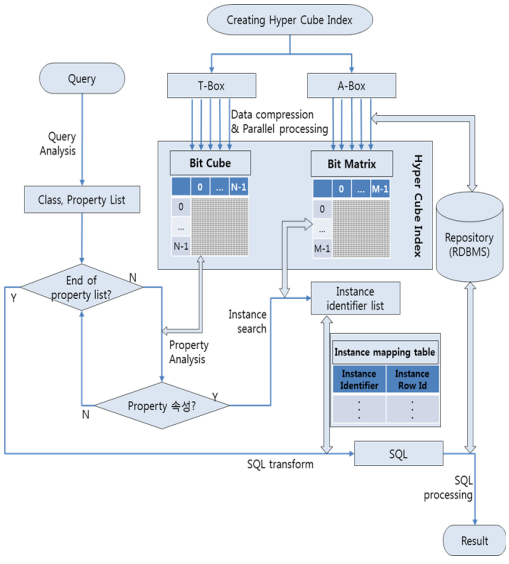
3.3 추론질의 처리

하이퍼 큐브 인덱스를 활용한 추론질의 처리과정은 <Figure 6>과 같다. 추론질의 처리를 위해 우선 하이퍼 큐브 인덱스를 생성하여야한다. 온톨로지 티-박스에 정의된 클래스 간의 관계 정보를 비트 큐브로 에이-박스의 정보를 토대로 인스턴스들 간 OWL의 5가지 프로퍼티 특성에 따른 추론결과를 비트 매트릭스로 압축과 병렬프로그래밍을 통해 하이퍼 큐브 인덱스를 생성한다. 비트 매트릭스를 생성할 때 비트 매트릭스의 인스턴스 식별자와 관계형 데이터베이스의 실제 인스턴스 매핑 테이블도 함께 생성하는데, 이는 비트 매트릭스 검색 결과인 인스턴스 식별자를 실제 관계형 데이터베이스에 저장된 데이터로 변환하기 위해 필요한 과정이다.

추론질의가 입력되어지면 질의 분석을 통해 해당 질의의 클래스와 프로퍼티 리스트를 추출한다. 다음으로 미리 생성된 하이퍼 큐브 인덱스의 비트 큐브를 활용하여 프로퍼티 리스트를 순차적으로 순회하여 5가지(이행적, 대칭적, 도치적, 함수적, 역함수적) 프로퍼티 특성의 존재 유무를 파악한다. 만약 5가지 프로퍼티 특성 중 어느 하나라도 존재할 경우 해당 클래스 인스턴스들간의 추론결과를 저장하고 있는 비트 매트릭스를 대상으로 검색을 실시하여 인스턴스 식별자 리스트를 추출한다. 프로퍼티 리스트의 마지막까지 순회하여 추출되어진 인스턴스 식별자 리스트를 인스턴스 매핑 테이블을 이용하여 실제 관계형 데이터베이스의 데이터로 변환하고, 이를 이용하여 원래 입력되어진 SparQL 질의문을 SQL 질의문으로 최종 변환하게 된다. 마지막으로 변환된 SQL 질의문을 관계형 데이터베이스 SQL 프로세서에 전달하여 추론결과를 획득한다.

예를 들어, A대학을 졸업한 모든 사람을 찾기 위해 'type Person ?X, hasAlumnus Auniversity ?X' 질의가 요구되고 온톨로지 정보에 hasAlumnus가 degreeFrom과 도치적(inverseOf) 관계가 정의되어 있으면, hasAlumnus 뿐만 아니라 degreeFrom을 통해서도 정의되어진 졸업생 역시 결과로 획득되어야한다. 예를 들어 '명지대 hasAlumnus 김인성', '명지대 hasAlumnus 송승재', '손창원 degreeFrom 명지대' 라는 트리플 정보가 있다면 명지대 졸업생으로 '김인성, 송승재' 뿐만 아니라 '손창원' 역시 검색되어져야 한다. 하이퍼 큐브 인덱스를 이용한 추론에서 위와 같은 추론질의가 입력되어지면, 해당 질의를 분석하여 클래스 리스트에 person과 university를 프로퍼티 리스트에는 hasAlumnus을 각각 저장한다. 다음으로 프로퍼티 리스트의 hasAlumnus에 대한 속성 정보를 하이퍼 큐브 인덱스의 비트

큐브를 통해 프로퍼티가 이행적, 대칭적, 도치적, 함수적, 그리고 역함수적 특성이 존재하는지 파악한다. hasAlumnus와 degreeFrom은 도치적 관계가 있으므로 이에 대한 추론 결과를 저장하고 있는 비트 매트릭스 파일 중 Auniversity을 행으로 포함하고 있는 파일을 검색 대상으로 결정하고 해당 비트 매트릭스 파일에서 특정 Auniversity 행을 기준으로 1의 값을 갖는 열 person의 인스턴스 식별자 리스트를 추출한다. 마지막으로 추출되어진 인스턴스 식별자 리스트를 인스턴스 매핑 테이블을 이용하여 관계형 데이터베이스의 데이터로 변환하고, 입력된 SparQL 질의문을 SQL 질의문으로 변환하여 관계형 데이터베이스에 처리를 요청한다. 결과적으로 hasAlumnus뿐만 아니라 degreeFrom을 통해서도 정의되어진 모든 졸업생을 추론해 낼 수 있다.



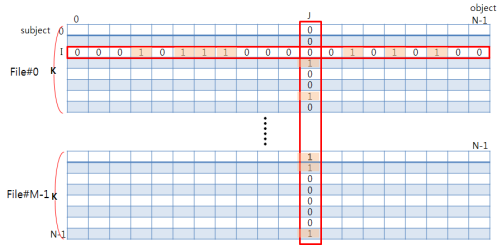
〈Figure 6〉 Inference Process Using Hyper Cube Index

4. 하이퍼 큐브 인덱스 구축방법

4.1 비트 매트릭스 구조

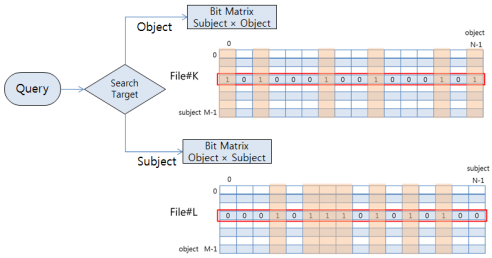
하이퍼 큐브 인덱스의 비트 매트릭스는 인스턴스들 간에 존재하는 OWL의 5가지 프로퍼트 특성(이행적, 대칭적, 도치적, 함수적, 역함수적)에 따른 추론 결과를 N×N 2차원 행렬로 표현하는데, 행은 주어(subject), 열은 목적어(object)인 주어×목적어 구조의 행렬이다. <Figure 7>과 같이 대용량 비트 매트릭스는 N개의 행을 K개씩 분할하여 M개의 파일에 나누어 저장된다. 추론 검색이 요청되어지면 기존에 모든 추론결과를 저장하고 있는 하이퍼 큐브 인덱스의 비트 매트릭스를 검색하여 결과를 획득하는데 이때, 특정 주어 I와 관계를 갖는 목적어 검색을 요구하는 질의를 처리하기 위해서는 해당 주어 I가 분포되어있는 특정 비트 매트릭스 파일 하나만을 대상으로 행 I에 대해 열 길이 N만큼 순회하여 요소 값이 1인 열을 검색하여 결과를 획득하면 된다. 하지만 모든 질의가 주어를 주고 이에 대한 목적어를 요구하는 형태만 있는 것이 아니므로 주어 검색을 요구하는 질의에 대해서도 최적화된 비트 매트릭스 구조도 고려되어져야 한다. 특정 목적어 J와 관계를 갖는 주어를 찾기 위해서는 J가 분포된 M개로 나눠진 모든 비트 매트릭스 파일들을 순회하면서 요소 값이 1인 주어를 검색해야하므로 총 M번의 파일 입출력이 발생한다. 이는 주어×목적어 비트 매트릭스 구조에서 주어 검색을 요구하는 질의를 처리하기 위해서는 비트 매트릭스 파일 수만큼의 파일 입출력이 발생하게 된다. 그러므로 주어 검색과 목적어

검색에 모두 효율을 보이는 비트 매트릭스 구조가 필요하다. 기존의 주어×목적어의 데이터 구조를 행은 목적어로 열은 주어로 치환하여 목적어×주어 구조로 변경하면 주어 검색을 요구하는 질의를 처리하기 위해서 나눠진 비트 매트릭스 파일 개수만큼이 파일 입출력이 아닌, 한 번의 파일 입출력으로 추론질의에 대한 결과를 반환할 수 있다.



〈Figure 7〉 Structure of Bit Matrix

두 가지 구조를 포함하는 비트 매트릭스를 이용한 추론 검색은 <Figure 8>과 같이, 우선 추론이 요구되어지면 질의를 분석하여 검색 대상을 파악한다. 목적어 검색이 요구되는 질의는 기존의 주어×목적어 구조의 비트 매트릭스를 통해 검색을 실시하고, 주어 검색이 요구되는 질의는 목적어×주어 구조의 비트 매트릭스를 이용하여 검색을 실시한다. 결과적으로 두 가지 구조를 포함하는 하이퍼 큐브 인덱스의 비트 매트릭스는 어떤 질의가 요구되어지더라도 하나의 비트 매트릭스 파일을 대상으로 검색을 수행하기 때문에 한 번의 파일 입출력만이 발생하게 된다. 그러므로 검색에 필요한 비트 매트릭스 파일 입출력 횟수가 비트 매트릭스 파일 수만큼이 발생하는 기존의 구조에서 어떠한 경우에도 한 번의 파일 입출력으로 검색이 가능한 두 가지 구조를 포함하는 비트 매트릭스를 생성함으로써 검색속도가 향상되는 효과가 있다.

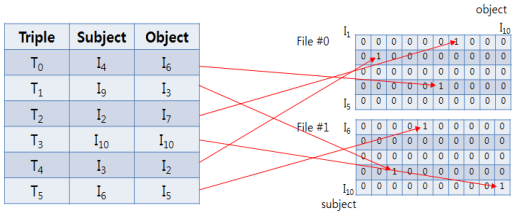


〈Figure 8〉 Inference of Bit Matrix

4.2 비트 매트릭스 생성 알고리즘

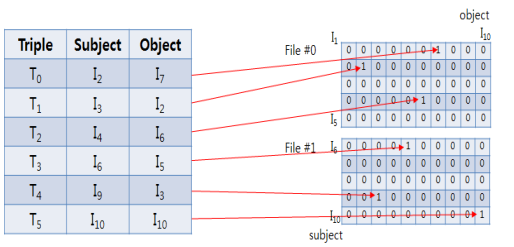
비트 매트릭스를 생성 시 대용량 비트 매트릭스를 한 번에 메모리에 적재 불가능하기 때문에 일부만을 생성하고 이를 파일로 출력하는데 이때, 파일로 출력하기 이전에 해당 비트 매트릭스에 관여하는 모든 트리플 정보를 반영해야지만 파일로 출력된 비트 매트릭스를 다시 메모리에 적재하지 않을 수 있다. 만약 파일로 출력하기 이전에 비트 매트릭스에 트리플 정보를 모두 반영하지 못한다면, 후에 파일로 출력된 비트 매트릭스 파일을 다시 메모리에 적재하고 트리플 데이터 반영을 실시해야하므로 잦은 파일 입출력이 발생한다. 만약 비트 매트릭스를 적재 가능한 메모리의 용량이 50비트이고 인스턴스가 I1부터 I10라면, 비트 매트릭스 크기는 10×10 = 100비트로 주어진 메모리 용량한계 때문에 2개의 파일로 나누어 저장된다. <Figure 9>와 같이 T0부터 T5까지 순차적으로 트리플을 탐색해가면서 비트 매트릭스를 생성할 때 최악의 경우 File #0과 File #1을 번갈아 가면서 메모리에 적재해야 하므로 6번의 파일 입출

력이 발생한다.



〈Figure 9〉 Generating of Bit Matrix by Unordered Triples

하지만 트리플 데이터에는 주어와 목적어 정보가 포함되어 있기 때문에 주어, 또는 목적어로 트리플 데이터를 정렬한다면 정렬되어 있는 데이터를 토대로 비트 매트릭스를 순차적으로 생성가능하게 된다. 예를 들어 <Figure 10>과 같이 주어를 기준으로 정렬된 트리플 데이터를 이용하여 주어×목적어 구조의 비트 매트릭스를 생성한다면, 트리플 데이터가 주어(행)를 기준으로 정렬되어 있으므로 T0부터 T5까지 순차적으로 트리플을 탐색해가면서 비트 매트릭스를 생성할 수 있어 이전 행에 다시 접근할 필요가 없어 추가적인 파일 입출력이 발생하지 않기 때문에 순차적으로 비트 매트릭스를 생성하는 것이 가능하게 된다.



〈Figure 10〉 Generating Bit Matrix by Ordered Triples

따라서 사전에 트리플 데이터를 정렬하고 주어와 목적어 정보를 이용하여 2차원 비트 매트릭스의 요소에 직접 접근하여 트리플 정보를 비트 매트릭스에 표현한다면 한번 파일로 출력된 비트 매트릭스를 재차 메모리에 적재하여 사용할 필요가 없어져, 정렬된 트리플 데이터를 한번만 순회하여 비트 매트릭스를 생성하는 것이 가능해 진다. 트리플 데이터 정렬을 위해 사용된 알고리즘은 병합정렬(Merge Sort)로 수행시간이 O(NlogN)이다. <Figure 11>과 같이 우선 트리플 데이터를 생성하고자하는 비트 매트릭스 구조에 따라 주어 또는 목적어로 정렬하고 정렬된 트리플 데이터를 순차적으로 비트 매트릭스에 표현한다. 결과적으로 하이퍼큐브 인덱스 생성 알고리즘은 트리플 데이터 정렬시간 O(NlogN)과 비트 매트릭스 생성시간 O(N)를 포함하는 알고리즘이다.

```
k = the number of maximum Bit Matrix of rows that can
      be loaded into memory;
file_index = 0;

triple = MergeSort(triple, object | subject);

for(i=0; i<Number of Triples; i++) {
    if((triple[i].subject / k) > file_index) {
        output bitmatrix;
        bitmatrix.init(k);
        file_index++;
    }
    subject = (triple[i].subject) % k;
    object = (triple[i].object);
    bitmatrix.set(subject, object);
}
```

〈Figure 11〉 Algorithm of Generating Bit Matrix

4.3 병렬프로그래밍

비트 매트릭스 생성 알고리즘에서 트리플 수와 인스턴스 수는 데이터 집합이 커질수록 기

하급수적으로 증가하여 하이퍼 큐브 인덱스 생성에 많은 시간이 소요되게 된다. 따라서 본 논문에서는 JOMP(OpenMP for Java)[5]를 적용하여 생성속도 향상을 도모한다. JOMP는 Unix 플랫폼과 Windows NT 플랫폼을 포함한 C/C++, Fortran에서 모든 아키텍처에서 병렬프로그램 개발을 위해 간단하고 유연한 메모리 공유 병렬 프로그래밍을 지원하는 OpenMP(Open Multi-Processing)[17]를 Java에서도 사용가능하도록 지원하는 API(Application Program Interface)이다. 병렬프로그램을 적용할 경우 이론상으로 프로세스 개수에 비례하여하여 수행속도가 향상되므로 결과적으로 하이퍼 큐브 인덱스 생성 시간 단축을 도모할 수 있다. 또한 JOMP는 병렬처리 지시어를 통해 동적으로 루틴을 구성하고 병렬화가 가능함으로 높은 확장성과 코드작성의 용이성을 나타낸다.

따라서 본 논문에서는 JOMP 병렬프로그래밍 기법을 하이퍼 큐브 인덱스 생성 알고리즘과 비트 매트릭스 검색 알고리즘에 적용하였다. 대표적으로 <Figure 12>와 같이 비트 매트릭스 생성 알고리즘에서 비트 매트릭스를 공유 메모리영역에 할당하고 트리플 수에 따른 반복문을 병렬처리 하였다. 공유 메모리영역의 비트 매트릭스와 트리플 정보를 병렬로 수행되는 스레드가 동시적으로 접근하여 비트 매트릭스

```
//omp for parallel shared(bitmatrix, triple) private(subject, object, i)
for(i=0; i<Number of Triples; i++){
    ...
    subject = triple[i].subject % k;
    object = triple[i].object;
    bitmatrix.set(subject, object);

    ...
}
```

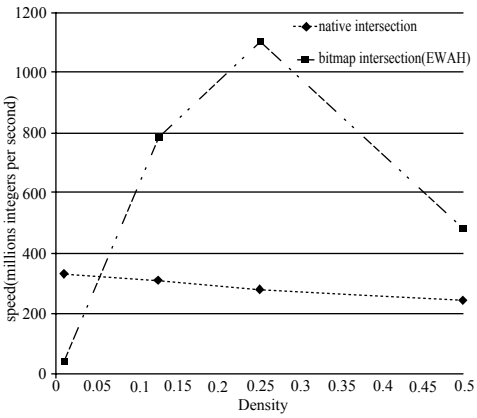
〈Figure 12〉 Algorithm of Generating Bit Matrix with JOMP

를 생성하게 되므로 생성속도가 향상된다.

4.4 비트 매트릭스 압축

비트 매트릭스는 기존의 관계형 데이터베이스 인덱스의 한 종류인 비트맵 인덱스(Bitmap index)와 매우 유사하다. 관계형 데이터베이스 시스템에서 비트맵 인덱스를 저장할 때, 일반적으로 Run-Length Encoding(RLE)[21] 알고리즘 기반의 Byte-aligned Bitmap Code(BBC)[1], Word-Aligned Hybrid Code(WAH)[28], 혹은 Enhanced Word-Aligned Hybrid(EWAH)[12, 13]와 같은 압축 알고리즘으로 비트맵을 압축 저장하여 공간을 절약한다. 이와 같은 알고리즘은 저장공간 절약뿐만 아니라, 압축된 비트맵의 압축해제 없이 비트연산이 가능하다는 장점을 가지고 있다. 또한 비트연산 속도에 있어서 비트맵을 이용한 비트연산보다 압축 알고리즘을 통한 비트연산이 보다 빠르다는 장점도 있다. WAH 알고리즘은 word 단위로 압축 및 연산을 실시하는 방법으로 byte 단위의 BBC보다 평균 10배 빠른 성능을 보이지만, 압축효율에서 최악의 경우 3% 데이터 확장이 발생할 수 있다. 이러한 WAH의 단점을 보완하고 보다 빠른 비트연산 처리를 위해 EWAH가 고안되었다. EWAH 알고리즘은 압축효율에서 최악의 경우에도 압축 이전의 크기보다 0.01% 이상 커지는 것을 방지하며, WAH에 비해 평균 3% 이상의 빠른 성능을 보인다[13]. <Figure 13>처럼 밀도에 따른 초당 비트연산 가능한 정수 수[14]에 따르면, EWAH를 이용한 비트연산 속도가 비트맵을 이용한 비트연산 속도보다 빠른 것을 확인할 수 있다. 따라서 하이퍼 큐브 인덱스의 비트 매트릭스 압축을 위하여 가장 좋은 성능

을 보이는 것으로 알려진 EWAH 압축기법을 이용하여 압축한다.

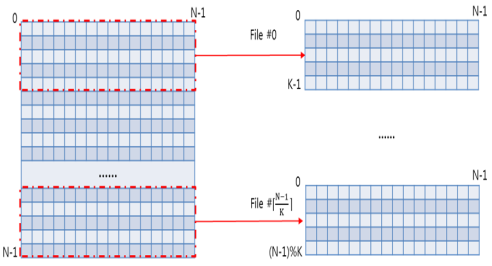


〈Figure 13〉 Bit-operation Speed According to the Density〔14〕

EWAH 압축 알고리즘을 적용하여 비트 매트릭스를 압축하더라도 비트 매트릭스 크기가 한 번에 메모리에 적재 가능하도록 압축되는 것을 보장하는 것은 아니다. 예를 들어 LUBM(200,0) 데이터 집합에서 비트맵 형태의 비트 매트릭스 크기는 49.35기가바이트로 EWAH 알고리즘을 적용하더라도 이론적으로는 최악의 경우 비트 매트릭스 크기가 오히려 0.01% 확장될 수도 있으므로 원래에서 약간 늘어날 수도 있다는 것을 의미한다. 만약 메모리 용량이 1기가 바이트라 할 때, LUBM(200,0) 데이터 집합에서 인스턴스는 651,112개이고 비트 매트릭스 크기는 651,112×651,112 ≒ 49.35기가 바이트로 메모리 용량 한계를 고려하여 비트 매트릭스를 한 번에 메모리에 적재 가능한 행의 개수만큼씩 분할하여 메모리에 적재하여야 한다. 따라서 메모리에 적재 가능한 비트 매트릭스 행의 개수를 K라 하면 (K × 651,112) 비트 ≤ 1기가 바이트로 K는 13,192보다 작거나 같아야 한다. K가 13,192라면 우선 0부터 13,191행까지의 비트 매트릭스를 메모리에 할당하여 트리플 정보를 반영하고 트리플의 주어(행)가 13,192에 도달하면 파일로 출력한다. 그리고 다음 비트 매트릭스 행인 13,192행부터 26,383행까지를 메모리에 할당하고 파일로 출력하는 작업을 반복하여 비트 매트릭스를 총 50개의 파일로 나누어 저장한다. 하지만 메모리에 적재 가능한 비트 매트릭스 행의 개수 K를 14,000이라한다면 13,193번째 행을 생성할 때, 비트 매트릭스 크기는 13,193×651,112비트로 1기가 바이트보다 크므로 메모리 오버플로우가 발생하고, K의 값이 5,000이라면 비트 매트릭스는 131개의 파일로 분할 저장되므로 131회 파일 입출력이 발생하게 된다. 따라서 비트 매트릭스를 여러 개의 파일로 분할하여 저장할 때, 메모리 오버플로우와 최소 파일 입출력 횟수를 고려한 메모리에 적재 가능한 비트 매트릭스 최대 행의 개수를 결정하고 압축된 비트 매트릭스를 효율적으로 분할하여 저장하는 방법이 필요하다.

비트 매트릭스 분할을 위해 본 논문에서는 N×N 2차원 행렬의 비트 매트릭스를 <Figure 14>와 같이 메모리에 적재 가능한 최대 비트 매트릭스 행의 개수 K로 전체 비트 매트릭스를 나누어, K×N로 파일로 저장한다. 이때 EWAH 알고리즘의 최악의 압축효율은 100.01%이므로 압축 알고리즘을 적용한 비트 매트릭스의 최대 크기는 1.00001×K×N비트이며, K의 값은 비트 매트릭스를 위해 할당 가능한 최대 메모리의 크기에 따라 달라진다. 비트 매트릭스를 위해 할당 가능한 메모리의 크기를 MS(Memory Size), 인스턴스의 개수 N으로 메모리에 적재

가능한 최대 행의 개수 K를 수식으로 표현하면 $1.00001 \times K \times N \leq MS$로서, $1 \leq K \leq \frac{MS}{1.0001 \times N}$ 이다. 그러므로 메모리 오버플로우를 발생하지 않고 최소한의 파일 입출력을 보장하는 최대 행의 수 K는 $[\frac{MS}{1.0001 \times N}]$ 이다. 이에 따라, 전체 행 N개를 메모리에 적재 가능한 행의 수 K로 나누어 구성하면 파일 당 저장되는 인덱스 데이터는 $K \times N$이며, 파일 수는 $\lceil N / K \rceil$ 이다. 그 결과로 비트 매트릭스를 메모리에 적재가 가능한 최대의 크기로 분할하여 최소의 파일 수에 나누어 저장할 수 있으며, 메모리 오버플로우 발생을 방지할 수 있다.



〈Figure 14〉 Bit Matrix Division

5. 실험 환경 및 결과

5.1 실험 환경 및 방법

본 논문의 실험에서는 하이퍼 큐브 인덱스의 우수성을 입증하기 위해 LUBM에서 제공하는 데이터 집합과 테스트 질의를 이용하여 성능평가 항목들에 대한 실험을 실시하였다. LUBM에서는 데이터 집합을 LUBM(N,0)으로 표현한다. N은 대학 수를 나타내며, 1, 5, 10, 20, 50, 100, 200개의 대학 수의 데이터 집합을 생성하여 실험을 수행하였으며, 실험에 사용된 시스템의 사양은 <Table 2>와 같다.

〈Table 2〉 Experimental Environment

	Specifications
CPU	Intel i5-3470 @ 3.2GHz
RAM	16.0GB
System	Windows 7 Enterprise K 64bit JDK 1.7.0(JVM Memory : 10GB) MySQL 5.5

실험 과정은 비트 매트릭스 압축과 병렬프로그래밍을 포함한 하이퍼 큐브 인덱스 생성 방법으로 하이퍼 큐브 인덱스를 생성하여 소요시간과 크기를 측정하여 생성속도와 압축률을 비교한다. 또한 병렬프로그래밍 적용 전후의 생성시간을 측정하여 하이퍼 큐브 인덱스 생성속도를 비교한다. 다음으로, 하이퍼 큐브 인덱스를 이용하여 추론질의를 수행하고 추론질의 결과를 모범 질의응답 결과와 비교함으로서 추론결과에 대한 완전성을 측정한다. 마지막으로 트리플 데이터를 저장하고 추론을 지원하는 상업용 관계형 데이터베이스 중 가장 널리 사용되는 Oracle[18]에서의 추론결과 저장시간을 측정하여 하이퍼 큐브 인덱스 생성시간과 비교하고, Oracle의 질의응답 시간을 측정하여 하이퍼 큐브 인덱스를 이용한 질의응답 시간과 비교한다. 각 실험 과정에서 측정된 결과를 분석하여 본 논문에서 제안한 하이퍼 큐브 인덱스 생성방법의 실용성과 추론 검색의 효율성, 그리고 Oracle과 비교를 통해 하이퍼 큐브 인덱스를 이용한 추론질의 처리의 우수성을 입증한다.

5.2 비트 매트릭스 압축

첫 번째 성능 평가 항목인 비트 매트릭스 압축은 본 논문에서 제안하는 비트 매트릭스 압축방법을 이용하여 비트 매트릭스의 크기를 측정함으로서 평가한다. 본 실험에서는 Java로 하이퍼 큐브 인덱스 생성을 구현하였고, EWAH 압축방법을 적용하여 비트 매트릭스 저장 공간을 더욱 절약하고, 비트연산 속도의 효율을 높였다. 비트맵으로 저장하였을 경우의 비트 매트릭스의 크기와 EWAH 압축방법을 적용하였을 때의 비트 매트릭스 크기를 <Table 3>에 정리하였다. 실험의 결과를 확인해보면, 비트 매트릭스 압축을 통해 저장공간을 효율적으로 줄일 수 있다.

〈Table 3〉 Bit Matrix Size by EWAH

	Before	After
LUBM(1,0)	1.37 MB	112 KB
LUBM(5,0)	31.57 MB	528 KB
LUBM(10,0)	121.02 MB	925 KB
LUBM(20,0)	518.33 MB	1.78 MB
LUBM(50,0)	3.09 GB	4.31 MB
LUBM(100,0)	12.36 GB	9.32 MB
LUBM(200,0)	49.35 GB	16.6 MB

5.3 하이퍼 큐브 인덱스 생성시간

하이퍼 큐브 인덱스 생성에 있어 비트 매트릭스에 EWAH 압축방법을 적용하였으며, 이는 비트연산에서 비트맵보다 빠른 속도를 보이고 데이터 압축을 통해 파일 입출력에서도 우수한 성능을 보인다. 또한, 본 논문에서는 하이퍼 큐브 인덱스 생성 알고리즘에서 반복 조건의 대상이 되는 트리플 수와 인스턴스 수는 LUBM의 데이터 집합(대학교 수)이 커질수록 기하급수적으로 증가하므로 수정된 하이퍼 큐브 인덱스 생성 알고리즘에 JOMP 병렬프로그래밍을 적용하여 생성속도 향상을 도모하였다. <Table 3>은 비트맵으로 비트 매트릭스를 저장할 때의 하이퍼 큐브 인덱스 생성속도와 EWAH 압축방법을 적용한 하이퍼 큐브 인덱스 생성속도, 그리고 병렬프로그래밍을 적용한 하이퍼 큐브 인덱스 생성속도 비교 결과이다. 결과적으로 EWAH 압축방법을 적용한 방법이 비트맵 방식보다 3배 이상의 빠른 속도를 보이며, 이에 병렬처리를 적용하였을 경우 10% 성능향상을 나타낸다.

〈Table 4〉 Time for Hyper Cube Index creation(sec)

	Bitmap	EWAH	EWAH + JOMP
LUBM (1,0)	0.598	0.452	0.420
LUBM (5,0)	2.993	1.450	1.319
LUBM (10,0)	6.200	2.602	2.368
LUBM (20,0)	49.206	5.399	4.913
LUBM (50,0)	1763247	63.7431	56.721
LUBM (100,0)	517.406	181.465	159.682
LUBM (200,0)	832.750	245.475	223.328

5.4 질의 결과에 대한 완전성

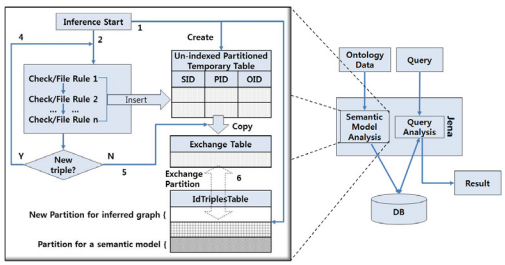
LUBM에서는 추론엔진의 성능 평가를 위하여 14가지의 질의를 제공하고 있다. 질의는 SparQL 형태로 제공이 되며, 각 질의에 대한 결과 값을 제공한다. LUBM 성능 평가 항목 중 하나인 질의 결과에 대한 완전성[31]은 (질의결과 ÷ LUBM에서 제공하는 결과)×100의 방법으로 측정하도록 정의하였다. 질의에 대한 완전성 결과는 <Table 4>와 같으며, 시맨틱 웹 도구로 관계형 데이터베이스를 이용하여 하이브리드 형태의 스키마를 제안한 추론 엔진 중 하나인 DLDB[19]에서는 OWL 프로퍼트 특성에 대한 추론을 지원하지 않기 때문에 LUBM 테스트 질의 중 이행적, 도치적 프로퍼트 특성에 대한 추론을 요구하는 11, 12, 13질의에서 0% 질의 결과 완전성을 보이지만, 하이퍼 큐브 인덱스는 OWL의 5가지 프로퍼트 특성에 따른 추론을 지원하므로 100% 질의 결과 완전성을 보장한다.

〈Table 5〉 Completeness(%)

Query	DLDB	Hyper Cube
1	100	100
2	100	100
3	100	100
4	100	100
5	100	100
6	100	100
7	100	100
8	100	100
9	100	100
10	100	100
11	0	100
12	0	100
13	0	100
14	100	100

5.5 하이퍼 큐브 인덱스 VS. Oracle

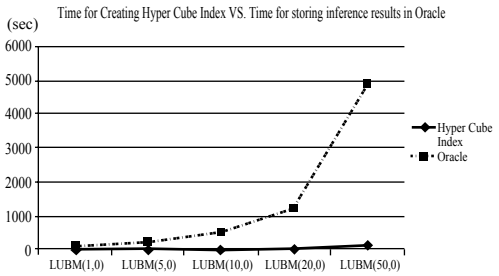
하이퍼 큐브 인덱스를 이용한 추론의 우수성을 입증하기 위해 트리플 데이터를 저장하고 추론을 지원하는 상업용 관계형 데이터베이스 중 가장 널리 사용되는 Oracle과 추론질의 응답시간을 비교분석하였다. Oracle[18]은 10g 이후 버전부터 Oracle Spatial의 일부분으로써 시맨틱 웹 데이터와 온톨로지에 대한 저장, 질의, 추론 기능을 지원하고 있다. 시맨틱 웹 기술을 지원하기 위한 별도의 관리자 계정(MDSYS)을 두고 그 하부에서 시맨틱 기술에 특화된 다양한 프로시저들을 제공하며, Jena API[9]를 지원하는 Jena Adaptor를 이용하여 Java 인터페이스를 제공한다.



〈Figure 15〉 Oracle Semantic Technology

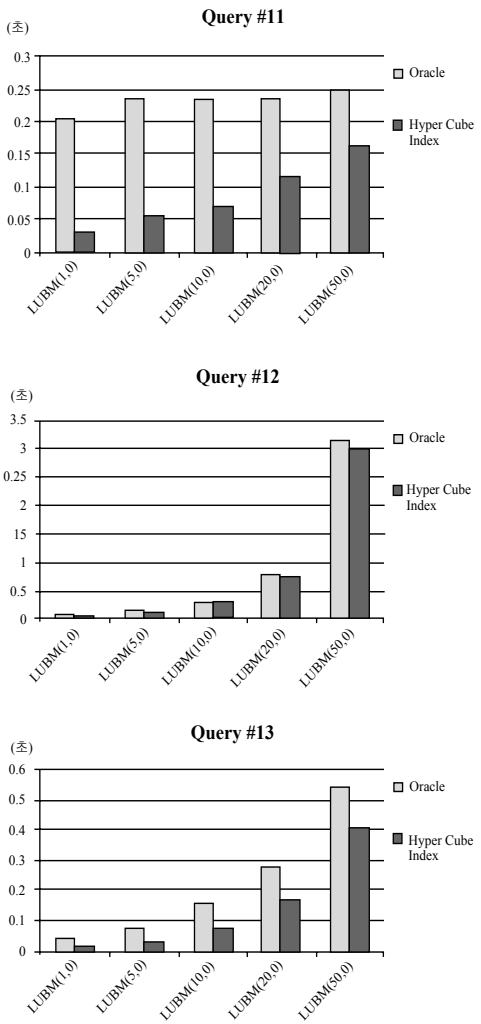
본 실험에서는 LUBM 데이터 집합과 11, 12, 13번 테스트 질의를 이용하여 Oracle Sematic Technology를 이용한 질의 응답시간을 하이퍼 큐브 인덱스를 이용한 질의 응답시간을 비교한다. Oracle에서는 추론질의 처리과정은 온톨로지 데이터 모델을 분석하여 추론 가능한 모든 정

보를 저장한 후에 저장되어진 추론결과를 대상으로 추론질의를 수행하여 결과를 획득하는 방법이다. 따라서 Oracle에서 추론결과 저장시간을 하이퍼 큐브 인덱스 생성시간과 비교 대상으로 삼았으며, 비교결과는 <Figure 16>과 같이 하이퍼 큐브 인덱스의 생성속도가 Oracle에 비해 빠르게 나타난다. 이유는 관계형 데이터베이스를 기반의 하이퍼 큐브 인덱스에서는 IsA, subClassOf, subPropertyOf와 같은 추론정보는 트리플 데이터를 데이터베이스에 적재 시 시맨틱 스키마로 반영되어 저장되지만, Oracle의 경우에는 IsA, subClassOf, subPropertyOf 등을 포함하는 추론 정보 모두를 저장해야 하므로 상대적으로 많은 추론결과 저장시간이 필요하게 되기 때문이다.



〈Figure 16〉 Time for index creation

다음으로 하이퍼 큐브 인덱스의 비트 매트릭스 파일 적재시간, 검색시간과 관계형 데이터베이스의 질의 응답시간을 포함하는 질의 처리시간과 Oracle의 질의 응답시간을 비교하였다. 결과는 <Figure 17>과 같으며, LUBM(50,0)의 데이터 집합까지는 하이퍼 큐브 인덱스를 이용한 질의 처리속도가 Oracle의 질의 응답속도와 비슷하거나 근소한 우수함을 보인다. 하지만 질의 11에서 보면 Oracle의 경우 데이터 집합과는 무관하게 동일한 응답시간을 보이지만 하이퍼 큐브 인덱스는 데이터 집합이 커짐에 따라 질의 처리시간이 비례적으로 커지는 것을 확인할 수 있다. 따라서 하이퍼 큐브 인덱스를 이용한 질의 처리시간은 LUBM(200,0)에서 Oracle과 비슷한 성능을 보이고 그보다 큰 데이터 집합에서는 Oracle의 질의 응답속도가 빠르게 나타날 수도 있을 것으로 예상된다.



〈Figure 17〉 Query Response Time

6. 결론 및 향후 연구과제

본 논문에서는 관계형 데이터베이스를 이용하여 대용량의 온톨로지에 대한 효율적 추론을 수행하기 위해 하이퍼 큐브 인덱스와 생성방식을 제안하였다. 추론은 프로퍼티의 특성에 따라 본 논문에서 정의된 알고리즘에 의해 수행되며, T-Box와 A-Box 각각에 대한 비트 큐브와 비트 매트릭스를 하이퍼 큐브에 포함시켜 정의함으로써, 추론을 수행할 수 있게 하였다. 추론은 단순히 비트 레벨의 논리적 연산을 통해 수행 하므로 계산 과정이 간단하며, 그 결과 또한 쉽게 획득할 수 있었다. 또한 본 논문의 실험은 우리 주변에서 쉽게 볼 수 있는 하드웨어 환경의 컴퓨터에서 실용적인 시간에 하이퍼 큐브 인덱스 생성과 이를 이용한 효율적인 추론이 가능함을 확인할 수 있었다. 또한 DLDB[19]에서 추론 결과를 획득할 수 없었던 LUBM 11, 12, 13 질의에 대해 100% 완전성을 보장하며, 특히 Oracle과의 질의 응답속도에서 확인한 것과 같이 대용량의 데이터에 대해서도 하이퍼 큐브 인덱스를 이용한 검색에 효율이 있음을 보여주었다. 다만, 질의처리 시간이 온톨로지가 커짐에 따라 추론질의 처리시간이 비례하여 선형적으로 증가함을 보이므로 이를 개선하기 위한 추가적인 연구가 필요하다.

만약 대용량 온톨로지를 위한 하드웨어 환경이 지원된다면, 하이퍼 큐브 인덱스 생성에서 큰 비중을 차지하는 파일 입출력이 감소될 것이므로 생성속도의 향상을 기대할 수 있다. 또한 데이터 적재에 소요되는 시간이 더 이상 필요치 않게 되므로 질의응답 속도 역시 향상될 수 있을 것이다.

하이퍼 큐브 인덱스는 관계형 데이터베이스에 저장된 온톨로지의 효율적인 추론을 위한 인덱스이다. 인덱스를 생성하고 이를 활용하기 위해서는 관계형 데이터베이스에 온톨로지 데이터를 적재하는 과정이 필요하며, 적재시간은 온톨로지 데이터가 커짐에 따라 크게 증가한다. 본 논문에서 제시한 생성방법을 통해 실용적인 시간 내에 하이퍼 큐브 인덱스 생성이 가능함이 확인할 수 있었지만, 빅 데이터(Big Data) 시대에 걸맞은 대용량 온톨로지를 실제 처리할 수 있는지에 대해 확신하기는 어려운 실정이다. 따라서 빅 데이터를 효율적으로 다룰 수 있는 기법을 적용하여 관계형 데이터베이스에 온톨로지 정보를 저장하고 하이퍼 큐브 인덱스를 생성하는 방법이 연구되고 실험을 통해 가능함이 실증되어져야 한다. 최근 빅 데이터를 효과적으로 처리하기 위해 하둡(Hadoop)[24]가 크게 관심을 받고 있다. 따라서 향후, 하둡을 이용한 빅 데이터 수준의 대용량 온톨로지 적재와 본 논문에서 제시한 하이퍼 큐브 인덱스 생성방법을 접목하여 효율적으로 추론질의를 처리하기 위한 연구가 진행되어야 할 것이다.

References

[1] Antoshenkov, G., “Byte-aligned bitmap compression,” In Proc. DCC’95, p. 476, 1995.

[2] Berners-Lee, T., Hendler, J., and Lassila, O., “The Semantic Web,” Scientific America, 2001.

[3] Brickley, D. and Guha, R. V., “Resource Description Framework(RDF) Schema Specication 1.0,” Candidate recommendation, World Wide Web Consortium, March 2000.

[4] Broekstra, J., Kampman, A., and van Harmelen, F., “Sesame : A generic Architecture for StOR and Querying RDF and RDF Schema,” In proc. of International Semantic Web Conference, Sardinia, Italia, pp. 54-68, 2002.

[5] Bull, J., Westhead, M., Kambites, M., and Obdr´zalek, J., “Towards OpenMP for Java,” In European Workshop on Open MP, 2000.

[6] CNN, “Issue of Fortune,” Vol. 163, No. 5, 2011.

[7] Extensible Markup Language(XML), http://www.w3.org/XML/.

[8] Grigoris Antoniou and Frank van Harmelen, “A Semantic Web Primer” 2nd Edition, The MIT Press, 2008.

[9] Jena, http://jena.apache.org/.

[10] Lassila, O. and Swick, R. R., “Resource Description Framework(RDF) : Model and Syntax Specication.,” Recommendation, World Wide Web Consortium, Feb. 1999.

[11] Lee, J. and Goodwin, R., “Ontology Management for Large-Scale E-Commerce Applications,” Electronic Commerce Research and Applications, Elsevier, pp. 7-15, Sept. 2005.

[12] Lemire, D., “Enhanced Word-Aligned Hybrid(EWAH),” http://code.google.com/p/javaewah/.

[13] Lemire, D., Kaser, O., and Aouiche, K., “Sorting improves word-aligned bitmap indexes,” Data and Knowledge Engineering, pp. 3-28, 2010.

[14] Lemire, D., “When is a bitmap faster than an integer list?,” http://lemire.me/blog/archives/2012/10/23/when-is-a-bitmap-faster-than-an-integer-list/.

[15] Lin, J., Lee, J., and Chung, C., “An Efficient Reasoning Method for OWL Properties using Relational Databases,” Journal of Korean Information Service System, Vol. 29 No. 1, pp. 92-103, 2010.

[16] McBirds, B., “Jena : A semantic web toolkit,” Institute of Electrical and Electronics Engineers Internet Computing, Vol. 6, No. 6, pp. 55-59, Nov. 2002.

[17] Open MP(Open Multi-Processing), http://www.openmp.org/.

[18] Oracle Semantic Technologies, http://www.oracle.com/technetwork/database/options/semantic-tech/index.html.

[19] Pan, Z. and Heflin, J., “DLDB : Extending Relational Databases to Support Semantic Web Queries,” In Proc. Practical and Scalable Semantic Systems, Sanibel Island, Florida, USA, pp. 109-113, 2003.

[20] Park, S. U., “Development of a Semantic Web Portal for Industry Knowledge Sharing,” The Journal of Society for e-Business Studies, Vol. 14, No. 4, pp. 195-214, 2009.

[21] Run-Length Encoding(RLE), http://en.w

ikipedia.org/wiki/Run-length_encoding.

[22] Semantic Web, http://semanticweb.org.

[23] Smith, M., Welty, C., and McGuinness, D., "OWL Web Ontology Language Guide," http://www.w3.org/TR/2004/REC-owl-features-20040210/#s3.3.

[24] The Apache Software Foundation, "Apache Hadoop," http://hadoop.apache.org/.

[25] Thomas, R. Gruber, "A translation approach to porTable ontologies," Knowledge Acquisition, Vol. 5, No. 2, pp. 199-220, 1993.

[26] Volz, R., Oberle, D., Staab, S., and Motik, B., "KAON SERVER : A Semantic Web Management System," In proc. of the Atlantic Web Intelligent Conference, Hungry, Budapest, p. 29, 2003.

[27] World Wide Web Consortium (W3C), http://www.w3.org/.

[28] Wu, K., Otoo, E. J., and Shoshani, A., "Optimizing bitmap indices with efficient compression," ACM Transactions on Database Systems, pp. 1-38, 2006.

[29] Wu, Z., Eadon, G., Das, S., Chong, E. I., Kolovski, V., Annamalai, M., and Srinivasan, J., "Implementing an inference engine for RDFS/OWL constructs and user-define d rules in Oracle," InProc.ICDE-2008, pp. 1239-1248, 2008.

[30] Yoo, D. H. and Suh, Y. M., "An Ontology -based Hotel Search System Using Semantic Web Technologies," The Journal of Society for e-Business Studies, Vol. 13, No. 4, pp. 71-92, 2008.

[31] Yuanbo, Guo, Zhengxiang, Pan, and Jeff, Heflin, "LUBM : A Benchmark for OWL Knowledge Base Systems," In Proc. of International Semantic Web Conference, Hiroshima, Japan, 2004.

저 자 소 개

송승재 (E-mail : sjsong@detnc.com)
2011년 명지대학교 컴퓨터공학과 (학사)
2013년 명지대학교 컴퓨터공학과 (석사)
2013년～현재 ㈜대양이티엔씨 유비쿼터스 기술연구소 연구원
관심분야 데이터베이스, 정보검색, 시맨틱 웹, 빅데이터

김인성 (E-mail : insungkim@promptech.co.kr)
2010년 명지대학교 컴퓨터공학과 (학사)
2012년 명지대학교 컴퓨터공학과 (석사)
2012년～현재 ㈜프람트테크놀로지 기술연구소 연구원
관심분야 데이터베이스, 정보검색, 시맨틱 웹, 소셜네트워크

전종훈 (E-mail : jchun@mju.ac.kr)
1986년 미국 University of Denver 컴퓨터과학 (학사)
1988년 미국 Northwestern University 컴퓨터공학 (석사)
1992년 미국 Northwestern University 컴퓨터공학 (박사)
2003년～현재 명지대학교 컴퓨터공학과 교수
2007년～현재 한국 전자거래학회 이사
2007년～현재 한국 정보과학회 데이터베이스 소사이어티 이사
관심분야 데이터베이스, 정보검색, 빅테이터, 시맨틱 웹