다차원 색인 구조를 위한 효율적인 압축 방법
(An Efficient Compression Method for Multi-dimensional Index Structures)

조 형 주 † 정 진 완 ††
(Hyung-Ju Cho) (Chin-Wan Chung)

요 약 지난 십년 동안, CPU의 발전 속도는 메모리나 디스크의 발전 속도를 훨씬 능가하였다. 이것이 압축 방법을 사용하여 데이타베이스 크기를 줄이거나 질의 비용을 줄일 수 있게 만들었다. 다양한 데이타베이스 연구 분야에서 압축 방법이 사용되고 있지만, 다차원 색인 구조를 압축하는 연구는 거의 없다. 본 논문에서는 다차원 색인 구조를 위한 HEM(Hybrid Encoding Method)이라는 압축 방법을 제안한다. HEM 압축 방법은 다차원 색인 구조의 크기 뿐만 아니라, 질의 비용도 크게 줄일 수 있다. 수학적인 분석과 다양한 실험을 통하여, 우리는 HEM 압축 방법이 기존에 제안되었던 압축 방법보다 색인 크기와 질의 비용 측면에서 우수하다는 것을 보여준다.
키워드 : 다차원 색인 구조, 압축 방법, R*-트리

Abstract Over the last decades, improvements in CPU speed have greatly exceeded those in memory and disk speeds by orders of magnitude and this enabled the use of compression techniques to reduce the database size as well as the query cost. Although compression techniques are employed in various database researches, there is little work on compressing multi-dimensional index structures. In this paper, we propose an efficient compression method called the hybrid encoding method (HEM) that is tailored to multi-dimensional indexing structures. The HEM compression significantly reduces the query cost and the size of multi-dimensional index structures. Through mathematical analyses and extensive experiments, we show that the HEM compression outperforms an existing method in terms of the index size and the query cost.
Key words : Multi-dimensional index structure, Compression method, R*-tree

1. 서 론

지난 십년 동안, CPU의 발전 속도는 메모리나 디스크의 발전 속도를 훨씬 능가하였다. 이런 기술의 진보가 압축과 압축 풀기를 위해서 추가적으로 요구되는 CPU 계산 비용의 증가에도 불구하고, 데이타베이스 분야에서 데이타베이스의 크기와 질의 비용을 줄이기 위해서 다양한 압축 방법을 사용하였다[1-6]. 현재까지 다양한 데이타베이스 분야에서 압축기술에 대한 활발한 연구가 진행되었다. 그러나 다차원 색인 구조를 위한 압축 방법에 대한 연구는 거의 없었다.

R*-트리[7]나 X-트리[8]와 같은 다차원 색인 기법들은 공간 데이타베이스, 멀티미디어 데이타베이스, 시공간 데이타베이스 분야에서 사용되는 다양한 질의 처리를 위해서 개발되었다. 특히, Gaede와 Gunther는 다양한 다차원 색인 구조를 상세하게 비교 설명하였다[9]. 다차원 색인 구조를 사용하여 질의를 처리할 때 소요되는 비용은 디스크 I/O 비용과 CPU 계산 비용으로 분리할 수 있다. 디스크 I/O 비용은 질의를 처리하기 위해서 디스크와 메모리 사이의 페이지 읽기/쓰기 회수를 의미하고, CPU 계산 비용은 메모리 상에서 비교와 같은 산술적인 계산 회수를 의미한다. 그러나, 윈도우, 근접, 조인을 포함하는 대부분의 질의 처리 비용을 계산할 때, 디스크 I/O 비용이 CPU 계산 비용보다 훨씬 크기 때문에, CPU 계산 비용을 줄이는 것보다 디스크 I/O 비용을 줄이는 것이 중요하다. 추가적으로, B+트리 [10] 와 같은 1차원 색인 구조와 비교했을 때, 다차원 색인 구조의 크기는 차원의 증가와 함께 커진다. 우리는 압축 방법을 사용하여 다차원 색인 구조의 크기와 질의 비용을 줄이려고 한다.

† 비 회 원 : 한국과학기술원 전자전산학과
hjcho@islab.kaist.ac.kr
†† 종신회원 : 한국과학기술원 전자전산학과 교수
chungcw@cs.kaist.ac.kr
논문접수 : 2002년 12월 27일
심사완료 : 2003년 9월 5일

본 논문에서 제안하는 HEM (Hybrid Encoding Method) 압축 방법을 가장 효율적인 다차원 색인 기법 중의 하나인 R*-트리[7]에 적용하였다. 그러나, HEM 압축 방법은 X-트리, buddy-트리[11], Quad-트리[12]와 같은 다양한 다차원 색인 기법에도 그대로 사용될 수 있다. 본 논문의 나머지 구성은 다음과 같다. 2장은 R*-트리의 크기와 질의 비용 계산식을 통하여 압축 방법의 필요성을 보여준다. 3장은 R*-트리에 대한 간단한 설명과 다차원 색인 구조를 압축하는 기존의 방법에 대하여 설명한다. 4장은 우리가 제안하는 HEM 압축 방법에 대해서 설명하고, 변경되는 데이타 구조와 알고리즘에 대하여 기술한다. 5장은 수학적인 분석을 사용해서 기존 압축방법보다 HEM 압축 방법이 우수하다는 것을 보여준다. 6장은 가공 데이타와 실제 데이타를 사용한 실험 결과를 사용하여 기존 압축방법보다 HEM 압축 방법이 우수하다는 것을 보여준다. 끝으로, 7장에서는 본 논문의 요약과 향후 연구 계획을 언급한다.

2. R*-트리의 크기와 질의 비용

이미 연구된 R*-트리의 크기와 질의 비용 계산식들은 압축 기술이 색인 구조의 크기와 질의 비용을 줄이는데, 크게 기여한다는 것을 보여준다[13,14]. 먼저 $SIZE_{R^*\text{-}tree}$는 R*-트리의 크기를 의미하고, $DA_{R^*\text{-}tree}$는 질의를 처리할 때 요구되는 디스크 I/O 비용을 의미한다. R*-트리의 크기는 데이타의 수와 노드의 평균적인 팬아웃(fanout, 노드안에 저장된 엔트리의 수)을 사용하여 식 (1)처럼 계산된다. 식 (1)에서 H는 트리 높이를 의미하고, N은 데이타의 수를 의미하고, f는 노드가 가지는 평균적인 팬아웃을 의미한다.

$$SIZE_{R^*\text{-}tree} = \sum_{h=1}^{H}\left(\left\lceil \frac{N}{f^h} \right\rceil\right) = \left\lceil \frac{N}{f} \right\rceil + \left\lceil \frac{N}{f^2} \right\rceil + \cdots + 1 \quad \text{where } H = \lceil \log_f N \rceil \qquad (1)$$

비슷한 방법으로, 주어진 질의 영역과 교차하는 데이타를 찾는 질의를 처리할 때, 필요한 디스크 I/O 비용은 식 (2)를 이용하여 계산된다. 식 (2)에서 d는 차원의 수를 의미하고, s는 주어진 질의 면적을 의미한다.

$$DA_{R^*\text{-}tree} = 1 + \sum_{h=1}^{H-1}\left(\sqrt[d]{\left\lceil \frac{N}{f^h} \right\rceil}\cdot s + 1\right)^d \qquad (2)$$

식 (1)과 식 (2)에서, 압축 방법을 사용하면 노드의 팬아웃 f가 커진다. 이것은 R*-트리의 크기와 디스크 I/O 비용을 감소하게 만든다. 특히, 버퍼의 크기가 제한되어있을 때, 색인 크기가 작아질 경우 버퍼를 효율적으로 사용하게 되어 디스크 I/O 비용은 크게 줄어든다.

3. 관련 연구

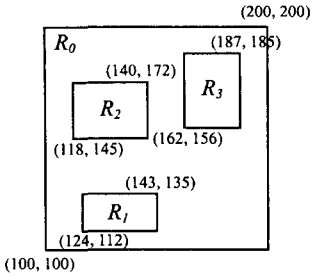
R*-트리[7]는 높이 균형 트리로써, 말단 노드는 <*ptr*, *MBR*>의 2개의 에트리뷰트를 가지는 엔트리들로 구성되어 있다. 여기서 *ptr*은 객체의 식별자를 나타내고, *MBR*은 객체를 포함하는 최소 사각형이다. 비말단 노드도 <*ptr*, *MBR*>로 이루어진 엔트리들로 구성되어 있다. 비말단 노드의 *ptr*은 자식 노드 식별자를 나타내고, *MBR*은 자식 노드를 포함하는 최소 사각형이다. 루트 노드를 제외한 모든 노드들은 2개 이상의 엔트리를 가지고, 노드의 크기는 보통 디스크 페이지 크기와 같다.

데이타베이스 분야에서 다양한 압축방법이 개발되었다. 이러한 추세는 CPU 발전 속도가 디스크나 메모리 발전 속도를 크게 능가하는 기술적인 상황과 데이타베이스에서 저장된 데이타의 양이 엄청나게 증가했다는 것이다. 그러나, 대부분 테이블에 저장된 데이타나 실제 데이타를 압축하는 방법에 대한 연구가 수행되었다. 또한 압축된 테이블에 대한 질의 처리나 최적화에 대한 연구가 수행되었다[1-6]. 최근에는 모바일 기술의 발전으로, PDA(Personal Digital Assistant)와 핸드폰과 같은 모바일 장치들이 폭 넓게 사용되고 있다. 모바일 장치들의 작은 기억 용량과 네트워크 밴드위드(network bandwidth)를 절약하기 위해서 압축 방법이 사용되고 있다[15]. 표 1은 몇 가지 표준 압축 방법들을 설명하고 있다.

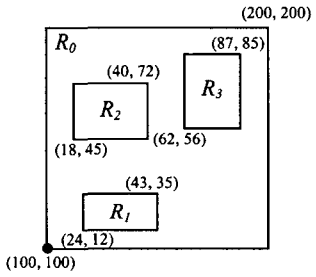
현재까지 다차원 색인 구조에 대한 압축방법으로는 FOR (Frame Of Reference)압축 방법이 제안되었다[3,16]. FOR 압축 방법은 페이지 단위의 옵셋 인코딩 압축 방법을 사용한다. FOR 압축 방법은 하나의 페이지에서 존재하는 값들의 범위가 이 값이 표현할 수 있는 범위보다 훨씬 작다는 사실을 이용한다. 예를 들어, 어떤 에트리뷰트의 타입이 32비트 정수일 경우에, 이 에트리뷰트의 표현 범위는 $0 \sim 2^{32} - 1$이다. 그러나 실제로 한 페이지 내에서 존재하는 에트리뷰트들의 범위가 2^{32}보다 작다면, 적은 공간으로 해당 에트리뷰트를 표현할

표 1 표준 압축 방법의 요약

압축 방법	설명
차등 인코딩 (differential encoding)	인접한 값들의 차이를 저장한다.
옵셋 인코딩 (offset encoding)	기준값으로 거리를 저장한다.
사전적 인코딩(dictionary encoding)	문자열을 하나의 숫자로 대응시킨다.
Lempel-Zip 77 또는 78	적응적인 사전적 인코딩 방법.
허프만 인코딩 (Huffman encoding)	자주 나오는 문자들을 더 짧은 비트로 표현한다.



(a) 압축 이전의 R-트리 노드



(b) 압축 이후의 R-트리 노드

그림 1 FOR압축 방식에 의한 R*-트리 노드 압축의 예

수 있다. FOR 압축 방법의 가장 큰 장점은 페이지 단위의 압축/해제를 제공하는 것이다. 그림 1은 R*-트리의 노드를 FOR 압축 방법을 이용하여 압축하는 예를 보여준다. 그림 1에 보여진 것처럼, FOR 압축 방법은 단순하게 하나의 노드에서 가장 작은 값을 기준으로 각 값까지의 옵셋을 계산하고, 이것으로 원래 값을 대신한다. R*-트리의 노드는 (*ptr, MBR*)들의 리스트로 구성되어 있기 때문에, 옵셋 인코딩은 *ptr*에서 가장 작은 값을 기준으로 다른 *ptr*까지 옵셋을 사용하여 표현하고, *MBR*에서 가장 작은 값을 기준으로 다른 *MBR*들을 표현한다. 그림 1(a)에서 R_1 = {(124,112), (143,135)}는 원점 (0,0)을 기준으로 표현되었다. 그림 1(b)에서 R_1 = {(24,12), (43,35)}는 노드에서 가장 작은 점 (100, 100)을 기준으로 표현되었다. 실제로 R_0는 x축의 범위가 [100, 200]이기 때문에, 32비트보다 적은 7 (= (log2 (200(100+1)()비트 정수를 이용하면 충분하다. 그러나, FOR 압축 방법은 효과적으로 다차원 색인 구조를 압축하지 못한다. 첫번째, MBR은 점이 아니라 사각형이기 때문에 1개의 옵셋 대신에 2개의 옵셋을 사용해야 한다. 두번째, *ptr*은 검색키가 아니기 때문에, MBR과 달리 하나의 노드에서 클러스터가 되지 않기 때문에, 옵셋 인코딩 방법은 효율적이지 않다.

4. HEM 압축 방법

HEM 압축 방법은 옵셋 인코딩과 차등 인코딩을 병합한 압축 방법이다. 하나의 노드는 (*ptr, MBR*)들의 리스트로 구성되어 있기 때문에, *ptr*을 압축하는 방법과 *MBR*을 압축하는 방법을 각각 다르게 적용한다. 옵셋 인코딩은 *MBR*을 압축할 때 사용되고, 차등 인코딩은 *ptr*을 압축할 때 사용된다.

4.1 옵셋 인코딩을 이용한 *MBR* 압축 방법

*MBR*을 압축하기 위해서, HEM 압축 방법은 FOR 압축 방법과 달리 2개의 옵셋을 사용한다. 그림 2는 2개의 옵셋을 사용해서 MBR을 압축하는 예를 보여준다.

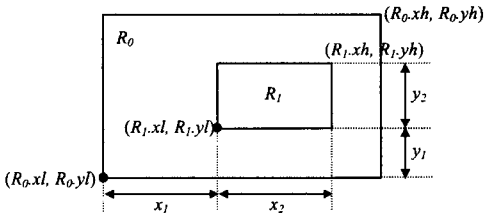


그림 2 2개의 옵셋을 이용한 MBR 압축 방법

그림 2에서 모든 좌표는 원점 (0,0)을 기준으로 표시되었다. 하나의 옵셋을 사용하여, R_1의 MBR을 다음과 같이 표현할 수 있다.

$$\begin{aligned} R_1 &= \{(x_1, y_1), (x_1 + x_2, y_1 + y_2)\} \\ &= \{(R_1.xl - R_0.xl, R_1.yl - R_0.yl), \\ &\quad (R_1.xh - R_0.xl, R_1.yh - R_0.yl)\} \end{aligned} \quad (3)$$

식 (3)은 각 차원에 대하여 1개의 옵셋을 사용해서 표현한 방법이고, 식 (4)[1]는 2개의 옵셋을 사용해서 표현한 방법이다.

$$\begin{aligned} R_1 &= \{(x_1, y_1), (x_2, y_2)\} \\ &= \{(R_1.xl - R_0.xl, R_1.yl - R_0.yl), \\ &\quad (R_1.xh - R_1.xl, R_1.yh - R_1.yl)\} \end{aligned} \quad (4)$$

x_2는 (x_1+x_2)보다 작고, y_2는 (y_1+y_2)보다 작기 때문에 2개의 옵셋을 (1개의 옵셋과 MBR의 한 변의 길이) 사용하는 것이 1개의 옵셋을 사용할 때 보다 효과적으로 압축할 수 있다. 즉, *MBR*을 표현하기 위해서 필요한 비트 수를 줄일 수 있다.

4.2 차등 인코딩을 이용한 *ptr* 압축 방법

*MBR*이 검색키가 되기 때문에, R*-트리는 공간적으로 인접한 *MBR*들을 하나의 노드에 넣으려고 노력한다. 따라서, 하나의 노드에 속하는 *MBR*들의 범위는 작게 된다. 반면에, *ptr*들은 검색키가 아니기 때문에, 노드 안에 있는 *ptr*들의 값은 클러스터링되지 않는다. 따라서, *ptr*을 압축하기 위해서는 옵셋 인코딩 방법이 효과적이지 못하다. 이러한 문제점을 해결하기 위해서, *ptr*들을

1) 이것은 1개의 옵션과 1개의 차등치를 이용한 방법으로 표현될 수 있다.

압축하기 위해서 옵셋 인코딩 대신에 차등 인코딩을 사용한다. 차등 인코딩의 압축률을 높이기 위해서, 엔트리들은 *ptr*을 기준으로 정렬한 후에, 이전 *ptr*로부터 차이를 저장한다. 그림 3은 차등 인코딩을 이용한 *ptr*압축의 예를 보여준다.

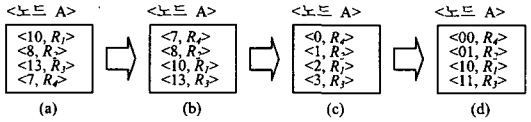


그림 3 차등 인코딩을 이용한 *ptr* 압축 방법의 예

그림 3(a)에서 노드 A는 4개의 엔트리로 구성되어 있다. R_1, R_2, R_3, R_4는 MBR을 나타내고, 10, 8, 13, 7은 *ptr*을 의미한다. 그림 3(b)에서, 4개의 엔트리들은 *ptr*을 기준으로 정렬된다. 그림 3(c)는 정렬된 4개의 엔트리에 대해서 차등 인코딩을 적용한 결과이다. 가장 큰 값이 3이기 때문에, 1개의 *ptr*을 표현하기 위해서 2 비트를 사용하면 된다. 그림 3(d)에서 00, 01, 10, 11은 0, 1, 2, 3을 각각 이진수로 표현한 것이다.

4.3 데이타 구조와 알고리즘

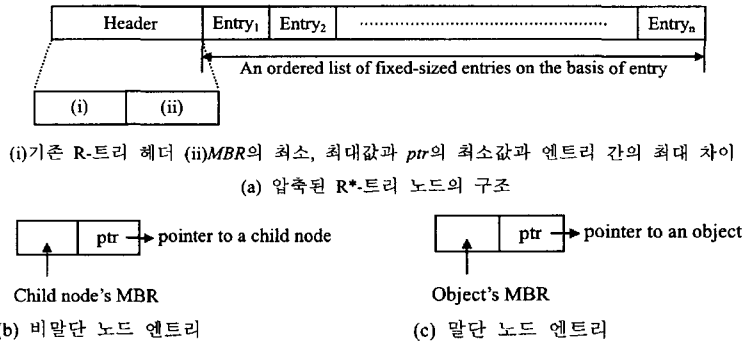
HEM 압축 방법을 사용하기 위해서, R*-트리 데이타 구조는 수정되어야 한다. 그림 4(a)에 보여준 것처럼, 각 노드에 속하는 *MBR*의 최소, 최대값 및 *ptr*의 최소값과, 인접한 엔트리 사이의 최대 차이를 헤더 정보에 추가한다. 이러한 추가적인 정보는 압축된 엔트리 정보를 압축이전의 엔트리로 복구할 때 사용된다. 하나의 노드에 속하는 압축된 엔트리는 고정 크기를 가진다. 그림 4(b)와 그림 4(c)는 비단말 노드 엔트리와 말단 노드 엔트리를 나타낸다. *MBR*은 옵셋 인코딩된 값을 저장하고, *ptr*은 차등 인코딩된 값을 저장한다.

HEM 압축 방법은 R*-트리 데이타 구조에 영향을 주었지만, R*-트리 삽입, 삭제, 검색 알고리즘들은 그대로 사용된다. HEM 압축 방법은 하나의 노드에 속하는 엔트리들은 고정 길이를 가진다. 그러나, 서로 다른 노드들에서는 엔트리 크기가 다를 수가 있다. 이것은 하나의 노드에 속하는 가변 길이를 가지는 엔트리들을 유지하기 위해서는 각각의 엔트리 길이 정보를 따로 유지해야 하고, 가변 길이 엔트리를 처리할 때, 복잡도가 증가하기 때문에 고정 길이를 가지는 엔트리를 사용한다. 또한, 압축되지 않은 노드에서는 항상 2개의 노드로 스플릿(split)이 되지만, 압축된 노드에서는 2개의 노드로 스플릿이 될 수 없는 경우에, 3개의 노드로 스플릿이 되기도 한다. 데이타를 삽입하는 과정이 색인 구조를 압축하는 과정이기 때문에 그림 5에 나타난 Compress 알고리즘이 추가되었다.

R*-트리에서, 삽입과 삭제는 단말노드에서 루트노드까지 하나의 경로를 따라서 이루어지기 때문에 HEM

```
1   Algorithm Compress (node N, entry E)
2   if (E.ptr < N.ptr_min) {
3     모든 entry들의 ptr에 (N.ptr_min - E.ptr)을 더한다
4     N.ptr_min := E.ptr
5     E.ptr := E.ptr – N.ptr_min
6     E를 노드의 처음 엔트리 앞에 삽입한다.
7   } else if (E.ptr > N.ptr_max) {
8     N.ptr_max := E.ptr
9     E.ptr := E.ptr – N.ptr_min
10    E를 노드의 마지막 엔트리 뒤에 삽입한다.
11  } else {
12    for each entry e which belongs to N {
13      if (E.ptr < e.ptr) {
14        E.ptr을 엔트리 e 앞쪽에 삽입한다
15        E.ptr := E.ptr – N.ptr_min
16        break;
17      }
18    }
19  }
20  if (E.mbr is fully contained in N.mbr) {
21    E.mbr을 식 (4.2)를 이용하여 옵셋 인코딩한다
22  } else {
23    N.mbr이 E.mbr을 포함할 수 있도록 확장한다.
24    모든 entry들의 mbr을 다시 변경된 N.mbr에 맞게 옵셋 인코딩한다
25    E.mbr을 식 (4.2)를 이용하여 옵셋 인코딩한다
26  }
```

그림 5 R*-트리 노드를 압축하는 Compress 알고리즘



(i)기존 R-트리 헤더 (ii)*MBR*의 최소, 최대값과 *ptr*의 최소값과 엔트리 간의 최대 차이

(a) 압축된 R*-트리 노드의 구조

(b) 비말단 노드 엔트리

(c) 말단 노드 엔트리

그림 4 HEM 압축 방법을 위한 R*-트리 데이타 구조

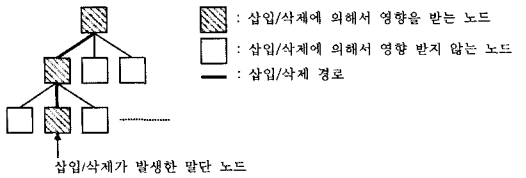
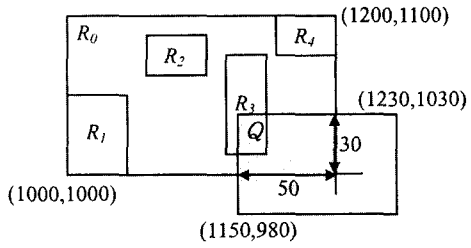
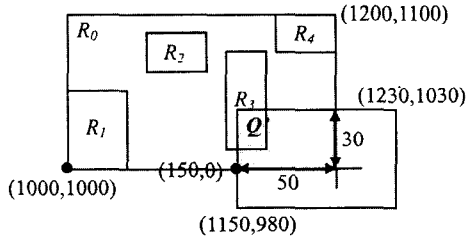


그림 6 삽입/삭제를 위한 말단 노드에서 루트 노드까지의 경로



(a) Q={(1150, 980), (1230, 1030)}



(b) Q'={(150, 0), (50, 30)}

그림 7 압축된 R*-트리를 위한 질의 윈도우 압축의 예

압축 방법은 경로 위에 존재하는 노드들에서만 발생한다. 다시 말하면, HEM 압축 방법은 FOR압축 방법처럼, 노드 단위로 압축과 해제가 발생한다. 그림 6에서 보여지는 것처럼, 삽입/삭제가 일어난 말단 노드에서 루트까지 경로에 있는 노드들만 변경된다.

검색 알고리즘은 R*-트리의 검색 방법을 그대로 사용하기 위해서는 압축된 노드들을 압축전의 노드로 복구해야 한다. 우리는 이러한 문제점을 해결하기 위해서 주어진 질의 윈도우를 압축해서 압축된 노드와 비교하는 방법을 사용한다. 즉, 압축된 노드를 복구하는 대신에, 질의 윈도우를 옵셋 인코딩을 사용해서 압축하고 비교한다. 그림 7은 질의 윈도우 Q를 옵셋 인코딩하는 예를 보여준다. R_1, R_2, R_3, R_4의 MBR은 이미 옵셋 인코딩으로 압축되었기 때문에, Q를 옵셋 인코딩으로 압축한 후에 비교하면 되기 때문에, 압축된 노드를 복구할 필요가 없다. 그림 7(a)에서 주어진 질의 윈도우 Q={(1150, 980), (1230, 1030)}는 그림 7(b)에서 Q'={(150, 0), (50, 30)}으로 변환된다. 왜냐하면, R_0와 교차하지 않는 질의 영역은 R_0의 자식 노드들 R_1, R_2, R_3, R_4와 교차하지 않기 때문에 고려할 필요가 없다.

5. 분 석

이 장에서는 우리는 수학적인 분석 방법을 사용하여, HEM 압축 방법이 FOR 압축 방법보다 압축 효과가 크다는 것을 보여준다. 모든 데이타들은 d-차원의 작업 공간 WS = $[0, 1)^d$ 위에서 균등하게 분포되어 있다고 가정한다 (이것을 균등분포 가정이라고 부른다[13,14]). 표 2는 이 장에서 사용되는 몇 개의 새로운 기호들에 대해서 간략하게 설명하고 있다.

표 2 기호들의 요약

기호	설명
N	데이타의 개수
f	노드의 평균적인 팬아웃
d	차원의 개수
M_h	레벨 h에 있는 노드들의 수
A_h	레벨 h에 있는 노드들의 MBR의 평균 면적
n_h	레벨 h에 있는 노드들의 MBR의 한 변의 평균 길이
e_h	레벨 h에 있는 엔트리들의 MBR의 한 변의 평균 길이, 다시 말하면, 레벨 h-1에 있는 노드들의 MBR의 한 변의 평균 길이

5.1 옵셋 인코딩을 이용한 *MBR* 압축

M_h는 레벨 h에 있는 노드들의 수라고 하자. M_h는 N와 f를 이용하면 계산된다.

$$M_h = \left\lceil \frac{N}{f^h} \right\rceil \tag{5}$$

예를 들어, 말단 노드들의 수는 $M_1 = \lceil N/f \rceil$ A_h는 레벨 h에 있는 노드들의 MBR의 평균 면적이라고 하자. M_h 개의 노드가 WS = $[0, 1)^d$을 차지하기 때문에, 1개의 노드의 MBR이 차지하는 면적 A_h는 다음과 같다.

$$A_h = \frac{1}{M_h} \tag{6}$$

n_h는 레벨 h에 있는 노드들의 MBR의 한 변의 평균 길이라고 하자. A_h가 레벨 h에 있는 노드들의 MBR의 평균 면적이기 때문에, n_h는 다음과 같다.

$$n_h = \sqrt[d]{A_h} \tag{7}$$

e_h는 레벨 h에 있는 엔트리들의 MBR의 한 변의 평균 길이라고 하자. 한 변을 차지하는 엔트리들의 개수는 $\sqrt[d]{f}$ 이기 때문에, e_h는 다음과 같다.

$$e_h = \frac{n_h}{\sqrt[d]{f}} \tag{8}$$

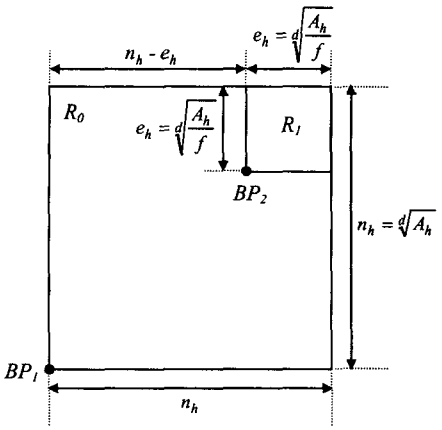


그림 8 n_h와 e_h의 관계

그림 8은 식 (7)과 식 (8)을 사용하여 n_h와 e_h사이의 관계를 보여준다. 그림에서 BP_1과 BP_2는 R_1을 옵셋 인코딩할 때 사용하는 기준점을 나타낸다.

MBR_{FOR}는 *FOR* 압축 방법을 사용하여 *MBR*을 표현할 때 필요한 크기를 의미하고, MBR_{HEM}은 HEM 압축 방법을 사용하여 *MBR*을 표현할 때 필요한 크기를 의미한다. FOR 압축 방법은 1개의 옵셋을 사용하여 *MBR*을 압축한다. 반면에, HEM 압축 방법은 2개의 옵셋을 사용하여 *MBR*을 압축한다. 그림 8에서 기준점 BP_1으로부터 R_1위의 한 점을 표현하기 위해서는 $2 \cdot \lceil \log_2 n_h \rceil$ 비트들을 요구한다. 차원의 개수가 d이기 때문에, MBR_{FOR}는 다음과 같다.

$$\mathrm{MBR_{FOR}} = 2 \cdot d \cdot \lceil \log_2 n_h \rceil \qquad (9)$$

HEM 압축 방법은 2개의 옵셋을 사용하기 때문에, BP_1에서 BP_2까지 길이(i.e., $n_h - e_h$)와 MBR의 한 변의 길이(i.e., e_h)를 표현할 수 있는 공간이 필요하다.

$$\mathrm{MBR_{HEM}} = d \cdot (\lceil \log_2 (n_h - e_h) \rceil + \lceil \log_2 e_h \rceil) \qquad (10)$$

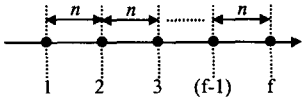
S_{MBR}은 하나의 노드에 있는 MBR들을 표현하기 위해서 FOR 압축 방법에서 요구하는 공간의 크기와 HEM 압축 방법에서 요구하는 공간의 크기 차이라고 하자. 노드의 팬아웃이 f이기 때문에, 1개의 노드는 f개의 MBR들을 가진다. 따라서, S_{MBR}은 다음과 같다.

$$\begin{aligned} S_{MBR} &= f \cdot (MBR_{FOR} - MBR_{HEM}) \\ &= f \cdot \{ (\lceil \log_2 n_h \rceil + \lceil \log_2 n_h \rceil) \\ &\quad - (\lceil \log_2 (n_h - e_h) \rceil + \lceil \log_2 e_h \rceil) \} \\ &= f \cdot \left\{ \log_2 \left(\frac{n_h}{n_h - e_h} \right) + \log_2 \left(\frac{n_h}{e_h} \right) \right\} \\ &= f \cdot \left\{ \log_2 \left(\frac{\sqrt[d]{f}}{\sqrt[d]{f} - 1} \right) + \log_2 \sqrt[d]{f} \right\} \\ &\cong f \cdot \log_2 \sqrt[d]{f} \end{aligned} \qquad (11)$$

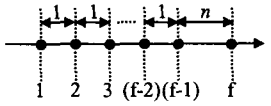
식 (8), 식 (9), 식 (10)을 넣고 정리하면 식 (11)을 얻을 수 있다. 식 (11)에서 $\frac{\sqrt[d]{f}}{\sqrt[d]{f} - 1} \cong 1$이기 때문에 $\log_2 \left(\frac{\sqrt[d]{f}}{\sqrt[d]{f} - 1} \right) \cong 0$.

5.2 차등 인코딩을 이용한 *ptr* 압축

*ptr*을 표현하기 위해서, FOR 압축 방법과 HEM 압축 방법에 의해서 요구되는 크기의 범위는 2가지 경우를 사용하여 나타낼 수 있다. 이 그림에서, n은 인접한 *ptr*사이의 최대 거리를 의미한다.



(a) *ptr*들의 값의 범위가 가장 큰 경우



(b) *ptr*들의 값의 범위가 가장 작은 경우

그림 9 *ptr*들의 값의 범위가 가장 큰 경우와 가장 작은 경우

그림 9(a)는 *ptr*의 범위가 가장 클 때의 모습을 나타내고, 그림 9(b)는 *ptr*의 범위가 가장 작을 때의 모습을 나타낸다. *ptr*은 객체의 식별자 또는 자식 노드를 가리키므로, 중복되지 않는다. PTR_{FOR}는 FOR 압축 방법을 사용하여 *ptr*을 표현할 때 요구되는 크기를 의미하고, PTR_{HEM}은 HEM 압축 방법을 사용하여 *ptr*을 표현할 때 필요한 크기를 의미한다. 그림 9(a)의 경우에, PTR_{FOR}와 PTR_{HEM}은 다음과 같다.

$$\mathrm{PTR_{FOR}} = \lceil \log_2 (f-1) \cdot n \rceil \qquad \mathrm{PTR_{HEM}} = \lceil \log_2 n \rceil \qquad (12)$$

그림 9(a)에서 하나의 노드는 f개의 *ptr*을 가지고 있기 때문에, *ptr*중에서 가장 작은 값과 큰 값의 차이는 $(f-1) \cdot n$이 되므로, $PTR_{FOR} = \lceil \log_2 (f-1) \cdot n \rceil$. 그림 9(a)의 경우에 PTR_{HEM}은 차등 인코딩을 사용하기 때문에, 두 *ptr*사이의 최대 거리를 표현할 수 있는 공간이 요구된다. 유사한 방법으로, 그림 9(b)의 경우에 PTR_{FOR}와 PTR_{HEM}은 다음과 같다.

$$\mathrm{PTR_{FOR}} = \lceil \log_2 \{ (f-2) + n \} \rceil \qquad \mathrm{PTR_{HEM}} = \lceil \log_2 n \rceil \qquad (13)$$

S_{PTR}은 하나의 노드에 있는 *ptr*들을 표현하기 위해서 FOR 압축 방법에서 요구하는 공간의 크기와 HEM 압축 방법에서 요구하는 공간의 크기 차이라고 하자. S_{PTR}은 식 (12)와 식 (13)을 사용해서 다음과 같이 표현할 수 있다. 식 (12)와 식 (13)은 한 개의 *ptr*을 표현하기 위해서 요구하는 공간이기 때문에, f를 곱해야 한다.

$$\mathrm{S_{PTR}} = f \cdot (\mathrm{PTR_{FOR}} - \mathrm{PTR_{HEM}}) \qquad (14)$$

식 (12), 식 (13), 식 (14)를 사용해서 S_{PTR}의 범위는

다음과 같다.

$$f \cdot \log_2\left(\frac{f-2}{n}+1\right) \le S_{PTR} \le f \cdot \log_2(f-1) \quad (15)$$

S_{NODE}는 FOR 압축 방법을 사용하여 노드를 표현할 때 필요한 크기와 HEM 압축 방법을 사용하여 노드를 표현할 때 필요한 크기 차이라고 하자. 하나의 노드는 MBR과 ptr로 구성되어 있기 때문에 S_{NODE}는 S_{MBR}과 S_{PTR}을 사용하여 구할 수 있다.

$$S_{NODE} = S_{MBR} + S_{PTR} \quad (16)$$

식 (11), 식 (15), 식 (16)을 사용해서 S_{NODE}의 범위는 다음과 같다.

$$f \cdot \left\{\log_2 f + \log_2\left(\frac{f-2}{n}+1\right)\right\} \le S_{NODE} \le f \cdot \{\log_2 f + \log_2(f-1)\} \quad (17)$$

6. 실 험

6.1 실험 환경

HEM 압축 방법과 FOR 압축 방법을 비교하기 위해서 우리는 가상 데이타와 실제 데이타를 사용해서 실험을 했다. 일련의 실험은 256MB의 메모리와 1.7GHz CPU를 가진 윈도우 2000 PC에서 수행되었다. 표 3은 실험에서 사용된 가상 데이타와 실제 데이타에 대하여 설명하고 있다. 사용된 실제 데이타인 Tiger/Line 데이타는 공개된 데이타를 사용하였다[17].

R*-트리는 이제까지 제안된 다양한 R-트리중에서 가장 우수하다고 알려져 있다. 노드 크기는 1KB이고, 2차원일 때, R*-트리의 팬아웃은 50이다. 압축 방법을 적용하지 않은 R*-트리 색인 구조의 모든 정보 (*MBR*의 좌표값과 *ptr*)는 4바이트 정수를 사용하였다.

HEM 압축 방법의 질의 비용의 감소 효과를 보여주기 위해서, 윈도우 질의를 수행할 때, 요구되는 노드 접근 회수를 조사하였다. 노드 접근 회수는 적당한 성능 비교의 방법이라고 할 수 있다. 왜냐하면, 노드 접근 회수가 많을수록, 실제 디스크 I/O 회수가 많아지기 때문이다.

표 3 실험에서 사용된 데이타들의 설명

데이타	설명
UNI	2차원 공간상에서 균등 분포를 하는 200,000개의 MBR로 이루어진 데이타.
TS	Iowa, Kansas, Missouri, Nebraska에 있는 194,971개의 도로 정보를 포함하는 MBR로 이루어진 Tiger Stream 데이타.
CAS	캘리포니아에 있는 98,451 개의 도로 정보를 포함하는 MBR로 이루어진 California Stream 데이타.
SEQP	캘리포니아에 있는 62,556개의 지명 정보를 나타내는 Sequoia Point 데이타

6.2 실험 결과

그림 10은 Winzip, FOR 압축 방법, HEM 압축 방법에 의한 압축된 R*-트리의 크기를 보여준다. R*-트리의 크기 압축률은 다음과 같이 정의된다.

$$\text{크기 압축률}(\%) = \frac{\text{압축 이후의 R*-트리 크기}}{\text{압축 이전의 R*-트리 크기}} \times 100$$

예를 들어, 압축이전의 R*-트리의 크기 10MB이고, 크기 압축률이 25%라면, 압축이후의 R*-트리의 크기는 2.5MB이다.

그림 10에 나타난 것처럼, Lempel-Zip 알고리즘에 기초한 Winzip 압축 방법은 평균적으로 55% 압축률을, FOR 압축 방법은 54%, HEM 압축 방법은 33% 압축률을 보였다. 그러나 Winzip 압축 방법은 질의를 수행하기 위해서는 항상 전체 인덱스 구조에 대해서 압축을 풀어야 하기 때문에, 다차원 인덱스 구조를 압축하는데 사용할 수가 없다. FOR 압축 방법과 HEM 압축 방법은 노드 단위의 압축/압축 해제 방법을 제공한다. 특히, SEQP 데이타는 지명 데이타이기 때문에 각 점들이 넓게 분포되어 있는 특성이 있다. 또한 지명 데이타는 사각형이 아닌 점으로 이루어져 있기 때문에, 식 (4.2)에서 말단 노드의 x_2=0, y_2=0이다. 왜냐하면, x_2, y_2는 사각형의 한 변의 길이를 의미하는데, 점은 변의 길이가 항상 0이기 때문이다. 그래서 FOR 압축 방법의 압축 효과가 좋지 않게 나타난다.

그림 11은 FOR 압축 방법과 HEM 압축 방법에 압축된 R*-트리에 대한 윈도우 질의를 수행한 결과를 보여준다. R*-트리의 질의 비용 압축률은 다음과 같이 정의된다.

$$\text{질의 비용 압축률}(\%) = \frac{\text{압축이후의 R*-트리 질의 비용}}{\text{압축이전의 R*-트리 질의 비용}} \times 100$$

식 (2)에서 나타난 것처럼, f가 클수록 노드 접근 회수가 크게 줄어든다. 질의 워크로드(workload)는 500개의 윈도우 질의들로 구성되었다. 질의 영역은 전체 영역의 1%에서 9%를 차지한다. 평균적으로, FOR 압축 방법은 57% 질의 비용 압축률을, HEM 압축 방법은 29% 질의 비용 압축률을 보였다. 주목할 만한 것은, 질의 영

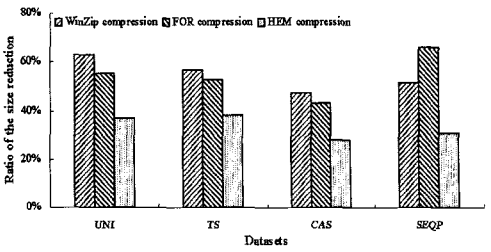


그림 10 다양한 데이타에 대한 크기 압축률의 비교

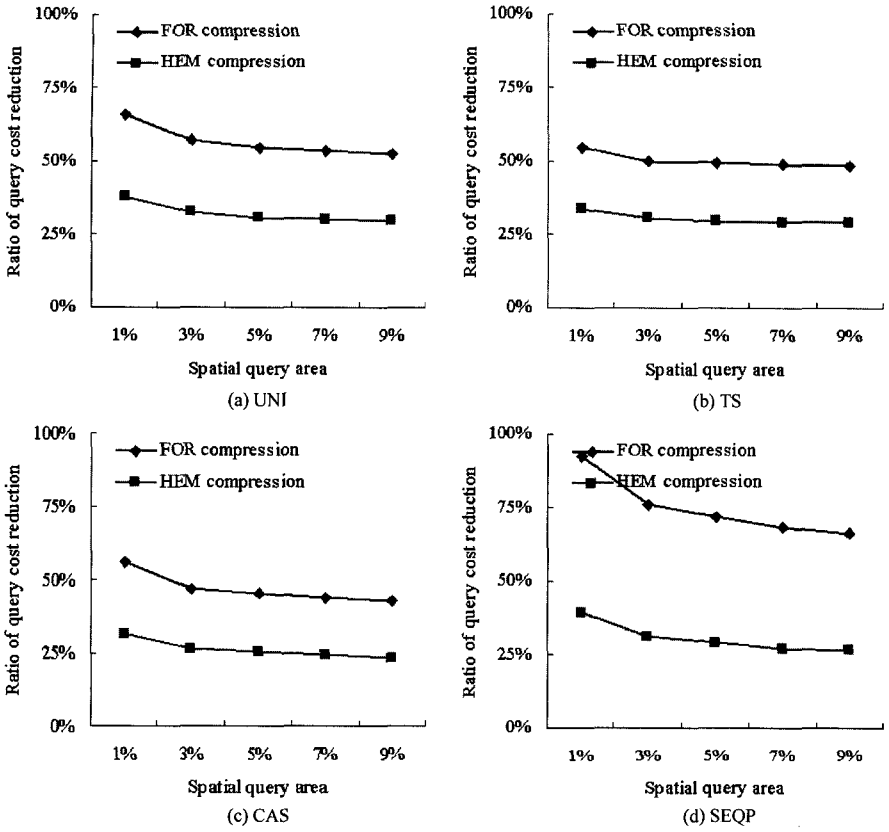


그림 11 다양한 데이타에 대한 질의 비용 압축률의 비교

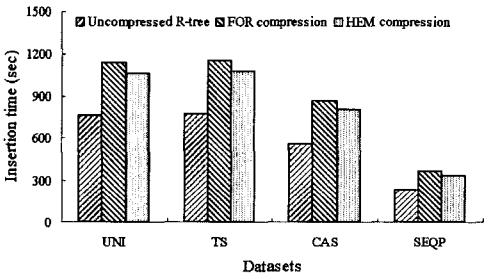


그림 12 다양한 데이타에 대한 삽입 시간의 비교

역이 커질수록 질의 비용 압축률이 작아진다는 것이다. 압축된 R*-트리의 경우 팬아웃이 크게 증가하기 때문에, 질의 영역이 클수록 효과가 좋다는 것을 의미한다. SEQP데이타의 경우는 그림 10에서 설명했던 것처럼, HEM 압축 방법의 압축효과가 FOR 압축 방법보다 크기 때문에, 그 차이가 그대로 질의 비용 압축률에도 반영되었다.

그림 12는 압축되지 않은 R*-트리에서의 삽입 시간과 압축된 R*-트리에서의 삽입 시간을 보여준다. 삽입 시간은 데이타들을 R*-트리에 삽입 하는데 경과한 시간을 측정했다. 압축된 R*-트리는 1개의 데이타를 삽입할 때마다, 몇 개의 노드들을 압축해야 하기 때문에 압축되지 않은 R*-트리에 데이타를 삽입하는 시간보다 길다. 그러나, 노드의 팬아웃이 크기 때문에, 노드의 스플릿이 훨씬 적게 발생한다. 평균적으로, FOR 압축 방법의 삽입 시간은 평균적으로 1.52 배 정도, HEM 압축 방법은 1.41배 정도 압축되지 않은 R*-트리보다 길다.

7. 결론 및 향후 연구

최근에, 압축기술은 다양한 데이타베이스 분야에서 연구되고 있다. 이것은 데이타베이스의 크기가 커지고, CPU의 발전 속도가 메모리나 디스크의 발전속도보다 훨씬 빠르다는 현실에 근거를 두고 있다. 그러나, 다차원 색인 구조를 압축하는 연구는 많이 수행되지 않았다. 우리는 다차원 색인 구조를 효과적으로 압축할 수 있는 HEM 압축 방법을 제안했다. HEM 압축 방법은 바람직한 3가지 특징을 가지고 있다. (i) 정보의 손실 없는 데이타의 압축과 해제 (ii) 압축된 상태에서 질의 가능한

압축 방법 (iii) 노드 단위의 데이타의 압축과 해제. 우리는 수학적인 분석과 다양한 실험을 통해서 HEM 압축 방법이 이전에 개발되었던 FOR 압축 방법보다 압축된 색인의 크기와 질의 비용 측면에서 우수하다는 것을 보여주었다.

다차원 색인 기술의 압축 방법을 실제 데이타베이스 분야에 적용하기 위해서는 질의 최적화기에 다차원 색인 구조를 이용한 질의 처리 비용 모델을 제공해야 한다. 우리는 압축된 다차원 색인 구조에 대한 질의 비용 계산식을 계획하고 있다.

참 고 문 헌

[1] B. R. Iyer and D. Wilhite: Data Compression Support in Databases. *VLDB*, 1994.

[2] W. K. Ng and C. V. Ravishankar: Relational Database Compression Using Augmented Vector Quantization. *ICDE*, 1995.

[3] J. Goldstein, R. Ramakrishnan, and U. Shaft: Compressing Relations and Indexes. *ICDE*, 1998.

[4] P. E. O'Neil and D. Quass: Improved Query Performance with Variant Indexes. *ACM SIGMOD*, 1997.

[5] M. A. Roth and S. J. Van Horn: Database Compression. *SIGMOD Record* 22(3), 1993.

[6] T. Westmann, D. Kossmann, S. Helmer, and G. Moerkotte: The Implementation and Performance of Compressed Databases. *SIGMOD Record* 29(3), 2000.

[7] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger: The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. *ACM SIGMOD*, 1990.

[8] S. Berchtold, D. A. Keim, and H. Kriegel: The X-tree: An Index Structure for High-Dimensional Data. *VLDB*, 1996.

[9] V. Gaede, and O. Günther: Multidimensional Access Methods. *ACM Computing Surveys* 30(2), 1998.

[10] D. Comer: The Ubiquitous B-Tree. *ACM Computing Surveys* 11(2), 1979.

[11] B. Seeger and H. Kriegel: The Buddy-Tree: An Efficient and Robust Access Method for Spatial Data Base Systems, *VLDB*, 1990.

[12] H. Samet: The Quadtree and Related Hierarchical Data Structures. *ACM Computing Surveys* 16(2), 1984.

[13] Y. Theodoridis and T. K. Sellis: A Model for the Prediction of R*-tree Performance. *ACM PODS*, 1996.

[14] I. Kamel and C. Faloutsos: On Packing R*-trees. *CIKM*, 1993.

[15] Z. Chen and P. Seshadri: An Algebraic Compression Framework for Query Results. *ICDE*, 2000.

[16] J. Goldstein, and R. Ramakrishnan: Squeezing the Most out of Relational Database Systems. *ICDE*, 2000.

[17] http://dias.cti.gr/~ytheod/research/datasets/spatial.html.

조 형 주

1997년 서울대학교 컴퓨터공학과 졸업(학사). 1997년 서울대학교 컴퓨터공학과 졸업(석사). 1999년~현재 한국과학기술원 전자전산학과 전산학 전공 박사과정 관심분야는 시간/공간/시공간 데이타베이스 질의처리, 인덱싱, 질의 최적화

정 진 완

정보과학회논문지 : 데이타베이스 제 30 권 제 1 호 참조