다차원 대용량 저밀도 데이타 큐브에 대한 고밀도 서브 큐브 추출 알고리즘

(Dense Sub-Cube Extraction Algorithm for a Multidimensional Large Sparse Data Cube)

이 석 룡 † 전 석 주 †† 정 진 완 †††
(Seok-Lyong Lee) (Seok-Ju Chun) (Chin-Wan Chung)

요 약 데이타 웨어하우스는 기업이나 사회 전반에서 사용되는 방대한 데이타를 저장하고, 효율적인 분석을 가능하게 하는 데이타 저장소로써, 점점 그 활용도가 증가하고 있다. 본 연구에서는 이러한 데이타 웨어하우스 구축 기술의 핵심이 되는 다차원 데이타 큐브(multidimensional data cube) 기술을 연구하는 데 목적이 있다.

고차원 데이타 큐브에는 필연적으로 내재하는 데이타의 희소성(sparsity)에 의한 검색 오버헤드가 있다. 본 연구에서는 이러한 오버헤드를 현격하게 감소시키는 알고리즘을 제시함으로써, 데이타 웨어하우스의 효율을 높이는 데 기여한다. 즉, 고차원의 희소 데이타 큐브에서 데이타가 조밀하게 밀집된 영역들을 찾아 그 영역을 중심으로 서브 큐브를 구축하여, 데이타 검색 시에 전체의 데이타 큐브를 대상으로 하지 않고 해당 서브 큐브 만으로 검색 대상을 제한시킴으로써 검색 효율을 높이는 알고리즘이다. 본 논문에서는 다차원 대용량의 희소 데이타 큐브로부터 밀도가 높은 서브 큐브를 찾기 위하여 비트맵과 히스토그램에 기반한 알고리즘을 제안하며, 실험을 통하여 제안한 알고리즘의 효용성을 보여준다.

키워드 : 데이타 큐브, 데이타웨어하우스

Abstract A data warehouse is a data repository that enables users to store large volume of data and to analyze it effectively. In this research, we investigate an algorithm to establish a multidimensional data cube which is a powerful analysis tool for the contents of data warehouses and databases.

There exists an inevitable retrieval overhead in a multidimensional data cube due to the sparsity of the cube. In this paper, we propose a dense sub-cube extraction algorithm that identifies dense regions from a large sparse data cube and constructs the sub-cubes based on the dense regions found. It reduces the retrieval overhead remarkably by retrieving those small dense sub-cubes instead of scanning a large sparse cube. The algorithm utilizes the bitmap and histogram based techniques to extract dense sub-cubes from the data cube, and its effectiveness is demonstrated via an experiment.

Key words : Data Cube, Data Warehouse

1. 서 론

최근에 여러 소스들로부터 수집된 데이타를 통합화된 스키마로 표현하고 저장하는 데이타 웨어하우스가 은행 및 재정서비스, 소매 유통과 생산 등의 많은 어플리케이션 영역에서 폭 넓게 사용되고 있다. 다양한 데이타 웨어하우스 기술 중에서 다차원 데이타베이스(multidimensional database)를 위한 데이타 모델인 데이타 큐브 기술이 점점 중요해지고 있다. 이것은 데이타 웨어하우스와 데이타베이스의 내용을 위한 강력한 분석 툴을 제공한다. 데이타 큐브의 어떤 속성은 측정 속성(measure attributes) 즉, 그것의 값이 관심의 대상이 되는 속성으로 선택되고 나머지 속성은 차원(dimension) 속성 또는 함수(functional) 속성으로 선택된다[5]. 데이타 큐브의

· 이 연구는 2006학년도 한국외국어대학교 교내학술연구비의 지원에 의하여 이루어진 것임
† 종신회원 : 한국외국어대학교 산업정보시스템공학부 교수
sllee@hufs.ac.kr
†† 비 회 원 : 서울교육대학교 컴퓨터교육학과 교수
chunsj@snue.ac.k
††† 종신회원 : 한국과학기술원 전자전산학과 및 Image Information Research Center
chungcw@cs.kaist.ac.kr
논문접수 : 2004년 7월 15일
심사완료 : 2006년 3월 22일

각 차원은 속성 값의 범위에 따라서 고정된 수의 구간으로 나누어지며, 따라서 데이타 큐브는 셀이라 불리는 같은 길이의 하이퍼-사각형 단위로 구성된다.

영역-합 질의(range-sum query)는 데이타 웨어하우스에서 속성들 간의 추세를 찾고 관계를 발견하는데 잘 알려진 도구이며, 이것은 지정된 질의 영역에서 선택된 셀들의 값들을 더하는 것이다. 데이타 큐브 자체를 엑세스하는 직접적인 방법(direct method)은 너무 많은 셀들을 엑세스해야 하므로 심각한 처리 오버헤드를 초래한다. 엑세스해야 되는 셀의 수는 질의에 의해 명시된 서브 큐브의 크기에 비례한다. 이러한 오버헤드를 극복하기 위해 프리픽스 섬(prefix-sum)접근법이 제안되었다. 이 접근법에서 영역-합 질의를 빠르게 하는 주요한 아이디어는 데이타 큐브의 다차원 프리픽스-합들을 미리 계산하는 것이다. 임의의 영역-합 질의는 2^d 개의 프리픽스-합들을 엑세스함으로 계산될 수 있다. 여기서 d는 차원이다. 비록 이러한 방법은 상당한 효율을 보여주나 데이타의 갱신시 많은 비용을 초래하므로 효율적이지 못하다. 데이타 큐브에서 한 셀의 변경은 프리픽스-섬 큐브(이하 PC)의 수많은 셀들을 변경하게 한다. 현재의 기업환경에서 큐브의 데이타 요소는 능동적으로 변경되어진다. 이것은 데이타 큐브 상의 갱신 파급(propagation)이 중요하게 간주되게 한다.

또 다른 문제는 데이타 큐브가 희소할 때 발생하는 저장 공간 오버헤드이다. 큐브의 차원이 커질수록 셀을 수용할 저장 공간은 지수적으로 증가한다. 그러나 현실적으로 고차원 데이타 큐브 내에 실제로 값이 존재하는 셀의 수는 큐브 전체 셀에 비해 매우 적다. 예를 들어 미국의 국세 조사국 (US Census Bureau)으로 부터의 실제 데이타를 고려해보자[7]. 분석을 위해 372 개의 속성에서 11개의 속성만 선택되었고, 측정 속성은 수입이고 10 개의 차원 속성은 나이, 결혼여부, 성별, 교육, 인종, 혈통, 가족형태, 세대구성, 연령 그룹과 노동 계층이다. 비록 데이타 큐브는 10차원이 되고 결과적으로 1천6백만이 넘는 셀을 가지나 큐브의 밀도는 약 0.001이다. 단지 15,985 개의 실제 값을 가지는 셀들이 존재한다[8]. 한편 대다수의 데이타 큐브는 여러 개의 밀집된 데이타 영역을 포함하며(고밀도 영역), 큐브의 나머지 공간에는 희박하게 분포된 점들을 포함하는 군집된 데이타 분포를 보여 준다[3].

이 논문에서는 저장 공간 오버헤드를 줄이기 위해 대용량 저밀도 데이타 큐브로부터 고밀도 서브-큐브들을 찾는 효과적인 알고리즘을 제안한다. 이러한 서브-큐브들은 갱신 파급을 최소화하면서 영역-합 질의에 응답하도록 처리된다. 큐브 내의 임의의 셀의 값을 변경하는 것은 단지 그 셀을 포함하는 서브-큐브에만 영향을 미치게 되며, 이것은 갱신 파급을 해당 서브-큐브 내에서만 이루어지도록 제한시킨다. 대용량 저밀도 큐브에서 고밀도 서브-큐브를 찾는 문제는 주어진 데이타 공간에서 군집된 데이타 포인트들을 찾는 문제와 유사하다. 데이타 포인트들의 군집을 찾는 문제는 다음과 같이 정의될 수 있다: 다차원 공간에서 데이타 포인트의 집합이 주어질 때 이러한 포인트들을 여러 개의 군집들로 나누되 각 군집 내의 포인트들은 유사한 특징을 가지는 반면 서로 다른 군집들에 있는 포인트들은 다른 특징을 갖도록 하는 것이다. 데이타 큐브에서 어떠한 군집에도 속하지 않는 포인트들은 아웃 라이어(outlier) 또는 노이즈(noise)로 언급한다. 다양한 군집화 방법들이 데이타베이스 분야에서 연구되고 있다. 그러나, 고밀도 서브-큐브를 찾는 문제는 원래 큐브의 고밀도 영역들로부터 사각형 형태의 서브-큐브를 발견하는 문제라는 점에서 기존의 군집을 찾는 방법들과는 다르다.

차원의 집합을 $\boldsymbol{D} = \{D_i \mid i = 1, 2, \cdots, d\}$라 하고, 각차원의 크기를 $|D_i|$로 가정하자. 여기서 D_i를 차원 속성이다. 그러면 d 차원의 데이타 큐브는 크기가 $|D_1| \times |D_2| \times \cdots \times |D_d|$ 인 $C[0:|D_1|-1]..[0:|D_d|-1]$로 표현된다. C 의 각 셀은 측정 속성 값을 포함하며 $c[j_1][j_2]..[j_d]$로 표현된다. 큐브 C는 다수의 서브-큐브를 포함하며 각 서브-큐브는 $SC[l_1:h_1]..[l_d:h_d]$로 표현되고 $0 \le l_i \le h_i \le |D_i|-1$ 이며, 여기서 $0 \le l_i \le h_i \le |D_i|-1$ 이다. 서브-큐브를 찾는 문제는 다음과 같이 정형적으로 표현된다:

Given: d-차원 데이타 큐브 $C[0:|D_1|-1]..[0:|D_d|-1]$, 서브-큐브내 최소 셀의 수 *minCells,* 히스토그램-빈에 대한 밀도 임계치(τ), 서브-큐브에 대한 밀도 임계치(δ)

Target: 정의된 밀도 조건을 만족하는 고밀도 서브-큐브들의 집합 SC를 찾는 것.

입력 파라미터 *minCells* 은 아웃라이어를 결정하는데 필요하다. 만일 추출된 서브-큐브가 이 값보다 적은 셀들을 가진다면 이 서브-큐브에 있는 값이 있는 셀들은 아웃라이어로 간주된다. 물론 이 값은 어플리케이션에 따라서 다양하게 결정될 수 있다. *minCells* 이 너무 작은 값을 가지면 중요치 않은 서브-큐브들을 발견하므로 메모리 활용도가 떨어진다. 반면에 너무 큰 값을 가지면 중요한 서브-큐브들을 놓치게 된다. 두개의 임계치 값 τ와 δ는 조건 파라미터로 고밀도 서브-큐브를 찾는데 사용된다. 이러한 임계치들은 3장에서 논의된다.

본 논문의 구성은 다음과 같다. 2장은 큐브 기술과 군집 방법에 관한 간단한 언급과 함께 관련 연구를 기술한다. 3장은 제안된 작업의 자세한 설명과 알고리즘을 포함한다. 제안된 방법에 대한 분석과 실험적인 결과가 4장에 주어지고 5장에서 결론을 맺는다.

2. 관련 연구

Ho 등[6]이 프리픽스 섬 큐브를 제안한 이래, 프리픽스 섬 큐브 상에서 갱신 파급을 최소화하려는 다양한 접근법들이 제안되었다. Geffner 등[5]은 직접적인 검색 방법과 프리픽스 섬 접근법 간의 질의-갱신에 균형을 맞추기 위하여 상대 프리픽스 섬 큐브(relative prefix-sum cube: RPC)를 제안하였다. 그러나 이 접근법은 고차원 및 대용량 데이타 큐브에서는 갱신 비용이 지수적으로 증가하기 때문에 비현실적이다. Chan and Ioannidis[2]은 두개의 직각(orthogonal) 차원에 기초한 계층적 큐브(hierarchical cubes)로 불리는 새로운 형태의 큐브를 제안하였다. 그들은 HBC(hierarchical band cube)가 Geffner 등[5]에 의해 제안된 알고리즘 보다 상당히 더 좋은 질의-갱신 트레이드오프(trade-off)를 가짐을 보였다. Geffner 등[4]은 PC를 재귀적으로 분해(decomposition)해서 얻어지는 동적 데이타 큐브(dynamic data cube)를 제안하였다. 그들은 또한 데이타 큐브의 각 차원이 같은 크기를 가진다고 가정하여 이러한 분해 기술에 의해 트리 구조를 만들었다. 그러나, 데이타 큐브가 고차원이고 대용량이면 트리가 너무 커지게 되므로 이러한 접근법을 적용하는 것이 어렵다. 그러므로 이러한 접근법은 심각한 처리 오버헤드를 초래하게 된다. 그러나, 이러한 접근법들은 비록 어느 정도까지 갱신 파급을 감소하였으나 여전히 갱신 파급 문제를 가지고 있다는 점에서 한계를 가진다. 왜냐하면 RPC는 PC를 약간 변환한 것이기 때문이다.

최근에 전 등[1]은 Δ-tree로 불리는 인덱스 구조를 사용하여 갱신 비용을 상당히 줄이는 효율적인 알고리즘을 제안하였다. 그러나, 위에 언급한 방법들의 갱신의 속도 향상은 검색 효율을 희생함으로써 얻어진 것이다. 많은 OLAP(on-line analytical processing) 어플리케이션에서 검색 효율의 희생을 최소화하면서 갱신 성능을 개선시키는 것이 중요한 이슈가 되고 있다. 더군다나, 위에서 언급한 방법들은 대용량 데이타 큐브에서의 저장 공간 축소의 문제에 대해서는 언급하지 않았다.

3. 제안된 기법

이 절에서는 밀도 함수에 근거하여 희박한 대용량 데이타 큐브에서 조밀한 서브 큐브를 찾는 방법에 관하여 논의한다. 각 차원에서 조밀한 구간을 찾고, 그것에 의거하여 조밀한 서브 큐브를 구축하는 효과적인 알고리즘을 제시한다.

그림 1은 제안한 방법을 그림으로 나타낸 것이다. 먼저 값이 있는 셀들을 각 차원마다 준비된 일 차원 배열에 사상시켜 각 차원에서의 조밀한 구간을 찾아낸다. 조밀한 구간에 의거하여 초기 서브 큐브를 생성한다. 생성된 각 서브 큐브에 대하여 다시 조밀한 구간을 찾아내고 이 구간에 의거하여 조밀한 서브 큐브를 생성한다. 이러한 단계가 서브 큐브가 충분히 조밀해질 때까지 반복된다. 다음에 이전 단계에서 생성된 후보 서브 큐브들이 주어진 밀도 임계 값에 대하여 정제 단계를 거친다. 서브큐브 확장 단계에서 근접하게 위치한 서브큐브들이 병합되며, 서브큐브 축소 단계에서는 후보 서브큐브들의 희박한 표면이 제거된다.

3.1 후보 서브 큐브 추출

데이타 큐브의 각 셀은 측정 속성 값을 가지며, 각 셀에 값이 있는 지의 여부에 관한 정보를 포함하기 위한 자료 구조를 유지한다. 각 셀은 ON(non-empty) 혹은 OFF(empty)의 정보를 갖는 한 비트로 표현된다. 그림 2는 2 차원의 16×16 데이타 큐브 C를 나타내며, 여기에서 33 셀이 값을 가지고 있다.

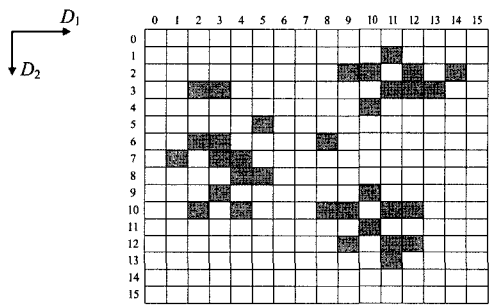

그림 2 2차원의 16×16 데이타 큐브C

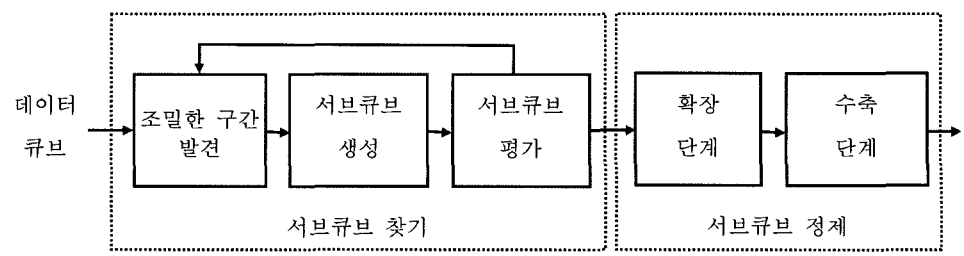

그림 1 주어진 데이타 큐브에서 조밀한 서브 큐브를 찾는 방법

3.1.1 밀집 구간 발견

이 절에서는 각 차원으로부터 밀집 구간을 발견하기 위한 두 가지 알고리즘(*Bit-map based*와 *Histogram-flattening based*)를 제시한다.

Bit-map based: 각 차원에 대하여 크기가 $|D_i|$인 배열을 가지며, 이 배열의 각 요소는 두 가지 상태, *true*(dense) 혹은 *false*(not dense)를 나타낸다. 그림 2의 큐브의 경우에 배열 각 요소의 해당 셀에 최소한 하나의 셀이 값을 가지고 있으면 true이고, 그렇지 않으면 false 값을 가지게 함으로써, 1차원에 대해서는 [1:5]와 [8:14]의 두 밀집 구간을 가지며, 2차원에 대해서는 [1:13]의 하나의 밀집 구간을 가진다.

그림 3과 4는 배열을 생성하기 위한 알고리즘 *Produce_Array* 와 하나의 차원에 대하여 밀집 구간을 찾기 위한 *Find_Dense_Intervals* 을 보여 준다.

Algorithm ***Produce_Array***
***Input*:** A cube $C[l_1:h_1]..[l_d:h_d]$
***Output*:** A set of *d one*-dimensional arrays $a_i[l_i:h_i]$ for dimension i=1,2,..,d
Step 0: /* Initialization */
for each array $a_i[l_i:h_i]$
fill all bins of the array with **false**
end for
Step 1: /* Project each cell of the cube to the *one*-dimensional arrays */
for each cell $c[j_1]...[j_d]$ of the cube $C[l_1: h_1]...[l_d:h_d]$, $(l_i \le j_i \le h_i)$
if the cell c is not empty **then**
$a_1[j_1]$, $a_2[j_2]$, ..., $a_d[j_d] \leftarrow$ **true**
end if
end for
Step 2: **return** a set of $a_i[l_i:h_i]$

그림 3 알고리즘Produce_Array

Algorithm ***Find_Dense_Intervals***
***Input*:** A *one*-dimensional array $a[l:h]$
***Output*:** A set of k dense intervals $a_k[l_k:h_k]$

$$\left(1 \le k \le \left\lceil \frac{h_{i,k} - l_{i,k} + 1}{2} \right\rceil\right)$$

Step 0: /* Initialization */
$k \leftarrow 1$
$current\_interval$, $a_c[l_c:h_c] \leftarrow$ null
Step 1: /* Finding dense intervals */
for each bin $a_i[j]$ of array $a_i[l_i:h_i]$ $(l_i \le j \le h_i)$
if $a_i[j]$ is not empty **then**
if $current\_interval$ is not null **then**
$h_c \leftarrow h_c + 1$
else
l_c, $h_c \leftarrow j$ // Initialize $current\_interval$
end if
else
if $current\_interval$ is not null **then**
$a_k \leftarrow current\_interval$
$k \leftarrow k + 1$
$current\_interval \leftarrow$ null
end if
end if
end for
Step 2: **return** a set of $a_k[l_k:h_k]$

그림 4 알고리즘 Find_Dense_Intervals

Histogram-flattening based: 이 방법에서는 밀집 구간을 발견하기 위하여 히스토그램을 이용한다. 비트맵 방법과 마찬가지로 각 차원마다 일 차원 배열을 유지하며, 각 배열의 빈(bin)은 그 빈에 해당하는 데이타 큐브 내의 값이 있는 셀의 수를 포함한다. 그림 2의 예에서, 차원 1의 배열은 16개의 빈을 가지고 있고, 값이 있는 셀의 수가 해당 빈에 기록된다. 이들 빈 중 밀도 임계 값(τ)을 초과하는 빈은 조밀한 빈으로 간주된다. 위의 그림에서 $\tau = 1$일 때, 차원 1에 대하여 두 개의 조밀 구간, [1:5]와 [8:14]를 구할 수 있고, 마찬가지로 차원 2에 대해서는 한 개의 조밀 구간 [1:13]을 구할 수 있다. 히스토그램 평활화(histogram flattening) 기법은 히스토그램 내의 각 빈의 값들의 차를 유연하게 하기 위하여 사용된다. 다음 그림은 히스토그램 평활화 기법을 나타낸 것이다.

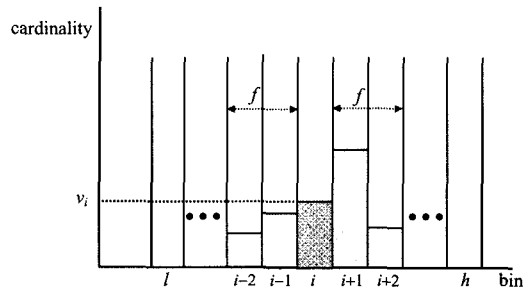

그림 5 $f = 2$ 일 때의 히스토그램 평활화 기법

그림 5에서 i번째 빈이 값 v_i를 갖는다 하자. v_i는 값이 있는 셀의 개수를 나타내며, 빈 i의 값은 v_i로 정해지는 것이 아니라 이웃 빈을 고려하여 히스토그램 평활화에 의해 구해진다. f를 평활화 요소(flattening factor)라 하자. $f = 0$이면 평활화가 일어나지 않는다. $f = 1$인 경우, 오른쪽과 왼쪽의 이웃 2빈이 빈 i의 값을 구하는 데 참여한다. 즉, 빈 i의 값은 $v_i' = (v_{i-1} + v_i + v_{i+1}) / 3$으로 구해진다. 유사하게 $f = 2$일 때는 4개의 이웃이 참여한다. 따라서, $v_i' = (v_{i-2} + v_{i-1} + v_i + v_{i+1} + v_{i+2}) / 5$이 된다. 일반적으로 k개의 빈을 가지는 구간 $[l:h]$에 대한 히스토그램 H에서, 빈 i의 값은 v_i'는 다음의 식으로 계산된다:

(1) $h - l \ge f$ 인 경우:

$$v_i' = \begin{cases} \dfrac{1}{f+i-l+1} \cdot \sum_{t=l}^{i+f} v_t, & \text{when } l \le i \le l+f-1, \\ \dfrac{1}{2f+1} \cdot \sum_{t=i-f}^{i+f} v_t, & \text{when } l+f \le i \le h-f, \\ \dfrac{1}{f-i+h+1} \cdot \sum_{t=i-f}^{h} v_t, & \text{when } h-f+1 \le i \le h. \end{cases}$$

(2) $h - l < f$ 인 경우:

$$v_i' = \frac{1}{h-l+1} \cdot \sum_{i=l}^{h} v_i \,. \quad \blacksquare$$

처음과 나중의 f개의 빈에 대한 v_i' 값은 그 값을 계산하기 위해 참여한 빈의 수가 $2f+1$ 보다 작기 때문에 다르게 취급되어야 한다. 즉, $f=2$일 때, 첫번째 빈의 v_1' 값은 $v_1' = (v_1 + v_2 + v_3) / 3$ 이 되고, 두 번째 빈의 값 v_2'는 $v_2' = (v_1 + v_2 + v_3 + v_4) / 4$ 이 된다. 히스토그램의 모든 빈에 대한 평활화 값을 구한 후, 각 값은 밀도 임계 값 τ에 대하여 그 값을 포함하고 있는 빈이 조밀한가를 판단하기 위하여 검사된다. τ 보다 큰 값을 갖는 빈은 조밀한 빈으로 간주된다. 조밀한 빈만을 추출하여 조밀 구간을 구성한다.

3.1.2 후보 서브 큐브의 구성

각 차원의 조밀 구간을 찾은 후, 그 구간에 의거하여 서브 큐브를 구성한다. $f=1$ 일 때 차원 1에서 두 개의 조밀 구간 [1:5] 와 [8:14]를 구하고, 차원 2에서 하나의 조밀 구간 [1:13]을 구한다. 이 구간들을 사용하여 그림 6(a)와 같이 두개의 초기 서브 큐브, SC_1[1:5, 1:13] 과 SC_2[8:14, 1:13]를 구성할 수 있다. 그러나 이 서브 큐브들은 희박한 영역을 포함할 수 있다. 따라서, 각 서브 큐브의 각 차원에 대하여 조밀 구간을 찾는 과정을 다시 수행한다. 서브 큐브 SC_1[1:5, 1:13]에 대하여, D_1에서 조밀 구간 [1:5]를 구하고, D_2에서 두 개의 조밀 구간 [3:3]과 [5:10]를 구함으로써, 두 개의 서브 큐브 SC_{11}[1:5, 3]과 SC_{12}[1:5, 5:10]을 구성한다. 유사하게 서브 큐브 C_2[8:14, 1:13]에 대하여 D_1에서 조밀 구간 [8:14]를 구하고, D_2에서 세 개의 조밀 구간 [1:4], [6:6], 그리고 [9:13]을 구하여 세 개의 서브 큐브 SC_{21}[8:14, 1:4], SC_{22}[8:14, 6], 그리고 SC_{23}[8:14, 9:13]을 구성한다. 이 과정을 더 이상 큐브를 분할할 수 없을 때까지 반복하여 그림 6(b)와 같이 5개의 후보 서브 큐브를 구할 수 있다.

다음 알고리즘 7과 8은 후보 밀집 서브 큐브를 발견하는 절차를 기술한다. 이러한 반복 과정으로 생성된 서브 큐브들은 서로 겹치지 않게 된다(*non-overlapping*).

3.2 서브 큐브의 정제 단계

반복 처리에서 생성된 후보 서브 큐브들은 큐브의 밀도에 근거하여 정제 단계를 거친다. 큐브 C의 밀도 *density*(C)는 조밀한 셀의 수 *numDenseCells*(C)를 전체 셀의 수 *numCells*(C)로 나눈 값으로 정의된다. 그러면, 앞의 예에서 큐브 C의 밀도는 *density*(C) = 33/(16*16) = 0.129가 된다. 큐브 C로부터 k 개의 서브 큐브 SC_1, SC_2, …, SC_k를 찾아 냈다면, 다음의 식으로 서브 큐브들의 평균 밀도를 계산할 수 있다.

$$density_{mean}(SC) = \frac{\sum_{i=1}^{k} numDenseCells(SC_i)}{\sum_{i=1}^{k} numCells(SC_i)}$$

확장 단계: 두 개의 근접한 서브 큐브는 사전에 정의된 조건을 만족하면 병합된다. 두 개의 서브 큐브를 병합하면 하나의 큰 서브 큐브를 생성하게 되므로 이 단계를 '확장' 단계라 한다. 두 개의 서브 큐브를 병합하기 위하여 병합 연산자를 다음과 같이 정의한다:

정의 1 (*병합 연산자* ⊕). 두 개의 d차원 큐브, $A[l_{1,A}:h_{1,A}, \ldots, l_{d,A}:h_{d,A}]$와 $B[l_{1,B}:h_{1,B}, \ldots, l_{d,B}:h_{d,B}]$가 병합되는 경우, 병합 연산자 ⊕는 $A \oplus B = C[l_{1,C}:h_{1,C}, \ldots, l_{d,C}:h_{d,C}]$로써 정의되며, 이때 $l_{i,C} = min(l_{i,A}, l_{i,B})$, $h_{i,C}$ =

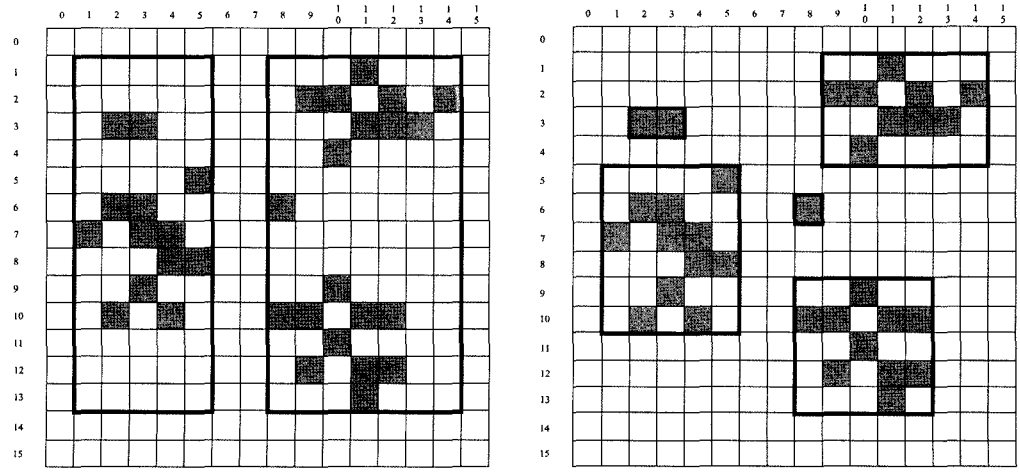

그림 6 (a) $f=1$일 때 생성된 두 개의 서브큐브, SC_1[1:5, 1:13]와 SC_2[8:14, 1:13], (b) 반복 처리에서 생성된 최종 5개의 서브큐브 SC_{111}[2:3, 3], SC_{121}[1:5, 5:10], SC_{211}[9:14, 1:4], SC_{221}[8, 6], SC_{231}[8:12, 9:13]

Algorithm ***Build_SubCubes***
Input: A cube $C[l_1:h_1]..[l_d:h_d]$
Output: A set ***SC*** of d-dimensional sub-cubes $A_k[l_{1,k}:h_{1,k}]..[l_{d,k}:h_{d,k}]$
Step 0: /* Projection */
/* ***SA***: a set of d *one*-dimensional arrays */
SA ← ***Produce_Array***(*C*)
Step 1: /* Finding dense intervals in ***SA*** */
/* SI_i: a set of dense intervals for a_i */
for each *one*-dimensional array $a_i[l_i:h_i]$
SI_i ← ***Find_Dense_Intervals***(a_i)
end for
Step 2: /* Building sub-cubes */
Generate sub-cubes by combining dense intervals of each dimension
(the number of sub-cubes generated: $|SI_1|\times|SI_2|\times\cdots\times|SI_d|$)
SC ← sub-cubes generated
Step 3: **return** ***SC***

그림 7 알고리즘 Build_SubCubes

Algorithm ***Find_Dense_SubCubes***
Input: A d-dimensional data cube $C[0:|D_1|-1]..[0:|D_d|-1]$
Output: A set ***SCD*** of d-dimensional dense sub-cubes
Step 0: /* Initialization */
stack ← ***Build_SubCubes***(*A*) // *stack*: a data structure to hold cubes
Step 1: /* Finding dense sub-cubes */
do while (*stack* is not empty)
cube ← pop()
if *cube* is *continuous* **then**
SCD ← ***SCD*** ∪ { *cube* }
else
stack ← ***Build_SubCubes***(*cube*)
end if
end while
Step 2: ***return SCD***

그림 8 알고리즘 Find_Dense_SubCubes

$max(h_{i,A}, h_{i,B})$, 그리고 i = 1, 2, ⋯, d 이다. ■

큐브 B가 큐브 A에 병합될 때, 병합에 의하여 증가된 셀의 $numCells(A\oplus B)$-$numCells(A)$가 된다. 한편, 병합에 의하여 증가된 조밀한 셀의 수는 큐브 B의 셀 수와 같게 된다. 즉, $numDenseCells(B)$로 표시할 수 있다. 따라서, 병합된 큐브의 확장된 부분의 밀도 $density(\Delta_{A,B})$는 다음의 식으로 표시된다.

$$density(\Delta_{A,B}) = \frac{numDenseCells(B)}{numCells(A \oplus B) - numCells(A)}$$

앞의 예에서 서브 큐브 SC_{111}[2:3, 3]이 서브 큐브 SC_{121}[1:5, 5:10]에 병합되는 경우를 고려해보자. 정의 1의 병합 연산 $SC_{111} \oplus SC_{121}$은 병합된 큐브 SC'[1:5, 3:10]을 생성한다. 병합된 큐브의 확장된 부분의 밀도 $density(\Delta A_{A,B})$는 2/10 = 0.20이 된다.

큐브의 병합은 특정 조건이 만족될 때에만 허용된다. 큐브 B가 큐브 A에 병합될 때, 만족되어야 하는 조건은 다음 식으로 표시할 수 있다:

$$density(\Delta_{A,B}) \geq \delta$$

밀도 임계 값 δ는 다양한 도메인의 특성에 따라 결정되며, 사용자에 의해 주어진다. 여기에 몇 가지 방법을 제시할 수 있는 데, 첫째 큐브 B가 큐브 A에 병합될 때 큐브 A의 밀도를 사용할 수 있다. 즉, $\delta = density(A)$가 된다. 두 번째 방법은 이전 단계에서 구한 서브 큐브들의 평균 밀도를 선택할 수 있다. 앞의 예에서, SC_{111}[2:3, 3]을 다른 서브 큐브에 병합하는 경우를 고려해보자. $density(\Delta)$는 C_{121}[1:5, 5:10]에 대해서는 0.20이고, C_{211}[9:14, 1:4]에 대해서는 0.07, C_{221}[8, 6]에 대해서 0.07, 그리고 C_{231}[8:12, 9:13]에 대해서 0.02가 된다. 첫 번째 경우, C_{121}[1:5, 5:10], C_{211}[9:14, 1:4], C_{221}[8, 6], C_{231}[8:12, 9:13]에 대해서 $density()$가 각각 0.37, 0.38, 1.00, 0.40이 되고, 이 값들은 모두 병합 조건을 만족시키지 못하므로, 어떠한 서브 큐브와의 병합도 일어나지 않는다. 그러나 만일 사용자가 임계 값을 명시적으로 0.15로 지정을 했다면, C_{111}[2:3, 3]은 C_{121}[1:5, 5:10]에

병합되게 되며, 새로운 서브 큐브 C_{121}'[1:5, 3:10]를 생성하게 된다.

수축 단계: 확장 단계에서 병합되어 생성된 서브 큐브는 그 표면에 희박한 영역을 포함할 수 있다. 따라서 그 표면의 밀도가 밀도 임계 값보다 작으면 해당 표면은 큐브로부터 제거된다. 2차원 큐브 $C[l_1:h_1, l_2:h_2]$는 4개의 표면($S_1[l_1:h_1, l_2]$, $S_2[l_1:h_1, h_2]$, $S_3[l_1, l_2:h_2]$, $S_4[h_1, l_2:h_2]$)을 가진다. 일반적으로 d차원의 큐브$C[l_1:h_1, \cdots, l_d:h_d]$는 $2d$개의 표면($S_1[l_1:h_1, l_2:h_2, \cdots, l_d]$, $\cdots$, $S_{2d}[h_1, l_2:h_2, \cdots, l_d:h_d]$)을 갖는다. 이러한 표면을 표면 슬라이스(*surface slices*)라 부른다. 수축 단계에서는 병합 과정에서 생성된 각 표면 슬라이스를 검사하여, 어떤 슬라이스의 밀도가 δ 보다 낮으면 그 슬라이스를 큐브로부터 제거한다. 예를 들어, 두 서브 큐브를 병합하여 생성된 서브큐브 C_{121}[1:5, 3:10]을 고려해 보자. 이 큐브는 S[1:5, 3], S[1:5, 10], S[1, 3:10], S[5, 3:10]의 4개의 슬라이스를 가지고 있고, 각 슬라이스의 밀도는 각각 0.4, 0.4, 0.125, 0.25가 된다. 만일 밀도 임계 값으로써 *density*(C) (= 0.129)를 선택한다면 표면 S[1, 3:10]이 제거되며, 결과적으로 수축된 서브큐브 C_{121}'[2:5, 3:10]을 생성하게 된다.

또한, 최종으로 생성된 서브 큐브들은 한 큐브가 가져야 하는 최소 셀의 수 *minCell*에 대하여 검사 과정을 거치게 된다. 이 값보다 작은 서브 큐브들은 제거되며, 이러한 서브 큐브 내의 값이 있는 셀들은 아웃라이어로 간주된다.

4. 실험과 분석

이 절에서는 본 논문에서 제안된 방법의 효율성을 보여주기 위해, 여러 가지 차원에서 다양한 데이타 분포에 의하여 생성된 데이타 세트에서의 실험 결과와 수학적인 분석을 보여준다. 실험은 주로 제안된 기법을 사용할 경우 얻을 수 있는 가장 큰 장점인 큐브의 저장 공간 감소와 제안된 기법의 질의 수행 시간 변화를 측정하였다. 이 실험을 위한 컴퓨팅 환경은 Pentium 4 1.7GHz 프로세서로써 512M 메모리와 80G 하드 디스크이다.

4.1 실험을 위한 준비

실험을 위해 2,3,4,5차원에 대해 각각 8개의 큐브 즉, 32개의 데이타 큐브를 생성하였다. 실험은 편의상 4개 차원(d = 2,3,4,5)에 대해서 수행되었으나 제안된 기법은 데이타 세트의 차원에 제한을 두지는 않는다. 이러한 큐브들은 군집된 데이타 분포를 가지도록 하되 각 군집들 내의 영역은 고밀도인 반면 바깥영역은 저밀도가 되도록 생성되었다. 다음 표 1은 데이타 큐브를 생성하는데 사용된 파라미터들을 보여 준다.

표 1 데이타 큐브를 생성하는데 사용된 파라미터들

차원	각 큐브의 크기	큐브의 갯수	값이 있는 셀의 수
2	1000 × 1000	8	2500
3	250 × 100 × 50	8	5000
4	150 × 60 × 50 × 30	8	10000
5	100 × 50 × 40 × 30 × 20	8	50000

표 1에서 사용된 파라미터 외에 3장에서 데이타 큐브로부터 서브 큐브를 찾는데 필요한 다양한 파라미터들을 논의했었다. 히스토그램 플래트닝 계수(f)에 관하여서 f = 0(비트맵 방법) 뿐만 아니라 f = 1,2,3,4,5을 사용하였다. 히스토그램-빈의 밀도 임계치(τ)과 서브 큐브의 밀도 임계치(δ)에 관하여 각 차원에서의 히스토그램 빈의 평균값과 서브 큐브들의 평균 밀도를 각각 채택하였다.

또한 서브-큐브당 최소 셀의 수, 즉 *minCells* 에 대해서는 16으로 하고 16 보다 적은 셀들을 가진 서브 큐브 내에 포함된 셀들을 아웃라이어로 간주하고 서브 큐브와 별도로 분리된 테이블에 저장하였다.

4.2 저장 공간

제안한 방법의 주된 이점 중 하나가 서브 큐브의 저장 공간의 감소이다. 제안한 방법에 의하여 생성된 서브 큐브의 볼륨은 원래의 데이타 큐브에 비하여 현격한 차이를 보여준다. 이 절에서는 저장 공간의 감소를 수학적 분석을 통해서, 그리고 실험 결과를 통하여 제시한다.

4.2.1 분석적 비교

제안한 방법의 주된 아이디어는 희소 데이타 큐브에 대하여 기존의 방법에서 제시하였던 하나의 대용량 프리픽스-섬 큐브를 구축하는 대신, 복수 개의 조밀한 서브 큐브를 찾아내어 다수의 작은 프리픽스-섬 큐브를 구축하는 것이다. 하나의 대용량 프리픽스-섬 큐브를 사용하는 것에 비하여 다수의 서브 큐브를 사용할 경우, 저장 공간의 감소는 다음과 같이 계산될 수 있다: 복수 개의 차원 $\boldsymbol{D} = \{D_i \mid i = 1, 2, \cdots, d\}$로 구성된 데이타 큐브 C를 고려해보자. 각 차원의 크기를 $|D_i|$로 나타내고 데이타 큐브는 C[0:$|D_1|$-1]..[0:$|D_d|$-1]로 표현한다. 큐브의 크기는 다음식으로 계산될 수 있다.

$$|C| = |D_1| \times \ldots \times |D_d| = \prod_{i=1}^{d} |D_i|$$

분석의 편의상 $|D_1| = |D_2| = \cdots = |D_d| = N$ 라 하면, $|C|$는 N^d이 된다. 큐브의 하나의 셀(즉, 측정 속성)을 저장하기 위한 저장 공간을 c라 하면, 큐브 C를 저장하기 위한 총 저장 공간은 큐브가 다차원 배열로 구현된다고 가정할 때 $c \cdot N^d$ 이 된다. 따라서 공간 복잡도는 $O(N^d)$이다.

한편, 제안한 방법에서 각 서브 큐브는 역시 다차원

배열로 구현되고 (MOLAP: Multi-dimensional OLAP), 서브 큐브에 포함되지 않는 셀(아웃 라이어)은 관계형 테이블에 저장된다(ROLAP: Relational OLAP). 제안한 방법에서 데이타 큐브에 대하여 m 개의 서브 큐브 (SC_j, $1 \le j \le m$)와 o 개의 아웃 라이어를 생성하였고, 차원 d 에 대하여 서브 큐브 SC_j 의 에지의 크기가 $e_{j,d}$라 가정하자. 그러면, 발견된 서브 큐브들의 전체 크기 $size(SC)$ 는 다음 식으로 표시된다:

$$size(SC) = \sum_{j=1}^{m} |SC_j| = \sum_{j=1}^{m} |e_{j,1} \times \ldots \times e_{j,d}| = \sum_{j=1}^{m} \prod_{i=1}^{d} e_{j,i}$$

비교의 편의상 $e_{j,1} = e_{j,2} = \cdots = e_{j,d} = n$ 라 하면 모든 서브 큐브를 구현하기 위한 저장 공간의 크기는 $m \cdot c \cdot n^d$ 이 된다. 아웃 라이어의 경우($index$, $value$)의 형태로 저장되게 되며, 여기에서 $index$ 는 d 차원 공간의 좌표 값이고 $value$ 는 측정 속성이다. 따라서, 하나의 아웃 라이어를 저장하기 위한 공간은 $d \times I + c$이 되며, 여기에서 I 는 하나의 차원의 좌표 값을 표현하기 위해 필요한 공간이다. 표현의 일반성(generality)을 상실하지 않고 $I = c$라 할 수 있다. 따라서, 모든 아웃 라이어를 저장하기 위한 공간은 $o \cdot c \cdot (d+1)$ 이 된다. 결론적으로, 제안한 방법의 공간 복잡도는 $O(m \cdot n^d + o \cdot d)$로 나타낼 수 있는 데, m 과 o 는 상수로 간주될 수 있고, n 은 N 에 비하여 매우 작으므로, 제안한 방법은 기존의 프리픽스-섬 방법에 비하여 저장 공간의 효율이 높다고 결론지을 수 있다. 다음 절에서는 실험을 통한 공간 효율을 보여준다.

4.2.2 실험 결과

저장 공간 효율을 평가하기위해 제안된 방법에 의해 생성된 서브 큐브들의 평균 밀도와 데이타 큐브의 밀도를 비교했다. 비교 파라미터 den_ratio 는 다음의 식으로 정의되어진다.

$$den\_ratio = \frac{density_{mean}(SC)}{density(DataCube)}$$

이 비율이 커질수록 저장 공간 활용도는 더욱 효율적이 되는 것은 명백하다. 보충적인 평가 파라미터로서 데이타 큐브 내의 값이 있는 셀에 대한 서브 큐브들 내의 값이 있는 셀의 비율, 즉 $nonE\_ratio$ 를 다음과 같이 정의한다.

$$nonE\_ratio = \frac{\sum_{j=1}^{m} numDenseCells(SC_j)}{numDenseCells(DataCube)}$$

그림 9와 10은 각각 den_ratio 과 $nonE_ratio$을 보여준다. 그림 9에서 den_ratio 는 2차원의 경우에 데이타 큐브 보다 8.9-22.3배 높고, 3차원의 경우에는 14.2-31.9 배, 4차원의 경우에는 18.9-35.6 배, 5차원의 경우에는 23.3-38.5 배가 높은 것을 보여 준다. 제안된 방법이 높은 저장 공간 효율을 달성하며 차원이 높아짐에 따라 이 비율도 증가함을 관찰할 수 있다. 이것은 더 높은 차원에서 더 뛰어난 기억 공간 절감의 효율을 기대할 수 있음을 의미한다.

또한, 제안된 방법은 바람직한 $nonE_ratio$비율을 보여준다. 2차원에서 0.88-0.98, 3차원에서 0.90-0.99, 4차원에서 0.91-0.98, 그리고 5차원에서는 0.88-0.93 의 비율을 각각 보여 준다. 이것은 대부분의 값이 있는 셀들이 서브 큐브에 포함됨을 의미한다. 결과적으로 제안된 방법에 의해 확인된 서브 큐브 들이 대부분의 값이 있는 셀들을 포함하면서 높은 저장 공간 효율을 달성한다고 결론지을 수 있다.

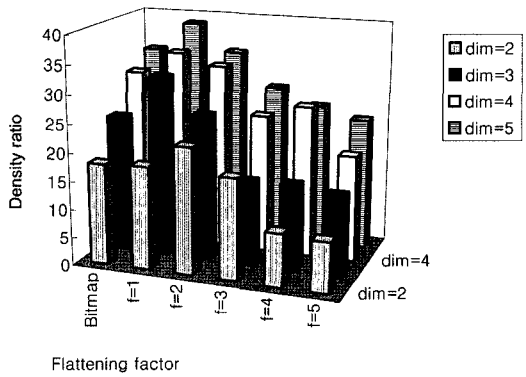

그림 9 차원과 평활화 계수에 따른 밀도율(Density ratio)

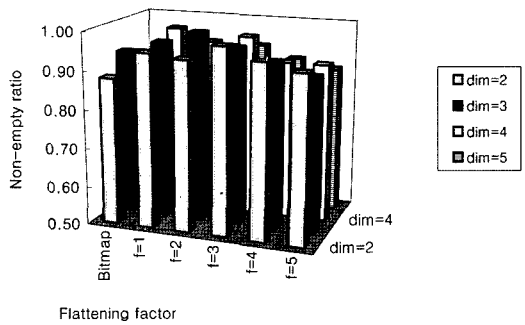

그림 10 차원과 평활화 계수에 따른 값이 있는 셀의 비율

4.3 질의 성능

질의 성능을 평가하기 위해 여러 가지 크기의 다양한 질의를 사용하였으며, 비교 대상은 PC 방법과 제안된 방법을 비교하였다. 왜냐하면 PC 방법이 갱신 파급의 어려움이 있지만 검색 효율에 있어서는 다른 방법들에 비해 더 좋은 성능을 보여주기 때문이다. 질의는 하이퍼 사각형으로 주어지며, 질의로 주어진 하이퍼 사각형 내

의 측정 속성(measure attribute)의 합을 구하는 방식으로 질의처리가 수행된다. 이 질의 하이퍼사각형에 내포되어 있거나 걸쳐있는 서브큐브로부터 측정 속성의 합을 각각 구한 후, 이로부터 전체의 합을 구한다. 질의는 소형, 중형, 대형의 3가지 그룹으로 분류된다. 소형 질의 그룹에서 질의 사각형의 각 테두리의 길이는 데이타 큐브의 대응되는 차원 길이의 0.3 배보다 작다. 중형 질의 그룹은 각 차원 길이의 0.3과 0.5 사이의 값으로 정해지고, 대형 질의 그룹은 각 차원 길이의 0.5 이상의 값으로 정해진다. 각 데이타 큐브와 각 그룹에 대해 5 개의 영역 질의를 생성하였다. 즉, 각 차원에 대해 120 질의를 생성하였고 성능을 비교하기위해 질의 결과로부터 평균값을 계산하였다.

질의 성능은 페이지 I/O의 수로 평가하였다. 왜냐하면 PC에서의 계산 시간은 I/O 시간에 비해 무시해도 좋을 만큼 작기 때문이다. 그림 11은 질의 크기와 차원에 따른 질의 성능의 비교이다. 그림에서 알 수 있듯이 제안된 방법이 적절하게 수행됨을 관측할 수 있다. 소형 그룹 질의의 경우 제안된 방법에서 page I/O는 5.5-18.8이고, PC 방법에서는 4.2-14.8으로 그다지 큰 차이를 보이지는 않는다. 예를 들면, 제안된 방법(SC)과 PC 방법의 page I/O 비율은 소형 질의 그룹에서 1.02에서 1.31으로써 이것은 실제 비즈니스 환경에 적절하게 사용할 수 있는 값이다.

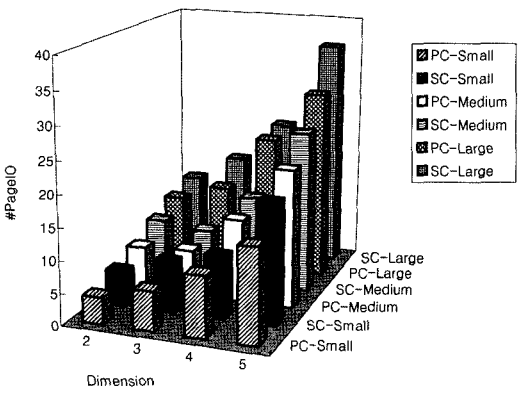

그림 11 질의 크기와 차원에 따른 질의 성능의 비교

5. 결 론

본 논문에서는 대용량 희소 데이타 큐브로부터 주어진 조건을 충족시키는 조밀한 서브 큐브를 발견하는 문제를 다루었다. 실제의 OLAP 환경에서 분석가들은 비즈니스 경향이나 기회 등을 발견하기 위하여 다양한 속성 사이의 관계를 분석하기를 원하며, 이때 데이타 큐브를 많이 사용하고 있다. 일반적으로 많은 데이타 큐브가 매우 희소하며 이것은 상당한 공간과 검색 오버헤드를 야기한다. 이 문제점을 해결하기 위하여 *bitmap based* 방법과 *histogram-flattening based* 방법을 사용하여 각 차원에서 밀집 구간을 찾고, 이 밀집 구간에 의하여 후보 서브 큐브를 추출하고, 반복적인 처리 과정에 의하여 최종 서브 큐브 집합을 얻는 알고리즘을 제안하였다.

제안한 기법의 효용성을 평가하기 위해 여러 종류의 크기를 갖는 큐브를 사용하였으며, 4가지의 차원(2,3,4,5)에서 실험을 수행하였다. 비록 실험에서는 데이타 세트 크기 및 차원이 제한되었지만, 제안된 기법은 보다 더 확장된 큐브 및 고차원에서도 실행될 수 있도록 설계되었다. 실험 결과 제안한 기법은 널리 사용되는 프리픽스-섬 큐브에 비하여 적절한 성능을 보이면서도, 큐브 저장 공간은 현저히 감소되었다. 또한 이러한 성능 개선은 차원이 높아질수록 더욱 증진됨을 관찰할 수 있었다. 향후 연구 계획으로써 보다 나은 검색 효율을 위하여 다수의 서브 큐브를 저장하고 검색할 수 있는 인덱스 구조에 대하여 연구할 예정이다.

참 고 문 헌

[1] S. J. Chun, C. W. Chung, J. H. Lee and S. L. Lee, Dynamic Update Cube for Range-Sum Queries, Proceedings of Int'l Conference on Very Large Data Bases, Italy, 2001, pp. 521-530.

[2] C. Y. Chan and Y. E. Ioannidis, Hierarchical cubes for range-sum queries, Proceedings of Int'l Conference on Very Large Data Bases, Scotland, 1999, pp. 675-686.

[3] D.W. Cheung, B. Zhou, B. Kao, H. Kan and S.D. Lee, Towards the building of a Dense-Region Based OLAP System, *Data and Knowledge Engineering*, Elsevier Science, V36, 1-27, 2001.

[4] S. Geffner, D. Agrawal, and A. El Abbadi, The Dynamic Data Cube, Proceedings of Int'l Conference on Extending Database Technology, Germany, 2000, pp. 237-253.

[5] S. Geffner, D. Agrawal, and A. El Abbadi, T. Smith, Relative prefix sums: an efficient approach for quering dynamic OLAP Data Cubes, Proceedings of Int'l Conference on Data Engineering, Australia, 1999, pp. 328-335.

[6] C. Ho, R. Agrawal, N. Megido, and R. Srikant, Range queries in OLAP Data Cubes, Proceedings of ACM SIGMOD Int'l Conference on Management of Data, 1997, pp. 73-88.

[7] U. S. Census Bureau, Census bureau databases, The online data are available on the web at http://www.census.gov/.

[8] J. S. Vitter and M. Wang, Approximate Computation of Multidimensional Aggregates of Sparse Data Using Wavelets, Proceedings of ACM

SIGMOD Int'l Conference on Management of Data, Pennsylvania, 1999, pp. 193-204.

이 석 룡

1984년 연세대학교 기계공학과 학사 1993년 연세대학교 산업공학과 전자계산 전공 석사. 2001년 한국과학기술원 정보 및통신공학과 컴퓨터공학전공 박사. 1984년~1995년 한국IBM 소프트웨어 연구소 선임연구원. 2002년~현재 한국외국어대학교 산업정보시스템공학부 부교수. 관심분야는 멀티미디어 데이타베이스, 데이타마이닝, 정보검색 등

전 석 주

1989년 경북대학교 전자공학과 컴퓨터공학전공. 2002년 한국과학기술원 정보및통신공학과 졸업. 2004년 서울교육대학교 컴퓨터교육과 교수. 현대중공업 중앙연구소 연구원. 관심분야는 데이타 마이닝, 데이타 웨어하우스와 OLAP, 멀티미디어 데이타베이스 등

정 진 완

1973년 서울대학교 공과대학 전기공학과 (학사). 1983년 University of Michigan 컴퓨터공학과(박사). 1983년~1993년 미국 GM 연구소 전산과학과 선임연구원 및 책임연구원. 1993년~현재 한국과학기술원 전산학전공 부교수 및 교수. 관심분야는 XML, 시맨틱웹, 멀티미디어 데이타베이스, 스트림 데이타 및 센서 네트워크 데이타베이스