SPEC: 데이타 웨어하우스를 위한 저장 공간 효율적인 큐브

(SPEC: Space Efficient Cubes for Data Warehouses)

전 석 주 † 이 석 룡 †† 강 흠 근 ††† 정 진 완 ††††

(Seok-Ju Chun) (Seok-Lyong Lee) (Heum-Geun Kang) (Chin-Wan Chung)

요 약 군집 질의는 사용자에 의해 명시된 질의 영역 내에서 큐브상의 군집 정보를 계산한다. 프리픽스-섬 기법에 기초한 기존의 방법들은 데이타의 누적된 합을 저장하기 위해 프리픽스-섬 큐브(PC)로 불리는 부가적인 큐브를 사용하므로 높은 저장공간 오버헤드를 초래한다. 이러한 저장공간 오버헤드는 기억장치의 추가적인 비용뿐만 아니라 업데이트의 부가적인 증식(propagation)과 더 많은 물리적 장치로의 접근시간을 유발시킨다. 본 논문에서는 대용량 데이타 웨어하우스에서 PC의 저장공간을 획기적으로 감소시킬 수 있는 'SPEC'으로 불리는 새로운 프리픽스-섬 큐브를 제안한다. SPEC은 PC내 셀들간의 종속에 의한 업데이트 증식을 감소시킨다. 이를 위해 대용량 데이타 큐브로부터 조밀한 서브큐브들을 발견하는 효과적인 알고리즘을 개발한다. 다양한 차원의 데이타 큐브와 여러가지 크기의 질의에 대해 폭 넓은 실험을 행하여 본 논문에서 제안한 방법의 효과와 성능을 조사한다. 실험적인 결과는 SPEC이 적절한 질의 성능을 유지하면서도 PC 저장공간을 상당히 감소시킴을 보여준다.

키워드 : 데이타 웨어하우스, 데이타 큐브, 프리픽스-섬 큐브, OLAP, 영역-합 질의, 클러스터링

Abstract An aggregation query computes aggregate information over a data cube in the query range specified by a user. Existing methods based on the prefix-sum approach use an additional cube called the prefix-sum cube(PC), to store the cumulative sums of data, causing a high space overhead. This space overhead not only leads to extra costs for storage devices, but also causes additional propagations of updates and longer access time on physical devices. In this paper, we propose a new prefix-sum cube called 'SPEC' which drastically reduces the space of the PC in a large data warehouse. The SPEC decreases the update propagation caused by the dependency between values in cells of the PC. We develop an effective algorithm which finds dense sub-cubes from a large data cube. We perform an extensive experiment with respect to various dimensions of the data cube and query sizes, and examine the effectiveness and performance of our proposed method. Experimental results show that the SPEC significantly reduces the space of the PC while maintaining a reasonable query performance.

Key words : Data warehouses, Data cube, Prefix-sum cube, OLAP, Range-sum query, Clustering

1. 서 론

데이타 큐브는 질의의 영역 내에 있는 중요한 애트리뷰트에 군집연산을 적용하는 영역-합 질의(range-sum query)라 일컫는 데이타 상에서의 유용한 분석 툴을 제공한다. 영역-합 질의는 데이타 웨어하우스에서 애트리뷰트들간의 추세를 찾고 관계를 발견하는데 매우 유용한 도구이다[1]. Ho 등[2]은 프리픽스-섬(Prefix-sum) 방법이라 불리는 데이타 큐브에서 영역-합 질의를 효율적으로 계산하는 알고리즘을 제안했다. 프리픽스-섬 방법의 기본 아이디어는 데이타 큐브의 많은 프리픽스-섬을 미리 계산하는 것인데 이러한 프리픽스-섬은 런타임시에 임의의 질의에 답하는 데 사용된다. 이러한 방법은 데이타 큐브의 크기에 상관없이 영역-합 질의가 상수시

· 본 연구는 대학 IT 연구센터 육성 지원 사업의 연구결과로 수행되었음
† 정 회 원 : 서울교육대학교 컴퓨터교육과 교수
chunsj@snue.ac.kr
†† 종신회원 : 한국외국어대학교 산업정보시스템공학부 교수
sllee@hufs.ac.kr
††† 비 회 원 : 우송공업대학 컴퓨터정보통신계열 교수
hgkang@woosongtech.ac.kr
†††† 종신회원 : 한국과학기술원 전산학과 교수
chungcw@cs.kaist.ac.kr
논문접수 : 2004년 1월 9일
심사완료 : 2004년 11월 2일

간에 처리됨을 보여주므로 매우 효율적이라는 것이 증명되었다. 그러나 프리픽스-섬 방법은 데이타 큐브에 데이타 요소들이 자주 변경될 때 PC를 유지하는 데에 매우 많은 비용이 드는 문제점을 갖고 있다. 따라서 데이타의 요소들이 자주 변경되는 응용에서는 사용할 수가 없다. 최근에 프리픽스-섬 방법에 근거하여 업데이트 시간을 줄이기 위한 많은 훌륭한 연구들이 행해졌다[1,3-6]. 그러나 이러한 방법들의 가장 큰 문제점은 원래의 데이타 큐브가 전형적으로 매우 희박할 때에 조차 PC는 매우 조밀 하다는 것이다. 그러므로 PC를 유지하기 위해서 많은 양의 저장 공간이 필요하게 된다. 이러한 저장공간의 부담은 기억장치의 추가적인 비용 뿐만 아니라 부가적인 업데이트 증식(update propagation) 을 유발하고 또한 물리적인 장치에 접근 시 더 많은 시간을 필요로 한다[7].

본 논문은 대용량 데이타 큐브를 공간효율적인 방법으로 저장하는 'SPEC'이라 불리는 새로운 프리픽스-섬 큐브를 제안한다. SPEC을 구축하기위해서 먼저 데이타 큐브의 조밀한 구간들을 찾고 이에 기초하여 주어진 임계치 값에 따른 조밀한 서브큐브들의 집합을 찾아낸다. 이렇게 찾은 서브큐브에 해당하는 작은 PC들을 만들어 SPEC을 구축하면 프리픽스-섬 방법에 비해 저장공간을 엄청나게 줄일 수 있다. 또한 델타큐브(△Cube)라 불리는 자료구조를 사용하는데 데이타 큐브에서 변경된 셀과 아웃라이어(outlier) 셀이 모두 델타큐브에 저장된다. 델타큐브는 PC에 있는 셀들간의 의존성에 의해 발생되는 업데이트 증식을 감소시킨다. 본 논문에서는 대용량 데이타 큐브로부터 어떻게 조밀한 서브큐브가 발견되고, 어떻게 아웃라이어 셀들이 표현되고 어떻게 질의와 업데이트가 효율적으로 처리되는 지를 보여준다.

본 논문의 공헌을 다음과 같이 요약할 수 있다.

- 대용량 데이타 웨어하우스에서 클러스터링 기법을 사용하여 저장공간을 엄청나게 줄이는 'SPEC' 이라 불리는 새로운 프리픽스-섬 큐브를 제안한다. 다양한 크기의 데이타 큐브들에 대해 분석을 하며 제안된 방법이 여러가지 차원에서 매우 효율적으로 수행됨을 실험적으로 평가한다.
- 희박한 데이타 큐브로부터 조밀한 서브큐브를 효율적으로 찾기위해서 이에 적합한 새로운 클러스터링 기법을 개발한다. 이 기법은 각 차원에서 조밀한 구간들을 효율적으로 찾아내며 이러한 조밀한 구간들에 기초하여 조밀한 서브큐브들을 만든다.

논문의 나머지 구성은 다음과 같다. 2절에서는 본 논문에서 제안한 방법을 자세히 설명하고 3절에서는 데이타 큐브로부터 조밀한 서브큐브들을 찾는 방법을 나타내고 4절에서 프리픽스-섬 방법과 제안된 방법에 대한 실험적인 평가가 주어진다. 마지막으로 5절에서 논문의 결론을 맺는다.

2. 저장공간 효율적인 큐브

경영 분석가들은 비즈니스의 경향과 경영상의 추가적인 기회의 포착을 위해서 여러 가지 다양한 속성들 사이의 관계를 파악하길 원하는데, 이러한 분석 작업에 필요한 데이타 큐브는 대개 고차원인 경우가 많다. 또한 고차원에서는 데이타 큐브가 희박해지는 경향이 있다. 데이타 웨어하우스에서 다차원의 데이타들이 작은 영역에서 조밀한 클러스터들을 많이 가지고 있고, 대부분의 나머지 공간에는 데이타가 희박하게 흩어져 있다. 다음 예는 매우 희박한 경우의 데이타 큐브를 보여주고 있다 [8].

표 1 미국 인구 통계 데이타의 속성과 속성 값의 도메인

속성	도메인	속성	도메인
나이	0-150	수입	0-99999
몸무게	1-500	인종	1-5
가정의 타입	1-5	결혼 유무	1-7
직업의 종류	0-8	교육 정도	1-17

예제 2.1 표 1에서와 같이 2×10^{14}개 이상의 셀을 가지고 있는 미국의 인구 통계 자료(US census)에 대한 8차원의 데이타 큐브를 생각해 보자. 미국 인구 통계 자료가 2억 명의 미국인들에 대한 자료를 저장하고 있다고 가정한다면, 이 데이타 큐브에서 Non-zero 셀은 많아야 전체의 0.0001% 뿐이다.

Ho등은 데이타 큐브에서 필요한 정보를 미리 계산해 놓고 질의의 크기에 관계없이 일정한 시간 내에 영역-합 질의를 수행하는 새로운 방법을 개발하였다. 이 방법은 누적 합을 저장하는 PC라고 불리는 큐브를 별도로 사용한다. 원래의 큐브가 희박하더라도 대응하는 PC는 누적 합을 저장하므로 조밀하게된다. 지금까지의 연구는 질의 수행시의 응답시간과 업데이트시간에만 관심을 가졌고 저장 공간의 비용에 대한 깊은 연구는 없었다. 그러나 현실적으로 PC를 사용하는 방법은 저장 공간의 문제와 셀들간의 종속성에서 기인하는 업데이트 증식 문제가 매우 심각하다. 만약 OLAP 시스템이 제한된 저장 공간을 가지고 있다면, PC를 구현하는 것은 현실적으로 불가능하다고 할 수 있다. 더구나 업데이트 증식 문제는 PC의 크기가 커질수록 더 증폭되는 경향이 있다. 본 논문에서 제안된 아이디어는 다음과 같이 요약된다.

- 미리 정의된 임계치를 초과하는 조밀한 서브큐브들을 찾는다. 한 개의 커다란 PC를 만드는 대신 각각의 조밀한 서브큐브에 대해 작은 크기의 여러 개의 PC를

만들어 SPEC을 구축한다. SPEC은 저장 공간의 필요량을 크게 감소시킬 수 있으며 업데이트 증식문제를 상당히 완화시킬 수있다. 조밀한 서브큐브를 찾는 방법은 3절에서 자세히 설명한다.

- 데이타 큐브 내의 한 셀의 값이 바뀔 때, SPEC 내의 PC를 직접 변경하는 경우의 업데이트 비용(UCOST)을 계산한다. 이때 UCOST는 SPEC에서 변경될 셀의 개수로 정의한다. 만약 UCOST가 정해진 일정한 값 이하이면 SPEC 내의 해당되는 PC를 직접 업데이트 하고 그 이상이면 변경된 셀을 델타큐브에 저장한다. 델타큐브의 크기는 삽입되는 셀의 개수가 증가함에 따라서 점차 증가한다. 델타큐브의 크기가 너무 커지면, 검색과 업데이트의 비용이 커진다. 그러므로 응용의 종류에 따라 매주, 매달 혹은 정해진 임계치 이상이 되면 델타큐브에 저장된 업데이트 정보들을 SPEC에 반영하기 위해 SPEC을 새로 구축하는 작업이 필요하다.

2.1 프리픽스 섬 방법

이 절에서 우리는 우리가 제안한 방법에 밀접한 관계가 있는 프리픽스 섬 큐브에 관한 배경 정보를 소개한다. 프리픽스 섬 방법에서는 데이타 큐브 C와 같은 크기의 PC가 사용되며 PC에는 C로부터 미리 계산된 다양한 프리픽스 섬(prefix-sum)을 저장한다. PC의 각 셀은 데이타 큐브의 각 셀을 포함한 그 셀까지의 모든 합을 저장한다. 그림 1은 6×8 데이타 큐브와 PC를 보여준다.

$PC[4,6]$는 $C[0,0]$에서 $C[4,6]$범위내의 모든 셀을 합한 값을 가진다. 따라서, 데이타 큐브 C의 전체의 합은 마지막 셀 $PC[5,7]$에서 찾을 수있다. $D = \{1,2,\cdots,d\}$를 차원의 집합을 나타내고 n_i가 각차원에 있는 셀의 수를 나타낸다고 가정하자. Ho 등[9]은 미리 계산된 프리픽스 섬을 저장하기 위해 $N=\prod_{i=1}^{d} n$개의 부가적인 셀을 필요로 하는 간단한 방법을 제안하였는데 이 방법은 임의의 영역 합을 구하기 위해서 2^d의 적절한 프리픽스 섬을 사용하여 2^d-1의 스텝만으로 계산 가능하다.

명시적으로, $0 \le x_i < n_i$이고 $i \in D$인 모든 x_i와 i에 대하여 $PC[x_1, x_2, \cdots, x_d] = Sum(0{:}x_1, 0{:}x_2, \cdots, 0{:}x_d) = \sum_{i_1=0}^{x_1} \sum_{i_2=0}^{x_2} \cdots \sum_{i_d=0}^{x_d} A[i_1, i_2, \cdots, i_d]$이다.

예를 들면, d가 2일때, 우리는 $0 \le x < n_1$이고 $0 \le y < n_2$에 대하여 $PC[x,y] = Sum(0{:}x, 0{:}y) = \sum_{i=0}^{x} \sum_{j=0}^{y} A[i, j]$이다.

그림 1은 2차원에 대한 데이타 큐브 $C[x_1, x_2]$와 그에 대응하는 $PC[x_1, x_2]$를 보여주는 예이다. 프리픽스 섬 방법은 매우 강력하여 데이타 큐브의 크기에 상관없이 상수시간에 영역-합 질의를 처리 할 수 있다.

2.2 SPEC 의 구축

이 절에서는 PC에 대한 기본 사항들을 소개하고 SPEC을 구축하는 알고리즘에 대해서 논의한다. $D = \{1, 2, \cdots, d\}$를 차원의 집합이라고 하고 n_i를 i번째 차원의 cardinality라고 하자. 그러면 d-차원의 데이타 큐브를 $C[0{:}n_1\text{-}1, \cdots, 0{:}n_d\text{-}1]$로써 표시할 수 있다. d-차원의 데이타 큐브에서 영역 합 질의를 계산하는 문제는 다음과 같은 식으로 공식화할 수 있다.

$$Sum(l_1{:}h_1, l_2{:}h_2, \cdots, l_d{:}h_d) = \sum_{i_1=l_1}^{h_1} \wedge \sum_{i_1=l_d}^{h_d} C[i_1, \cdots, i_d]$$

모든 $j \in D$에 대하여 범위 인자인 l_j , h_j는 사용자가 지정하는데 보통은 미리 알 수가 없다. 앞으로 d차원의 PC를 $PC[0{:}n_1\text{-}1, \cdots, 0{:}n_d\text{-}1]$으로 표시한다. PC는 C의 미리 계산된 프리픽스-섬을 저장하는데 사용된다. 모든 $0 \le x_i \le n_j$와 모든 $j \in D$에 대해서 다음 식에 주어진 대로 프리픽스-섬을 미리 계산할 것이다.

$$PC[x_1, x_2, \cdots, x_d] = Sum(0{:}x_1, 0{:}x_2, \cdots, 0{:}x_d) = \sum_{i_1=0}^{x_1} \sum_{i_2=0}^{x_2} \cdots \sum_{i_d=0}^{x_d} C[i_1, \cdots, i_d] \quad (1)$$

Ho 등은 d차원의 영역-합을 계산할 때에, 2^d개의 셀 값을 가지고 2^d - 1단계 만에 계산하도록, $N=\prod_{i=1}^{d} n_i$개의 셀에 프리픽스-섬을 미리 계산해 놓는 방법을 제시하였다. 보조정리 2.2는 C에 대한 임의의 영역-합 질의가 어떻게 PC의 2^d개의 셀값을 가지고 계산되는지를 설명한다. 식 2의 왼편은 C의 영역-합을 나타내고, 오른편은 PC의 원소인 2^d개의 덧셈 항으로 구성되는데,

Index	0	1	2	3	4	5	6	7
0	4	5	2	8	3	7	5	6
1	2	1	5	3	7	2	4	2
2	5	3	9	3	4	7	1	3
3	3	5	6	1	8	5	1	6
4	3	2	1	4	7	8	6	4
5	6	2	2	6	1	9	5	2

(a) 데이타 큐브 C

Index	0	1	2	3	4	5	6	7
0	4	9	11	19	22	29	34	40
1	6	12	19	30	40	49	58	66
2	11	20	36	50	64	80	90	101
3	14	28	50	65	87	108	119	136
4	17	33	56	75	104	133	150	171
5	23	41	66	91	121	159	181	204

(b) PC

그림 1 6×8 데이타 큐브와 프리픽스 섬 큐브 PC

각 항의 부호는 모든 $s(j)$의 곱으로 구해진다. 기호 표시의 편의를 위해서 어떤 $j \in D$에 대해서 $x_j = -1$이라면 $PC[x_1, x_2, \cdots, x_d] = 0$ 라고 놓는다.

보조정리 2.2 [1] 모든 $j \in D$에 대해서 $s(j) = \begin{cases} i, & \text{if } x_j = h_j \\ -1, & \text{if } x_j = l_j - 1 \end{cases}$라고 하자. 그러면 모든 $j \in D$에 대해

$$Sum(l_1{:}h_1, l_2{:}h_2, \cdots, l_d{:}h_d) = \sum_{\forall x_j \in [l_j-1, h_j]} \left\{ \left(\prod_{j=1}^{d} s(j) \right) * PC[x_1, x_2, \cdots, x_d] \right\} \quad (2)$$

SPEC은 밀도가 높은 서브큐브로부터 만들어진 PC들의 집합이다. $SPEC_s = \{PC_k \mid k=1, 2, \cdots, m\}$이라고 하고 m은 SPEC내 서브큐브의 개수라고 하자. k번째 PC인 PC_k는 k번째 서브큐브 $C[pl_1{:}ph_1, \cdots, pl_d{:}ph_d]$로부터 만들어지며 PC_k를 만드는 식은 다음과 같다.

$0 \le pl_i \le x_i \le ph_i \le n_i$인 모든 x_i와 모든 $i \in D$에 대해서 $PC_k[x_1, x_2, \cdots, x_d]$는 다음과 같이 계산되어진다.

$$PC_k[x_1, x_2, \cdots, x_d] = Sum(pl_1{:}x_1, pl_2{:}x_2, \cdots, pl_d{:}x_d) = \sum_{i_1=pl_1}^{x_1} \sum_{i_2=pl_2}^{x_2} \cdots \sum_{i_d=pl_d}^{x_d} C[i_1, \cdots, i_d] \quad (3)$$

예제 2.3 그림 2(a)에서 보여준 8×8인 데이타큐브를 생각하자. 데이타 큐브는 희박하고 uniform하게 분포되어 있지 않다. 밀도가 높은 서브큐브를 찾는 과정에서 5개의 서브큐브가 발견되어지고 각각 서브큐브에 대해서 PC를 만들고 그림 2(b)에 보이는 것과 같이 SPEC을 구축한다.

보조정리 2.4는 PC_k에서 영역-합 질의를 어떻게 2^d개의 PC_k원소들로부터 계산하는지를 보여준다. 식 (4)의 왼편은 PC_k에 대한 C의 영역-합을 나타내고, 오른편은 PC_k의 원소인 2^d개의 덧셈 항으로 구성되는데, 각 항의 부호는 모든 $s(j)$의 곱으로 구해진다. 기호 표시의 편의를 위해서 어떤 $j \in D$에 대해서 $x_j < pl_j$ 이라면 $PC[x_1, x_2, \cdots, x_d] = 0$라고 놓는다.

보조정리 2.4 모든 $j \in D$에 대해서 $s(j) = \begin{cases} 1, & \text{if } x_j = h_j \text{ or } x_j = ph_j \\ -1, & \text{if } x_j = l_j - 1 \end{cases}$라고 하자. 그러면 모든 $j \in D$에 대해 X_j와 $Sum_k(l_1{:}h_1, l_2{:}h_2, \cdots, l_d{:}h_d)$는 다음과 같다.

$$X_j = \begin{cases} \{ph_j\}, & \text{if } l_j \le pl_j \le ph_j \le h_j \\ \{h_j\}, & \text{if } l_j \le pl_j \le h_j < ph_j \\ \{l_j - 1, h_j\}, & \text{if } pl_j < l_j \le h_j \le ph_j \\ \{l_j - 1, ph_j\}, & \text{if } pl_j < l_j \le ph_j < h_j \end{cases} \text{ 이고,}$$

$$Sum_k(l_1{:}h_1, l_2{:}h_2, \cdots, l_d{:}h_d) = \sum_{\forall x_j \in X_j} \left\{ \left(\prod_{j=1}^{d} s(j) \right) * PC_k[x_1, x_2, \cdots, x_d] \right\} \quad (4)$$

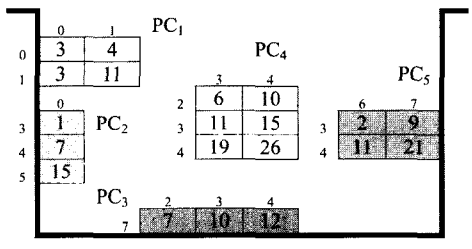
이다.

2.3 델타큐브 (△Cube)

델타큐브는 데이타 큐브의 변경된 셀과 아웃라이어 셀들만을 포함한다. 그러므로, 델타큐브는 매우 희박하다. 과학계산 분야에서 2-3차원의 희박행렬을 적절한 희박 자료 구조를 사용하여 표현하는 상당한 연구가 행해졌다. 그러나 고차원 희박 자료구조는 과학 영역에서 많은 주목을 받지 못했다[9]. 최근에 chun등 [4]은 동적 OLAP 환경에서 업데이트를 효율적으로 관리하기위해 '델타트리'를 제안했다. 이러한 방법의 기본 아이디어는 데이타 큐브의 변경된 부분들을 별도로 델타트리에 저장하여 관리하자는 것이다. 이것은 런-타임시 업데이트 비용을 엄청나게 줄여준다. 델타트리의 생성과정은 R*-tree[10] 와 동일하다. 데이타 큐브셀이 바뀔 때마다 데이타 큐브셀의 변경된 부분(△)과 그것의 공간적인 위치가 트리에 저장된다. 공간적으로 서로 근접한 변경된 셀들은 대응되는 MBR(minimum bounding rectangle)로 클러스터링된다. 말단 노드는 변경된 셀의 △값과 위치 인덱스를 포함한다. 중간노드는 자식노드의 주소, 자식노드를 감싸는 MBR들과 하위 서브트리 노드의 모든 △값의 합인 Σ△값을 저장한다. 업데이트를 효율적으로 관리하기위해서 델타트리로부터 개념과 구조를 도입하여 델타큐브를 구축한다. 델타큐브는 초기에 데이타 큐브의 아웃라이어셀을 포함한다는 점에서 델타트리와 다르다. 영역-합 질의를 수행하기위해 델타큐브를 검색할 때 델타큐브에서 질의와 무관한 MBR들은 효율적으로

Index	0	1	2	3	4	5	6	7
0	3	1				1		
1		7	5					6
2				6	4			
3	1	8		5			2	7
4	6			8	3		9	3
5	8							
6		4				7		
7	3		7	3	2			3

(a) 데이타큐브 C



(b) SPEC

그림 2 SPEC의 기본적 개념

제거된다.

예제 2.5 그림 3은 델타큐브의 기본 구조를 나타낸다. 그림 3(a)에서 보여지는 것처럼 초기에 델타큐브는 데이타 큐브의 아웃라이어셀들을 포함한다. 즉, 델타큐브의 최하위에있는 각 MBR은 이러한 아웃라이어 셀들을 포함한다. 그림 3(b)는 델타큐브의 트리 구조를 보여준다.

2.4 질의 및 업데이트

영역-합 질의 Q의 질의 영역이 $(l_1{:}h_1, l_2{:}h_2, \cdots, l_d{:}h_d)$와 같이 주어질 때 SPEC과 델타큐브를 이용하여 질의 Q에 대한 답을 얻을 수 있다. $Sum(Q)$를 질의 Q의 답을 돌려주는 함수, $SPEC_Sum(Q)$는 SPEC내의 PC들로부터 계산되는 함수이고 $\triangle_Sum(Q)$는 델타큐브로부터 계산되는 함수라고 가정한다.

$sum_k(Q)$를 SPEC내 k번째 PC로부터 계산된 값을 돌려주는 함수라고하면 이는 보조정리 2.4로부터 쉽게 구해진다. 일반적으로 $Spec_Sum(Q)$값을 얻기 위해 영역 합 질의를 계산하는데 필요한 PC의 수를 m개라고 가정하면 영역-합 질의 Q의 답은 다음과 같다.

$$Sum(Q) = Spec\_Sum(Q) + \triangle\_Sum(Q) = \sum_{k=1}^{m} Sum_k(Q) + \triangle\_Sum(Q)$$

예제 2.6 그림 4에 보이는 것과 같이 영역-합 질의 Q즉, $Sum(4{:}7, 2{:}6)$가 주어질 때 (점선박스) SPEC과 델타큐브를 이용하여 Q의 답을 얻을 수가 있다. 그림 4-(a)에 보여지는 것과 같이 PC_4와 PC_5가 질의 Q의 답을 구하는데 관여하는 PC 들이다. 따라서, Q의 답은 다음과 같이 구해진다.

$Sum(4{:}7, 2{:}6) = PC_4[4,4] - PC_4[3,4] + PC_5[7,4] + \triangle\_Sum(4{:}7, 2{:}6) = 26 - 19 + 21 + 7 = 35.$

최근의 동적 OLAP환경에서 데이타 큐브셀들은 자주 변경이되어진다. 한 셀이 바뀔 때마다 업데이트 비용(UCOST)를 계산한다. 여기서 UCOST는 SPEC내 해당하는 PC를 업데이트하기위해 접근되는 셀의 수이다. 만일 UCOST가 주어진 임계시간 보다 작으면 직접 SPEC내 해당하는 PC를 직접 업데이트 하고 크거나 같으면 변경된 셀을 델타큐브에 저장하여 관리한다.

정의 2.7 업데이트 비용(UCOST)

$C[x_1, x_2, \cdots, x_d]$를 데이타 큐브에서 변경된 셀이고 $PC_k[pl_1{:}ph_1, \cdots, pl_d{:}ph_d]$는 SPEC 내에 있는 k번째 PC라고 가정하자. 셀의 변경에 따른 업데이트 비용은 PC_k를 업데이트 하기 위해 접근되어지는 셀들의 수이다.

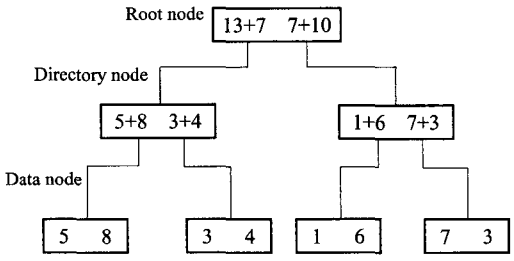
따라서 업데이트 비용은 다음과 같이 정의되어진다.

$$UCOST_k(C[x_1, x_2, \cdots, x_d]) = \begin{cases} \prod_{i=1}^{d}(ph_i - x_i + 1), & \text{iff } pl_i \le x_i \le ph_i, \\ 0, & \text{Otherwise} \end{cases}$$

3. 서브큐브 찾기

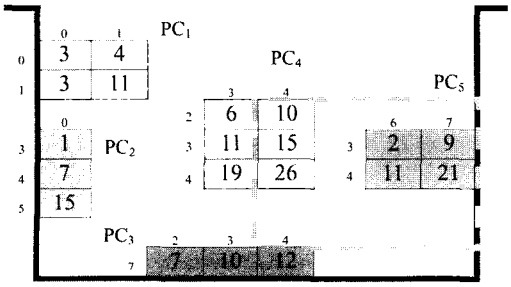
Index	0	1	2	3	4	5	6	7
0						1		
1			5					6
2								
3		8						
4								
5								
6		4				7		
7	3							3

(a) 델타큐브



(b) 델타큐브의 트리 구조

그림 3 델타큐브의 기본 개념



(a) SPEC

Index	0	1	2	3	4	5	6	7
0						1		
1			5					6
2								
3		8						
4								
5								
6		4				7		
7	3							3

(b) 델타큐브

그림 4 SPEC과 델타큐브를 이용한 영역-합 질의의 예제

이 장에서는 밀도 함수에 근거하여 희박한 대용량 데이타 큐브에서 조밀한 서브큐브를 찾는 방법에 관하여 논의한다. 데이타베이스 분야에서 다양한 클러스터링 방법들이 연구되었다[11-15]. 그러나 조밀한 서브큐브를 찾는 방법은 그 형태가 하이퍼사각형으로 제한된다는 면에서 기존의 클러스터링 기법과는 다른 방식으로 다루어 져야 한다. 각 차원에서 조밀한 구간을 찾고, 그것에 의거하여 조밀한 서브큐브를 구축하는 효과적인 알고리즘을 제시한다. 희박한 데이타 큐브 C는 여러 조밀한 영역을 가질 수 있고, 이 영역들은 서브큐브 $SC[l_1:h_1, \cdots, l_d:h_d]$로 표현될 수 있다. 여기에서, l_i과 h_i는 각각 하한과 상한이며 $0 \le l_i \le h_i \le n_i-1$이 된다. 따라서, 조밀한 서브큐브를 찾는 문제는 다음과 같이 정형화될 수 있다.

Given : d-차원의 데이타 큐브$C[0:n_1-1, \cdots, 0:n_d-1]$, 큐브당 최소 셀 수 $minCells$, 밀도 임계값 δ_1, δ_2

Target : 주어진 조건을 만족하는 조밀한 서브큐브를 찾으시오.

입력 변수 δ_1은 각 차원에서 조밀한 구간을 결정하기 위해 사용되고, δ_2는 생성된 서브큐브의 병합 및 수축을 제어하기 위하여 사용된다. 입력 변수 $minCells$은 델타큐브에 저장될 주변점을 결정하기 위해 사용된다. 서브큐브가 이것보다 적은 수의 셀을 가지면 그 안에 있는 값을 가지고 있는 셀들은 아웃라이어로 취급된다. 이 값은 애플리케이션에 따라 경험적으로 결정된다. 너무 작은 값은 중요하지 않은 서브큐브를 찾게 만들어 메모리 효율을 저하시킨다. 반대로 너무 큰 값은 의미있는 서브큐브를 발견해내지 못하게 한다.

3.1 제안한 방법의 개요

그림 5에 제안한 방법의 개요가 도시되어 있다. 먼저 값이 있는 셀들을 표시하고, 이 셀들을 각 차원마다 준비된 일 차원 배열에 사상시켜 각 차원에서의 조밀한 구간을 찾아낸다.

조밀한 구간에 의거하여 초기 서브큐브를 생성한다. 생성된 각 서브큐브에 대하여 다시 조밀한 구간을 찾아내고 하고 이 구간에 의거하여 조밀한 서브큐브를 생성한다. 이러한 단계가 서브큐브가 충분히 조밀해질 때까지 반복된다. 다음에 이전 단계에서 생성된 후보 서브큐브들이 주어진 밀도 임계 값에 대하여 정제 단계를 거친다. 서브큐브 확장 단계에서 근접하게 위치한 서브큐브들이 병합되며, 서브큐브 축소 단계에서는 후보 서브큐브들의 희박한 표면이 제거된다.

3.2 값이 있는 셀의 마킹

데이타 큐브의 각 셀은 측정 속성 값을 갖으며, 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 6은 2차원의 16×16 데이타 큐브 C를 나타내며, 여기에서 33개 셀이 값을 가지고 있다.

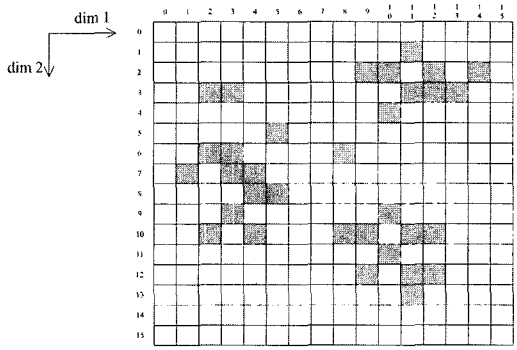


그림 6 2차원의 16×16 데이타 큐브C

3.3 각 차원에서 조밀 구간 찾기

각 차원의 조밀 구간을 찾기 위한 히스토그램 평활화에 기초한 알고리즘(*histogram-flattening based algorithm*)을 소개한다. 각 차원마다 크기가 n_i인 일 차원 배열을 유지하며, 각 빈(bin)은 그 빈에 해당하는 데이타 큐브 내의 값이 있는 셀의 수를 포함한다. 그림 6의 예에서, 차원 1의 배열은 16개의 빈을 가지고 있고, 값이 있는 셀은 해당 빈에 투영된다. 밀도 임계 값 δ_1을 초과하는 빈은 조밀한 빈으로 간주된다. 그림 6에서 δ_1 = 1일 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 기법은 히스토그램 내의 각 빈의 값들의 차를 유연하게 하기 위하여 사용된다.

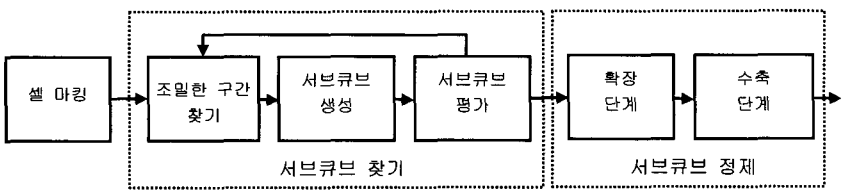


그림 5 주어진 데이타 큐브에서 조밀한 서브큐브를 찾는 방법의 개요

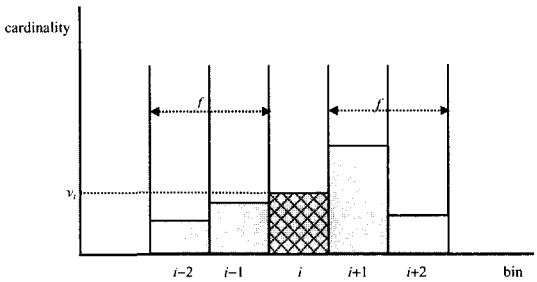


그림 7 히스토그램 평활화 기법

그림 7은 한 차원에 관한 히스토그램을 보여준다. i번째 빈이 값 v_i를 갖는다고 하자. v_i는 마크된 셀의 개수를 나타내며, 각각의 값이 있는 셀을 해당 차원에 투영함으로써 구해진다. 빈 i의 값은 v_i로 정해지는 것이 아니라 이웃 빈을 고려하여 히스토그램 평활화에 의해 구해진다. f를 평활화 요소라 하자. f = 0이면 평활화가 일어나지 않는다. f = 1인 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}) / 3으로 구해진다. 유사하게 f = 2일 때는 4개의 이웃이 참여한다. 따라서, v_i' = (v_{i-2} + v_{i-1} + v_i + v_{i+1}+ v_{i+2}) / 5이 된다. 일반적으로 k개의 빈을 가지는 구간 [l:h]에 대한 히스토그램 H에서, 빈 i의 값은 v_i'는 다음의 식으로 계산된다.

$$v_i' = \frac{1}{2f+1} \cdot \sum_{t=i-f}^{i+f} v_t$$

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=2일 때, 첫 번째 빈의 v_i'값은 v_1' = (v_1 + v_2 + v_3) / 3이 되고, 두 번째 빈의 값 v_2'는 v_2' = (v_1 + v_2 + v_3 + v_4) / 4이 된다. 히스토그램의 모든 빈에 대한 평활화 값을 구한 후, 각 값은 밀도 임계 값 δ_1에 대하여 그 값을 포함하고 있는 빈이 조밀한가를 판단하기 위하여 검사된다. δ_1보다 큰 값을 갖는 빈은 조밀한 빈으로 간주된다. 조밀한 빈만을 추출하여 조밀 구간을 구성한다. 위의 예에서 f = 0, 1, 2일 때 차원 1에 대한 히스토그램이 표 2에 나타나 있다. f = 1, δ_1 = 1일 때 조밀 구간 [1:5]와 [8:13]을 구할 수 있고, f = 2, δ_1 = 1일 때 조밀 구간 [1:14]를 구할 수 있으며, f = 2, δ_1 = 2일 때, 조밀 구간 [1:4]와 [9:13]을 구할 수 있다.

3.4 후보 서브큐브의 구성

각 차원의 조밀 구간을찾은 후, 그 구간에 의거하여 서브큐브를 구성한다. f = 1일 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 8(a)와 같이 두 개의 초기 서브큐브, SC_1[1:5, 1:13]과 SC_2[8:14, 1:13]를 구성할 수 있다. 그러나, 이 서브큐브들은 희박한 영역을 포함할 수 있다. 따라서, 각 서브큐브의 각 차원에 대하여 조밀 구간을 찾는 과정을 다시 수행한다. 서브큐브 SC_1[1:5, 1:13]에 대하여, 차원 1에서 조밀 구간 [1:5]를 구하고, 차원 2에서 두 개의 조밀 구간 [3:3]과 [5:10]를 구함으로써, 두 개의 서브큐브 SC_{11}[1:5, 3]과 SC_{12}[1:5, 5:10]을 구성한다. 유사하게 서브큐브 C_2[8:14, 1:13]에 대하여 차원 1에서 조밀 구간 [8:14]를 구하고, 차원 2에서 세 개의 조밀 구간 [1:4], [6:6], 그리고 [9:13]을 구하여 세 개의 서브큐브 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 8(b)와 같이 5개의 후보 서브큐브를 구할 수 있다.

3.5 서브큐브의 정제단계

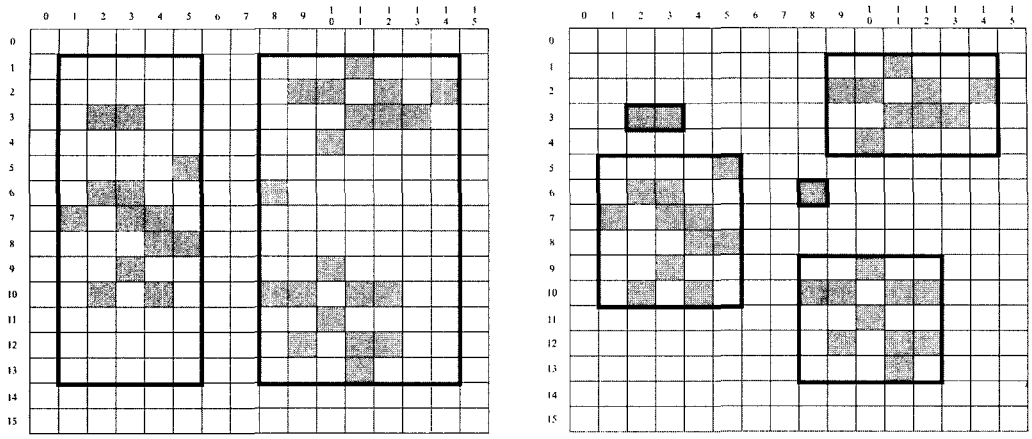
반복 처리에서 생성된 후보 서브큐브들은 큐브의 밀도에 근거하여 정제 단계를 거친다. 큐브 C의 밀도 $density(C)$는 조밀한 셀의 수 $numDenseCells(C)$를 전체 셀의 수 $numCells(C)$로 나눈 값으로 정의된다. 그러면, 앞의 예에서 큐브 C의 밀도는 $density(C)$ = 33/(16*16) = 0.129가 된다. 큐브 C로부터 k개의 서브큐브 SC_1, SC_2, …, SC_k를 찾아 냈다면, 다음의 식으로 서브큐브들의 평균 밀도를 계산할 수 있다.

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

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

표 2 f = 1, 2, 3일 때, 차원 1에 대한 히스토그램

f \ Bin	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
f=0	0	1	3	4	3	2	0	0	2	3	4	5	4	1	1	0
f=1	0.50	1.33	2.67	3.33	3.00	1.67	0.67	0.67	1.67	3.00	4.00	4.33	3.33	2.00	0.67	0.50
f=2	1.33	2.00	2.20	2.60	2.40	1.80	1.40	1.40	1.80	2.80	3.60	3.40	3.00	2.20	1.50	0.67



(a) (b)

그림 8 (a) f = 1일 때 생성된 두 개의 서브큐브, SC1[1:5, 1:13] 와 SC2[8:14, 1:13], (b) 반복 처리에서 생성된 최종5개의 서브큐브 SC111[2:3, 3], SC121[1:5, 5:10], SC211[9:14, 1:4], SC221[8, 6], SC231[8:12, 9:13].

정의 3.1 (*병합 연산자* ⊕). 두 개의 d차원 큐브, $A[l_{1,A}:h_{1,A}, ..., l_{d,A}:h_{d,A}]$와 $B[l_{1,B}:h_{1,B}, ..., l_{d,B}:h_{d,B}]$가 병합되는 경우, 병합 연산자 ⊕는 A⊕B = $C[l_{1,C}:h_{1,C}, ..., l_{d,C}:h_{d,C}]$로써 정의되며, 이때 $l_{i,C} = min(l_{i,A}, l_{i,B})$, $h_{i,C} = max(h_{i,A}, h_{i,B})$, 그리고 i = 1, 2, ⋯, d이다.

큐브 B가 큐브 A에 병합될 때, 병합에 의하여 증가된 셀의 $numCells(A⊕B)-numCells(A)$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((A,B)는 다음의 식으로 표시된다.

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

앞의 예에서 서브큐브 SC111[2:3, 3]이 서브큐브 SC_{121}[1:5, 5:10]에 병합되는 경우를 고려해보자. 정의 4.1의 병합 연산 SC111(SC121은 병합된 큐브 SC([1:5, 3:10]을 생성한다. 병합된 큐브의 확장된 부분의 밀도 density((AA,B)는 2/10 = 0.20이 된다.

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

$$density(\triangle_{A,B}) \geq \delta_2$$

밀도 임계 값 δ_2는 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, δ_2 = $density(A)$가 된다. 두 번째 방법은 이전 단계에서 구한 서브큐브들의 평균 밀도를 선택할 수 있다. 앞의 예에서, SC_{111}[2:3, 3]을 다른 서브큐브에 병합하는 경우를 고려해보자. $density(\triangle)$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. 첫번째 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 $density(\triangle)$가 각각 0.37, 0.38, 1.00, 0.40이 되고, 이 값들은 모두 병합 조건을 만족시키지 못하므로, 어떠한 서브큐브와의 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 서브큐브 C_{121}'[1:5, 3:10]를 생성하게 된다.

수축 단계 : 확장 단계에서 병합되어 생성된 서브큐브는 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$C[l_1:h_1, ⋯, l_d:h_d]$는 $2d$개의 표면($S_1[l_1:h_1, l_2:h_2, ⋯, l_d]$, ⋯, $S_{2d}[h_1, l_2:h_2, ⋯, l_d:h_d]$)을 갖는다. 이러한 표면을 표면 슬라이스(*surface slices*)라 부른다. 수축 단계에서는 병합 과정에서 생성된 각 표면 슬라이스를 검사하여, 어떤 슬라이스의 밀도가 δ_2보다 낮으면 그 슬라이스를 큐브로부터 제거한다. 예를 들어, 두 서브큐브를 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 제

거되며, 결과적으로 수축된 서브큐브 C_{121}'[2:5, 3:10]을 생성하게 된다.

또한, 최종으로 생성된 서브큐브들은 한 큐브가 가져야 하는 최소 셀의 수 *minCell*에 대하여 검사 과정을 거치게 된다. 이 값보다 작은 서브큐브들은 제거되며, 이러한 서브큐브 내의 값이 있는 셀들은 아웃라이어로 간주되어 델타큐브에 저장되게 된다.

4. 실 험

본 논문에서 제안된 방법의 효과와 효율성을 보여주기 위해서, 여러 차원의 다양한 분포의 실험 데이타를 사용하여 다양하게 실험하였다. 실험은 사용자의 영역 질의의 실행 시간과 데이타 큐브에서 필요한 저장공간의 양을 기준으로 효율성을 측정하였다. 본 실험은 512M 메인 메모리와 80G 하드 디스크와 Pentium 4 1.7GHz의 PC에서 수행되었다. 다음절에 실험의 준비 내용과 결과의 분석을 보여준다.

4.1 실험 준비

실험을 위해서 데이타 큐브 안에 인위적으로 여러 개의 밀도가 큰 서브큐브를 만들었다. 서브큐브 밖의 영역은 밀도가 낮도록 했다. 이러한 데이타 큐브에서 3절에서 설명된 서브큐브를 찾는 방법을 적용하여 서브큐브들을 발견해 내었다. 데이타 큐브를 생성할 때 사용한 인자들은 다음과 같다. 차원 d, 데이타 큐브의 크기 s, 영이 아닌 셀의 개수 z, 표 3은 인자들의 실제 값을 보여준다. 실험은 편의상 2, 3, 4, 5차원에서 수행되었다. 그러나 본 방법은 데이타의 차원에 제한을 두지 않는다. 각 차원별로 10개씩 모두 40개의 데이타 큐브를 만들었다.

3절에서 소개한 데이타큐브에서 서브큐브를 찾는 알고리즘에는 다양한 인수가 적용된다. 그 중에서 평활화 요소(f)로서 2가 적용되었다. 따라서 4개의 인접한 빈의 밀도로써 빈의 밀도를 계산하게 한다. 서브큐브를 찾아내는 동안 각 차원에서 밀도가 높은 구간을 찾게 되는데, 각각의 차원에서 히스토그램 빈들의 평균값을 밀도 경계치 δ_1으로 사용하였다. 서브큐브들을 합하고 축소시키는데 사용되는 밀도 경계치 δ_2로는 데이타큐브의 밀도가 사용되었다. 서브큐브안의 셀의 최소개수인 *minCells*로서 16을 적용하였다. 16개 이하의 셀을 가진 서브큐브에 있는 모든 영이 아닌 셀들의 데이타는 아웃라이어로 간주되고 델타큐브에 저장된다.

질의의 수행 성능을 평가하기 위해 여러가지 크기의 질의를 사용했다. 데이타 큐브의 각각의 차원의 길이를 기준으로하여 사각형 형태 질의의 한변의 길이가 50%, 40%, 30%, 20%, 10%되도록 질의를 만들었다. 각각의 데이타큐브에서 크기가 다른 각각의 질의마다 10개씩의 질의를 만들어 한가지 차원에서 500개의 질의를 만들었고 그 실행 결과를 평균하였다.

표 3 데이타큐브를 만드는데 사용된 인자

d	s	z
2	1000×1000	4000
3	250×100×50	10000
4	150×60×50×30	30000
5	100×50×40×30×20	90000

4.2 실험 결과

4.2.1 저장공간 감소 비율

제안된 방법의 가장 중요한 장점은 전체 서브큐브의 저장공간량이 데이타큐브의 저장공간량에 비해서 훨씬 작아서 저장공간 필요량이 현저히 감소한다는 것이다. 만들어진 서브큐브들은 SPEC에 저장된다. 그림 9는 차원이 증가함에 따라서 PC의 저장공간 필요량이 급격히 증가하는 반면, SPEC의 저장공간 필요량은 서서히 증가하는 것을 보여주고 있다. 그림 10은 PC대 SPEC의 저장공간 필요량의 차원에 따른 비율을 보여주고 있다. 그림에서 차원이 증가함에 따라서 그 비율이 감소한다는 것을 알 수 있다. 이것은 차원이 높아지면 저장공간 필요량의 감소 효과가 커지는 것을 의미한다. 실험결과는 SPEC이 PC저장공간의 약 82-93%를 감소시킴을 보여준다.

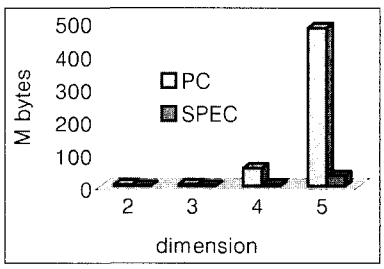


그림 9 SPEC과 PC의 저장공간 필요량 비교

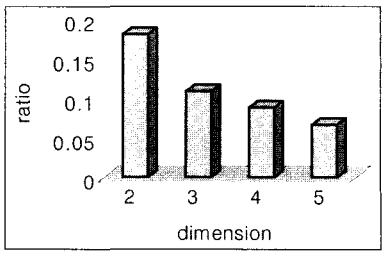
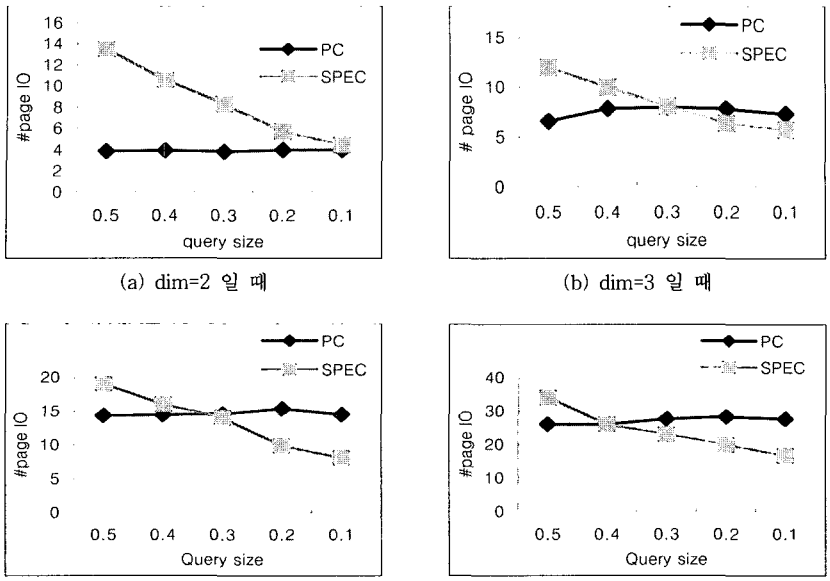


그림 10 PC에 대한 SPEC의 저장공간 필요량의 비율



(a) dim=2 일 때 (b) dim=3 일 때

(c) dim=4 일 때 (d) dim=5 일 때

그림 11 데이타 큐브의 차원이 각각 2~5 차원 일 때 페이지 I/O 비교

4.2.2 질의 성능

PC와 SPEC의 저장공간은 매우 커므로 PC와 SPEC를 모두 디스크에 저장하였다. 따라서 질의성능을 비교하기위해 계산에 필요한 페이지 I/O의 수를 PC와 SPEC대해서 각각 구하였다. 질의 성능을 페이지 I/O회수로 측정한 본 실험에서는 2~5차원의 데이타 큐브와 각각의 큐브에 대해 여러가지 질의 크기로 30번의 질의를 수행한 후 그 결과 값을 평균하였다. 그리고 제안한 방법이 델타큐브에서의 결과도 필요로하므로 델타큐브에서의 페이지 I/O회수를 SPEC에서의 페이지 I/O회수에 더하였다. 그림 11은 SPEC을 사용할 때 질의 크기가 커지면 페이지 I/O회수가 증가하는 것을 보여준다. 이것은 질의가 커지면 질의와 겹치는 서브큐브의 수가 늘어나기 때문에 페이지 I/O회수가 증가하는 경우이다. 또한 차원이 높아질수록 큐브는 희박해져서 질의와 겹치는 서브큐브의 수 또한 줄어들게된다. 실험결과는 SPEC이 PC에 비해 저장공간을 현저하게 적게 사용하면서도 질의 수행 시간은 PC를 사용하는 경우와 비슷함을 보여준다.

5. 결 론

본 논문은 프리픽스-섬 접근법에 기초한 기존의 OLAP 방법들에서의 저장공간 오버헤드의 문제를 고려했다. 실제 OLAP 환경에서 분석가들은 사업의 추세와 기회를 찾기위해 다양한 애트리뷰트들간의 관계를 탐험하길 원한다. 이와 같은 분석을 위한 데이타 큐브는 일반적으로 고차원이되며 데이타 큐브가 고차원이 될수록 데이타 큐브는 매우 희박하게 된다. 이것이 대용량 데이타 웨어하우스에서 클러스터링 기법을 사용하여 PC의 저장공간을 획기적으로 감소하는 'SPEC'으로 불리는 새로운 기법을 제안한 동기이다. 기본 아이디어는 하나의 거대한 PC를 만드는 대신에 미리 정한 임계치를 만족하는 조밀한 서브큐브의 집합을 발견하여 다수의 작은 PC들을 만드는 것이다. 이것은 대용량 희박 데이타 큐브의 PC에 대한 저장공간 요건을 획기적으로 감소시킨다.

또한 *histogram-flattening based* 알고리즘으로 불리는 데이타 큐브의 각차원에서 조밀한 구간들을 찾는 효과적인 알고리즘을 개발했고 이 알고리즘에 기초하여 조밀한 서브큐브들을 찾는 방법을 제안했다. 다양한 데이타 셀으로 폭넓은 실험을 수행하였고 여러가지 차원의 데이타 큐브와 질의 크기들에 대해 제안한 방법의 저장공간 감소와 성능을 조사했다. 실험적인 결과는 제안된 방법이 적절한 질의 성능을 유지하면서 PC의 저장공간의 약 82-93%를 감소시킴을 보여준다. 앞으로 연구과제로 고차원 데이타 큐브를 위한 더 효율적인 클러스터링 알고리즘을 개발하는 것이다.

참 고 문 헌

[1] W. Liang, H. Wang, and M. E. Orlowska, "Range Queries in dynamic OLAP data cubes," Data & Knowledge Engineering, Vol. 34, pp. 21-38, 2000.

[2] C. Ho, R. Agrawal, N. Megido, and R. Srikant,

"Range queries in OLAP Data Cubes," ACM SIGMOD Conference, pp. 73-88, 1997.
[3] C.-Y. Chan, and Y. E. Ioannidis, "Hierarchical cubes for range-sum queries," VLDB Conference, Scotland, pp. 675-686, 1999.
[4] Seok-Ju Chun, Chin-Wan Chung, Ju-Hong Lee, and Seok-Lyong Lee, "Dynamic Update Cube for Range-Sum Queries," VLDB Conference, Italy, pp. 521-530, 2001.
[5] S. Geffner, D. Agrawal, and A. El Abbadi, "The Dynamic Data Cube, EDBT Conference," Germany, pp. 237-253, 2000.
[6] S. Geffner, D. Agrawal, A. El Abbadi, and T. Smith, "Relative prefix sums : an efficient approach for quering dynamic OLAP Data Cubes," ICDE Conference, Australia, pp. 328-335, 1999.
[7] M. Riedewald, D. Agrawal, A. E. Abbadi, and R. Pajarola, "Space-Efficient Data Cubes for Dynamic Environments," DaWaK conference, pp. 24-33, 2000.
[8] M. Riedewald, D. Agrawal, and A. E. Abbadi, "pCube : Update-Efficient Online Aggregation with Progressive Feedback and Error Bounds," SSDBM conference, pp. 95-108, 2000.
[9] S. Goil and A. Choudhary, "BESS : Sparse data storage of multi-dimensional data for OLAP and data mining," Technical report, Northwestern University, 1997.
[10] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger, "The R*-tree : an efficient and robust access method for points and rectangles," ACM SIGMOD Conference, New Jersey, pp. 322-331, 1990.
[11] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan, "Automatic subspace clustering of high dimensional data for data mining applications," ACM SIGMOD Conference, Washington, pp. 94-105, 1998.
[12] M. Ester, H. P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," KDD Conference, Oregon, pp. 226-231, 1996.
[13] R. T. Ng and J. Han, "Efficient and effective clustering methods for spatial data mining," VLDB Conference, Chile, pp. 144-155, 1994.
[14] S. Guha, R. Rastogi, and K. Shim, "CURE : An efficient clustering algorithm for large databases," ACM SIGMOD Conference, Washington, pp. 73-84, 1998.
[15] T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH : An efficient data clustering method for very large databases," ACM SIGMOD Conference, Canada, pp. 103-114, 1996.

전 석 주

2002년 한국과학기술원 정보및통신공학과 박사. 2003년 9월~2003년 12월 서강대학교 정보통신대학원 강사. 1989년 2월~1995년 9월 현대중공업 중앙연구소 연구원 1급. 2004년 4월~현재 서울교육대학교 컴퓨터교육과 전임강사. 관심분야는 데이타 마이닝, 데이타 웨어하우스와 OLAP, 멀티미디어 데이타베이스 등

이 석 룡

1984년 연세대학교 기계공학과 학사 1993년 연세대학교 산업공학과 전자계산전공 석사. 2001년 한국과학기술원(KAIST) 정보및통신공학과 박사. 1984년 1월~1995년 2월 한국IBM 소프트웨어 연구소 선임연구원. 1995년 3월~2002년 2월 안산1대학 조교수. 2002년 3월~현재 한국외국어대학교 산업정보시스템공학부 부교수관심분야는 데이타베이스, 데이타 마이닝, 멀티미디어 정보검색, 시계열데이타 처리

강 홈 근

1992년 2월 한국과학기술원 전산학과 석사. 2004년 8월 한국과학기술원 전산학과 박사. 1992년~1995년 한국전자통신연구원 연구원. 1997년~현재 우송공업대학 컴퓨터정보통신계열 조교수. 관심분야는 데이타웨어하우스, OLAP, 분산시스템

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 전산학과 부교수 및 교수. 관심분야는 XML, 멀티미디어 데이타베이스, GIS, 웹 정보검색, 객체지향 데이타베이스