고차원 멀티미디어 데이터 검색을 위한 벡터 근사 비트맵 색인 방법

박 주 현[†] · 손 대 온[††] · 낭 종 호[†††] · 주 복 규[††††]

요 약

고차원 데이터 공간에서의 효과적인 검색을 위해 최근 VA-file[1], LPC-file[2] 등과 같이 벡터 근사에 기반을 둔 필터링 색인 방법들이 연구되었다. 필터링 색인 방법은 벡터를 근사한 작은 크기의 색인 정보를 사용하여 근사 거리를 계산하고, 이를 사용하여 질의 벡터와 유사하지 않은 대부분의 벡터들을 빠른 시간 안에 검색 대상에서 제외한다. 즉, 실제 벡터 대신 근사 벡터를 읽어 디스크 I/O 시간을 줄여 전체 검색 속도를 향상시키는 것이다. 하지만 VA-file이나 LPC-file은 근사 거리를 구하는 방법이 순차 검색과 같거나 복잡하기 때문에 검색 속도 향상 효과가 그리 크지 않다는 문제점을 가지고 있다. 본 논문은 이러한 근사 거리 계산 시간을 줄이기 위하여 새로운 비트맵 색인 구조를 제안한다. 근사 거리 계산속도의 향상을 위하여, 각 객체의 값을 특성 벡터 공간상의 위치를 나타내는 비트 패턴으로 저장하고, 객체 사이의 거리를 구하는 연산은 실제 벡터 값의 연산보다 속도가 훨씬 빠른 XOR 비트 연산으로 대체한다. 실험에 의하면 본 논문이 제안하는 방법은 기존 벡터 근사 접근 방법들과 비교하여 데이터 읽기시간은 더 크지만, 계산 시간을 크게 줄임으로써 전체 검색 속도는 순차 검색의 약 4배, 기존의 방법들보다는 최대 2배의 성능이 향상되었다. 결과적으로, 데이터베이스의 속도가 충분히 빠른 경우 기존의 벡터 근사 접근법의 필터링을 위한 계산 시간을 줄임으로써 더욱 검색 성능을 향상 시킬 수 있음을 확인할 수 있다.

키워드 : 멀티미디어 검색, 고차원 색인

Vector Approximation Bitmap Indexing Method for High Dimensional Multimedia Database

Joohyoun Park[†] · Deaon Son[††] · Jongho Nang[†††] · Bokgyu Joo[††††]

ABSTRACT

Recently, the filtering approach using vector approximation such as VA-file[1] or LPC-file[2] have been proposed to support similarity search in high dimensional data space. This approach filters out many irrelevant vectors by calculating the approximate distance from a query vector using the compact approximations of vectors in database. Accordingly, the total elapsed time for similarity search is reduced because the disk I/O time is eliminated by reading the compact approximations instead of original vectors. However, the search time of the VA-file or LPC-file is not much lessened compared to the brute-force search because it requires a lot of computations for calculating the approximate distance. This paper proposes a new bitmap index structure in order to minimize the calculating time. To improve the calculating speed, a specific value of an object is saved in a bit pattern that shows a spatial position of the feature vector on a data space, and the calculation for a distance between objects is performed by the XOR bit calculation that is much faster than the real vector calculation. According to the experiment, the method that this paper suggests has shortened the total searching time to the extent of about one fourth of the sequential searching time, and to the utmost two times of the existing methods by shortening the great deal of calculating time, although this method has a longer data reading time compared to the existing vector approximation based approach. Consequently, it can be confirmed that we can improve even more the searching performance by shortening the calculating time for filtering of the existing vector approximation methods when the database speed is fast enough.

Key Words : Multimedia Retrieval, High Dimensional Indexing

1. 서 론

컴퓨터 처리 능력 향상과 디지털 카메라와 같은 디지털 미디어 생상 장비의 발달은 인터넷이나 PC에 이미지, 동영상과 같은 멀티미디어 데이터의 양을 폭발적으로 증가시키는 결과를 가져왔다. 동시에 효과적인 내용 기반 멀티미디어 검색(Content based Multimedia Retrieval)에 대한 요구가 증가하고 있으며 최근 이에 대한 연구가 널리 진행되어 오고 있다. 일반적으로 내용 기반 멀티미디어 검색은 동영상, 이미지와 같은 멀티미디어 객체 자체보다는 객체로부터 추출한 색상, 텍스쳐 등과 같은 고차원의 특성 벡터를 사용

※ 이 논문은 2006년도 두뇌한국21사업에 의하여 지원되었습니다.
† 준 회 원 : 서강대학교 대학원 컴퓨터학과 박사과정
†† 정 회 원 : LG전자 MC연구소 주임연구원
††† 정 회 원 : 서강대학교 컴퓨터학과 교수
†††† 종신회원 : 홍익대학교 컴퓨터정보통신공학과 교수
논문접수 : 2006년 3월 7일, 심사완료 : 2006년 6월 1일

하여 검색을 수행한다. 따라서 차원 증가에 따라 급격하게 색인 성능이 저하되는 소위 "Dimensionality Curse"[7, 8] 문제를 가지고 있는 R*-tree[3], X-tree[4], 그리고 SR-tree[6] 등과 같은 기존의 데이터 클러스터링 기반의 다차원 색인 기법은 내용 기반 멀티미디어 검색에 사용하기에 매우 부적합하다고 할 수 있다. 최근 이러한 "Dimensionality Curse" 문제를 해결하기 위한 연구들이 활발히 진행되고 있다.

고차원 공간에서의 검색 속도를 향상시키기 위한 연구는 크게 차원 축소법(Dimensionality Reduction Approach)[11, 12], 근사 최대 근접 이웃 접근법(Approximate Nearest Neighbor Approach)[13], 그리고 벡터 근사를 사용한 필터링 방법(Filtering Approach using Vector Approximation) [1, 2]등으로 나누어 생각할 수 있다. 차원 축소법이나 근사 최대 근접 이웃 접근법등은 빠른 시간 안에 검색 결과를 찾을 수 있으나, 검색 결과가 순차 검색 결과와 다를 수 있다는 문제점을 가지고 있다. 반면에, VA-file[1], LPC-file[2] 등과 같은 벡터 근사를 사용한 필터링 색인 방법은 정확한 검색 결과를 얻을 수 있다. 검색 속도 또한 앞의 두 방법보다는 느리지만 순차 검색 보다 좋은 성능을 보여주는 것으로 알려져 있다.

필터링 색인 방법들은 검색에 이용되는 특성 벡터들을 작은 크기의 근사 벡터로 표현한 후, 이를 검색에 이용하여 실제 특성 벡터를 읽는 I/O 시간을 줄임으로서 전체 검색 시간을 단축한다. 즉, 근사 벡터를 사용하여 계산된 근사 거리를 기준으로 질의 객체와 멀리 떨어져 있는 객체들을 검색 대상에서 제외해 실제 특성 벡터를 읽어 실제 거리를 계산해야 하는 객체의 수를 줄이는 것이다. 하지만 [1, 2]의 실험 결과를 보면 이러한 필터링 기반 색인 방법들의 검색 속도 향상 효과가 그리 크지 않음을 알 수 있다. 이러한 결과가 나타나는 이유는 I/O 시간의 줄임 폭은 크지만 근사 거리 계산 시간은 실제 거리를 계산하는 것과 같거나 오히려 증가했기 때문이다.

본 논문은 비트 연산을 통하여 근사 거리 계산 시간을 큰 폭으로 단축시킬 수 있는 필터링 색인 기반의 **벡터 근사 비트맵 색인 방법**을 제안한다. 제안한 색인 방법은 검색 집합 내의 각 객체의 특성 벡터를 벡터 공간상의 위치 정보를 표현하는 작은 크기의 비트패턴으로 만들어 색인 정보로서 저장한다. 검색은 두 단계로 구성되는데, 저장된 색인 정보를 사용하여 계산된 근사 거리를 기준으로 질의 벡터와 멀리 떨어져 있는 객체들을 검색 대상에서 제외시키는 필터링 단계와 제외되지 않은 객체를 대상으로 구성된 후보 집합으로부터 최종 검색 결과를 만들어내는 단계로 구성된다. 근사 거리는 질의 객체를 위한 비트패턴과 각 객체의 비트패턴간의 XOR연산을 사용하여 계산함으로써 필터링 시간을 큰 폭으로 단축시키는 효과를 보여주게 된다.

실험에 의하면 본 논문이 제안하는 방법은 벡터 근사 접근방법에 기반한 다른 알고리즘들과 비교하여 벡터 선택률(vector selectivity)은 근소하게 높아서 입출력 시간은 더 크지만, 필터링을 위한 계산 시간을 크게 줄임으로써 전체 수행속도 면에서는 순차 검색보다는 4배, 기존의 방법들 보다는 2배 더 빠른 성능을 보여주었다.

2. 벡터 근사 비트맵 색인을 사용한 검색

본 장에서는 데이터 근사 방법을 기본으로 하여 정확한 근접 이웃을 구함과 동시에 검색 속도를 큰 폭으로 향상 시킬 수 있는 고차원 벡터 집합을 위한 비트맵 색인 방법에 대해서 설명한다.

2.1 비트맵 색인 생성 방법

비트맵 색인은 전체 데이터 공간상에서의 특정 객체의 공간적인 위치를 그대로 반영하여 비트맵에 반영된다는데 있다. 비트맵의 정보가 곧 데이터 공간상의 위치이기 때문에 비트맵 자체만으로 데이터 공간상에서의 거리 정보를 직접 계산할 수 있게 한다.

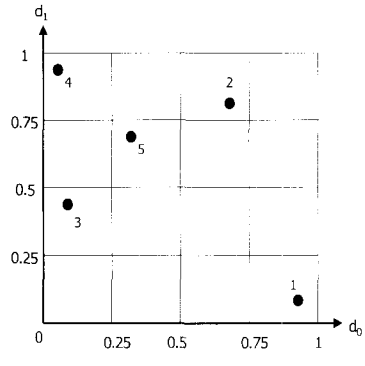
비트맵 색인을 구성하려면 먼저 모든 셀의 크기가 균등하도록 데이터 공간을 분할해야 한다. 각 차원을 b개의 구간으로 균등하게 분할한다고 가정하자. 비트맵 색인은 b개의 구간을 b개의 비트로 표현하고 이것은 하나의 차원에 대한 객체의 비트열이 된다. 비트열은 하나의 차원에 대하여 그 객체의 공간적인 위치를 표시한다. 기본적으로 모든 비트는 0(false)값을 가지고 객체가 존재하는 구간에 해당하는 비트만 1(true)의 값을 가지게 한다. 하나의 객체는 이러한 비트열을 차원의 개수만큼 가지게 된다. 따라서 하나의 비트맵은 대응하는 객체의 벡터가 각 차원마다 어느 구간에 위치하고 있는 가를 표현할 수 있게 된다. 각 차원을 b개의 구간으로 분할하고, 차원의 개수가 D이며 전체 객체의 개수가 N이라고 할 때, 전체 비트맵 색인의 크기는 $b \cdot D \cdot N$비트이다. 또한, 하나의 객체에 대한 비트맵의 크기는 $b \cdot D$비트이다. 비트맵을 생성하는 알고리즘은 (그림 1)과 같다.

```
// M_{i,j} : i번째 객체의 j차원의 값이 속한 구간의 번호
// D : 차원의 개수
// N : 객체의 개수
// b : 하나의 차원에 할당되는 비트(bit) 개수
// Bit_{i,j} : i번째 객체를 근사 하는 비트열의 j번째 비트 값 (0혹은 1)

Algorithm MakeBitmap do
    loop i to N do
        loop j to D do
            if ( M_{i,j} )
                Bit_{i,M_{i,j}} := 1
            else
                Bit_{i,j} := 0
        end
    end
end
```

(그림 1) 비트맵 생성 알고리즘의 의사 코드

예를 들어 (그림 2)와 같이 2차원 공간에 5개의 객체가 분포되어 있고, 각 차원을 4개의 구간으로 균일하게 분할하였다고 하자. 객체 1을 예를 들어 살펴보면, 차원 0에서의 위치는 4번째 구간이므로 차원 0에 대응하는 비트열은 0001이고 차원 1에서의 위치는 0번째 구간이므로 대응하는 비트열은 1000이다. 따라서 객체 1의 비트맵은 [0001 1000]이다. 이러한 방법으로 객체 2,3,4,5에 대해 적용하여 각 객체에 대한 비트맵을 생성할 수 있으며 <표 1>은 각 객체에 대한 비트맵 생성 결과를 보여준다. <표 1>에서 볼 수 있는 바와 같이 제안하는 비트맵 색인 방법의 각 객체에 대한 비트맵의 크기는 VA-file의 경우보다 크다. 만약 VA-file과 같은 크기의 비트열을 사용한다면 분할 할 수 있는 구간의 개수가 줄어들기 때문에 VA-file에 비해 벡터 선택률이 낮아질 가능성이 높아지게 된다. 이러한 벡터 선택률의 손해에도 불구하고 비트맵을 사용하는 이유는 벡터 선택률의 손해로 입는 시간보다 근사 거리 계산 시간의 단축으로 인한 이득이 더 크기 때문이다.



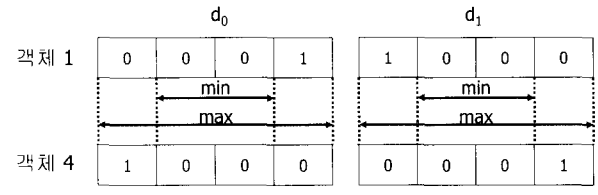
(그림 2) 데이터 분포의 예

〈표 1〉 데이터를 비트맵으로 근사화한 예

	벡터 데이터	비트맵	VA-file[1]
객체 1	0.9 0.1	0001 1000	11 00
객체 2	0.6 0.8	0010 0001	10 11
객체 3	0.1 0.4	1000 0100	00 01
객체 4	0.1 0.9	1000 0001	00 11
객체 5	0.3 0.7	0100 0010	01 10

2.2 근사 거리 예측 방법

비트맵 색인을 사용하여 근사 거리를 구하는 방법은 매우 간단하다. 우선 질의 객체를 비트맵으로 바꾼 다음 데이터베이스의 비트맵과 질의 비트맵과의 거리를 계산한다. 비트맵 간의 거리 계산은 매우 직관적이다. 예를 들어 <표 1>에서 객체 1의 비트맵 [0001 1000]과 객체 4의 비트맵 [1000 0001]사이의 근사 거리를 구해보자. 객체간의 거리는 각 차원 별로 독립적으로 이루어지므로 차원0의 거리는 [0001]과 [1000] 사이의 거리이고, 차원1의 거리는 [1000]과 [0001] 사이의 거리이다. 먼저 차원0에서의 [0001]과 [1000]의 거리는 직관적으로 두 객체가 차원0에서 존재하는 구간 사이에 2개의 구간이 존재하므로 2개 구간 거리보다는 크고 4개 구간 거리보다는 작다. 즉, (그림 3)에서 보이는 것처럼, 차원 0에서의 두 벡터의 최소 거리는 2개 구간이고 최대 거리는 4개 구간이 된다. 만약 두 객체간 거리 측정 방법이 각 차원별 차의 합인 L_1 거리 측정 방법이고 하나의 구간의 크기가 c 라면 객체 1과 객체 4 간 거리의 최소값은 $(2+2)\cdot c = 4c$ 이고, 최대값은 $(4+4)\cdot c = 8c$ 이다.



(그림 3) 비트맵 색인을 통한 근사 최대 최소 거리

〈표 2〉 데이터를 수정된 비트맵으로 표현한 예

	벡터 데이터	기존 비트맵	변환된 비트맵
객체 1	0.9 0.1	0001 1000	0001 1111
객체 2	0.6 0.8	0010 0001	0011 0001
객체 3	0.1 0.4	0001 0100	0001 0111
객체 4	0.1 0.9	1000 0001	1111 0001
객체 5	0.3 0.7	0100 0010	0111 0011

직관적으로 각 객체의 차원 단위 거리는 두 객체의 대응하는 차원의 1로 표시된 비트사이의 0으로 표시된 비트 개수를 계산하면 됨을 알 수 있다. 이러한 계산을 간단히 하기 위하여 전술한 비트맵 생성 방법을 (그림 4)와 같이 수정한다. 즉, 한 번의 XOR 연산을 사용하여 거리를 측정하기 쉽도록 각 차원의 1로 표시된 비트의 오른쪽의 나머지 비트를 1로 채우는 것이다. 예를 들어 객체4의 첫 번째 차원에 해당하는 비트열인 [1000]은 [1111]로 변환한다. 객체4의 두 번째 차원에 해당하는 비트열인 [0001]은 1의 오른쪽 비트가 없으므로 그대로 [0001]이다. 2.1절에서 예를 들었던 (그림 2)에 해당하는 비트맵을 변환한 새로운 비트맵은 <표 2>와 같다.

```
// M_{i,j} : i번째 객체의 j차원의 값이 속한 구간의 번호
// D : 차원의 개수
// N : 객체의 개수
// b : 하나의 차원에 할당되는 비트(bit) 개수
// Bit_{i,j} : i번째 객체를 근사 하는 비트열의 j번째 비트 값 (0혹은 1)

Algorithm MakeBitmap do
    loop i := 0 to N do
        loop j := 0 to D do
            if (j ≥ M_{i,j}) Bit_{i,M} := 1
            else Bit_{i,j} := 0
        end
    end
end
```

(그림 4) 수정된 비트맵 생성 알고리즘의 의사 코드

변환된 비트맵을 사용하여 앞서 예를 들었던 객체 1과 객체 4간의 근사거리를 계산해보자. 객체 1의 비트맵 [0001 1111]과 객체 4의 비트맵 [1111 0001]을 XOR 비트 연산을 수행하면 [1110 1110]이 된다. 비트 연산을 수행한 결과인 [1110 1110]의 거리는 1의 개수로 판단할 수 있다. XOR 비트 연산의 결과인 [1110 1110]의 1의 개수는 6이다. 한편, 이전의 (그림 3)의 예를 통해서 알 수 있었던 것처럼 객체 1과 객체 4의 최대 거리는 $8 \cdot c$ 이고 최소 거리는 $4 \cdot c$ 이다. 즉, XOR 비트 연산 결과의 1의 개수는 최대 거리보다 차원 수만큼 작고, 최소 거리보다 차원 수만큼 큼을 알 수 있다. 정리하면, d 차원의 임의의 두 객체의 비트맵의 XOR 연산 결과 k 를 사용하여 계산한 두 객체 간 최대 근사 거리 d_{max} 와 최소 근사 거리 d_{min} 은 다음과 같다.

$$d_{max} = (k + d) \cdot c$$
$$d_{min} = (k - d) \cdot c \quad \text{(식 1)}$$

(그림 5)는 수정된 비트맵 생성 알고리즘을 통하여 구성된 비트맵 색인을 이용하여 두 객체간의 최대 근사 거리와 최소 근사 거리를 구하는 알고리즘을 보여준다. BitTableInitialize는 한 바이트 내부의 1의 개수를 세기위한 테이블인 BitTable의 값을 계산하기위한 초기화 함수이다. BitTable의 각 원소의 값은 인덱스의 값에 해당하는 1의 개수를 저장하고 있어 한 바이트 내부의 1의 개수를 최대한 빠르게 계산할 수 있다.

```
// Bitmap_o : 객체 o를 근사화 하는 비트열
// Bitmap_q : 질의 객체 q를 근사화 하는 비트열
// BitTable[k] : k의 값에 해당하는 1의 개수를 저장한 정적
배열
global BitTable[256]
Algorithm GetBitmapDistance(Bitmap_o, Bitmap_q)do
    Bitmap_o = Bitmap_o XOR Bitmap_q
    return GetBitCount(Bitmap_o)
end

Algorithm GetBitCount(Bitmap) do
    return BitTable[Bitmap]
end

Algorithm BitTableInitialize do
    for k := 0 to 256 do
        BitTable[k] = NumOf1in(k);//i를 2진수로 변
환하였을 경우
                                    //  '1'의 개수를
계산한다.
    end
end
```

(그림 5) 객체 간 근사거리를 계산하는 알고리즘의 의사 코드

2.3 k개의 근접 이웃 검색 알고리즘

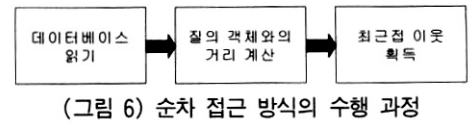
비트맵 색인을 사용한 k 개의 근접 이웃의 검색은 두 단계로 이루어진다. 첫 번째 단계에서, 모든 벡터 근사를 순차적으로 검사하고, 질의 벡터로부터 각 셀까지의 거리의 하한 경계 d_{min} 과 상한 경계 d_{max} 를 계산한다. 만약 어떤 근사(셀)에 대해, 그것의 d_{min} 이 지금까지 발견된 k번째 최소 상한 경계를 초과한다면 그 벡터를 제거할 수 있다. 이 단계의 끝에서, 최소 경계를 갖는 벡터들이 질의 벡터에 대한 k 개의 근접 이웃에 대한 후보로 결정된다. 두 번째 단계에서는 d_{min} 의 오름차순으로 실제 벡터를 읽어서 후보 집합을 걸러낸다. 이 단계에서 후보들 중에서 그 것의 d_{min} 이 k번째 근접 이웃까지의 거리와 같거나 초과하는 근사를 만나면 검색을 마칠 수 있고, 지금까지의 k 개의 근접 이웃이 검색 결과가 된다.

3. 알고리즘의 이론적 분석

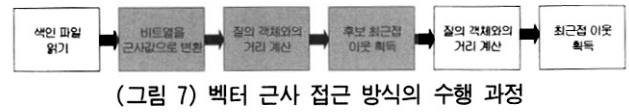
기존의 벡터 근사 접근법에서의 색인은 비트열로 저장되며 수많은 셀로 분할한 데이터 공간상의 특정한 셀의 번호를 의미한다. 벡터 근사 접근법에서는 실제 데이터가 아닌 색인 파일 만을 이용하여 질의 벡터와 유사하지 않은 대 부분의 벡터들을 필터 아웃시킨다. 이를 위해서는 질의 객체와 모든 객체와의 근사 거리를 구해야 하는데 이를 위해서 비트열을 근사 값으로 변환하는 과정이 필요하다. 즉, 하나의 객체에 대한 근사 거리는 비트열을 근사 벡터로 변환한 다음 질의 벡터와 비교하여 근사 거리를 구한다. (그림 6)은 순차 접근 방식의 수행 과정을 나타내고 (그림 7)은 기존의 벡터 근사 접근법의 수행 과정을 보여준다.

제안하는 방법은 기존 방법에서의 2번째 블록인 비트열을 근사 값으로 변환하는 과정이 필요 없다. 따라서 수행과정을 (그림 8)과 같다.

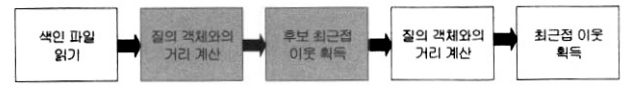
순차 검색에서 2번째 블록의 질의 객체와의 거리 계산과 비교하여 기존 방법의 3번째 블록의 거리 계산은 동일하게 전체 객체의 개수인 N회 수행되고 계산 과정은 더 복잡하므로 순차 검색의 계산 시간(T_{Linear})보다 기존 방법의 계산 시간($T_{VA\ based}$)이 더 크다. 순차 검색과 기존 방법들의 계산 시간을 수식 화하면 다음과 같다. 여기서 계산 시간은 모든 객체와 질의 객체 사이의 거리를 구하는 시간이며 근접 이



(그림 6) 순차 접근 방식의 수행 과정



(그림 7) 벡터 근사 접근 방식의 수행 과정



(그림 8) 제안하는 방법의 수행 과정

〈표 3〉 기존의 방법과 제안하는 방법의 검색 시간의 비교 (검색 시간 = 근사 거리 계산 시간 + 디스크 입출력 시간)

	근사 거리 계산 시간	전체 디스크 입출력 시간
순차 검색	$(T_{add}+T_{sub})\cdot D\cdot N$	$T_{element}\cdot D\cdot N$
VA-file	$2\cdot(T_{add}+T_{sub}+T_{Trans})\cdot D\cdot N+(T_{add}+T_{sub})\cdot m\cdot D$	$T_{approx}\cdot D\cdot N+T_{element}\cdot D\cdot m$
LPC file	$T_{LPC}=T_{VA}+(T_{mul}+T_{add}+T_{sub}+T_{cos})$	$(T_{approx}\cdot D+S_{radius}+S_{angle})\cdot N+T_{element}\cdot D\cdot m$
GC-tree	LPC-file과 동일	$(T'_{approx}\cdot D+S_{radius}+S_{angle})\cdot N+T_{element}\cdot D\cdot m$
제안하는 방법	$T_{proposed}=(T_{XOR}+T_{Count})\cdot b$	$T_{approx}\cdot D\cdot N+T_{element}\cdot D\cdot m$

웃을 검색하는 시간은 생략하였다. 먼저 순차 검색의 계산 시간(T_{Linear})은 각 차원의 값의 차의 합이므로 다음의 (식 2)와 같다.

$$T_{Linear} = (T_{add} + T_{sub})\times D \qquad (식\ 2)$$

여기서 D는 차원의 전체 개수이고 T_{add}는 덧셈을 수행하는데 걸리는 시간, T_{sub}는 뺄셈을 수행하는데 걸리는 시간, T_{mul}는 곱셈을 수행하는데 걸리는 시간을 의미한다. VA-file[1], LPC-file[2], GC-tree[5]의 계산 시간은 각각 (식 3), (식 4), (식 5)과 같다.

$$T_{VA} = 2\cdot(T_{add} + T_{sub} + T_{Trans})\cdot D \qquad (식\ 3)$$

$$T_{LPC} = T_{VA} + (T_{mul} + T_{add} + T_{sub} + T_{cos}) \qquad (식\ 4)$$

$$T_{GC-tree} = T_{LPC} \qquad (식\ 5)$$

벡터 근사 접근 방법은 근사 거리의 최대, 최소를 구해야 하므로 최소한 순차 검색보다 2배 더 크다. 또한 각 차원의 비트열을 근사 값으로 변환하는데 걸리는 시간(T_{Trans})도 필요하다. LPC file의 계산 시간(T_{LPC})은 VA-file의 계산시간(T_{VA})과 마찬가지로 벡터가 존재하는 셀의 근사 값을 구한 후 셀 내부의 지역 극좌표의 값을 계산하는 과정이 추가적으로 필요하므로 곱셈과 코사인을 수행하는 시간($T_{mul}+T_{cos}$)이 더 필요하다. 그리고 GC-tree의 계산 시간($T_{GC-tree}$)은 LPC-file의 방법을 그대로 사용하므로 T_{LPC}와 동일하다. 요컨대, 각 색인 방법들의 계산 시간이 순차 검색의 계산 시간보다 큼을 알 수 있다. 반면에 제안하는 알고리즘의 계산 시간을 수식 화하면 다음의 (식 6)과 같다.

$$T_{proposed} = (T_{XOR} + T_{Count})\cdot b \qquad (식\ 6)$$

T_{XOR}은 한 바이트의 비트열에 대하여 XOR 비트 연산을 수행하는데 걸리는 시간이고 T_{Count}는 한 바이트이 비트열 내부의 1로 셋팅된 비트의 개수를 세는데 걸리는 시간이다. b는 하나의 객체에 할당된 바이트 수이다. 기본적으로 T_{XOR}과 T_{Count}는 기존 방법의 속도 기준인 T_{add}, T_{sub} 등과 비교하여 작고, 차원의 개수와 관계없이 연속적인 바이트 단위로 연산을 수행하기 때문에 기존의 방법과 비교하여 훨씬 속도가 빠르다. <표 3>은 앞서 설명한 기존색인 방법들과 제안하는 색인 방법의 전체 검색 시간 비교 결과를 보여준다.

계산시간을 제외한 시간은 디스크의 입출력 시간이다. 전체 디스크 입출력 시간은 전체 색인파일을 읽어 들이는 시간과 필터링 후 남은 후보의 개수 m 만큼 실제 벡터 데이터를 읽어 들이는 시간으로 구분된다. 만약 색인파일의 크기가 일정하다면 m의 크기에 따라서 디스크의 입출력 시간이 크게 좌우된다. m의 크기는 데이터의 분포에 따라 조금씩 달라질 수 있지만 데이터가 균등하게 분포되어있다고 가정하면 기존의 각 방법의 m의 크기의 비비교가 가능하다. m은 곧 데이터의 개수가 일정할 경우 벡터 선택률(Vector Selectivity)이 의미하는 것과 같다. 데이터가 균일하게 분포되어있다고 가정하면 벡터 선택률은 데이터 공간을 얼마나 조밀하게 나누는가에 따라 좌우된다. 따라서 기존의 각 방법과 제안하는 방법과의 벡터 선택률은 다음 (식 7)과 같이 비교될 수 있다.

$$VS_{LPC} = VS_{GC-tree} \le VS_{VA} \le VS_{proposed} \qquad (식\ 7)$$

VA-file은 하나의 벡터에 b개의 비트가 할당된다면 데이터 공간을 2^b개로 분할한다. 그러나 제안하는 방법은 빠른 연산을 위하여 b개로 할당하므로 분할된 셀의 크기가 더 크므로 벡터 선택률은 떨어질 수밖에 없다. 그러나 떨어진 벡터 선택률에 의하여 증가된 디스크 입출력 시간의 증가분 보다 계산과정의 단순화로 감소된 계산시간의 감소분이 더 크다면 전체 검색 시간을 줄일 수 있을 것이다.

본 장에서 분석한 바와 같이 제안하는 알고리즘은 객체 사이의 거리를 계산하는 속도 면에서 기존의 방법에 비하여 월등하게 향상되고 또한 최근접 이웃을 계산 하는 속도에서도 기존의 방법보다 개선되었지만 벡터 선택률은 더 크다. 그러나 벡터 선택률의 상승으로 인한 성능감소 폭보다 계산 속도 감소로 인한 성능향상 폭이 더 크다면 전체 성능은 증가할 것이다.

4. 실험 및 결과 분석

실험은 Intel Pentium 4 2.6GHz의 CPU와 512MB의 메모리를 장착하고 Windows XP 운영체제를 사용하는 컴퓨터에서 수행하였다. 실험에서 사용한 고차원 멀티미디어 데이터는 버클리 풍경 이미지 집합으로부터 추출한 MPEG-7

Color Structure 기술자[8]를 사용하였다.

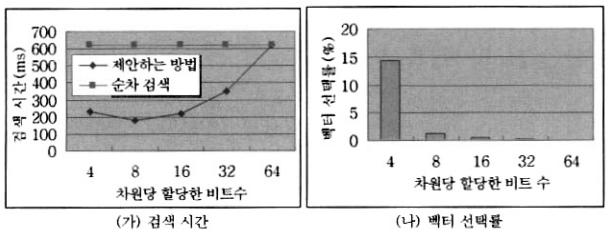
4.1 벡터 근사 정도에 따른 성능 실험

벡터 근사 정도란 색인 정보가 실제 벡터를 얼마나 정확하게 표현하고 있는 가를 말하는데, 제안하는 색인 방법에서 벡터 근사 정도는 객체의 한 차원을 나타내는 비트 수로 표현할 수 있다. 즉, 차원 당 할당되는 비트 수가 클수록 벡터 근사 정도가 높아지고, 적을수록 벡터 근사 정도는 낮아지게 된다. 그런데, 색인 정보의 근사 정도가 높아지면 벡터 선택률이 낮아질 수 있지만 색인 정보의 크기가 커져 I/O 시간이 늘어나기 때문에 적절한 크기의 비트 수를 선택하는 것은 매우 중요하다. (그림 9)는 각 차원 당 할당된 비트 수에 따른 성능 변화 그래프를 보여준다. 실험한 비트 수는 4비트, 8비트, 16비트, 32비트 그리고 64비트이고 실험에 사용된 데이터의 차원의 개수는 256개이다. 근접 이웃의 개수는 다른 논문에서도 주로 사용하는 10개를 선택하였다. 실험 결과의 신뢰도를 높이기 위해 각 실험의 수행 시간은 100개의 랜덤하게 선택된 질의 객체에 대한 수행 시간을 평균한 값이다. <표 4>은 각 차원 당 할당된 비트 수에 따른 색인 파일의 크기이다.

(그림 9)-(가)의 실험 결과에 의하면 각 차원 당 8비트를 할당했을 때 수행 시간이 가장 짧은 것을 알 수 있다. 반면에 64비트를 할당한 경우는 순차 검색보다 느린 성능을 보여주는데, 이는 64비트를 할당한 색인 파일의 크기가 실제 벡터 데이터 파일보다 더 커졌기 때문이다. 결국, 4비트를 제외하고는 전체적으로 색인 파일의 크기가 작을수록 검색 속도가 빨라진다. 즉, 4비트의 경우는 벡터를 근사 하는 정도가 너무 낮아서 필터 아웃시키는 객체의 개수가 급격히 커져 실제 데이터를 읽는 횟수가 증가하기 때문일 것이라고 예측할 수 있다. (그림 9)-(나)는 차원 당 할당한 비트 수에 따른 벡터 선택률을 나타낸 그래프이다. 벡터 선택률은 전체 객체의 개수에 대한 필터링 후 남은 후보 객체 리스트의 비율이다. 벡터 선택률이 낮을수록 실제 데이터를 읽는 횟수가 줄어들기 때문에 전체 검색 시간은 줄어들게 된다. (그림 9)-(나)의 결과에서 볼 수 있는 바와 같이 각 차원에 대하여 4비트를 할당한 경우 벡터 선택률이 급격하게 증가한다. 따라서 실제 데이터를 읽는 횟수가 증가하여 4비트의 경우는 전체 수행 속도가 8비트에 비하여 좋지 않음을 알 수 있다. 결과적으로 본 실험을 통하여 제안하는 방법은 각 차원 당 8비트를 할당하였을 때 가장 성능이 좋은 것을 알 수 있었고 순차 검색보다 약 3배의 성능 향상이 있는 것을 확인할 수 있다.

〈표 4〉 차원 당 할당된 비트 수에 따른 색인 파일의 크기

비트 수	4	8	16	32	64
색인 파일의 크기 (KByte)	3,242	6,311	12,619	25,235	50,467



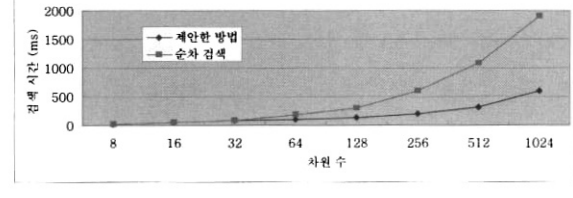
(가) 검색 시간 (나) 벡터 선택률

(그림 9) 차원 당 할당된 비트 수에 따른 검색 시간과 벡터 선택률

4.2 차원 증가에 따른 성능 실험

차원의 저주(Dimensionality Curse)[7]은 객체의 차원이 증가함에 따라 클러스터링 기반의 색인을 적용한 검색 성능이 급격하게 하락하여 오히려 순차 검색보다 느려지는 현상을 의미한다. 본 논문에서는 제안하는 방법이 차원의 저주 문제를 해결할 수 있음을 보여주기 위해 차원 증가와 검색 시간에 대한 실험을 수행하였다. (그림 10)은 8~1024 차원을 가진 데이터 집합에 대해 각각 검색을 수행하고 검색에 소요된 시간을 순차 검색 시간과 비교하여 나타낸 그래프이다. 실험 결과에 의하면 64차원 이하의 경우에는 순차 검색과 별 차이가 없으나 차원이 커질수록 순차 검색과의 검색 시간 차이가 점점 더 커지는 것을 확인할 수 있다.

32, 64, 128, 256차원의 데이터 집합은 MPEG-7 XM (eXperience Model)[9]을 사용해 25,168개의 이미지로부터 MPEG-7 Color Structure 기술자를 직접 추출하였다. 하지만 나머지 차원의 데이터 집합은 직접 추출이 불가능하므로 다른 데이터 집합으로부터 변형하여 생성하였다. 즉, 8, 16차원의 데이터 집합은 256차원 데이터 집합에서 분산이 높은 차원을 각각 8, 16개를 선택하여 구성하였고, 512, 1024 차원의 데이터 집합은 256차원 데이터 집합의 각 벡터의 원소 값들에 대한 평균과 분산을 유지하는 범위 안에서 각 차원에 임의의 값을 할당하여 구성하였다.



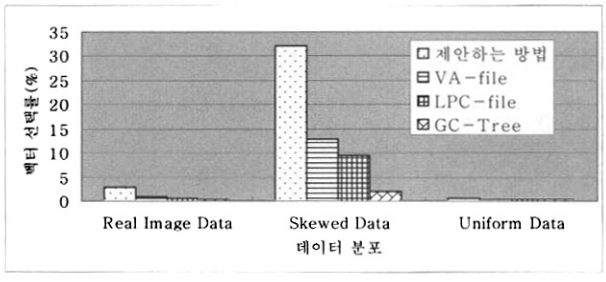
(그림 10) 차원 수에 따른 검색 시간 (25,168개 이미지)

4.3 기존 방법들과의 성능 비교

본 장에서는 제안하는 방법과 기존의 방법인 VA-file, LPC-file 그리고 GC-tree와의 성능을 비교한다. 제안하는 알고리즘의 각 차원 당 비트 수는 가장 성능이 우수한 8비트를 선택하였다. 동일한 환경에서의 비교를 위하여 VA-file과 LPC-file의 각 차원 당 비트 수도 8비트로 유지하였고 GC-tree의 경우 차원당 초기 비트수를 2비트로 하였고 각 차원의 데이터 분포에 따라 적절히 비트가 할당된다. 256차원의 총 25168개의 실제 이미지 데이터를 무작위 조합하여 평균 이미지를 만들어내어 50000개, 100000개, 200000개의 데이터에 대하여 실험하였다. 제안하는 방법과 기존의 방법들로 생성된 색인 파일의 크기는 <표 5>와 같다.

〈표 5〉 객체의 개수에 따른 각 방법들의 색인 파일 크기 (KByte)

객체의 개수	실제 벡터 크기	제안하는 방법	VA-file	LPC-file	GC-tree
25,168	50,337	6,311	6,805	6,854	4,155
50,000	100,002	12,427	12,921	12,971	7,479
100,000	200,003	24,251	24,745	24,795	14,210
200,000	400,005	48,174	48,668	48,718	26,999

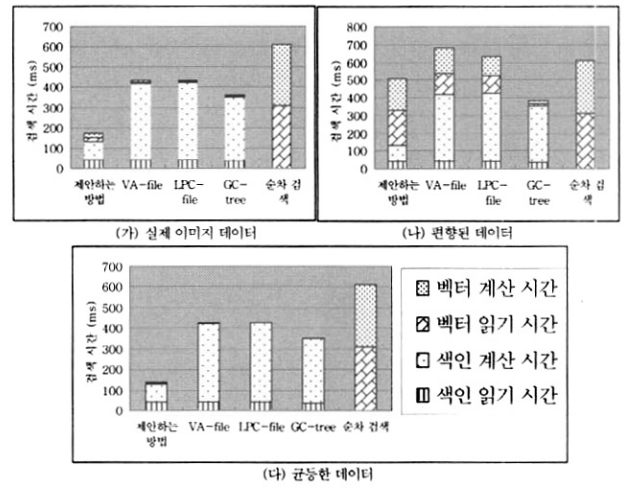


(그림 11) 기존의 방법과의 벡터 선택률 비교

본 논문이 제안하는 방법은 앞에서도 언급하였듯이 같은 크기의 색인 파일을 가지는 경우, 기존의 방법보다 색인 파일의 근사 정도가 낮기 때문에 벡터 선택률이 기존 방법에 비하여 높을 수밖에 없다. 특히 데이터의 분포가 불균등할 경우 데이터 공간을 적응적으로 나누는 GC-tree에 비하여 벡터 선택률이 크게 높아질 수 있다. (그림 11)은 50,000개의 실제 이미지 데이터(real data), 데이터 분포가 균일하게 분포된 인공적인 데이터(uniform data), 그리고 80%가 거의 비슷한 값을 가지도록 편향된 인공적인 데이터(skewed data)에 대한 각 방법의 벡터 선택률을 비교한 것이다.

(그림 11)에서 볼 수 있는 바와 같이 제안하는 방법은 기존의 방법에 비하여 벡터 선택률이 좋지 않다. 특히 두 번째 항목의 편향된 데이터에서는 많은 차이가 있다. 첫 번째 항목의 실제 이미지 데이터에서의 실험에서 비교 대상 중 가장 좋은 GC-Trec과 비교하여 2.7% 정도가 차이가 난다. 이는 제안하는 방법이 기존의 방법에 비하여 1350개 정도의 객체를 더 읽어야 하는 것이다. 따라서 제안하는 방법은 실제 벡터 데이터를 읽는 시간과 그 것의 거리를 계산하는 시간이 기존의 방법과 비교하여 최대 2.7% 더 크다. 그러나 전체 계산 시간은 실제 벡터를 읽는 시간과 그것의 거리를 계산하는 시간 이외에도 색인 파일을 읽는 시간과 읽은 색인으로 거리계산을 수행하여 필터링하는 시간도 포함된다. 따라서 3장에서 분석한 바와 같이 전체 검색 시간을 계산 순서대로 색인 읽기 시간(T_{io}), 색인 계산 시간(T_{ic}), 실제 벡터 읽기 시간(T_{vo}) 그리고 실제 벡터 계산 시간(T_{vc})로 나누어서 측정해 보았다. (그림 12)는 서로 다른 분포를 가진 데이터에 대한 각 방법의 검색에 소요되는 부분 시간을 누적 그래프로 나타낸 것이다.

(그림 12)-(나)의 결과에서 볼 수 있듯이 제안한 방법은 편향된 분포의 데이터에 대해서는 좋지 않은 결과를 낸다. 그러나 실제 이미지 데이터와 균등한 분포의 데이터에서는



(그림 12) 각 방법의 입출력 시간과 계산 시간 비교

제안한 방법이 가장 좋은 결과를 보여준다. 각 그래프의 부분 시간에서 알 수 있듯이 실제 벡터를 읽는 시간과 계산하는 시간은 색인파일을 읽는 시간과 계산하는 시간에 비하면 상대적으로 매우 작다. 기존 방법 중 가장 좋은 성능을 내는 GC-tree와 제안하는 방법과의 입출력 시간의 차이는 25ms 이고 계산시간은 233ms 이나 차이가 난다. 제안한 방법을 구현한 환경은 4.1장에서 언급하였듯이 일반적인 PC환경에서 구현하였다. 실험 결과에 의하면 계산 시간에 비하여 입출력 시간의 비중은 그리 크지 않다. 이렇듯 데이터베이스가 하드디스크에 존재하고 운영체제에 의한 캐시의 영향이 클 때, 검색 성능에 크게 영향을 미치는 것은 계산 속도이다. 제안한 방법은 입출력 시간의 손해를 조금 보지만 비트 연산을 통하여 계산 속도를 향상시킴으로써 전체 성능을 향상시켰다.

5. 결론 및 앞으로의 연구 방향

본 논문은 필터링을 위한 계산 시간을 줄이기 위하여 질의 객체와 전체 데이터들 사이의 빠른 거리 계산을 위한 새로운 비트맵 색인 구조를 제안한다. 제안하는 방법은 각 객체를 특성 벡터 공간상의 위치를 나타내는 비트 패턴으로 저장하고, 객체 사이의 거리를 구하는 연산은 실제 벡터 값의 연산이 아닌 XOR 비트 연산으로 대체한다. 비트 연산은 실제 벡터 값의 산술 연산보다 속도가 월등히 빠르기 때문에 빠른 속도로 객체들 간의 거리를 구할 수 있다. 실험에 의하면 본 논문이 제안하는 방법은 기존 벡터 근사 접근 방법들과 비교하여 벡터 선택률은 근소하게 높아서 입출력 시간은 더 크지만, 계산 시간을 크게 줄임으로써 전체 수행속도 면에서는 순차 검색보다는 약 4배, 기존의 방법들 보다는 약 2배의 성능이 향상되었다. 결과적으로 데이터베이스의 입출력 속도가 충분히 빠른 경우 기존의 벡터 근사 접근 방법의 필터링 속도를 빠르게 함으로써 전체 검색 속도를 향상 시킬 수 있음을 확인할 수 있었다.

고차원 멀티미디어 데이터베이스 검색은 현재에도 많은 연구가 이루어지고 있으며 아직까지도 개선해야 할 연구 분야를 많이 가지고 있다. 제안하는 방법은 디스크 I/O 속도가 빠른 환경에서 의미가 있다. 만약 디스크 I/O 속도가 매우 느린 환경이라면 기존의 벡터 선택률을 최대한으로 낮추는 방법이 더 유리할 것이다. 또한 분포가 어느 한쪽으로 편향된 데이터의 경우 균등한 범위로 공간을 분할하는 제안하는 방법은 적합하지 않다. 따라서 데이터의 분포에 따라 적응적으로 공간을 분할하면서도 비트 연산을 통하여 계산 속도를 향상시킬 수 있는 방법에 대한 연구가 지속되어야 할 것이다. 본 논문이 제안하는 색인 및 검색 방법은 MPEG-7 기술자 데이터베이스 검색[10], 유전자 정보 데이터베이스 검색 등의 고차원 데이터베이스의 빠른 검색을 필요로 하는 분야에 널리 이용될 수 있을 것이다.

참 고 문 헌

[1] R. Weber, H. Schek, and S. Blott, "A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces," Proc. of the Int'l Conf. on VLDB, pp.194-205, 1998.

[2] G. Cha, X. Zhu, D. Petkovic, and C. Chung, "An Efficient Indexing Method for Nearest Neighbor Searches in High-Dimensional Image Databases," IEEE Trans. on Multimedia, Vol.4, No.1, pp.76-87, 2002.

[3] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: An efficient and robust access method for points and rectangles," Proc. of ACM SIGMOD Int'l Conf. on Meanagement of Data, pp.322-331, 1990.

[4] S. Berchtold, D. Keim, and H. Kriegel, "The X-tree: An Index Structure for High-Dimensional Data," Proc. of the Int'l Conf. on Very Large Data Bases, pp.28-39, 1996.

[5] G. Cha, and C. Chung, "The GC-Tree: A High-Dimensional Index Structure for Similarity Search in Image Databases," IEEE Trans. on Multimedia, Vol.4, No.2, pp.235-247, 2002.

[6] N. Katayama, and S. Satoh, "The SR-Tree: An Index Structure for High-Dimensional Nearest Neighbor Queries," Proc. of the ACM SIGMOD Int'l Conf. on Management of Data, pp.369-380, 1997.

[7] P. Indyk, and R. Motwani, "Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality," Proc. of the ACM Symp. on the Theory of Computing, pp.604-613, 1998.

[8] ISO/IEC JTC1/SC29/WG11 Information Technology Multimedia Content Description Interface-Part3: Visual, 2001.

[9] ISO/IEC JTC1/SC29/WG11 MPEG-7 Visual part of eXperience Model Version 11.0, 2001.

[10] B. Manjunath, P. Salembier, and T. Sikora, Introduction to MPEG-7 Multimedia Content Description Interface, JOOHN WILEY & SONS, 2002.

[11] K. Chakrabarti, and S. Mehrotra, "Local Dimensionality Reduction: A New Approach to Indexing High Dimensional Spaces," Proc. of the Int'l Conf. on VLDB, pp.89-100, 2000.

[12] K. Kanth, D. Agrawal, and A. Singh, "Dimensionality Reduction for Similarity Searching in Dynamic Databases," Proc. of the ACM SIGMOD Int'l Conf. on Management of Data, pp.166-176. 1998.

[13] E. Kushilevitz, R. Ostrovsky, and Y. Ravani, "Efficient Search for Approximate Nearest Neighbor in High Dimensional Spaces," Proc. of the ACM Symposium on the Theory of Computing, pp.614-623, 1998.

박 주 현

e-mail : parkjh@sogang.ac.kr
1999년 서강대학교 컴퓨터학과(학사)
2002년 서강대학교 컴퓨터학과(공학석사)
2002년~현재 서강대학교 컴퓨터학과 박사과정
관심분야 : 동영상분석, 멀티미디어검색

손 대 온

e-mail : son@sogang.ac.kr
2003년 서강대학교 컴퓨터학과(학사)
2005년 서강대학교 컴퓨터학과(공학석사)
2006년~현재 LG전자 MC연구소 주임 연구원
관심분야 : 무선인터넷, HD방송

낭 종 호

e-mail : jhnang@sogang.ac.kr
1986년 서강대학교 전자계산학과(학사)
1988년 한국과학기술원 전자계산학과 (공학석사)
1992년 한국과학기술원 전자계산학과 (공학박사)
1992년~1993년 Fujitsu 연구소 연구원
1993년~현재 서강대학교 컴퓨터학과 교수
관심분야 : 멀티미디어 검색, 모바일 멀티미디어

주 복 규

e-mail : bkjoo@hongik.ac.kr
1977년 서울대학교 계산통계학과(학사)
1980년 한국과학기술원 전산학과 (공학석사)
1990년 메릴랜드대학교 전산학과 (공학박사)
1990년~1998년 삼성전자 중앙연구소 수석연구원
1998년~2000년 (주) 동양시스템즈 연구소장
2001년~현재 홍익대학교 컴퓨터정보통신공학과 교수
2004년~현재 아시아태평양 첨단망협회(APAN) 학술위원회 의장
관심분야 : 인터넷 프로토콜, 네트워크 보안, 소프트웨어 재사용