GC-트리 : 이미지 데이타베이스를 위한 계층 색인 구조

(GC-Tree: A Hierarchical Index Structure for Image Databases)

차 광 호 †

(Guang-Ho Cha)

요 약 멀티미디어 데이타의 사용이 증가함에 따라 고차원 이미지 데이타에 대한 효율적인 색인과 검색 기법이 크게 요구되고 있다. 그러나 많은 노력에도 불구하고 현재의 다차원 색인 기법들은 고차원 데이타 공간에서 만족할 만한 성능을 보여주지 못하고 있다. 이러한 소위 차원의 저주를 해결하기 위해 최근에 차원을 줄이거나 근사 해를 구하는 등의 접근법이 시도되고 있지만 이러한 방법들은 근본적으로 정확도의 상실이라는 문제를 갖고 있다. 정확도의 보존을 위해 VA-file, LPC-file등과 같이 벡터 근사에 기반한 기법들이 최근에 개발되었다. 그러나 이 기법은 검색 성능이 색인 파일의 크기에 큰 영향을 받으며, 한번에 큰 검색 공간을 줄이는 계층 색인 구조의 장점을 상실한다. 본 논문에서는 이미지 데이타베이스에서 유사성 질의를 위한 새로운 계층 색인 구조인 GC-트리를 제안한다. GC-트리는 밀도 함수에 기초하여 데이타 공간을 적응적으로 분할하고, 색인 구조를 동적으로 생성한다. 이러한 특성을 갖는 GC-트리는 군집화된 고차원 이미지 데이타 검색에 훌륭한 성능을 나타낸다.

키워드 : 고차원 색인, 유사성 검색, 최근접 질의, 이미지 데이타베이스

Abstract With the proliferation of multimedia data, there is an increasing need to support the indexing and retrieval of high-dimensional image data. Although there have been many efforts, the performance of existing multidimensional indexing methods is not satisfactory in high dimensions. Thus the dimensionality reduction and the approximate solution methods were tried to deal with the so-called dimensionality curse. But these methods are inevitably accompanied by the loss of precision of query results. Therefore, recently, the vector approximation-based methods such as the VA-file and the LPC-file were developed to preserve the precision of query results. However, the performance of the vector approximation-based methods depend largely on the size of the approximation file and they lose the advantages of the multidimensional indexing methods that prune much search space. In this paper, we propose a new index structure called the GC-tree for efficient similarity search in image databases. The GC-tree is based on a special subspace partitioning strategy which is optimized for clustered high-dimensional images. It adaptively partitions the data space based on a density function and dynamically constructs an index structure. The resultant index structure adapts well to the strongly clustered distribution of high-dimensional images.

Key words : high-dimensional indexing, similarity search, nearest neighbor query, image database

1. 서 론

이미지 데이타베이스는 이미지를 종종 d 개의 수치적 특성으로 표현하고, 그 특성 벡터와 유사성을 이용하여 저장하고 검색한다. 특성 벡터는 d-차원 벡터 공간 상의 점으로 볼 수 있고, 유사성은 그 공간 내에서 거리의 척도로 볼 수 있다. 이러한 시스템에서 중요한 문제는 주어진 질의 이미지와 가장 유사한 k 개의 이미지를 찾는 것이다. 이 문제는 먼저 질의 이미지를 대응하는 특성 벡터로 바꾸고, 그 벡터에 대한 k 개의 최근접 이웃(nearest neighbors: NNs)을 찾아서 해결한다. 그러면 기본 문제는 k 개의 NNs을 어떻게 효율적으로 찾는가 하는 것이다(k -NN문제).

저차원 또는 중차원(예를 들어, 10차원 이하)의 특성 벡터를 갖는 응용에서는 기존의 다차원 색인 기법(mul-

· 본 연구는 정보통신부의 정보통신기초기술 연구지원사업(정보통신연구진흥원)으로 수행한 연구결과입니다.

† 종신회원 : 숙명여자대학교 멀티미디어학과 교수
ghcha@sookmyung.ac.kr

논문접수 : 2003년 5월 23일
심사완료 : 2003년 11월 3일

tidimensional indexing methods: MIMs)을 이용하여 이 문제를 훌륭하게 해결할 수 있다. 그러나 지금까지, 예를 들어 100 차원 이상의, 고차원 특성 벡터를 갖는 응용에서는 효과적인 해결책이 발견되지 못했다. 고차원에서는 데이타 공간의 분할에 기초하는 기존의 MIMs의 성능은 질의 객체와 전체 데이타베이스 객체를 하나씩 비교하는 순차 탐색보다 성능이 더 나빠진다[1-4]. 따라서 주된 문제는 차원의 저주(*dimensionality curse*)[5] -차원이 높아짐에 따라 검색 성능이 급격히 하락하는 현상- 를 해결하는 것이다.

데이타베이스 시스템에서 *k-NN* 문제를 해결하기 위한 다양한 접근법이 연구되어 왔다. 직관적으로는 객체들이 다차원 공간상의 점으로 표현되기 때문에 MIMs을 사용하여 *k-NN* 질의 처리의 속도를 높이는 것이 자연스러워 보인다. 그러나, 대부분의 MIMs은 고차원 데이타 집합에 대해서 성능이 좋지 못하며, 현재의 MIMs이 고차원 공간에서 검색 공간을 잘라낸다는 것은 매우 힘들다[6,7].

고차원 공간에서 이러한 MIMs이 갖는 한계를 해결하기 위해 최근에는 다른 방향에서의 접근이 시도되었고, 차원 축소 접근법[8-10], 근사 최근접 이웃 접근법[1-3], 다중 공간 채움 곡선 접근법[11,12], 벡터 근사 접근법[4,6,7]의 4가지로 크게 분류할 수 있다. 이 중에서 벡터 근사 접근법을 제외한 나머지 접근법은 질의 처리 결과에서 어는 정도의 정확도를 포기한다는 의미에서 일종의 근사 접근법이라고 할 수 있다. 벡터 근사 접근법은 기존의 MIMs은 검색 공간을 잘라내지 못하므로 검색 중에 모든 특성 벡터들을 디스크에서 읽어야 한다는 가정 하에, 특성 벡터들을 근사값으로 표현하고 검색 시에 이 근사값을 차례로 읽고 특성 벡터 자체는 이 근사값을 통해 필터링하여 그 중 적은 일부만 실제로 디스크에서 읽도록 하는 것이다. 그러나 이 방법은 검색 성능이 색인 파일의 크기에 영향을 받으며, 한번에 큰 검색 공간을 줄이는 기존의 계층적 색인 구조의 장점을 상실한다.

본 논문에서는 차원의 저주를 해결하는 새로운 접근법으로써 고차원 데이타 집합에서 클러스터와 국외자를 구분해낸 다음, 데이타 집합의 분할을 클러스터 부분에 집중하고, 이 클러스터들을 계층 구조로 구성함으로써 고차원 데이타 집합에 대해서도 트리 형태의 계층적 색인 구조를 형성할 수 있음을 보인다.

2. GC-트리

GC-트리는 실제 데이타의 적은 부분만 접근하는 벡터근사 기법의 장점과 검색 공간의 많은 부분을 잘라내는 다차원 색인 기법의 장점을 결합한다. 이렇게 하기 위해 GC-트리는 데이타 집합을 분석하고 그에 따라 데이타 공간을 분할하며, 분할된 계층구조에 기반하여 계층적으로 색인을 형성한다.

2.1 밀도 기반 공간 분할

GC-트리는 데이타 집합의 밀도에 기초하여 데이타 공간을 계층적으로 분할한다. 데이타 집합의 밀도를 근사하기 위해 데이타 공간을 서로 겹치지 않는 사각형 셀로 분할하고, 각 셀에 놓이는 벡터의 수를 계산한다. 이것을 위해 각 차원을 같은 수의 동일한 길이의 간격으로 분할한다. 이것은 각 셀이 같은 크기를 가짐을 의미하고, 따라서 각 셀 내부에 놓이는 벡터의 수는 그 셀의 밀도를 근사하는데 사용될 수 있다.

정적 데이타베이스에서는, 한 셀의 밀도를 전체 데이타 벡터의 수에 대해 그 셀에 놓이는 벡터의 비율로 정의한다. 그러나 동적 데이타베이스 환경에서는, 특히, 처음 데이타베이스를 구축하는 경우에는 전체 데이타 벡터의 수에 상대적인 밀도를 계산하기 어려우므로 조밀한 셀과 희박한 셀을 결정할 밀도 임계치(density threshold)를 계산하기 어렵다. 따라서 GC-트리에서는 한 셀의 밀도를 특정 영역을 이진 분할(binary partitioning)로 2^d 개의 셀로 분할할 때(여기서 d는 차원의 수) 디스크 페이지 용량(즉, fanout)에 상대적으로 한 셀에 있는 데이타 벡터의 비율로 정의한다. 데이타 공간에서의 한 영역은 물리적으로 디스크의 한 페이지로 사상되고, 한 디스크 페이지가 수용할 수 있는 객체의 최대수를 나타내는 수 P가 규정된다. 즉, P는 페이지 용량을 나타낸다. 한 페이지에 삽입된 데이타의 수가 P를 초과하면 일반적으로 그 페이지를 두 개로 분할한다. 우리는 한 셀 c의 밀도가 특정 밀도 임계치 τ를 초과할 때 셀 c를 조밀하다고 하고, 그렇지 않을 경우 희박하다고 한다. 아울러, 조밀한 셀을 클러스터(cluster)라고 하고, 희박한 셀에 포함된 데이타 벡터들을 국외자(outliers)라 부른다. 만약 밀도 임계치 τ를 페이지 용량의 절반보다 크게 결정하면, 객체가 삽입되어 한 영역을 분할할 때 최대로 하나의 클러스터만 생성될 수 있다.

밀도 기반 분할의 기본 아이디어는 다음과 같다: (1) 데이타 공간을 분할할 때 클러스터를 규정하고; (2) 국외자들에 의해 점유된 공간은 검색 시에 잘라내기 어려우므로 분할을 클러스터들의 부분 공간에만 집중한다; (3) 같은 분할 레벨에 존재하는 국외자들은 함께 처리한다.

국외자들은 전체 부분 공간에 넓게 퍼져있어서 조그만 영역으로 묶기 어렵고, *k-NN* 탐색을 위한 큰 검색 범위($k\text{-}NN^{sphere}$)와 국외자들이 존재하는 큰 영역이 거의 교차하므로 검색 시에 광범위한 공간에 분산된 국외자들을 잘라내기는 거의 힘들다. 따라서 우리는 색인 구조의 하나의 노드에 한 특정 부분 공간에서 생성된 모

든 국외자를 모으고, 검색 공간 $k\text{-}NN^{sphere}$와 클러스터들이 교차하는 확률을 줄이기 위해 공간 분할은 클러스터들에만 집중한다.

저차원 색인에서는 가능하면 데이타 공간을 균형 있게 분할하는 것이 유리하다고 알려져 있다. 그러나, 고차원 공간에서의 균형 분할은 각 분할에 대해 큰 경계 사각형(bounding rectangles)을 생성하여 검색 시에 해당 부분을 잘라내기 어렵게 만든다.

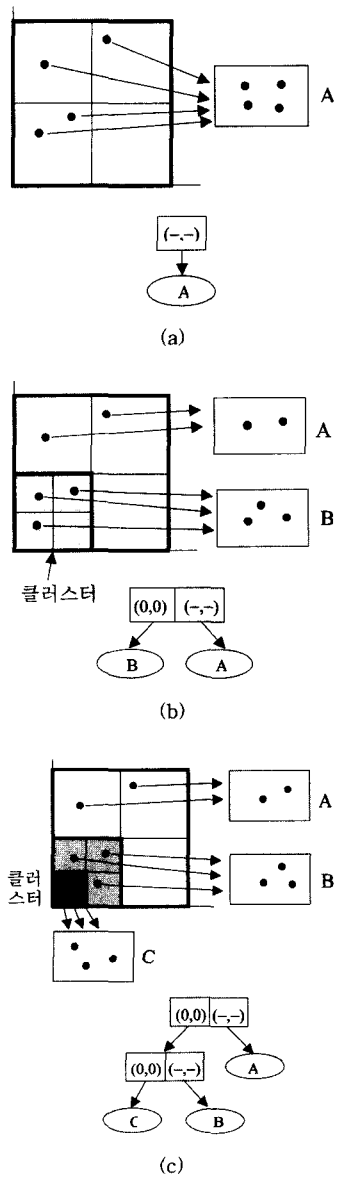


그림 1 밀도 기반 분할에서의 공간 분할과 GC-트리 색인 형성 과정

GC-트리는 데이타 삽입에 따라 한 클러스터 또는 희박 셀에 해당하는 디스크 페이지가 범람할 때마다 그 데이타 공간을 한번에 모든 차원의 중앙에서 분할한다. 따라서 그 공간은 동일한 크기의 2^d 개의 셀로 분할된다. 분할한 공간에서 클러스터와 국외자를 규정하고 데이타 벡터를 위치에 따라 해당 노드에 분산시킨다. 만약 분할로 인해 새로운 클러스터가 생성되면 그것을 분할 계층 구조에서 범람 노드의 아래 레벨 노드로 만든다. 그렇지 않으면, 새로운 노드를 할당하여 범람 객체를 삽입한 다음, 단순히 범람 노드에서 새로운 노드로 연결한다.

그림 1은 반복되는 데이타의 삽입으로 2 차원 공간을 분할하는 과정과 GC-트리 색인의 성장 과정을 보여준다. 처음에는 하나의 공간(셀)이 존재하고 이것은 전체 데이타 공간에 대응한다. 개념적으로 많은 데이타 벡터가 이 셀에 삽입될 것이고, 그 셀에 해당하는 디스크 페이지가 존재한다. 그림 1(a)에서, 데이타 공간은 이미 4 개의 셀로 분할되어 있다. 밀도 임계치 τ는 3/4이라고 가정한다. 즉, 페이지 용량은 4이고 한 클러스터는 최소한 3 개의 데이타 벡터를 포함해야 하고, 3 개 보다 적은 수의 벡터를 포함하는 셀에 있는 벡터들은 국외자로 취급한다. 그림 1(a)는 데이타베이스가 전체 데이타 공간에서 4 개의 벡터를 포함하는 초기 상태를 나타낸다. 하나의 루트 노드와 하나의 리프 노드가 존재한다. 리프 노드는 희박 셀에 존재하는 4 개의 벡터(국외자)에 해당하는 벡터 근사를 포함하고 있다. 루트 노드의 셀 벡터는 전체 데이타 공간을 나타낸다. 기호 '_'는 주어진 한 셀에서 한 차원에 대한 전체 도메인을 나타낸다. 그림 1(b)에서 새로운 벡터가 첨가되고 새로운 클러스터가 생성되었다. 범람 공간을 2^2 개의 셀로 분할하고 새로운 클러스터를 분할 노드의 하위 레벨 노드로 만든다. 그림 1(b)에서 새로운 노드 B는 셀 벡터가 (0,0)인 클러스터를 나타내고 국외자들은 노드 A에 재분산된다. 그림 1(c)는 셀 B를 분할한 뒤의 구조를 보여준다.

2.2 색인 구조

GC-트리 색인 구조는 디렉토리와 데이타 노드의 두 개의 요소로 구성된다. 디렉토리는 내부 노드와 리프 노드로 구성된다. 디렉토리 노드는 색인에 사용되고 데이타 노드는 실제 데이타를 저장하는데 사용된다. 각 노드는 데이타 공간의 한 영역에 해당되고 한 디스크 페이지로 사상된다. 클러스터에 해당하는 영역은 같은 길이의 변을 갖는 하이퍼사각형이다. 국외자 영역은 원래 영역에서 클러스터에 해당하는 영역들을 제외하고 남는 영역이다. 그러나 국외자 영역에 대한 셀 벡터는 원래 영역의 셀 벡터로 표현한다. 클러스터에 해당하는 노드를 클러스터 노드, 국외자에 해당하는 노드를 국외자 노

드라고 부른다.

리프 노드의 요소는 실제 벡터에 대한 LPC-file[6] 근사 <r, θ>와 실제 벡터 p가 저장되는 데이타 노드를 가리키는 포인터로 이루어진다. 여기서 r은 벡터 p가 있는 셀의 원점 O와 벡터 p간의 거리로 계산되고, θ는 벡터 p와 셀의 원점에서 반대편 모서리까지의 대각선 간의 각도로 계산된다. 내부 노드의 요소는 하위 레벨 노드가 점유하는 영역에 대응하는 셀 벡터와 하위 레벨 노드를 가리키는 포인터로 구성된다. 그림 2와 3은 내부 노드와 리프 노드의 구조를 나타낸다. 리프 노드의 LPC_j는 해당 벡터의 극좌표, 즉, <r, θ>를 나타낸다.

국외자의 삽입으로 노드가 범람할 때는 새로운 노드를 할당하고 그 노드를 단순히 범람 노드에 연결한다. 만약 클러스터 벡터의 삽입으로 클러스터 노드가 범람하면 새로 할당하는 노드는 색인 트리의 하위 레벨 노드가 된다. 즉, 공간 분할 계층 구조와 그것을 나타내는 색인 트리 계층 구조 간에 일 대 일 대응 관계가 존재한다.

요소의 갯수	셀 벡터1	자식1 포인터	셀 벡터i	...	자식i 포인터	범람 페이지 포인터

그림 2 내부 노드 구조

요소의 갯수	LPC_1	객체1 포인터	...	LPC_j	객체j 포인터	범람 페이지 포인터

그림 3 리프 노드 구조

예제 1: 공간 분할 계층 구조와 GC-트리의 색인 트리 계층 구조와의 관계를 보이기 위해 그림 4와 5에 4-단계 색인을 갖는 2-차원 GC-트리를 나타내었다. 그림 4에서, 차원당 2 비트를 사용하여 2-차원 데이타 공간을 미리 4 × 4 개의 셀로 분할하였다. 밀도 임계치 τ는 $3/P$로 가정하였다. P는 리프 노드의 팬아웃이다. *Root* 디렉토리 노드는 3개의 엔트리를 갖고 있고, 셀 벡터(00, 11)을 갖고 있는 두번째 디렉토리 엔트리는 그림 5의 노드 C2를 가리키고, 그림 4에서는 셀 C2를 나타낸다. 차례로 이것은 클러스터 C4와 국외자 영역으로 더 세분화된다. 그림 4의 셀 C4는 클러스터 C5와 국외자 영역으로 세분화되고, 그림 5의 GC-트리에 해당 노드가 존재한다. 데이타 점들은 <c_i, r, θ>로 근사되고, LPC(local polar coordinate)[6] 정보 <r, θ>는 리프 노드에 표현되고, 셀 벡터 c_i는 리프 노드의 내부 부모 노드의 색인 엔트리로 표현된다. 상위 노드의 색인 엔트리에 있는 셀 벡터는 하위 노드의 셀 벡터들의 공통 접두사(prefix)로 사용된다. 예를 들어, C3의 셀 벡터

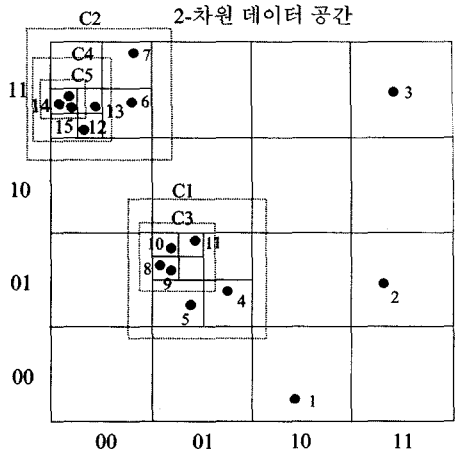


그림 4 데이타 공간의 비균등 분할

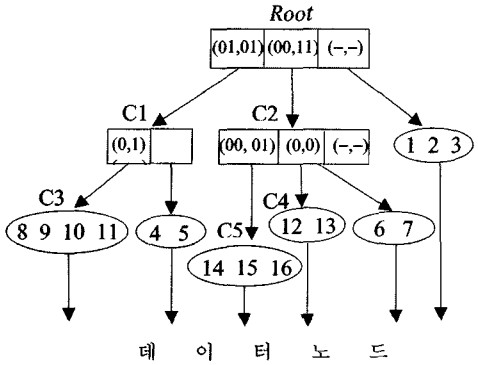


그림 5 그림 4의 분할에 대한 색인

(010, 011)은 상위 노드 색인 엔트리의 셀 벡터(01, 01)을 자신의 셀 벡터(0, 1)의 앞에 붙여 얻을 수 있다. 하위 노드 셀 벡터를 나타내는데 이 공통 접두사를 사용함으로써 GC-트리는 내부 노드의 팬아웃을 높일 수 있고 디스크 접근 회수를 줄일 수 있다. 밀도 기반 비균일 데이타 분할과 함께 이것은 GC-트리가 디스크 접근 회수를 줄일 수 있는 장점을 제공한다.

3. 알고리즘

여기서는 GC-트리 색인 구축 알고리즘 및 GC-트리를 이용한 k-NN 탐색 알고리즘을 제시한다.

3.1 삽입

알고리즘 **Insert**는 GC-트리를 내려가면서 새로운 객체를 수용할 적합한 리프 노드를 찾는다. 다른 색인 트리와 달리 GC-트리는 상향식으로 트리가 성장하지 않는다. 한 노드 N의 범람은 N과 같거나 하위 레벨에서 새로운 노드 N'을 할당하고, 엔트리들을 클러스터와 국

외자로 구별하여 두 노드에 분산시키고, 필요하면 디렉토리 엔트리를 부모 노드에 첨가하여 처리된다.

Algorithm Insert(Node *N*, Object *O*)
// 객체 *O*를 *N*이 루트인 GC-트리에 삽입한다.
{
1.만약 *N*이 NULL이면 // 이것이 초기 상태이다.
{
1.1. 디렉토리 엔트리 (*r*, θ, *Ptr*)를 갖는 리프 노드를 생성한다. 여기서 (*r*, θ)는 입력 객체 *O*의 지역 극좌표이고 *Ptr*은 *O*의 데이타 노드를 가리킨다. 입력 객체가 놓이는 지역 셀 벡터 *Cell*은 부모 노드에 저장되고 GC-트리의 탐색을 안내하는데 사용된다.
1.2. 색인 엔트리(*Cell*, *Ptr*)를 갖는 내부 노드를 생성한다. 여기서 *Cell*은 셀 벡터이고 그것의 각 요소 는 "_"이다. *Ptr*은 객체 *O*의 지역 극좌표를 포함하는 리프 노드를 가리킨다.
1.3. *Root*는 위의 내부 노드를 가리킨다.
}
2. 그렇지 않으면 {
2.1. 입력 객체 *O*를 삽입할 리프 노드 *L*을 선택하기 위해 *O*의 특성 벡터에 대해 정확 매치 질의를 수행한다.
2.2. 만약 노드 *L*에 여유 공간이 있으면 객체 *O* 를 *L*에 넣는다.
2.3. 그렇지 않으면 **SplitLeaf(*L*, *O*)**을 호출한다.
}
}

Algorithm SplitLeaf(Node *N*, Object *O*)
{
1. 새 노드 *N'*을 할당한다.
2. GC-트리의 공간분할 원칙에 따라 범람 노드 *N*이 점유하고 있던 공간을 분할하고 클러스터와 국외자를 구분한다.
3. 만약 클러스터를 발견하면 {
3.1. 국외자들을 노드 *N*으로 보내고 클러스터 점들을 노드 *N'*으로 보낸다.
3.2. 노드 *N*과 *N'*을 위한 디렉토리 엔트리 *E*와 *E'*을 만든다.
3.3. N_p를 노드 *N*의 부모 노드라 한다.
3.4. 만약 노드 N_p에 여유 공간이 있으면 엔트리 *E*와 *E'*을 N_p에 저장한다.
3.5. 그렇지 않으면 **SplitNonleaf(N_p, *E*, *E'*)**을 호출한다.
}
4. 그렇지 않으면 {
4.1. 객체 *O*를 노드 *N'*에 삽입한다.
4.2. 노드 *N*을 노드 *N'*에 연결한다.
}
}

Algorithm SplitNonleaf(Node *N*, Entry *E*, Entry *E'*)
{
1. 새 노드 *N'*을 할당한다.
2. E_p를 범람 노드를 가리키는 ***N***에 있는 디렉토리 엔트리라고 한다.
3. 만약 E_p가 클러스터 노드를 가리킨다면 {
3.1. *E*와 *E'*를 *N'*에 넣는다.
3.2. E_p가 *N'*를 가리키도록 조정한다.
3.3. 노드 *N*과 *N'*를 저장한다.
}
4. 그렇지 않으면 { // E_p가 국외자 노드를 가리키는 경우 //
4.1. E_p를 *E'*로 바꾼다.
4.2. *E'*를 *N'*에 넣는다.
4.3. 노드 *N*을 노드 *N'*에 연결한다.
4.4. 노드 *N*과 *N'*를 저장한다.
}
}

3.2 삭제

GC-트리에서의 삭제는 매우 간단하다. 즉, 단순히 노드가 비면 그 노드를 삭제하고, 부모 노드에서 해당 노드에 대한 엔트리만 조정하면 되므로 특별히 삭제 알고리즘을 기술하지 않는다. 그림 4와 5의 예를 들어 삭제를 설명해보자. 그림 5에서 객체 4, 5가 삭제되었다고 하자. 그러면 단순히 객체 4, 5가 들어있던 노드를 삭제하고, 해당 노드에 대한 정보를 갖고 있는 부모 노드의 엔트리를 삭제한다. GC-트리에서는 R-트리와 같은 색인 구조와 달리 색인 트리의 높이의 균형이나 각 노드가 갖는 엔트리 수의 균형을 맞추려고 시도하지 않는다. 따라서 GC-트리에서는 데이타 집합의 군집성이나 편향성을 이용하여 트리를 오히려 비균형으로 유지함으로써 탐색 시에 잘라내는 비율을 높이고자 한다. GC-트리의 이러한 기본 정신에 의해 삽입이나 삭제 시에 범람이나 과부족(underflow) 처리가 굉장히 간단해지는 것이다. 이것은 [13]의 피라미드 기법의 정신과도 맥을 같이 한다.

3.3 *k*-NN 탐색 알고리즘

k-NN 탐색 알고리즘은 질의 벡터 *q*의 *k* 개의 최근접 이웃을 찾는다. 알고리즘은 벡터 근사 접근법처럼 2단계로 구성된다. 그러나 전체 근사 파일을 순차적으로 읽는 벡터 근사 접근법과 달리 분기-및-제한(branch-and-bound) 기법을 사용할 수 있다. GC-트리는 두 개의 우선 순위 큐 *bl_list*와 *cand_list*, 그리고 *k*-요소 배열 *knn*의 3개의 전역 구조를 사용한다. 배열 *knn*은 그 시점까지 진행된 *k* 개의 *NN*을 포함하고, 실행이 완료되면 최종 결과를 포함한다. *bl_list*는 해당 내부 노드에 대한 분기와 함께 질의 벡터로부터 노드까지의 최소 및 최대 거리를 포함한다. *cand_list*는 후보들을 포함한다. *bl_list*와 *cand_list* 는 min-heap로 구현된다. *cand_list*에는 또한 각 후보로부터 질의 벡터까지의 하한 경계치 d_{min}과 상한 경계치 d_{max}를 유지한다.

GC-트리의 내부 노드와 리프 노드에서 사용되는 탐색 경로 설정 정보는 다르다: 내부 노드에서는 셀 벡터, 리프 노드에서는 LPC가 사용된다. 따라서, 그들 간의 거리 경계치 d_{min}과 d_{max}이 다르다. 내부 노드에 대한 경계치는 다음과 같이 결정된다:

$$d^2_{\min} = \sum_{i=0}^{d} l_i^2 \text{ 여기서 } l_i = \begin{cases} q_i - m[r_{p_i}+1] & , r_{p_i} < r_{qi} \\ 0 & , r_{p_i} = r_{qi} \\ m[r_x+1] - q_i & , r_{p_i} > r_{qi} \end{cases}$$

$$d^2_{\max} = \sum_{i=0}^{d} u_i^2 \text{ 여기서 } l_i = \begin{cases} q_i - m[r_{p_i}] & , r_{p_i} < r_{qi} \\ \max(q_i - m[r_{p_i}], & \\ \quad m[r_{p_i}+1] - q_i) & , r_{p_i} = r_{qi} \\ m[r_{p_i}+1] - q_i & , r_{p_i} > r_{qi} \end{cases}$$

여기서 q_i는 *q*의 *i*-번째 차원의 요소로서 셀 r_{qi} 내에 존재한다. r_{pi} 는 데이타베이스 벡터 *p*가 놓이는 차원 *i*

에서의 셀이다. $m[j]$는 주어진 차원에서 j-번째 분할의 시작 위치이다.

리프 노드의 d_{min}과 d_{max}는 다음과 같이 계산된다:

$d_{min}^2 = |p|^2 + |q|^2 - 2|p||q| \cos|\theta_1 - \theta_2|$

$d_{max}^2 = |p|^2 + |q|^2 - 2|p||q| \cos(\theta_1 + \theta_2)$

여기서 θ_1(=∠AOD)은 벡터 p와 p가 있는 셀의 주 대각 간의 각도이고, θ_2(=∠BOD)는 벡터 q와 셀의 주 대각간의 각도이다(그림 6 참조). 즉, d_{min}과 d_{max}는 두 벡터 p와 q간의 각도 ϕ ($0° \le \phi \le 180°$)가 각각 최소, 최대일 때 결정된다. 두 벡터 p와 q간의 최소, 최대 각도는 각각 $|\theta_1 - \theta_2|$와 $(\theta_1 + \theta_2)$로 결정된다.

첫 번째 탐색 단계에서, 알고리즘 **k_NN_Search**는 GC-트리의 최상위 레벨의 분기들을 검사하고 각 분기에 대해 d_{min}과 d_{max}를 계산한 다음, d_{min}의 오름차순으로 각 분기를 탐색한다. k-NN 탐색의 잘라내는 원칙은 동적이므로 - 탐색 반경은 q와 그것의 k-번째 NN 간의 거리 $k\text{-}NN^{dist}$ - 방문하는 노드의 순서는 탐색 성능에 영향을 미친다. 자신의 d_{min}이 $k\text{-}NN^{dist}$를 초과하는 객체는 더 나은 k개의 후보를 이미 선정한 상태이므로 삭제할 수 있다. 첫 번째 단계의 끝에서 *cand_list*는 탐색 후보들의 집합을 포함한다.

두 번째 단계에서 벡터 근사 접근법처럼 d_{min}의 오름차순으로 실제 벡터를 방문하여 후보 집합을 정제하고, 최후로 k 개의 최근접 벡터를 찾는다. 함수 **Read-NodeFromGCtree**는 입력 변수로 제공되는 노드 번호에 대응하는 디스크 페이지를 읽는다. 함수 **GetFrom-BranchList**는 *bl_list*로 부터 가장 작은 d_{min}를 갖는 분기 리스트를 변수 *bl*로 가져온다.

Algorithm k_NN_Query(Query *q*, Integer *k*)

{

// *MAX*: 데이타베이스에 있는 임의의 두 점 간의 최대 거리 //

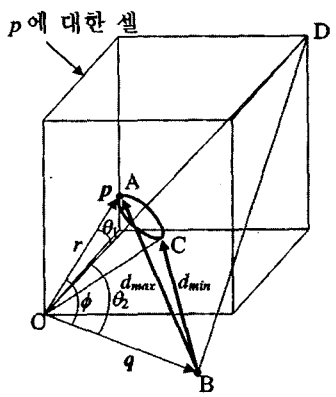


그림 6 3-차원 공간에서 리프 노드에 대한 하한치 d_{min}과 상한치 d_{max}

// NN: 최근접 이웃 //
// k-NN: k-번째 최근접 이웃 //
// *oid(x)*: *x*의 객체 식별자 //
// *dist(x)*: *x*와 ***q*** 간의 거리 //
// $d_{min}(x)$: *x*와 ***q*** 간의 d_{min} //
// *bl_list*: 분기 리스트를 저장하는 최소 히프 //
// *cand_list*: 후보 집합을 저장하는 최소 히프 //
// *bl*: *bl_list*에 있는 요소를 나타내는 변수로써 객체 번호와 d_{min}, d_{max}로 구성된다. //
// *knn*: 요소 [*oid(x)*, *dist(x)*]를 갖는 크기 *k*의 배열 //
// *knn*에 대한 삽입은 크기 순으로 이루어진다. 즉, 새 요소 *x*는 *dist(x)* 순으로 *knn* 의 올바른 위치에 삽입된다. //

1. for *i* := 0 to *k* do {
   knn[*i*].*dist* := *MAX*.
  }
2. $k\text{-}NN^{dist}$:= *MAX*.
3. *bl.node_no* := *Root*; // *Root*는 GC-트리의 루트이다.

// 단계 1
4. do {
   next_node := **ReadNodeFromGCtree**(*bl.node_no*);
   k_NN_Search(*next_node*, *q*, *k*);
  } while (**GetFromBranchList**(&*bl*) ≠ NULL);

// 단계 2
5. for *i* := 0 to *k* do {
   knn[*i*].*dist* := *MAX*;
  }
6. while (*cand_list* 에서 후보 *O*를 읽고, $d_{min}(O) \le k\text{-}NN^{dist}$ 이면) do {
  6.1. *O*에 해당하는 벡터 ***p***를 읽는다.
  6.2. If (*dist* (***p***) < $k\text{-}NN^{dist}$) {
    6.2.1. NN [*oid*(*O*), *dist* (***p***)]을 *knn*에 삽입한다.
    6.2.2. $k\text{-}NN^{dist}$:= *dist*(*k*-NN in *knn*)
    }
  }
}

Algorithm k_NN_Search(Node *N*, Query *q*, Integer *k*)

{
  1. 만약 *N* 이 리프 노드이면 {
  *N*에 있는 모든 엔트리 N_i 에 대해 {
  1.1. N_i 와 질의 객체 ***q*** 간의 d_{min}과d_{max} 를 계산한다.
  1.2. If (d_{min} ($k\text{-}NN^{dist}$) {
    1.2.1. [$oid(N_i)$, d_{min}, d_{max}]를 *cand_list* 에 삽입한다.
    1.2.2. If (d_{max} < $k\text{-}NN^{dist}$) {
      NN [$oid(N_i)$, d_{max}]를 *knn*에 삽입한다.
      $k\text{-}NN^{dist}$:= *dist*(*k*-NN in *knn*).
     }
    }
  }
 }
2. 그렇지 않으면 {
  *N*에 있는 모든 엔트리 N_i 에 대해 {
  2.1. N_i 와 질의 객체 ***q*** 간의 d_{min}과d_{max} 를 계산한다.
  2.2. [N_i, d_{min}]를 최소-히프 *bl_list*에 삽입한다.
  2.3. If (d_{max} < $k\text{-}NN^{dist}$) {
    NN [$ptr(N_i)$, d_{max}]를 *knn*에 삽입한다.
    $k\text{-}NN^{dist}$:= *dist*(*k*-NN in *knn*).

```
        }
    }
  }
}
```

4. 성능 평가

본 고차원 데이타를 위한 계층 색인 구조의 성능 평가를 위해 GC-트리에 대한 광범위한 실험을 수행하였고, IQ-tree[14], VA-file[4], LPC-file[6], 순차 탐색과 그 성능을 비교하였다. 실험은 256 MB의 메모리를 갖는 Intel Pentium III 800 MHz 프로세서와 Microsoft Windows 2000 운영 체제 하에서 수행되었다.

실험을 위해 13,724 개의 256-차원의 이미지 히스토그램 데이타를 사용하였고, 이것은 IBM QBIC[5]의 이미지 데이타베이스에서 사용한 우표 집합과 사진을 포함한다. 다양한 실험을 위해 본 논문에서는 이 이미지 데이타 집합의 분산은 그대로 유지하면서 100,000 개의 데이타 집합으로 확장하였다. 새로운 이미지 벡터 v를 생성하기 위해 원래 데이타 집합에서 한 벡터 u를 무작위로 선택하고 u가 놓이는 셀 c를 찾은 다음, 거기에서 두 개의 벡터를 선택하여 각 차원 i, $0 \le i \le 255$, 에서 평균을 계산하여 새로운 벡터 v를 만들었다.

또한 무작위 분포를 하는 256-차원의 데이타 집합과, Zipf의 법칙을 따르는 편향 분포를 하는 256-차원의 데이타 집합에 대해서도 실험을 수행하였다. Zipf 분포는 다음과 같이 정의되었고, 사용한 z 값은 0.5.이다:

$$f(i) = \frac{1/i^z}{\sum_{j=1}^{N} 1/j^z}, i = 1,2,...,N$$

모든 실험에서, 유클리드 거리 측정법 L_2를 사용하였고, 찾는 최근접 이웃의 수 k는 10으로 고정시켰다. 사용한 디스크 페이지 크기는 8 KB이다. 1,000 개의 10-NN 질의를 수행하고 그 결과를 각각 평균하였다. 질의 벡터 q는 실제 이미지 데이타 집합에서 임의로 선정되었다. 임계치 τ는 8/15이다. 즉, 한 디스크 페이지의 용량은 15이고, 한 셀이 8 개 이상의 벡터를 포함하면 클러스터로 간주한다.

4.1 검색시 디렉토리 노드의 삭제(pruning) 비율

대부분의 트리 구조의 다차원 색인 구조는 고차원 공간의 희박성으로 인해 $k\text{-}NN^{sphere}$ 및 경계 영역의 크기가 커서 k-NN 탐색 시에 디렉토리 노드를 잘라 내기가 어렵다. 따라서 디렉토리 노드를 잘라내는 비율은 색인 구조의 성능을 평가할 수 있는 좋은 척도가 된다. 그림 7은 디렉토리 노드 삭제 비율에 대한 실험 결과이다. 실제 이미지 데이타 집합에서 GC-트리의 삭제 비율은 60%를 넘는다. 이 결과로 부터, GC-트리는 이미지와 같이 클러스터된 데이타 집합에서 좋은 탐색 성능을 보임을 예측할 수 있다. 그러나, Zipf 분포와 무작위 데이타 분포를 갖는 데이타 집합에서의 삭제 비율은 실제 데이타 집합에서의 그것보다 못하다. 예를 들어 무작위 데이타 집합에서는 어떠한 노드도 삭제하지 못했다. 모든 데이타 집합에서 GC-트리는 IQ-tree보다 우수하며, GC-트리의 우수한 삭제 비율은 클러스터된 점들만 포함하는 타이트한 경계 영역에 기인한다. LPC-file과 VA-file은 전체 근사 화일을 읽기 때문에 본 실험에는 포함하지 않는다.

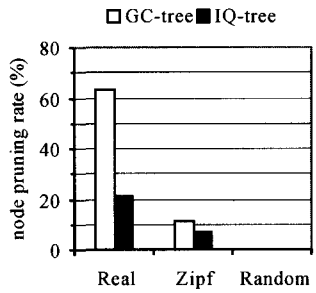


그림 7 디렉토리 노드 삭제 비율

4.2 벡터 선택률(vector selectivity) 실험

벡터 선택률은 데이타베이스에 있는 전체 벡터 수에 대한 방문한 벡터의 비율이다. k-NN 탐색 알고리즘에서, 첫 번째 단계에 대한 벡터 선택률은 첫 번째 단계에서 삭제되지 않고 남은 벡터 근사의 비율이다. 남은 근사에 대응하는 실제 벡터는 두 번째 단계에서 실제로 방문해야 할 가능성을 갖고 있다. 두 번째 단계에서의 벡터 선택률은 전체 데이타 벡터의 수에 대한 실제 방문한 벡터의 비율이다. 벡터 선택률은 디스크 접근에 대한 추정치가 되므로 검색 성능을 평가할 수 있는 좋은 측정치가 된다.

그림 8~10은 벡터 선택률 실험 결과이다. 세로 축은 벡터 선택률을 나타낸다. 결과에서 보듯이 GC-트리의 벡터 선택률은 IQ-tree, VA-file, LPC-file보다 우수하다. 이 우수성은 고밀도의 클러스터를 갖는 부공간을 찾기위해 데이타 공간을 적합하게 분할하는 GC-트리의 특성과 고밀도의 부공간에 더 많은 수의 비트를 할당하는데서 나온다. 따라서 GC-트리의 근사를 구분하는 능력은 커진다. 이것은 GC-트리가 실제 벡터 중에서 적은 부분만을 접근하는 벡터 근사 접근법의 장점을 보존함을 의미한다. GC-트리의 초점이 비록 계층 색인 기법의 장점을 벡터 근사 기법에 적용하는 것이지만, GC-트리는 각 셀 벡터에 할당하는 비트의 수를 적절히 바꿀 수 있어서 벡터 선택률을 높인다.

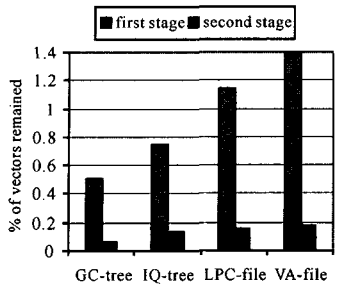


그림 8 벡터 선택률 실험(실제 이미지 데이타)

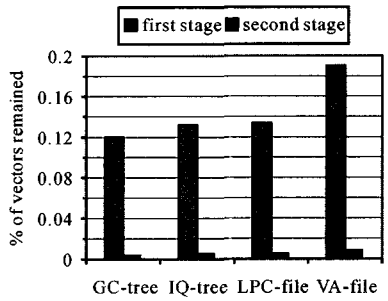


그림 9 벡터 선택률 실험(Zipf 데이타)

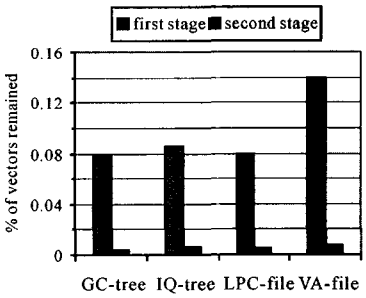


그림 10 벡터 선택률 실험(무작위 데이타)

4.3 디스크 접근 회수 실험

디렉토리 노드 삭제와 벡터 선택률 실험은 검색 성능을 추정하는데 사용될 수 있지만 그것은 근사 파일이나 색인 파일 자체를 읽는 비용을 고려하지 않으므로 검색 성능을 완전히 나타낸다고 볼 수는 없다. 그림 11은 10-NN 질의를 수행하기 위해 실제 벡터, 근사, 색인을 포함하여 전체 디스크 접근 회수를 나타낸다. 실제 이미지 데이타 집합에 대해, GC-트리에 의해 수행되는 디스크 접근 회수는 IQ-tree, VA-file, LPC-file의 그것과 비교할 때 매우 적다. 이러한 성능 향상은 GC-트리의 밀도에 기반한 데이타 공간 분할과 그 분할을 계층적으로 구성한데에 기인한다. 이 실험 결과는 데이타의 성질을 분석하여 정교한 분할 기법을 적용하면 고차원 데이타를 색인하기 위해 계층 색인 기법을 사용할 수 있음

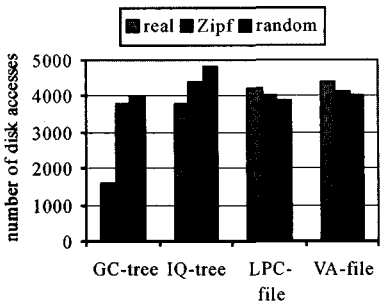


그림 11 디스크 접근 회수 실험

을 나타낸다. Zipf과 무작위 데이타 집합의 경우에는 디렉토리 노드를 잘라내는 비율이 실제 데이타 집합의 경우와 비교할 때 작으므로, GC-트리의 성능은 벡터 근사 기법(VA-file, LPC-file)과 비교하여 약간 낮거나 비슷하다.

4.4 실행 시간 실험

그림 12는 10-NN 질의를 수행하는데 걸린 시간을 보여준다. 수평 축에 있는 Scan은 실제 벡터 자체를 하나씩 검사하는 순차 탐색이다. k-NN 탐색의 디스크 접근 형태는 근본적으로 무작위이므로, 순차 탐색은 많은 디스크 시작 시간을 절약할 수 있어서 고차원 데이타 집합에 대해서 많은 무작위 디스크 접근을 하는 정교한 트리 형태의 색인 기법보다 우수한 성능을 나타내는 경우가 많다. 따라서 순차 탐색은 고차원 데이타에 대한 색인 기법의 성능을 평가하는 좋은 척도가 된다. 그림 12에서 보듯이 GC-트리는 IQ-tree, VA-file, LPC-file, 순차 탐색에 비해 훨씬 빠른 탐색 속도를 보인다.

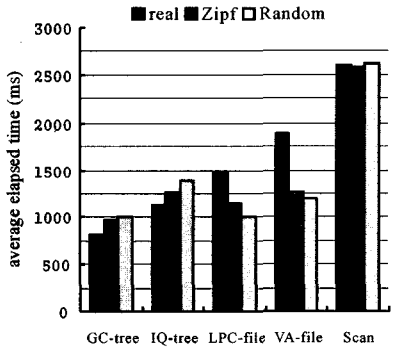


그림 12 10-NN에 대한 평균 탐색 시간

4.5 실험 인자에 대한 분석

여기에서는 실험 인자, 즉, 편중도, 군집화, 밀도 임계치, 최근접 이웃의 수의 변화에 따른 GC-트리의 성능의 변화를 살펴본다. 먼저 실험 데이타의 편중도 및 군집화에 대해 살펴보면, GC-트리의 경우, 편중도와 군집화

비율이 높을수록 상대적으로 좋은 성능을 나타낸다. 실제로 GC-트리는 데이타 집합의 편중도와 군집화되는 성질을 이용한 것이다. 본 논문의 실험에서 우리는 무작위 데이타 집합, Zipf 분포 데이타 집합, 실제 이미지 데이타 집합을 사용하였는데, 실험에 사용한 이미지 데이타의 경우, 특정 차원과 영역들에 대한 편중도와 군집화 비율이 다른 두 데이타 집합보다 매우 높다.

밀도 임계치의 경우, 그 값이 작을수록 클러스터 노드를 형성할 수 있는 조건이 약해지므로 다수의 클러스터 노드를 형성할 수 있다. 따라서 전체 노드 수가 늘어나고 트리의 키가 커질 수 있다. 반대로, 밀도 임계치 값이 클수록 GC-트리는 LPC-file과 같은 벡터 근사 접근법의 순차 파일로 접근한다. 본 논문의 실험에서는 디스크 페이지 크기를 8 KB로 하였고 밀도 임계치 τ = 8/15로 하였다. τ = 8/15, 9/15, 10/15, 11/15, 12/15에 대한 실험에서 실제로 τ = 8/15일 때 GC-트리가 상대적으로 좋은 성능을 나타내었다.

최근접 이웃의 수에 대해 살펴보면, k-NN 질의에서 k의 크기에 어느 정도 비례하여 필터링 비율과 디스크 접근 회수가 결정된다. 그림 13은 실제로 GC-트리에서 k의 크기를 변화시키면서 실제 데이타 집합에 대해 k-NN 질의를 수행함에 있어서 첫 번째 단계와 두 번째 단계에서 선택률을 실험한 결과이다. 아래 그림 13의 세로 축은 전체 벡터 수에 대해 첫 번째 단계에서 필터링되지 않고 남은 벡터의 비율과 두 번째 단계에서 실제로 읽은 벡터의 비율을 나타내고, 가로 축은 k의 크기를 나타낸다. 그림 14는 k-NN 질의 처리의 두 번째 단계에서 실제 디스크 접근 회수이다. 그림 13과 14를 살펴보면 k의 크기에 어느 정도 비례하여 선택률이 증가하고, 디스크 접근 회수가 증가함을 볼 수 있다. LPC-file, VA-file, IQ-tree에서도 그래프의 모양은 비슷한 형태로 증가하며, 상대적인 성능 비교는 거의 차이를 보이지 않는다.

Zipf 분포와 무작위 데이타 집합에서는 실제 이미지 집합과 같은 잘라내기 비율을 보여주지 못하고 있는데 그 이유는 실제 이미지 데이타 집합은 아주 클러스터되어있는 데이타이고 GC-트리는 이러한 클러스터되는 데이타의 성질을 이용한 색인 기법이다. 반면에 Zipf 분포의 데이타 집합은 상당히 편향된 분포를 갖고 있지만 실제로 많은 클러스터를 찾아내기는 힘들고, 무작위 데이타 집합에서는 클러스터를 거의 찾지 못한다. 아울러, 모든 기법들의 실제 이미지에 대한 벡터 선택률은 Zipf과 무작위의 경우보다 매우 큰 벡터 선택률을 보여주고 있다. 이러한 차이에도 불구하고 실제 데이타에 대한 디스크 접근 회수에서는 GC-트리의 결과에서만 큰 차이를 보이고 다른 기법에서는 Zipf, 무작위 분포와 큰 차

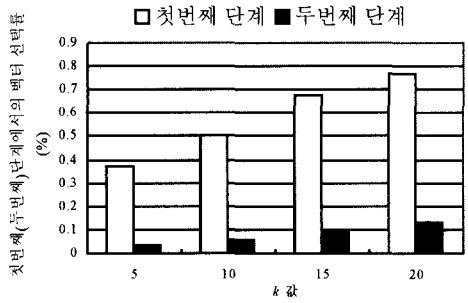


그림 13 GC-tree에서 다양한 k에 대한 선택률 성능 실험

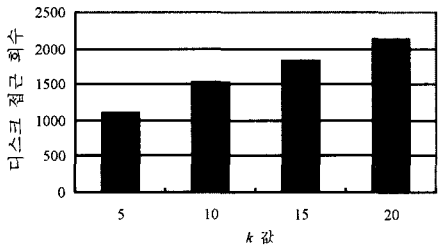


그림 14 GC-tree에서 다양한 k에 대한 디스크 접근 회수 실험

이를 보이지 않고 있는데, 그 이유는 다른 기법들은 색인을 형성하는 근사 화일을 순차적으로 다 읽어야하는 반면에 GC-트리는 색인을 형성하는 트리의 많은 부분을 탐색 시에 잘라내기 때문이다.

5. 결 론

이 논문에서 우리는 고차원 이미지 데이타 집합에 대해 k-NN 질의를 효율적으로 수행하기 위한 새로운 계층 색인 기법인 GC-트리를 제시하였다. 이 기법은 밀도 기반 공간 분할과 그 분할 계층 구조를 반영하는 색인 구조에 기반한다. 실험을 통한 성능 평가에서, GC-트리는 순차 탐색 및 최근에 개발된 벡터 근사 기법 (*VA-file, LPC-file*) 및 *IQ-tree* 보다 우수한 성능을 보임을 보였다. GC-트리 설계의 근본 아이디어는 벡터 근사 접근법의 장점과 계층 색인 기법의 장점을 결합하는데 있다. 그러나 이 두 접근법은 서로 다른 설계 원칙과 목표를 가지고 있어서 결합하기가 어려우며, 지금까지 개발된 대부분의 계층 색인 구조는 고차원 데이타 공간에서 순차 탐색보다도 성능이 나빠지는 결과를 보였다. 그러나 GC-트리는 데이타 집합을 주의 깊게 분석함으로써 이미지와 같은 고차원 데이타에 대해서도 계층 구조를 갖는 색인 기법을 사용할 수 있음을 보였고, 다양한 실험을 통해 그것을 입증하였다.

참 고 문 헌

[1] Arya, S., Mount, D.M., Netanyahu, N.S., Silverman, R., and Wu, A.Y., "An Optimal Algorithm for Approximate Nearest Neighbor Searching in Fixed Dimensions," *Journal of the ACM*, 45(6), 891~923, Nov. 1998.

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

[3] Kushilevitz, E., Ostrovsky, R. and Y. Rabani, "Efficient Search for Approximate Nearest Neighbor in High Dimensional Spaces," *Proc. of the ACM Symp. on the Theory of Computing*, 614~623, 1998.

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

[5] Niblack, N., et al., "The QBIC Project: Querying Images By Content Using Color, Texture, and Shape," *Proc. of the SPIE Conf. on Storage and Retrieval for Image and Video Databases II*, 173~187, 1993.

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

[7] 차광호, 정진완, "적응분할과 벡터근사에 기반한 고차원 이미지 색인 기법", 정보과학회논문지: 데이타베이스, 제29권 제2호, pp. 128~137, 2002. 4.

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

[9] Kanth, K.V.R., Agrawal, D. and Singh, A., "Dimensionality Reduction for Similarity Searching in Dynamic Databases," *Proc. of the ACM SIGMOD Int'l Conf.*, 166~176, 1998.

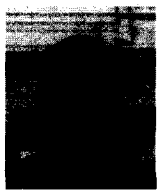
[10] Lin, K.-I., Jagadish, H.V., and Faloutsos, C., "The TV-tree: An Index Structure for High-Dimensional Data," *The VLDB Journal*, 3(4), 517~542, 1994.

[11] Megiddo, N. and Shaft, U., "Efficient Nearest Neighbor Indexing Based on a Collection of Space-Filling Curves," *Technical Report RJ 10093, IBM Almaden Research Center*, Nov. 1997.

[12] Shepherd, J., Zhu, X. and Megiddo, N., "A Fast Indexing Method for Multidimensional Nearest Neighbor Search," *Proc. of the IS&T/SPIE Conf. on Storage and Retrieval for Image and Video Databases VII*, 350~355, 1999.

[13] Berchtold, S., Boehm, C., and Kriegel, H.-P., "The Pyramid-Technique: Towards Breaking the Curse of Dimensionality," *Proc. of the ACM SIGMOD Int'l Conf. on Management of Data*, 142~153, 1998.

[14] S. Berchtold, C. Boehm, H.V. Jagadish, H.-P. Kriegel, and J. Sander, "Independent Quantization: An Index Compression Technique for High-Dimensional Data Spaces," *Proc. of the IEEE Data Engineering*, pp. 577~588, 2000.



차 광 호

1984년 부산대학교 계산통계학과 졸업(학사). 1989년 한국과학기술원 전산학과 졸업(석사). 1997년 한국과학기술원 정보및통신공학과 졸업(박사). 1986년~1987년 삼성전자(연구원). 1989년~1996년 데이콤(선임연구원). 1997년~2001년 동명정보대학교(조교수). 1999년~2000년 IBM Almaden Research Center(Visiting Scientist). 2002년~현재 숙명여자대학교 멀티미디어과학과(조교수). 관심분야는 멀티미디어 정보검색, XML 데이타베이스, Bioinformatics