CS-트리 : 고차원 데이터의 유사성 검색을 위한 셀-기반 시그니쳐 색인 구조

송 광 택† · 장 재 우††

요 약

최근 고차원 색인 구조들이 멀티미디어 데이터베이스, 데이터 웨어하우징과 같은 데이터베이스 응용에서 유사성 검색을 위해 요구된다. 본 논문에서는 고차원 특징벡터에 대한 효율적인 저장과 검색을 지원하는 셀-기반 시그니쳐 트리(CS-트리)를 제안한다. 제안하는 CS-트리는 고차원 특징 벡터 공간을 셀로써 분할하여 하나의 특징 벡터를 그에 해당되는 셀의 시그니쳐로 표현한다. 특징 벡터 대신 셀의 시그니쳐를 사용함으로써 트리의 깊이를 줄이고, 그 결과 효율적인 검색 성능을 달성한다. 또한 셀에 기반하여 탐색 공간을 효율적으로 줄이는 유사성 검색 알고리즘을 제시한다. 마지막으로 우수한 고차원 색인 기법으로 알려져 있는 X-트리와 삽입시간, k-최근접 질의에 대한 검색 시간 그리고 부가저장 공간 측면에서 성능 비교를 수행한다. 성능비교 결과 CS-트리가 검색 성능에서 우수함을 보인다.

CS-Tree : Cell-based Signature Index Strucutre for Similarity Search in High-Dimensional Data

Kwang-Taek Song† · Jae-Woo Chang††

ABSTRACT

Recently, high-dimensional index structures have been required for similarity search in such database applications as multimedia database and data warehousing. In this paper, we propose a new cell-based signature tree, called CS-tree, which supports efficient storage and retrieval on high-dimensional feature vectors. The proposed CS-tree partitions a high-dimensional feature space into a group of cells and represents a feature vector as its corresponding cell signature. By using cell signatures rather than real feature vectors, it is possible to reduce the height of our CS-tree, leading to efficient retrieval performance. In addition, we present a similarity search algorithm for efficiently pruning the search space based on cells. Finally, we compare the performance of our CS-tree with that of the X-tree being considered as an efficient high-dimensional index structure, in terms of insertion time, retrieval time for a k-nearest neighbor query, and storage overhead. It is shown from experimental results that our CS-tree is better on retrieval performance than the X-tree.

키워드 : 멀티미디어 정보 검색(multimedia information retrieval), 고차원 색인 기법(high-dimensinal index structure)

1. 서 론

최근 멀티미디어 데이터베이스, 데이터마이닝, 데이타웨어하우징, 의료 정보검색 시스템등의 데이터베이스 응용분야에서는 유사성에 기반한 검색 질의 형태가 널리 요구되고 있다[1, 2]. 유사성 검색은 주어진 질의 객체와 유사한 객체들의 집합을 찾는 질의로써, 실제 객체들간의 유사성을 측정하는 것이 아니라 객체로부터 추출된 n(≥1)개의 특징값(feature value)들간의 유사성에 기반한다. 이를 위해 먼저, 객체로부터 특징벡터를 추출하며 이러한 특징벡터간의 거리를 측정함으로써 두 객체의 유사성을 측정하게 된다.

유사성에 기반한 검색 질의 중, k-최근접 객체 탐색 질의는 주어진 질의 객체와 가장 유사한 k(≥1)개의 최근접 객체를 찾는 질의이다. k-최근접 객체 탐색 질의와 같은 유사성 질의를 얼마만큼 효율적으로 지원하느냐는 차세대 대용량 멀티미디어 데이터베이스 시스템의 주된 이슈이다. 기존의 GIS와 같은 응용 분야에서는 저장되는 객체의 특징의 수가 상대적으로 많지 않기 때문에 k-최근접 데이터를 탐색하는데 큰 문제가 되지 않는다. 하지만 데이터베이스는 매우 대량화되어지고 멀티미디어 객체로부터 추출되는 특징 벡터도 고차원화(대개 10차원 이상) 되는 경향이 있기 때문에, 이에 대한 k-최근접 데이터 탐색은 상당히 많은 디스크 접근 횟수를 요구한다. 따라서 이를 해결하기 위한 고차원 인덱싱 구조에 대한 연구가 활발히 진행되고 있다. 실제 차원이 증가함에 따라 제안되는 고차원 색인 기법들의 성능은 급격히

† 준 회 원 : 전북대학교 대학원 컴퓨터공학과
†† 종신회원 : 전북대학교 컴퓨터공학과 교수, 전기전자회로합성연구소
논문접수 : 2000년 7월 18일, 심사완료 : 2001년 6월 27일

저하하며, 10~15차원 이상에서는 순차탐색과 동일한 검색 결과를 보이는 것으로 증명되었다. 이러한 차원이 증가함에 따라 발생하는 현상을 "Dimensional Curse"라고 한다[3].

대용량의 멀티미디어 데이터베이스에서 보다 빠른 k-최근접 탐색 질의를 지원하는 것은 멀티미디어 정보 검색과 같은 응용분야에서는 매우 중요한 시스템 요구 사항이다. 따라서 본 연구에서는 차원이 증가함에 따라 발생하는 트리-기반 고차원 색인 기법의 비효율성을 극복하기 위해 특징 벡터의 시그니쳐(signature)를 이용한 새로운 고차원 색인 구조를 제안한다. 제안하는 셀-기반 시그니쳐 트리(CS-트리 : Cell-based Signature Tree)는 고차원 특징 벡터 공간을 셀로써 분할하고 특징 벡터는 셀의 시그니쳐로 표현되어 트리에 저장된다. 특징 벡터 대신 시그니쳐를 사용하여 트리의 깊이가 낮아짐으로써 검색을 효율적으로 수행할 수 있다. 또한 셀에 적합한 새로운 가지치기 거리를 이용한 유사성 검색 알고리즘을 제시한다.

본 논문의 구성은 다음과 같다. 2장에서는 고차원 색인 구조에 대한 기존 연구를 분석한다. 3장에서는 본 논문에서 제안한 시그니쳐를 이용한 새로운 고차원 색인 구조를 제시하고, 아울러 이에 기반한 k-최근접 탐색 질의 알고리즘을 기술한다. 4장에서는 제안한 고차원 색인 구조의 성능을 평가하고, 마지막으로 5장에서는 결론 및 향후 연구 방향을 제시한다.

2. 관련 연구

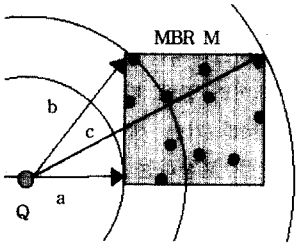
2.1 트리-기반 고차원 색인 기법

대용량의 멀티미디어 데이터베이스를 다루는데 있어서 주된 이슈는 유사성 검색을 효율적으로 지원하는 것이다. 최근 이를 위한 다수의 색인 기법들이 연구되었다. 첫째, SS-트리(Similarity Search tree)[4]는 유사도를 평가하는 척도를 사용하여 질의와 이미지 데이터의 유사성을 비교하여 유사도가 높은 이미지 데이터를 검색하는데 적합하도록 설계된 동적인 색인 구조이다. R-트리 계열의 색인 구조[5]가 데이타 공간을 MBR (Minimum Rectangle Region)로 분할하는 반면, SS-트리는 구 영역(spherical region)으로 데이타 공간을 분할한다. 이는 유사성에 기반한 검색을 용이하게 하지만 겹침 영역이 많이 발생함으로써 검색성능을 저하시킨다. 둘째, SR-트리[6]는 SS-트리와 비슷한 구조로써 노드를 MBR과 구(spherical region)로써 표현하며 SS-트리의 겹침 영역 문제를 해결하기 위해 제안 되었다. 그러나 노드에 MBR 정보와 구 영역(spherical region) 정보를 전부 포함하고 있기 때문에 트리의 팬-아웃(fan-out)이 떨어져 차원이 증가함에 따라 성능이 급격히 저하된다. 셋째, TV-트리[7]는 차원이 증가함에 따라 트리 노드의 팬-아웃에 따른 성능 저하 문제를 해결하기 위해 제안되었다. TV-트리의 기본 개념은 고차원 특징 벡터를 색인하기 위해 사용되는 차원의 수를 객체의 수와 트리의 깊이에 따라 가변적으로 적용하는 것이다. 즉, 루트 노드에 근접한 노드들에 대해서는 단지 몇 개의 차원만을 사용해서 높은 팬-아웃을 갖고, 트리의 깊이가 깊어질수록 보다 많은 차원들을 사용함으로써 더욱 노드들의 분별력을 높이도록 하였다. 따라서 기존의 색인 기법들이 특징 벡터 차원의 수가 증가할 수록 검색 속도와 저장 공간의 요구량이 급속하게 증가되는 문제점을 해결하고, 고차원의 특징 벡터를 효율적으로 다룬다. 하지만 특징들이 중요도에 따라 우선 순위가 부여되어야 하는 전제를 포함하고 있다. 마지막으로, X-트리[8]는 기존의 색인구조들이 차원이 증가함에 따라 겹침영역이 증가하여 검색성능이 현저히 저하되는 문제점을 방지하기 위해 제안된 고차원 색인 구조이다. X-트리 구조는 분할시 겹침영역(overlap)이 최소가 되지 못할 때는 분할하지 않고 노드의 크기가 가변적으로 확장될 수 있는 슈퍼노드를 사용한다. X-트리 구조는 저차원에서는 계층적인 디렉토리 구조를 사용하고, 고차원으로 갈수록 겹침영역이 증가되기 때문에 기억공간이 절약되고 빠른 접근이 가능한 선형적인 디렉토리 구조를 사용한다. X-트리는 고차원 특징벡터 공간에서 k-최근접 탐색 질의와 같이 유사성에 기반한 검색을 효율적으로 지원하는 적절한 색인 구조의 하나로 평가됨에도 불구하고, 차원이 증가함에 따라(16차원이상) 순차탐색과 동일한 검색 성능을 보이는 것으로 증명되었다[3].

2.2 k-최근접 객체 탐색

Roussopoulos는 R-트리를 이용하여 다차원 특징 벡터 공간에서 주어진 질의 객체와 가장 가까운 객체의 점을 찾는 방법을 제시하였다[9]. 이 방법은 트리에서 검색되는 노드의 수를 줄이기 위해 다차원 검색 공간에서 주어진 질의 점으로부터 객체 또는 검색 공간을 나타내는 최소경계사각형까지의 최소거리(minimum distance : MINDIST)와 최대거리(min-max distance : MINMAXDIST)를 이용한 가지치기 전략을 이용한다. 탐색 알고리즘에서 가지치기(pruning)란 원하는 객체를 포함하지 않는 노드들을 접근 후보 노드 리스트에서 제거하는 연산이고, 가지치기 전략은 후보 노드 리스트에서 제거할 노드들을 찾기 위한 방법이다. 가지치기 전략에서 MINDIST와 MINMAXDIST를 이용하여 불필요한 탐색 영역을 제거함으로서 탐색 영역을 줄이고자 하였다. MINDIST는 질의 점 Q에서 최소경계사각형 M의 가장 가까운 변 또는 점까지의 최소거리로서 정의되며, MINMAXDIST는 질의 점 Q와 객체를 포함하고 있는 최소경계사각형 M과의 최대거리들 중에서 최소거리이다. 즉, Q로부터 MBR M을 구성하는 모든 축에서 가장 먼 점까지의 거리들 중에서 최소가 되는 거리를 의미한다. 반대로 MAXDIST는 그 중에서 최대가 되는 거리를 의미한다. (그림 1)은 질의 점과 최소경계사각형과의 MINDIST, MINMAXDIST 그리고 MAXDIST를 나타낸다. MAXDIST는 질의 점 Q로부터 MBR M에 있는 모든 객체를 포함할 수 있는 거리이다.

최적의 k-최근접 질의 알고리즘은 트리를 검색하는 동안



(그림 1) MINDIST와 MINMAXDIST

k-최근접 거리와 겹치는 모든 노드를 방문한다[10, 11]. 방문할 필요가 없는 노드들을 방문대상에서 제외시키기 위해 MINDIST와 MINMAXDIST를 조합하여 다음과 같은 전략을 사용한다.

S1. 질의 점 Q로부터 k번째 최소경계사각형 R'까지의 MINMAXDIST(Q, R')보다 MINDIST(Q, R)가 더 큰 값을 갖는 최소경계사각형 R이 존재하면, R은 k-최근접 이웃 객체를 포함하지 않기 때문에 R에 해당하는 노드는 검색대상에서 제외한다.

S2. 질의 점 Q로부터 객체 O까지의 거리가 k번째 최소경계사각형 R에 대한 MINMAXDIST(Q, R)보다 크다면, 객체 O를 k-최근접 객체 대상에서 제외한다.

S3. 질의 점 Q로부터 k번째 객체 O까지의 거리보다 큰 MINDIST(Q, R)를 갖는 모든 최소경계사각형 R은 검색대상에서 제외한다.

3. 셀-기반 시그니쳐 트리

3.1 셀-기반 시그니쳐 트리 구조

트리 구조의 고차원 색인 구조에서 특징 벡터의 차원이 증가함에 따라 특징 벡터가 노드에서 차지하는 비중은 그 만큼 커지게 된다. 즉 노드의 팬-아웃은 감소하고 트리의 깊이는 증가하여, 이는 결국 성능 저하를 초래하게 된다. 이를 극복하고자 본 논문에서는 특징 벡터의 요약으로서 시그니쳐에 기반한 색인 구조를 제안한다. 예를 들어, 10차원의 특징 벡터는 <0.15, 0.31, 0.0, 0.5, 0.76, 0.84, 0.66, 0.32, 0.77, 0.98>으로 표현되며, 여기서 각 차원의 값은 정규화를 통해 D[0, 1]의 값을 갖도록 표현된다. 이러한 고차원 데이터에 대한 시그니쳐 표현은 특징 벡터의 각 차원의 값에 대한 요약들의 집합이므로, 각 차원에 해당하는 벡터의 성질은 유지하되 비트 단위로 표현되므로 벡터 크기를 현저히 줄일 수 있다. 이를 위해 데이터 공간을 셀(cell) 단위로 나누고, 셀에 대한 시그니쳐를 생성한다. 이렇게 만들어진 시그니쳐는 각각의 셀을 대표하는 값이 되며, 셀에 대한 정보를 포함하게 된다. 또한 차원에 따라 데이터 공간을 균등하게 분할함으로써, 셀에 대한 시그니쳐는 데이터 공간에서의 어떤 영역의 범위를 가지게 되고, 시그니쳐를 통해 쉽게 이 범위값을 구할 수 있다.

일반적인 MBR(minimum bounding rectangle)은 트리-기반 색인 구조에서 자식 데이터를 모두 포함하는 최소 경계 영역으로써, 최소 경계(lower bound)와 최대 경계(upper bound)로써 표현할 수 있다. 즉 d차원 특징 벡터 공간과 이의 공간에서의 MBR R은 ((l0, u0), (l1, u1), ···, (ld-1, ud-1))이 된다. 이러한 MBR 정보는 특징벡터의 두 배에 해당하는 저장 공간을 요구함으로써 차원이 증가함에 따라 트리 노드의 팬-아웃이 감소하게 된다. 이러한 MBR을 시그니쳐로 변환하여 저장하기 위하여 먼저 MBR을 셀에 기반한 CMBR(Cell-based MBR)로 변환한다. 이에 대한 변환은 정의 1과 같다.

정의 1. 각 차원축에 대한 최저값이 0, 최대값이 1인 D[0, 1]의 d차원 특징 벡터 공간과 이의 공간에서의 MBR $R = ((l_0, u_0), (l_1, u_1), \cdots, (l_{d-1}, u_{d-1}))$이 주어졌을 때, 이에 해당하는 셀 기반 공간에서의 CMBR CR은 다음과 같이 정의된다.

$$CR = ((cl_0, cu_0), (cl_1, cu_1), \cdots, (cl_{d-1}, cu_{d-1}))$$
$$cl_i = \lfloor l_i \times 2^b \rfloor \times (1/2^b),\ cu_i = \lceil u_i \times 2^b \rceil \times (1/2^b)$$
(단, $0 \le i < d$)

여기서 b 는 각 차원에 대해 할당된 시그니쳐 비트 수이다. 만일 b = 2일 경우, 각 차원에 대해 할당된 시그니쳐 비트 수는 2가 되기 때문에 각 차원 축은 0과 1사이로 정형화되었을 때 0.25, 0.5, 0.75으로 구분되는 4개의 영역으로 분할되게 된다. b의 값이 크면 특징벡터 공간은 보다 세밀한 셀 공간으로 분할되게 되어 가지치기 거리인 MINMAXDIST가 작아짐으로써 가지치기 효과를 증가시킬 수 있는 반면, 고차원의 특징벡터를 시그니쳐로 저장하는 효과를 상실하게 되는 문제점을 갖게된다. 본 연구에서는 실험에 의해 b = 8일 때 최적의 성능을 얻을 수 있어 그 값을 사용한다. MBR R의 1차원에 대한 최저값이 0.35, 최대값이 0.78 이면, 정의 1에 의해 CMBR CR의 1차원에 대한 최저값은 0.25, 최대값은 1이 된다. 이렇게 생성한 CMBR에 대해 시그니쳐를 생성하여 저장하며 시그니쳐 생성은 정의 2와 같다.

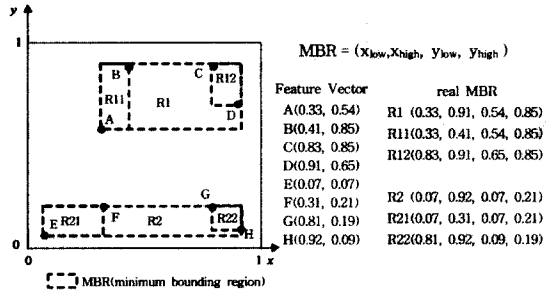
정의 2. 셀 기반 공간에서의 CMBR $CR = ((cl_0, cu_0), (cl_1, cu_1), \cdots, (cl_{d-1}, cu_{d-1}))$ 에 대한 시그니쳐 S(CR)은 다음과 같이 생성된다.

$$S(CR) = \{Sl_0Su_0Sl_1Su_1 \cdots Sl_iSu_i\},\ (\text{단, } 0 \le i < d)$$
$$Sl_i = SIG(cl_i \times 2^b),\ Su_i = SIG(cu_i \times 2^b - 1).$$

SIG는 주어진 값을 2진수로 b-비트만큼 표현하는 함수이다. 예로써, CR의 1차원에 대한 최저값이 0.25, 최대값이 1이라면 시그니쳐는 01, 11이 된다.

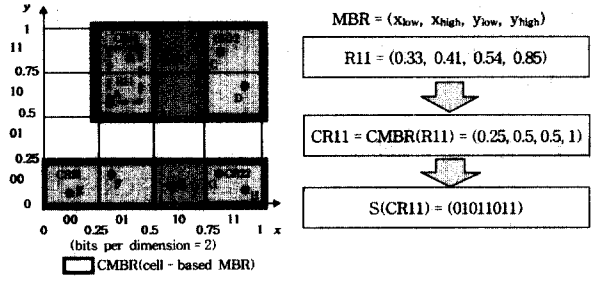
(그림 2)는 2차원 특징 벡터 공간에서 객체들과 객체들을 포함하는 MBR들을 나타내고 있다. 여기에서 객체 포인트는 A(0.33, 0.54)부터 H(0.92, 0.09)까지 존재하고, 포인트 A

와 B를 포함하는 MBR R11(0.33, 0.41, 0.54, 0.85)와 포인트 C와 D를 포함하는 MBR R12(0.83, 0.91, 0.65, 0.85)가 존재한다. 아울러 R11과 R12를 포함하는 MBR R1(0.33, 0.91, 0.54, 0.85)이 존재한다. 마찬가지로 R21(0.07, 0.92, 0.07, 0.21), R22(0.81, 0.92, 0.09, 0.19)가 있으며 이들을 포함하는 R2 (0.07, 0.92, 0.07, 0.21)가 존재한다.



(그림 2) 2차원 특징벡터 공간

(그림 3)은 각각의 차원에 대해서 2-비트 시그니처를 사용할 경우, 2차원 공간에서의 셀에 의한 공간 분할을 나타낸다. 각 차원의 시그니처 크기를 2비트로 할당하므로 각 차원에 대한 데이터 공간은 4개의 부분으로 나뉘어져, 2차원 공간에서 총 16개의 셀을 가지게 된다. 차원에 따른 데이터 공간에서 표현할 수 있는 객체들의 실제값이 0에서 1사이인 경우, 각각의 셀을 나누는 점의 좌표는 0, 0.25, 0.50, 0.75, 1 이 된다. 한편 객체 포인트 A, B를 포함하는 실제 MBR R11의 셀에 기반한 CMBR CR11은 정의 1에 의해 (0.25, 0.5, 0.5, 1)이 된다. 또한 이에 대한 최종적인 시그니처 S(CR11)은 정의 2에 의해 (01011011)이 된다. 마찬가지로 이와 같은 과정을 통해 모든 MBR에 대한 최종적인 시그니처를 구할 수 있다. <표 1>은 (그림 2)에서의 각각의 MBR들에 해당하는 CMBR과 이에 대한 시그니처를 나타낸다.

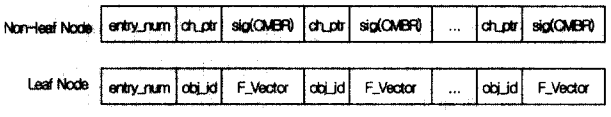


(그림 3) 2차원 셀 공간에서의 CMBR

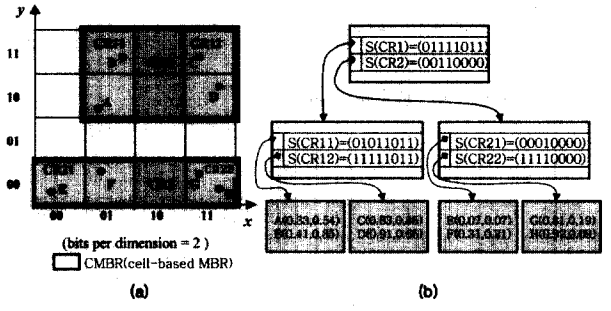
〈표 1〉 (그림 2)의 셀 공간에서 MBR의 시그니처 변환

CMBR	S(CMBR)
CR1 = (0.25, 1 , 0.5 , 1)	(01111011)
CR11 = (0.25, 0.5 , 0.25 , 1)	(01011011)
CR12 = (0.75, 1 , 0.5 , 1)	(11111011)
CR2 = (0 , 1 , 0 , 0.25)	(00110000)
CR21 = (0 , 0.5 , 0 , 0.25)	(00010000)
CR22 = (0.75, 1 , 0 , 0.25)	(11110000)

본 논문에서 제안하는 셀-기반 시그니처 트리(CS-트리)는 기존의 트리-기반 색인 구조에 시그니처를 적용한 방법으로, 트리 내부 노드에 실제 특징 데이터 값에 따른 정보를 시그니처로써 저장하는 방법이다. 트리-기반 색인 구조의 중간 노드는 자식 노드의 객체를 모두 포함하는 경계영역(bounding region)으로 표현된다. CS-트리에서는 중간 노드(internal node)에 셀을 포함하는 CMBR에 대한 시그니처(sig(CMBR))를 저장하고, 리프노드(leaf node)에 객체로부터 추출한 특징벡터(F-Vector)를 저장한다. (그림 4)는 노드의 구조를 보이고 있다. (그림 5)는 (그림 3)의 특징벡터 및 CMBR에 대한 CS-트리의 구조를 나타낸다. 예를 들어, 전체 데이터 공간은 셀로서 분할이 되며, 객체 A, B를 포함하는 CR11과 객체 C, D를 포함하는 CR12 가 CR1 에 포함될 때, 실제 객체(즉 A, B)는 트리의 리프 노드에 저장이 되며 CR1, CR11, CR12 의 CMBR은 시그니처로 변환되어 트리의 중간노드에 저장된다. (그림 5)의 (b)는 (a)의 각 객체들과 CBMR의 시그니처가 트리 구조로 저장되어 있는 것을 나타낸다.



(그림 4) 셀-기반 시그니처 트리의 노드 구조



(그림 5) 셀-기반 시그니처 트리 구조

제안하는 CS-트리에서의 삽입 알고리즘은 다음과 같다. 먼저 트리의 루트로부터 시작하며, 삽입할 객체 포인트를 포함할 최적의 서브 경로를 찾는다. 객체를 삽입하므로써 변경되는 노드간의 경계 영역, 노드 영역의 증가, 또는 겹침 영역등을 고려한다. 라인 1에서는 객체가 삽입될 리프노드를 찾아간다. 객체를 포함하는 리프노드가 존재할 경우 해당 리프노드가 선택되며, 객체를 포함하는 리프노드가 존재하지 않을 경우, 객체를 포함함으로써 경계영역이 최소확장을 하는 리프노드를 선택하게 된다. 라인 4에서는 리프노드에 객체를 삽입한다. 이때 리프노드에서 오버플로우(overflow)가 발생하면 노드를 분할하게 되는데, 트리 구조에 바탕을 둔 CS-트리의 분할 알고리즘은 좌표축의 하나에 대해 수직 분할 방법을 사용한다. 분할 축의 선택은 경계 영역, 노드 영역의 증가, 또는 겹침등의 최소화 등과 같은 위

상적인 특성을 고려한다. 분할과정은 먼저 분할 차원을 선택하기 위해 최소 Margin를 갖는 차원 축을 선택하며 앤트리 개수 분할은 해당 차원 축에 대해 정렬한 다음 분할 했을 경우 겹침영역과 DeadSpace가 최소가 되는 축을 선택하게 된다. DeadSpace는 경계영역에서 객체가 존재하지 않는 영역을 의미한다. 자식노드에서 발생하는 분할로 인하여 변경되는 구조는 상위 노드의 CMBR 시그니쳐에 반영시킨다. 한편 삽입시, 객체 특징 벡터가 한 곳에 밀집되더라도 트리의 전체적인 구성은 균형 트리 형태를 유지한다. 하지만 객체 특징 벡터가 한 곳에 많이 밀집한 경우, 노드간의 겹침이 발생하게 되어 평균 검색 성능이 저하하는 문제점을 갖는다. 밀집된 형태의 분포를 갖는 데이터 집합에 대해서 특징 벡터 공간을 보다 세밀한 셀로써 분할할 경우 이러한 문제점을 보완할 수 있다.

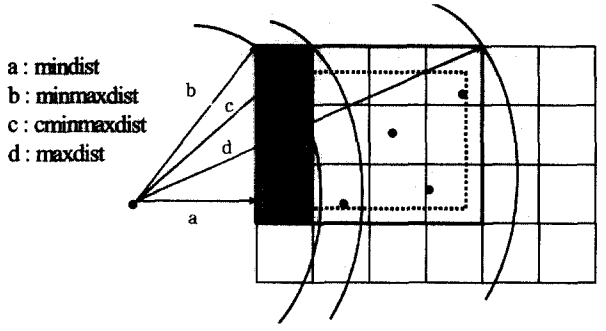
Input :
An index tree ***T***, input feature object ***p***;
Output :
The new CS-Tree that results after the insertion of ***p***;
Variable :
1. Choose the best branch to follow, descend three until the leaf node L is reached ;
2. Insert the feature object ***p*** into the leaf node L ;
3. If leaf L overflows then
4. Split the leaf into two leaves;
5. Extract the CMBRs from the signature of CMBRs that have been changed
6. Update the signature of CMBRs that have been changed ;
7. Split an internal node if overflow occurs;
8. End.

(Algorithm Insertion)

3.2 유사성-기반 검색 알고리즘

본 논문에서 제안하는 CS-트리의 중간 노드는 셀에 기반한 경계(bounce) 시그니쳐를 저장하고 있다. 따라서 트리의 노드의 MBR은 이러한 셀을 포함하는 영역이 된다. 기존의 MINMAXDIST는 MBR 내의 적어도 하나의 객체를 포함할 수 있는 거리를 의미한다. 따라서 k-최근접 탐색에서 k번째 MINMAXDIST는 질의 객체와 가장 가까운 k개의 객체를 보장할 수 있는 거리이다[9]. 제안하는 CS-트리에서의 CMBR은 객체들의 최소경계영역이 아니라 셀들의 최소경계영역이기 때문에, 셀-기반 시그니쳐 트리에서는 기존의 k-최근접 탐색에서 사용했던 가지치기 거리 MINMAXDIST 를 그대로 사용할 수 없다.

(그림 6)은 이러한 문제점을 보이고 있다. b는 질의 포인트의 MBR에 대한 기존 MINMAXDIST를 의미하고 있다. 하지만 이는 셀에 대한 공간 분할 특성으로 인하여 적어도 하나의 객체를 포함하고 있음을 보장하지 못한다. 따라서 본 논문에서는 셀을 저장하고 있는 노드들의 CMBR에 대해 적어도 하나의 객체를 포함할 수 있음을 보장할 수 있는 새로운 거리 개념인 CMINMAXDIST를 정의한다.



(그림 6) MINMAXDIST와 CMINMAXDIST의 예

정의 3. 임의의 점 P와 CMBR R 간의 거리 CMINMAXDIST(P, R)은 다음과 같이 정의한다.

$$CMINMAXDIST(P,R) = \min_{1\le k\le n}\left(|p_k - crm_k|^2 + \sum_{\substack{i\neq k \\ 1\le i\le n}} |p_i - crM_i|^2\right)$$

여기에서 crmk는 CMBR R의 평면들(hyper-plains) 중 각 차원 축과 직교하는 두 개의 평면들 중 점 P로부터 가까운 쪽의 평면까지의 거리에 적어도 하나의 셀을 포함할 수 있도록 cd를 고려한 거리이다. cd는 하나의 셀 폭을 의미한다.

$$crm_k = \begin{cases} s_k + cd & \text{if } p_k \le \frac{s_k+t_k}{2} \wedge p_k < s_k \\ s_k & \text{if } p_k \le \frac{s_k+t_k}{2} \wedge p_k \ge s_k \\ t_k - cd & \text{if } p_k > \frac{s_k+t_k}{2} \wedge p_k > t_k \\ t_k & \text{if } p_k > \frac{s_k+t_k}{2} \wedge p_k \le t_k \end{cases}$$

crMi는 각 차원 축과 직교하는 두 개의 평면들 중 점 P로부터 먼 쪽의 평면까지의 거리로서 다음과 같이 정의된다.

$$crM_i = \begin{cases} s_i & \text{if } p_i \le \frac{s_i+t_i}{2} \\ t_i & \textit{Otherwise.} \end{cases}$$

CMINMAXDIST(P, R) 거리는 셀의 특징을 고려하여 CMBR CR 내의 셀 중에서 객체가 저장되어 있는 적어도 하나의 셀을 포함할 수 있는 거리를 보장함으로써 이를 이용한 정확한 가지치기를 수행할 수 있다.

제안하는 k-NN 알고리즘은 새로운 거리 CMINMAXDIST에 기반하고 있다. 큐 리스트 PQ는 트리에서 탐색해야 할 노드들의 리스트이며, *dmin*(*q*, *Reg*(*N*))에 의해 노드들을 정렬하여 가지고 있다. 알고리즘은 k-최근접 객체를 보장할 수 있는 거리 k_nn_dist와 겹치는 모든 영역을 탐색한다. 이는 최적의 검색 알고리즘에 기반한다[11]. 라인5에서 k_nn_dist는 PQ에 저장되어 있는 노드들에 대한 *dcmax(k)* 과 리프 노드에서 발견한 객체에 대한 *NNq(k)* 중에서 작은값으로 재계산된다. 라인 6은 알고리즘에 의한 탐색이 k_nn_

dist 보다 작은 거리의 dmin(q, Reg(N))를 갖는 노드가 PQ에 존재하지 않을 때 중단됨을 보인다.

```
Input :
    index tree T, query object q ;
Output :
    NNq (k) : k-nearest neighbor of q ;
Variable :
    PQ : queue list of node to search
    Nc : current node ;
    k_nn_dist : distance of k-nearest neighbor ;
    Reg (Nc) : CMBR, region of Nc ;
    d (q, pi) : distance between point q and point pi ;
    dmin (q, Reg (N)) : MINDIST between q and Reg (N)
    dcmax (k) : CMINMAXDIST(k) of q in PQ

1. Initialize PQ with a pointer to the root node of T ;
2. Compute k_nn_dist; Let k_nn_dist = ∞ ;
3. While PQ ≠ ∅ do :
4.     Extract the first entry from PQ, referencing node N ;
5.     Compute k_nn_dist : Let k_nn_dist = min (dcmax (k),
                                k_nn_dist ) ;
6.     If dmin (q, Reg (N)) ≥ k_nn_dist then exit, else read N ;
7.     If N is a leaf node then :
8.     For each point pi in N do :
9.     If d (q, pi ) < k_nn_dist then :
10.    Insert pi into NNq (k ), k_nn_dist = d (p, pi ) ;
11.    else : // N is an internal node
12.    For each child node Nc of N do :
13.    If dmin(q, Reg (Nc )) < k_nn_dist :
14.    Update PQ performing an ordered insertion of the pointer
                                to Nc ;
15.    Update dcmax (k ) in PQ ;
16.    End.
```

(Algorithm k-NN)

4. 실험 평가

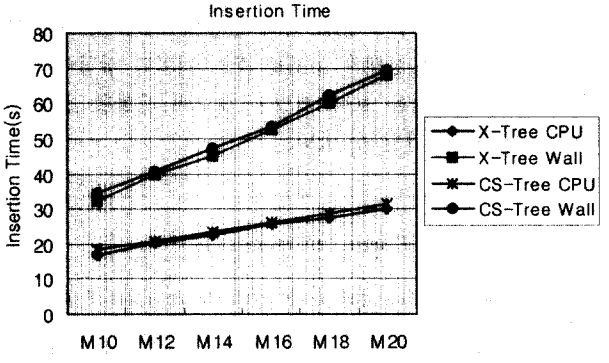
본 논문에서 제안한 CS-트리는 SUN Enterprise 150상에서 Solaris 2.6 운영체제에서 구현되었다. 아울러 성능평가를 위해 각각 100,000건의 Synthetic 데이터 셋(set)과 Real 데이터 셋을 이용한다. 랜덤하게 생성한 Synthetic 데이터 셋은 균등 분포를 이루고 있다. Real 데이터 셋은 애니메이션 MPEG 동영상에서 100,000 개의 I-프레임으로부터 추출한 색상 특징 벡터 셋이다. 본 장에서 X-트리와 삽입 시간, k-최근접 질의에 대한 검색 시간 그리고 부가저장공간 측면에서 성능 비교를 수행한다. <표 2>는 실험에서 사용한 CS-트리 구성의 인자를 나타낸다.

〈표 2〉 실험인자

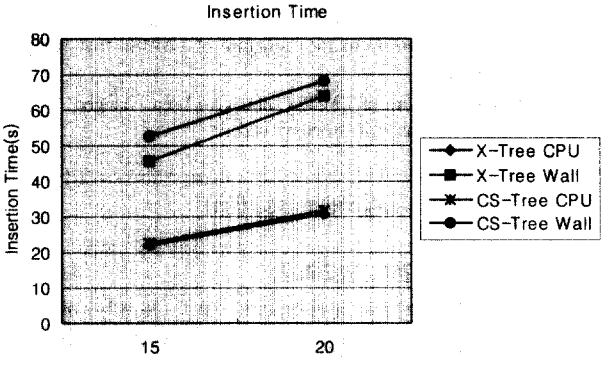
	Synthetic Data	Real Data
실험데이타 수	10,0000	
페이지 크기	4,096 bytes	
검색인자 k	100	
시그니쳐 비트수 b	8	

4.1 삽입 시간

삽입 시간은 각 차원에 따른 특징 벡터들로 구성된 데이터 100,000 건을 인덱스 파일에 저장할 때 소요되는 시간이다. (그림 7)의 (a)는 Synthetic 데이터를 인덱스에 삽입하는 경우 소요되는 시간을 나타낸다. 가로축은 차원을 나타내며, 세로축은 전체 데이터 100,000 건을 삽입하는 경우 걸리는 시간이다. 삽입시간은 크게 CPU 시간과 Wall 시간으로 구분되어진다. CPU 시간은 메모리에서의 연산 시간을 의미하며, Wall 시간은 디스크 I/O 시간을 포함한 전체 시간을 의미한다.



(a) Synthetic 데이터



(b) Animation 데이타

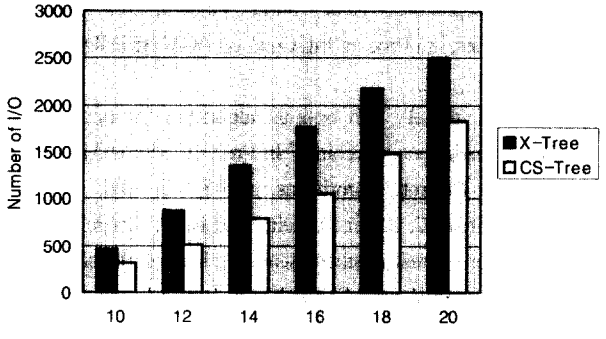
(그림 7) 삽입 시간

CPU 시간과 Wall 시간 측면에서 모두 X-트리와 CS-트리는 유사한 삽입성능을 나타낸다. 전체 삽입 시간인 Wall 시간에서 X-트리는 10차원 데이터일 경우 약 32초, 20차원 데이터일 경우 약 68초가 소요되며, 제안하는 CS-트리는 10차원 데이터일 경우 약 34초, 20차원 데이터일 경우 약 69초가 소요된다. CS-트리에서는 특징 벡터에 대한 시그니쳐 표현을 위한 연산이 추가되어 X-트리보다 시간이 약간 더 필요하며, 이는 전체 삽입 시간에 비해서는 매우 미약하다. 아울러 (그림 7)의 (b) Animation 데이터의 경우, 전체 삽입 시간 측면에서 X-트리는 10차원 데이터일 경우 약 45초, 20차원 데이터일 경우 약 64초가 소요되며, 제안하는 CS-트리는 15차원 데이터일 경우 약 52초, 20차원 데이터일 경우 약 68초가 소요된다. 전체적으로 Synthetic 데이터

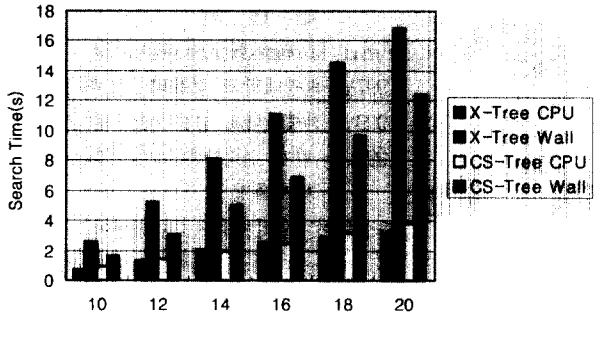
와 Animation 데이터 모두에 대해 CS-트리의 삽입 시간은 X-트리와 비슷한 결과를 나타낸다.

4.2 검색 시간

검색 성능 평가를 위해 대표적인 유사성 검색 질의인 k-최근접 질의 검색에 대한 검색시간(search time)을 측정한다. 성능평가를 위해 탐색 과정에서 요구되는 디스크 I/O와 탐색 시간을 측정한다. k-최근접 질의의 검색 인자 k는 100 이며 질의 횟수는 100번으로 평균 값을 이용한다. (그림 8)의 (a)는 k-최근접 질의를 위해 요구되는 디스크 I/O 횟수를 나타낸다. X-트리는 10차원의 경우 디스크 I/O 접근 횟수가 467번, 20차원의 경우 2502번이며, CS-트리는 10차원의 경우 디스크 I/O 접근 횟수가 305번, 20차원 일 경우 1829번으로 전체적으로 감소한다. 아울러 (그림 8)의 (b)의 전체 검색 시간인 Wall 시간에서 X-트리는 20차원일 경우 약 17초, CS-트리는 약 12초로, CS-트리가 보다 우수한 검색성능을 보인다.



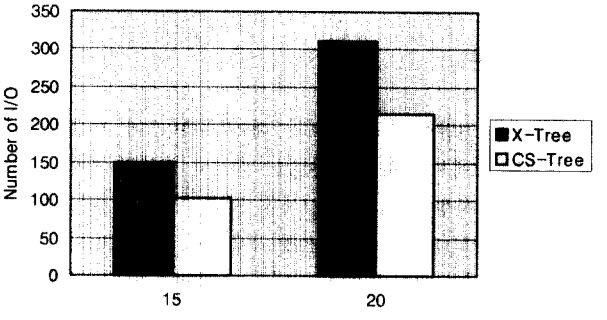
(a) 디스크 페이지 I/O 횟수



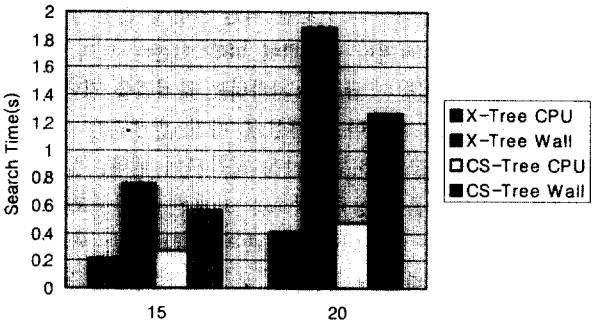
(b) 검색 시간 (CPU, Wall)

(그림 8) Synthetic 데이터 셋에 대한 k-NN 검색 성능 평가 (k=100)

(그림 9)는 Real 데이터에 대한 k-최근접 질의 검색(k=100)을 위한 디스크 I/O 횟수와 검색시간을 나타낸다. 검색 시간에서 CPU 시간은 X-트리와 CS-트리가 비슷한 결과를 보인 반면, 디스크 I/O 횟수의 감소로 인하여 전체 탐색 시간인 Wall 시간은 X-트리에 비해 CS-트리가 많은 성능 개선을 보이고 있다. 즉, X-트리는 15차원 데이터에 대해 약 0.8초, 20차원일 경우 약 1.9 초가 소요된 반면, CS-트리는 15차원 데이터에 대해 약 0.6초, 20차원 일 경우 약 1.3초를 요구된다. 애니메이션 데이터에 대한 성능 결과는 제안하는 CS-트리가 X-트리에 비해 전체 탐색시간인 Wall 시간을 기준으로, 15차원과 20차원에서 평균 약 50% 이상의 우수한 검색 성능을 보인다.



(a) 디스크 페이지 I/O 횟수



(b) 검색 시간 (CPU, Wall)

(그림 9) Real 데이터 셋에 대한 k-NN 검색 성능 평가 (k=100)

4.3 부가 저장 공간

부가 저장 공간 비율은 인덱스 파일에 100,000건의 데이터를 저장하여, 원래의 데이터 셋의 사이즈를 기준으로 인덱스 파일에서 부가적으로 사용한 저장공간 비율을 의미한다. X-트리 및 CS-트리와 같은 트리-기반 구조는 데이터에 따라 전체 노드의 구조가 영향을 받으므로 부가 저장 공간이 데이터의 특성에 따라 다르게 나타난다. 순수 데이터를 저장하는데 필요한 저장 공간을 기준으로 각 색인 기법에서 요구되는 부가저장공간 비율을 보면, Synthetic 데이터에 대해 X-트리는 약 260%의 부가저장 공간을 요구하며, CS-트리는 약 255%의 부가저장 공간을 요구하여 비슷한 성능을 나타낸다. 마찬가지로 Real 데이터에 대해서도 부가저장공간 측면에서 X-트리와 비슷한 성능을 나타낸다. <표 3>은 두 기법의 부가저장공간 비율을 나타낸다. 부가저장 공간 측면에서는 시그니쳐를 저장하는 CS-트리가 X-트리에 비하여 약간 더 적은 부가저장공간을 요구한다.

〈표 3〉 부가저장공간

		X-Tree	CS-Tree
Sythetic Data Set	10 차원	262%	258%
	12 차원	259%	255%
	14 차원	258%	251%
	16 차원	257%	251%
	18 차원	261%	254%
	20 차원	262%	253%
Real Data Set	10 차원	382%	375%
	15 차원	342%	330%

5. 결 론

유사성에 기반한 검색 질의는 데이터베이스에 저장된 대량의 객체들 중에서 사용자가 원하는 객체와 유사한 객체를 찾는 질의이다. 유사성에 기반한 검색을 위해서는 객체로부터n(>1) 개의 특징 벡터(feature vector)를 추출하여 데이터베이스에 저장한 다음, 사용자가 검색하려고 하는 객체의 특징 벡터와 가장 유사한 값을 갖는 객체를 찾는다. k-최근접 데이터 탐색 질의는 유사성에 기반한 대표적인 검색 질의로서, 이를 효율적으로 지원하기 위해 다수의 고차원 색인 기법에 관한 연구가 수행되었다. 그러나 차원이 증가함에 따라 기존의 트리-기반 색인 구조의 검색 성능은 급격히 저하하는 문제점을 지닌다.

본 연구에서는 차원이 증가함에 따라 발생하는 기존 트리 기반 색인 구조의 비효율성을 극복하기 위해, 셀 영역에 기반한 시그니처를 이용한 새로운 고차원 색인구조를 제안하였다. 제안하는 CS-트리는 기존의 트리 기반 색인 기법이 차원 증가에 따라 팬-아웃이 급격히 떨어지는 문제점을 해결하고자, 특징벡터의 시그니처(signature)를 저장한다. 또한 효율적인 k-최근접 탐색 질의를 지원하기 위해 셀-기반 k-최근접 탐색 알고리즘을 제안하였다. 아울러 본 연구에서 제안한 CS-트리를 기존 고차원 색인 구조인 X-트리와의 성능 비교를 수행한 결과, 삽입 시간은 비슷한 반면, 평균 50%의 검색 성능이 개선됨을 보였다.

앞으로의 연구방향은 대용량의 데이터베이스로부터 보다 빠른 유사성 검색을 지원하기 위한 근사 k-최근접 탐색 알고리즘을 연구하는 것이다.

참 고 문 헌

[1] C. Faloutsos, W. Equitz, M. Flickner, W. Niblack, D. Petkovic, and R. Barber, "Effecient and effective querying by image content," Journal of Intelligent Information Systems, Vol.3, No.3-4, pp.231-262, July. 1994.

[2] J. M. Hellerstein, J. F. Naughton, and A. Pfeffer, "Generalized search trees for database systems," In Proc. of the 21st Int. Conf. on VLDB, pp.562-573, Sept. 1995.

[3] V. Pestov, "On the geometry of similarity search : Dimensionality curse and concentration of measure," Technical Report RP-99-01, School of Mathematical and Computing Sciences, Victoria University of Wellington, New Zealand, January. 1999.

[4] D. A. White and R. Jain, "Similarity Indexing with the SS-tree," In Proc. of the 12th Int. Conf. on Data Engineering, pp.516-523, 1996.

[5] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree : an efficient and robust access method for points and rectangles," In Proc. of Int. Conf. on ACM SIGMOD, pp.322-331, 1990.

[6] N. Katayama and S. Satoh, "The SR-tree : an index structure for high-dimensional nearest neighbor queries," In Proc. of Int. Conf. on ACM SIGMOD, pp.369-380, 1997.

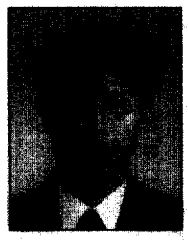
[7] K. I. Lin, H. Jagadish, and C. Faloutsos, "The TV-tree : an index structure for high dimensional data," VLDB Journal, Vol.3, pp.517-542, 1994.

[8] S. Berchtold, D. Keim, and H.-P. Kriegel, "The X-tree : an index structure for high-dimensional data," In Proc. of the 22nd Int. Conf. on VLDB, pp.28-39, 1996.

[9] N. Roussopoulos, S. Kelley, and F. Vincent, "Nearest neighbor queries," In Proc. of Int. Conf. on ACM SIGMOD, pp.71-79, 1995.

[10] S. Arya, et. al., "An optimal algorithm for approximate nearest neighbor searching," In Proc. ACM-SIAM Symposium on Discrete Algorithms, pp.573-582, 1994.

[11] S. Berchtold, C. Bohm, D. Keim, and H.-P. Kriegel, "A cost model for nearest neighbor search in high-dimensional data space," In Proc. of ACM PODS Symposium on Principles of Database System, pp.78-86, 1997.



송 광 택

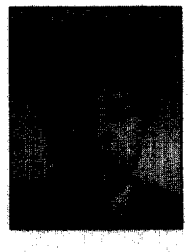
e-mail : ktsong@dblab.chonbuk.ac.kr

1997년 원광대학교 컴퓨터공학과(공학사)

1999년 전북대학교 컴퓨터공학과(공학석사)

1999년~현재 전북대학교 컴퓨터공학과 박사과정

관심분야 : 멀티미디어 데이타베이스, 멀티미디어 정보검색, 고차원 색인기법



장 재 우

e-mail : jwchang@dblab.chonbuk.ac.kr

1984년 서울대학교 전자계산기공학과 (공학사)

1986년 한국과학기술원 전산학과(공학석사)

1991년 한국과학기술원 전산학과(공학박사)

1996년~1997년 Univ. of Minnesota, Visiting Scholar.

1991년~현재 전북대학교 컴퓨터공학과 부교수

관심분야 : 멀티미디어 데이타베이스, 멀티미디어 정보검색, 하부저장구조