산업기술연구(강원대학교 산업기술연구소 논문집), 제20권, 2000.
Journal of Industrial Technology, Kangwon Nat'l Univ., Korea, No. 20, 2000.

Singular Value Decomposition 기반 고차원 인덱스 구조

A High-Dimensional Index Structure Based on Singular Value Decomposition

김 상 욱* Charu Aggarwal** Philip S. Yu***
Kim, Sang-Wook Charu Aggarwal Philip S. Yu

Abstract

The nearest neighbor query is an important operation widely used in multimedia databases for finding the object that is most similar to a given query object. Most of techniques for processing nearest neighbor queries employ multidimensional indexes for effective indexing of objects. However, the performance of previous multidimensional indexes, which use N-dimensional rectangles or spheres for representing the capsule of the object cluster, deteriorates seriously as the number of dimensions gets higher. This paper proposes a new index structure based singular value decomposition resolving this problem and the query processing method using it. We also verify the superiority of our approach through performance evaluation by performing extensive experiments.

Keywords : multimedia databases, nearest neighbor queries, high dimensional indexing

I. 서론

최근접 질의(nearest neighbor query)는 멀티미디어 데이터베이스에서 유사한 객체를 찾기 위하여 가장 널리 사용되는 질의의 하나이다[3][8][10][11]. 기존의 최근접 질의 처리 기법들은 대부분 객체들의 효과적인 인덱싱을 위하여 R*-트리[2], X-트리[3], SR-트리[14] 등의 트리 구조를 가지는 다차원 인덱스(multidimensional index)를 사용한다. 그러나 기존의 다차원 인덱스들은 차원 수가 높아짐에 따라 그 성능이 크게 떨어지는 것으로 알려져 있다[2][12].

본 논문에서는 최근접 질의를 효과적으로 처리할 수 있는 새로운 인덱싱 방법을 제시하고자 한다. 또한, X-트리, 순차 검색 등 기존 기법들과의 다양한 실험을 통한 성능 평가에 의하여 제안된 기법의 우수성을 검증한다.

II. 연구 동기

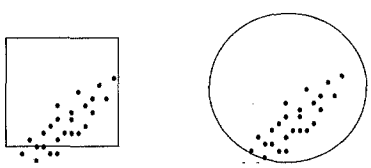
최근접 질의를 처리하기 위한 기존의 기법들은 브랜치-앤드-바운드(branch and bound)를 기반으로 한다. 브랜치-앤드-바운드는 전체 탐색 공간에서 주어진 상한 값(upper bound) 이상인 영역들을 이후의 조사 대상에서 제외시킴으로써 탐색을 효과적으로 수행하는 전통적인 방법이다[5]. 기존의 기법들

* 강원대학교 컴퓨터정보통신공학부 조교수
** Software Tools and Techniques Team, IBM T.J. Watson Research Center, Research Staff Member
*** Software Tools and Techniques Team, IBM T.J. Watson Research Center, Manager

본 연구는 한국과학재단 99 해외 Post-Doc 방문 연구 프로그램과 강원대학교 멀티미디어연구센터를 통한 정보통신부 정보통신 우수대학원 사업의 연구비 지원을 받았습니다.

[3][8][10][11]은 다차원 인덱스를 이용하여 전체 객체들을 다수의 객체 클러스터(cluster)들의 집합으로 구분하고[2], 주어진 질의 점으로부터 현재까지 구한 가장 가까운 최근접 객체(nearest neighbor object)보다 먼 곳에 있는 클러스터는 미리 조사 대상에서 제외하는 방식을 사용한다.

다차원 인덱스내의 각 엔트리는 고유의 클러스터와 대응되며, 클러스터내의 모든 객체 점들을 포함하는 N차원 캡슐(capsule)을 사용하여 클러스터를 표현한다. 좋은 검색 성능을 얻기 위해서는 캡슐이 클러스터내의 객체 점들을 모두 포함하되, 내부의 죽은 영역(dead space)[1]의 크기가 가능한 작게 형성되어야 한다. 현재, 제안된 대부분의 다차원 인덱스에서는 캡슐의 형태로서 N차원 사각형(rectangle)을 사용하며, 몇몇 기법에서는 N차원 원(sphere)을 사용하기도 한다[4][7][13]. 캡슐의 형태로 사각형 혹은 원을 사용하는 기존의 기법에서는 클러스터내에 상관 관계(correlation)[16]가 커질수록 캡슐내의 죽은 영역의 크기가 커진다. 그림 2.1은 2차원 공간에서 상관 관계를 가지는 한 객체 클러스터를 투영한 것이다. 클러스터가 축들 간에 상관 관계를 갖는 경우, 사각형과 원의 형태를 가지는 기존의 캡슐 표현 방식에서는 죽은 영역이 매우 커짐을 볼 수 있다. 축들간의 상관 관계가 커질수록 이러한 죽은 영역은 커지게 된다. 이러한 죽은 영역의 증가는 최근접 질의의 처리 성능을 저하시키는 주요 원인이 된다.



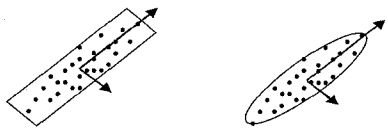
(a) 사각형 형태의 캡슐. (b) 원 형태의 캡슐.

그림 2.1. 상관 관계가 있는 클러스터에서의 캡슐의 형태와 죽은 영역의 관계

III. 제안하는 기법

3.1. 축 시스템 변환

본 논문에서는 죽은 영역이 증가하는 문제를 해결하는 근본적인 방법으로서 해당 클러스터에 맞는 새로운 축 시스템(axis system)을 사용하는 것을 제안한다. 즉, 해당 클러스터 내의 객체들의 분포를 조사함으로써 죽은 영역을 최소화할 수 있는 축 시스템을 채택하고, 이를 기준으로 해당 클러스터를 포함하는 캡슐을 결정하는 것이다. 그림 3.1(a)는 그림 2.1에 나타난 클러스터를 새로운 축 시스템과 사각형 캡슐을 이용하여 표현한 것이다. 두 화살표는 새롭게 채택된 새로운 축 시스템을 의미한다. 이렇게 새로운 축 시스템을 사용하는 경우, 죽은 영역의 크기가 현저하게 감소함을 볼 수 있다. 차원의 수가 커질수록 이러한 죽은 영역의 감소 효과는 더욱 커지게 된다.



(a) 사각형 캡슐의 사용. (b) 타원 캡슐의 사용.

그림 3.1. 변환된 축 시스템의 적용

3.2. 캡슐 형태의 표현

본 논문에서는 원과 사각형을 조합함으로써 죽은 영역의 크기를 최소화하는 다양한 형태의 N차원 캡슐을 표현하는 방식을 이용한다. 먼저, $\{r_1, r_2, ..., r_k\}$를 구한다. 여기서, r_i는 변환된 축 시스템에서 클러스터내의 객체들이 갖는 i-번째 축의 평균값이다. 이후부터는 클러스터의 분포 형태에 맞도록 변환된 새로운 축 시스템을 기준으로 논의를 전개한다. 또한, 설명의 편의상 $r_1 \le r_2 \le r_3 \ldots \le r_k$ 라 가정한다[3]. 다음에는 $\{ r_i \}$를 $m(\le k)$개의 그룹 $[r_1, .., r_{f(1)}]$, $[r_{f(1)+1}, .., r_{f(2)}]$, ..., $[r_{f(m-1)+1}, r_{f(m)}]$로 분할한다. 여기서, $f(i)$는 $r_{f(i)}/r_{f(i-1)+1} < 2$를 만족하도록 선택된다. 예를 들어, $\{ r_i \}$가 {1, 2, 3, 5, 8, 12, 15, 22, 26, 40}인 10차원 공간 클러스터의 경우, 위의 방법을 적용하면, [1, 2], [3, 5], [8, 12, 15], [22, 26, 40]의 네 그룹이 형성된다.

클러스터를 위한 캡슐은 다음의 식들 표현하는 다차원 원들의 교집합으로 구성되는 공간에 의하여 결정된다. $x_1^2 + ... + x_{f(1)}^2 \le a_1^2$, $x_{f(1)+1}^2 + ... + x_{f(2)}^2 \le a_2^2$, $x_{f(2)+1}^2 + ... + x_{f(3)}^2 \le a_3^2$, ..., $x_{f(m-1)+1}^2 + ... + x_{f(m)}^2 \le a_m^2$. 여기서, a_i는 i-번째 그룹에 의하여 표현되는 다차원 원의 지름을 의미한다. 그림 3.2는 3차원 공간에서 클러스터의 분포에 따라 가능한 모든 분할 방식과 이에 의한 캡슐의 형태를 도면화 한 것이다.

2) 대부분의 다차원 인덱스들은 트리 구조를 사용하므로, 인덱스의 상위 단계에서 하위 단계로 내려갈수록 보다 세분화된 클러스터들을 만나게 된다.

3) 물론, 일반적인 경우에는 이 조건이 만족하지 않으므로 먼저 r_i의 값에 따라 정렬해야 한다.

3.3. 아웃 라이어 처리

캡슐의 기본 형태는 분할의 결과에 의하여 결정되지만, 각 그룹의 형태를 결정하는 원의 지름 혹은 사각형의 변은 a_i 값에 의하여 결정된다. 가장 간단한 방법의 하나는 중심점과 이로부터 가장 멀리 떨어진 객체와의 거리를 a_i 값으로 이용하는 것이다. 이 경우, 클러스터 내의 모든 객체들은 결정된 캡슐 내에 포함된다.

그러나 이 방법의 큰 문제점은 죽은 영역이 매우 커질 수 있다는 것이다. 그림 3.3에 나타난 클러스터 c1의 예를 살펴보자. 대부분의 객체들은 2차원 공간의 인접한 위치에 밀집해 있는 반면, 하나의 객체 o1은 이와 매우 떨어진 위치에 존재한다. 모든 객체를 포함하도록 캡슐 1과 같은 형태를 취하는 경우 죽은 영역이 지나치게 많아지고, 이 결과 최근접 질의의 성능이 크게 저하된다. 반면, 지나치게 동떨어진 객체를 클러스터부터 제외하고, 캡슐 2와 같은 형태를 취하면 대부분의 죽은 영역을 제거할 수 있다. 이때, 클러스터로부터 제외되는 객체를 아웃 라이어(outlier)라 정의한다.

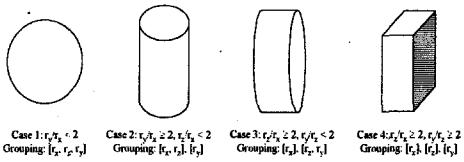


그림 3.2. 3차원 공간에서 캡슐 형태의 결정

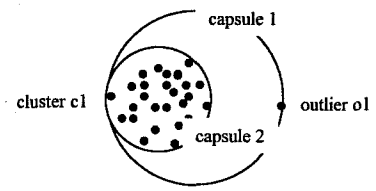


그림 3.3. 클러스터, 아웃 라이어, 캡슐의 관계

본 연구에서는 ai 값을 먼저 결정함으로써 해당 캡슐의 형태를 결정하고, 이 캡슐 밖에 존재하는 객체들은 모두 아웃 라이어라 간주한다.

아웃 라이어는 클러스터와 독립적으로 관리되므로 질의 점의 위치에 관계없이 항상 액세스해야 하는 객체들임을 의미한다. 따라서 아웃 라이어 수가 지나치게 많아지면, 이를 별도로 처리하는 부분에서 병목현상이 발생하게 된다. 본 연구에서는 위의 정리 1과 다양한 실험에 의한 분석 과정을 통하여 a_i를 위한 합리적인 값으로서 $\mu+2\sigma$를 이용하기로 결정하였다.

3.4. 인덱스 구조 및 질의 처리 방안

그림 3.4에 나타난 바와 같이 제안된 인덱스 구조는 다수의 엔트리들로 구성되는 디렉토리와 페이지들의 집합으로 구성된다. 디렉토리는 주기억장치 내에서 관리되며, 페이지는 디스크 내에서 관리된다. 각각의 디렉토리 엔트리는 클러스터와 일대일 대응되며, 클러스터에 적합한 축 시스템, 그룹 수, 각 축이 참여하는 그룹 번호, 각 그룹의 ai 값을 정보로서 갖는다. 실제 객체들은 디스크 내에 존재하는 페이지 내에 저장된다. 같은 클러스터 내에 포함되는 객체들의 페이지들은 연결 리스트(linked list) 형태로 관리되며, 이 연결 리스트의 시작 페이지를 해당 디렉토리 엔트리가 가리키게 된다. 디렉토리 엔트리 수는 응용 환경에 따라 조절할 수 있다.

이러한 인덱스는 이미 존재한 객체들의 집합을 대상으로 다음과 같은 방식으로 일괄 구성된다: (1) 각 클러스터를 위한 시드 객체를 객체 집합으로부터 무작위로 선택한다. (2) 각 객체를 차례로 읽어들여 소속될 클러스터를 파악한다. 소속될 클러스터를 파악하는 기준으로는 해당 객체와 시드 객체간의 거리의 역을 사용한다. (3) 각 클러스터에 적합한 최적의 축 시스템을 결정한다. 주어진 클러스터에 가장 적합한 축 시스템은 Principal Component Analysis에서 널리 사용되는 SVD(singular value decomposition) 기법[6]을 이용한다. (4) 각 클러스터를 위한 캡슐의 형태와 ai 값을 결정한다. (5) 각 클러스터에서 캡슐 밖에 존재하는 아웃 라이어들을 파악하여 마지막 디렉토리 엔트리에 옮긴다.

이러한 인덱스 구조를 기반으로 최근접 질의는 다음과 같은 방식으로 처리된다. (1) 마지막 디렉토리 엔트리가 가리키는 아웃 라이어를 조사하여 질의 점과 가장 가까운 객체 Omin과 이 객체가 질의 점으로부터 떨어진 거리를 Dmin을 찾아낸다[4]. (2) 마지막 엔트리를 제외한 모든 엔트리들을 질의 점과 해당 엔

4) 마지막 디렉토리 엔트리가 가리키는 아웃 라이어들은 모든 클러스터의 캡슐 밖에 있는 객체들을 수집하여 함께 저장한 것으로서 질의 조건에 관계없이 항상 조사되어야 한다. 이들의 조사를 다른 엔트리들의 조사가 끝난 후, 최종적으로 하는 것을 고려할 수도 있다. 그러나 O_{min}에 해당되는 객체가 아웃 라이어로 존재하는 경우, 클러스터들을 모두 조사해야 하는 경우가 발생하므로 커다란 성능 저하를 유발한다. 반면, 이들을 가장 먼저 조사하는 제안된 기법에서는 작은 D_{min} 값을 조기에 확보함으로써 O_{min}을 포함할 가능성이 없는 클러스터를 미리 조사 대상에서 제외시킬 수 있는 장점이 있다.

트리가 표현하는 캡슐간의 거리 Dentry를 계산하고, 이 값을 기준으로 정렬한다[5]. (3) 이 정렬된 순서로 각 엔트리를 조사하여 엔트리가 조건 (Dmin > Dentry)를 만족하면, 그 엔트리와 대응되는 클러스터를 조사하여 새로운 Dmin과 Omin을 찾아낸다. 만일, 이 조건을 만족하지 않으면, 현재까지 찾은 Dmin과 Omin을 반환하고, 질의 처리를 종료한다.

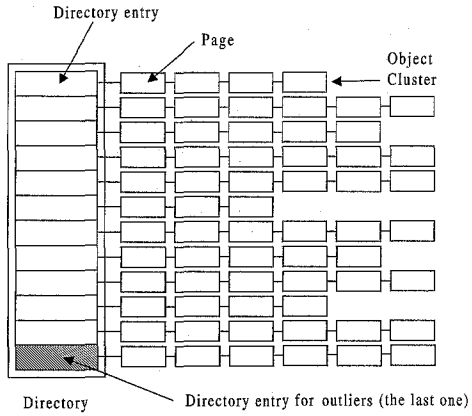


그림 3.4. 인덱스 구조

Ⅳ. 성능 평가

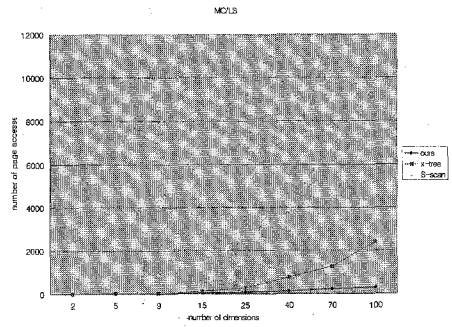
본 실험에서는 제안된 기법의 비교 대상으로서 X-트리[2]와 순차 검색(sequential scan)을 채택한다. 성능 평가 지수로는 DBMS 연산의 비용에 대부분을 차지하는 페이지 액세스 수를 사용한다. 페이지 크기로는 실제 DBMS에서 가장 널리 사용하는 4K 바이트를 세 가지 기법 모두에서 공통적으로 사용한다. 제안된 기법에서 디렉토리 엔트리의 수는 1,000으로 설정하였다. 실험에서 사용된 객체 집합들은 2차원에서 100차원까지의 다양한 차원 수를 가진다. 각 객체 집합은 클러스터들의 집합으로 구성되며, 전체 100,000개의 객체들을 포함한다. 객체 집합은 클러스터 내의 객체 수에 따라 다음과 같이 분류한다.

- MC(many clusters): 500 ~ 1,000의 객체들을 가지는 많은 수의 클러스터들이 존재한다.
- FC(few clusters): X를 5,000 ~ 10,000의 객체들을 가지는 적은 수의 클러스터들이 존재한다.

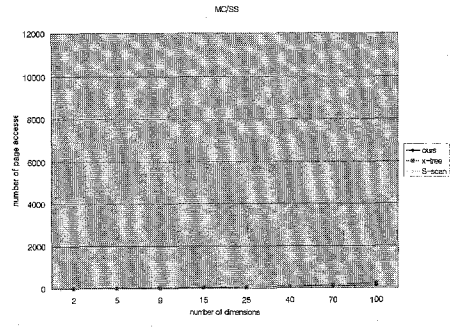
또한, 각 클러스터가 차지하는 공간 내의 영역 크기에 따라 다음과 같이 분류한다.

- LS(large standard deviation): 클러스터내의 객체들이 넓은 공간 내에서 분포한다.
- SS(small standard deviation): 클러스터내의 객체들이 좁은 공간 내에서 분포한다.

이러한 X, Y의 가능한 값의 설정에 따라 실험에서는 MC/LS, MC/SS, FC/LS, FC/SS의 네 가지 조합의 객체 세트들을 사용한다. 그림 4.1에서와 같이 최근접 질의 처리를 대상으로 한 실험 결과에 의하면, 제안된 기법은 15차원 이하의 저차원 객체 집합에 대해서는 X-트리와 유사한 성능을 보였다. 또한, 25차원 이상의 고차원 객체 집합에 대해서는 객체 분포에 관계없이 X-트리 및 순차 검색에 비하여 우수한 성능을 보이는 것으로 나타났다. 이것은 제안된 기법이 고차원 데이터의 인덱싱을 위한 매우 효과적인 방법임을 의미하는 것이다.

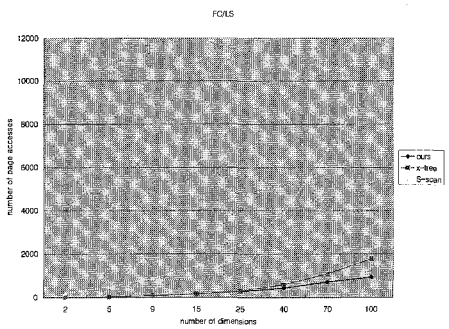


(a) MC/LS

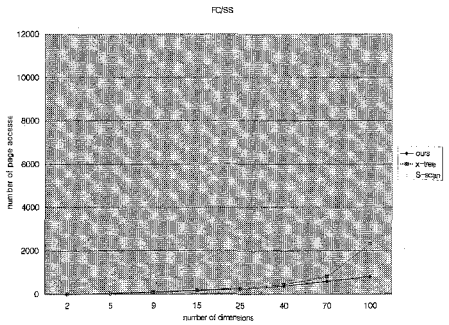


(b) MC/SS

5) 정렬은 실제 디렉토리 엔트리 자체를 대상으로 하는 것이 아니라 <entryNum, distance>의 쌍으로 구성되는 요약 엔트리를 대상으로 한다. entryNum은 디렉토리 엔트리 번호를 의미하며, distance는 질의 점과 entryNum과 대응되는 클러스터의 캡슐간의 거리를 의미한다. 이와 같이 요약 엔트리를 사용하도록 함으로써 정렬시의 CPU 시간을 최적화 할 수 있다.



(c) FC/LS



(d) FC/SS

그림 4.1. 차원 수 증가에 따르는 최근접 질의 처리 성능

V. 결론

멀티미디어 데이터베이스 응용에서는 최근접 질의의 효과적인 지원이 매우 중요하다. 이를 위하여 제안된 기존의 다차원 인덱스들은 차원 수가 높아짐에 따라서 성능이 급격히 저하되는 문제점을 가진다. 본 논문에서는 이러한 성능 저하가 클러스터 캡슐내의 죽은 영역의 크기 증가로 인한 것임을 밝히고, 이를 해결하기 위한 새로운 인덱싱 및 최근접 질의 처리 방안을 규명하기 위하여 다양한 실험을 통한 성능 평가를 수행하였다. 향후 연구 방향으로는 (1) 제안된 인덱스를 트로 구조로 확장하는 방안, (2) 제안된 기법의 k-최근접 질의로의 적용 방안, (3) 최근접 질의 처리 절차의 병렬 수행 방안, (4) 제안된 인덱스의 동적 환경으로의 적용 방안 등을 고려하고 있다.

Acknowledgment

본 논문은 정보통신부에서 주관하는 정보통신 우수시범학교 지원사업에 의하여 수행되었음.

참 고 문 헌

[1] N. Beckmann et al., "The R*-tree: an Efficient and Robust Access Method for Points and Rectangles," In Proc. Intl. Conf. on Management of Data, ACM SIGMOD, pp. 322-331, May 1990.

[2] S. Berchtold, D. A. Keim, and H.-P. Kriegel, "The X-tree: An Index Structure for High-Dimensional Data," In Proc Intl. Conf. on Very Large Data Bases, VLDB, pp. 28-39, 1996.

[3] S. Berchtold et al., "Fast Nearest Neighbor Search in High-Dimensional Space, In Proc. Intl. Conf. on Data Engineering, IEEE, pp. 209-218, 1998.

[4] P. Ciaccia, M. Patella, P. Zezula, "M-tree: An Efficient Access Method for Similarity Search in Metric Spaces," In Proc Intl. Conf. on Very Large Data Bases, VLDB, pp. 426-435, 1997.

[5] E. Horowitz, S. Sahni, Fundamentals of Computer Algorithms, Computer Science Press, 1978.

[6] I.T. Jolliffe, Principal Component Analysis, Springer-Verlag 1986.

[7] N. Katayama and S. Satoh, "The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries," In Proc. Intl. Conf. on Management of Data, ACM SIGMOD Conference, pp. 369-380, 1997.

[8] F. Korn et al., "Fast Nearest Neighbor Search in Medical Image Databases, In Proc. Intl. Conf. on Very Large Data Bases, VLDB, pp. 215-226, 1996.

[9] A. M. Law and W. D. Kelton, Simulation Modeling and Analysis, McGraw-Hill Company, New York, 1982.

[10] N. Roussopoulos, S. Kelley, F. Vincent, "Nearest Neighbor Queries," In Proc. Intl. Conf. on Management of Data, ACM SIGMOD, pp. 71-79, 1995.

[11] T. Seidl and H.-P. Kriegel, "Optimal Multi-Step k-Nearest Neighbor Search," In Proc. Intl. Conf. on Management of Data, ACM SIGMOD, pp. 154-165, 1998.

[12] R. Weber, H.-J. Schek, and S. Blott, "A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces, In Proc. Intl. Conf. on Very Large Data Bases, VLDB, pp. 194-205, 1998.

[13] D. A. White and R. Jain, "Similarity Indexing with the SS-tree," In Proc. Intl. Conf. on Data Engineering, IEEE, pp. 516-523, 1996.