고차원 데이터의 효율적인 최근접 객체 검색 기법

김 진 호† · 박 영 배††

요 약

피라미드 기법은 n-차원 공간 데이터를 1차원 데이터로 변환하여 B+-트리로 표현하며, n-차원 데이터 공간에서 하이퍼큐브 영역질의 처리로 발생하는 "차원의 저주현상"에 영향을 받지 않게 검색 시간 문제를 해결하고 있다. 또 구형 피라미드 기법(SPY-TEC)은 피라미드 기법의 공간 분할 전략을 응용하여 유사도 검색에 적합한 구 영역질의 방법을 사용하고 검색 성능을 개선하고 있다. 하지만 유사도 검색의 응용에서 영역질의는 범위를 지정하는데 어려움이 있어 최근접 질의가 더 효율적이며, 기존의 제안된 인덱스 기법들은 특정 분포의 데이터에 대해서만 우수한 성능을 보이는 단점이 있다. 따라서 이 논문에서는 멀티미디어 데이터와 같은 고차원 데이터의 검색 성능을 향상시키기 위해 제안되었던 PdR-트리를 이용하여 최근접 객체 검색 기법을 제안한다. 다양한 분포의 모의 데이터와 실제 데이터를 이용하여 실험한 결과, PdR-트리가 피라미드 기법과 구형 피라미드 기법보다 검색 성능이 향상되었음을 보이고 있다.

Efficient Searching Technique for Nearest Neighbor Object in High-Dimensional Data

Jin-Ho Kim† · Young-Bae Park††

ABSTRACT

The Pyramid-Technique is based on mapping n-dimensional space data into one-dimensional data and expresses it as a B+-tree. By solving the problem of search time complexity the pyramid technique also prevents the effect of "phenomenon of dimensional curse" which is caused by treatment of hypercube range query in n-dimensional data space. The SPY-TEC applies the space division strategy in pyramid method and uses spherical range query suitable for similarity search so that improves the search performance. However, nearest neighbor query is more efficient than range query because it is difficult to specify range in similarity search. Previously proposed index methods perform well only in the specific distribution of data. In this paper, we propose an efficient searching technique for nearest neighbor object using PdR-Tree suggested to improve the search performance for high dimensional data such as multimedia data. Test results, which uses simulation data with various distribution as well as real data, demonstrate that PdR-Tree surpasses both the Pyramid-Technique and SPY-TEC in views of search performance.

키워드 : PdR-트리(PdR-Tree), 피라미드 기법(Pyramid-Technique), 구형 피라미드 기법(SPY-TEC), 최근접 질의(Nearest Neighbor Query)

1. 서 론

내용 기반 이미지 검색(CBIR : Content Based Image Retrieval)이나 증권시장의 주식성향의 시 순차 분석(Time Sequence Analysis)등의 비 정형화된 데이터를 취급하는 응용에서는 고차원 데이터를 위한 인덱스 기법과 정확하게 일치하는 데이터 검색보다는 유사도(similarity)를 기반으로 한 데이터를 검색하는데 중점을 두고 있다.

비 정형화된 응용에서 고차원 데이터를 처리하기 위한 대표적인 인덱스 기법으로 R-트리[19], R*-트리[18], Hilbert R-트리[16], TV-트리[13], X-트리[10], SR-트리[9]가 있고, 시계열 패턴을 위한 인덱스 기법으로 dR-트리[23] 등이 제안되었다. 그러나 제안된 인덱스 기법들은 검색 시 차원이 증가함에 따라 노드간의 겹침현상(overlap)으로 인하여 검색 성능이 순차 검색에 미치지 못하는 비효율적인 인덱스 구조가 된다.

이러한 문제점을 해결하기 위하여 탐색공간을 2d개의 피라미드로 분할하여 고차원 데이터를 1-차원 값으로 변환하는 피라미드 기법[4]을 구 형태로 분할하도록 구형 피라미드 기법(SPY-TEC)[21]이 제안되었고, 구형 피라미드에 dR-트리를 이용한 형태의 고차원의 동적 인덱스 구조인 PdR(Pyramid doughnuts Range)-트리[20]가 제안되었다.

유사도 검색(similarity search)을 이용한 응용에서는 두

† 준 회 원 : 명지대학교 대학원 컴퓨터공학과
†† 정 회 원 : 명지대학교 컴퓨터공학과 교수
논문접수 : 2003년 10월 13일, 심사완료 : 2004년 2월 3일

가지 종류의 질의가 존재한다[12]. 하나는 주어진 객체에 주어진 유사도 범위 안의 모든 객체를 검색하는 범위 질의(Range Query)이고, 다른 하나는 주어진 객체에 가장 유사한 k개의 객체를 검색하는 최근접 질의(Nearest Neighbor Query)이다.

내용기반 이미지 검색의 응용에서 '질의 이미지와 모양이 일정 수준 비슷한 모든 이미지를 검색하라'와 '질의 이미지와 모양이 가장 비슷한 이미지를 검색하라'는 두 가지 질의가 있다고 가정해보자. 전자는 범위 질의의 예로서 질의 이미지의 주어진 유사도 범위를 만족하는 이미지가 검색하기 위해 유사도 범위를 지정하는데 어려움이 많아 사용이 어렵다는 단점이 있다. 반면에 후자는 최근접 질의의 예로서 지정된 유사도 범위가 아닌 질의 이미지를 중심으로 유사도 범위를 확장하면서 검색하므로 일반 사용자가 사용하기에는 더 유용하다고 할 수 있다.

이 연구에서는 영역질의에 대해 성능향상을 보인 PdR-트리[20]의 연구에서 제안하지 못했던 최근접 질의 처리에 대한 알고리즘을 제안하고, 특정 분포에서만이 아니라 다양한 분포(Global Uniform : GU, Local Uniform : LU, Global Gauss : GG, Local Gauss : LG)에서의 모의 데이터를 이용한 실험을 통해 PdR-트리의 우수성을 증명하고자 한다.

이 논문은 다음과 같은 기여도(contribution)를 가진다.

① R-트리 기반 인덱스의 문제점이었던 노드간의 겹침 현상은 중심으로부터 거리를 기반으로 한 영역정보(dR : doughnuts Range)를 이용하면 노드간의 겹침은 발생하지 않는다.

② 고차원 데이터의 최근접 객체 검색 시, PdR-트리에 저장된 겹침이 없는 영역정보(dR : doughnuts Range)를 이용하면 기존 피라미드나 구형 피라미드 기법보다 더 향상된 검색 성능을 발휘 할 수 있다.

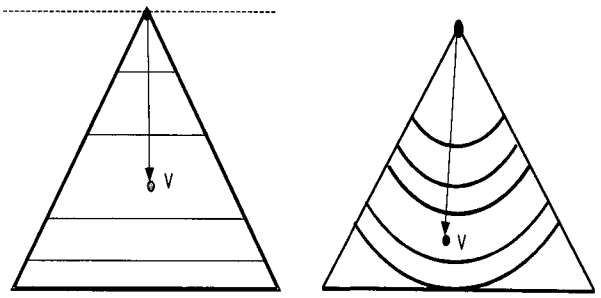
③ 실제 응용 데이터들은 다양한 분포를 이루기 때문에 일반적으로 고려할 수 있는 분포의 데이터 특성에 따른 검색 기법의 성능 분석이 필요하다. PdR-트리는 다양한 분포(GU, LU, GG, LG)에서도 기존 기법들 보다 더 향상된 성능을 발휘하므로 다양한 분포를 이루는 여러 응용에 적용할 수 있다.

2. 관련 연구

2.1 피라미드, 구형 피라미드, PdR-트리

피라미드 기법은 분할된 피라미드에 대한 페이지 분할을 (그림 2.1)(a)에서 보는 바와 같이 데이터 공간의 중심점에서의 높이 값을 이용하고, 구형 피라미드 기법의 경우는 (그림 2.1)(b)에서 보는바와 같이 중심점에서의 거리 값을 이용한다.

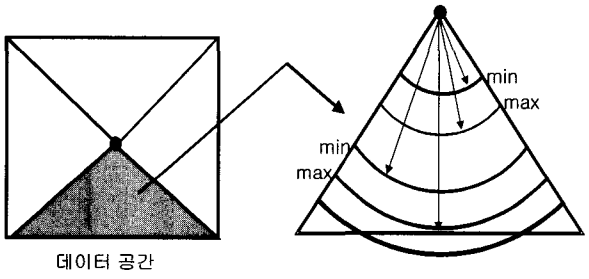
그러나 두 기법 모두 전체 입력 데이터를 데이터 공간에 적재시켜 놓고 일정한 데이터 개수에 의해 페이지를 분할하는 방법을 사용한다. 따라서 이와 같은 방법을 이용하면 삽입과 삭제연산을 적용할 경우에 전체 데이터 구조를 재구성해야 하는 문제점이 발생한다. 또한 B+-트리를 기반으로 하고 있어 하위 노드의 영역정보를 가지고 있지 않다.



(a) 피라미드 기법 (b) 구형 피라미드 기법

(그림 2.1) 피라미드 기법과 구형 피라미드 기법의 데이터 페이지 분할 방법 비교

PdR-트리는 (그림 2.2)와 같이 데이터의 삽입과 삭제연산이 가능하도록 각각의 피라미드에 새로운 데이터가 입력될 때마다 중심점 (0.5, 0.5, …, 0.5)에서 거리에 의한 영역정보(dR : doughnuts Range)를 기반으로 페이지 단위로 분할하여 인덱스를 구성한다.



(그림 2.2) PdR-트리의 데이터 페이지 분할 방법

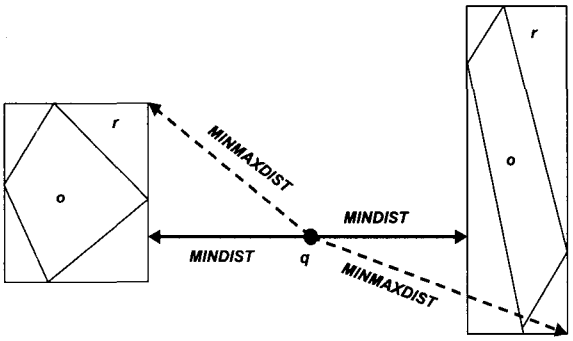
PdR-트리에서 데이터 페이지 분할은 해당 노드의 오버플로 발생 시 페이지 내에 포함된 데이터 중에서 거리상 정중앙에 위치한 데이터를 기준으로 분할한다. 이렇게 분할되어 만들어진 영역정보(dR)에는 겹침(overlap)이 발생하지 않는다는 것이 PdR-트리의 장점이다.

2.2 최근접 질의 처리

2.2.1 KNN(k-nearest neighbor) 알고리즘[14]

KNN 알고리즘은 R-트리의 최근접 질의로서 제안된 알고리즘으로서 트리를 깊이 우선 순회(depth-first traversal)하여 k개의 최근접 객체 후보의 목록(ActivateBranchList)를 유지하는 것이다. KNN에서는 방문해야할 노드를 정렬하기 위해 MINDIST와 MINMAXDIST라는 개념을 제안하고 있다.

MINDIST는 (그림 2.3)과 같이 질의 객체 q로 노드 n의 경계 사각형 r의 경계상에서 가장 가까운 지점까지의 거리로 하한을 의미하고, MINMAXDIST는 q에서 가장 멀리 있는 꼭지점에 이웃하는 r의 가장 가까운 꼭지점까지의 거리로 하한을 의미한다. 이 두 거리 함수를 이용하여 상향과 하향의 가지치기(prunning)을 위한 전략을 제안하고 있다.



(그림 2.3) MINDIST와 MINMAXDIST

2.2.2 INN(incremental nearest neighbor) 알고리즘[2]

INN 알고리즘은 최적-우선순회(best-first traversal)라는 개념을 제안하여 점진적으로 최근접 객체를 검색하는 알고리즘이다. KNN 알고리즘은 k개의 최근접 객체를 찾기 위해 미리 k값을 지정하여 알고리즘을 호출하여야 한다. 그러므로 (k+1) 최근접 객체에 대해서는 다시 (k+1)의 값을 가지고 호출되어야 한다.

이런 문제점을 해결하기 위하여 INN은 필요한 경우, 점진적으로 근접 객체를 얻는 거리 브라우징(distance browsing)이라는 개념을 제안한다. 이 개념은 거리를 기준으로 하는 우선순위 큐(Priority Queue)를 사용하여 데이터베이스를 브라우징하는 것으로 다양한 실험을 통하여 INN이 KNN보다 우수함을 보여주고, KNN 알고리즘의 MINMAXDIST를 사용하지 않고 MINDIST만을 사용하는 가지치기(prunning) 전략을 제안하고 있다.

3. PdR-트리 : 최근접 질의 처리

INN 알고리즘은 효율적인 최근접 질의 처리 방법이지만, 고차원 데이터에서는 그렇게 좋은 성능을 보여주지는 않는다. 그러나 이것은 INN 알고리즘의 문제가 아니라 고차원 데이터 공간에서 효율적인 인덱싱이나 질의 처리가 불충분한 R-트리 기반의 인덱스 구조를 사용하기 때문이다. R-트리 기반의 인덱스 구조는 차원이 증가할수록 거의 모든 노드간의 겹침 현상이 발생하여 모든 노드를 검색해야 하는 결과를 초래하므로 순차 검색보다 성능이 떨어질 수 밖에 없다.

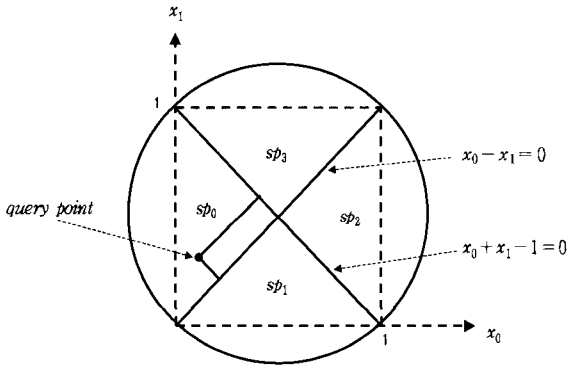
3.1 확장 INN 알고리즘

PdR-트리는 R-트리와는 구조적으로 다르기 때문에 원래 INN 알고리즘을 사용할 수 없다. 그러므로 PdR-트리에서 사용할 수 있도록 확장된 INN 알고리즘을 사용한다.

3.1.1 최소 거리의 계산 : MINDIST

INN 알고리즘에서는 가지치기를 위해 KNN 알고리즘의 MINMAXDIST를 사용하지 않으므로MINDIST만을 사용하고, SPY-TEC에서는 특성상 질의 점에서 구형 피라미드까지의 최소 거리와 질의 점에서 경계 슬라이스까지의 최소 거리를 계산하지만, PdR-트리에서는 질의 점에서 구형 피라미드 까지의 최소거리를 사용하지 않는다.

SPY-TEC에서는 질의 점과 구형 피라미드간의 최소 거리의 관계는 (그림 3.1)에서 보여주는 것과 같으며 각 구형 피라미드와의 최소 거리는 [정의 3.1]에 의해 계산한다. j는 질의 점 q가 속한 피라미드의 번호이고, i는 해당 피라미드의 번호이다.



(그림 3.1) 질의 점과 구형 피라미드간의 최소 거리

[정의 3.1] 질의 점 q 와 구형 피라미드 sp_i 간의 최소 거리

$$MINDIST(q,\ sp_i) = \begin{cases} 0 & \text{if } i = j \\ d_q & \text{if } |i-j| = d \\ \dfrac{|q_j - q_i|}{\sqrt{2}} & \text{if } i < d \\ \dfrac{|q_j + q_i - 1|}{\sqrt{2}} & \text{if } i > d \end{cases}$$

PdR-트리에서 질의 점과 경계 슬라이스(*BS* : *Bounding Slice*)간의 최소 거리를 구하는 방법과 질의 점과 영역정보(*dR* : *doughnuts Range*)간의 최소 거리를 구하는 방법은 동일하다.

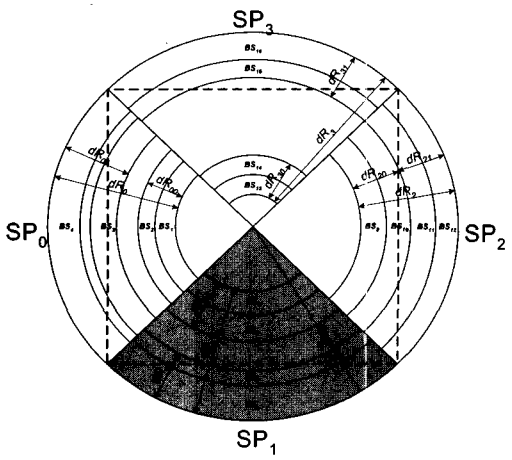
질의 점과 각 경계 슬라이스(Bounding Slice)나 영역정보(dR)간의 최소 거리를 계산하는 경우, 발생 가능한 3가지 경우와 최소 거리의 정의는 다음 [정의 3.2]와 같다. 영역

정보와의 최소 거리를 구할 경우는 BS_l을 dR_l로 치환하여 계산한다.

[정의 3.2] 질의 점 q와 경계 슬라이스 BS_l간의 최소 거리

경우 1 : ($i=j$: BS_l이 q를 포함하는 구형 피라미드에 속한 경우)

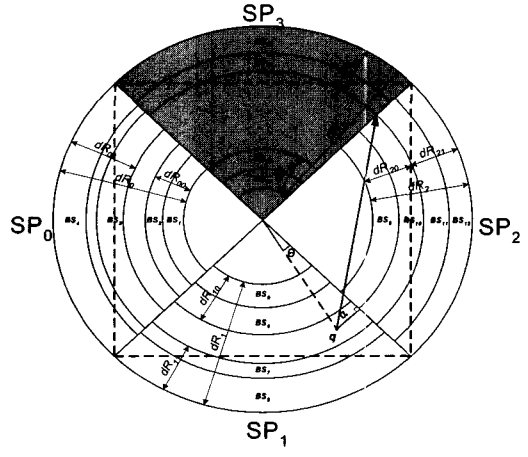
$$MINDIST(q, BS_l) = \begin{cases} |d_q - \max(BS_l)| & \text{if } d_q > \max(BS_l) \\ 0 & \text{if } \min(BS_l) \le d_q \le \max(BS_l) \\ |d_q - \min(BS_l)| & \text{if } d_q < \min(BS_l) \end{cases}$$



(그림 3.2) BS_l까지의 최소 거리 : 경우 1

경우 2 : ($|i-j|=d$: BS_l이 q의 반대쪽 구형 피라미드에 속한 경우)

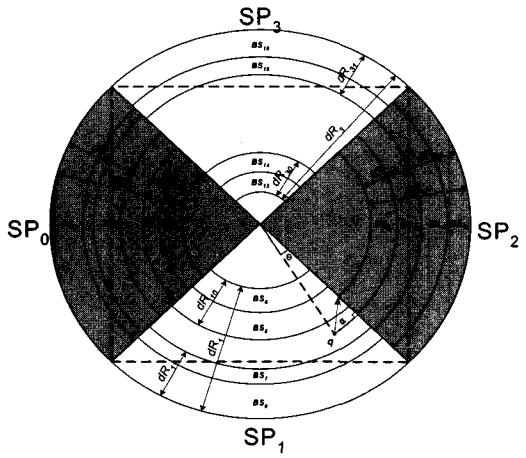
$$MINDIST(q,\ BS_l) = \sqrt{d_q^2 + \min(BS_l)^2 - 2d_q \min(BS_l)\cos(\theta + \frac{\pi}{2})}$$



(그림 3.3) BS_l 까지의 최소 거리 : 경우 2

경우 3 : (*otherwise* : BS_l이 q와 인접한 구형 피라미드에 속한 경우)

$$MINDIST(q, BS_l) = \begin{cases} \sqrt{|\delta - \max(BS_l)|^2 + \alpha^2} & \text{if } \delta > \max(BS_l) \\ \alpha & \text{if } \min(BS_l) \le \delta \le \max(BS_l) \\ \sqrt{|\delta - \min(BS_l)|^2 + \alpha^2} & \text{if } \delta < \min(BS_l) \end{cases}$$



(그림 3.4) BS_l까지의 최소거리 : 경우 3

3.1.2 최근접 처리 알고리즘

최근접 처리 알고리즘은 (알고리즘 3.1)과 같다. 1번 행에서 우선 순위 큐(Priority Queue)를 생성한 후, 2~5번 행에서 [정의 3.1]을 이용해서 질의 점과 dR_0~dR_3 까지의 최소 거리를 계산한 후, 각 피라미드에 대한 정보와 거리 순으로 정렬하여 큐에 입력한다.

7번 행은 큐에서 원소 하나를 추출하고, 원소의 타입에 따라 다음과 같은 연산을 수행한다.

원소의 타입이 영역정보(dR)인 경우는 영역정보에 속한 하위 원소의 타입에 따라 모든 하위 원소와 질의 점간의 거리를 [정의 3.2]를 이용하여 계산하고 거리 순으로 정렬하여 큐에 입력한다(9~17행).

원소의 타입이 경계 슬라이스(BS)인 경우는 경계 슬라이스에 속한 모든 실제 객체와 질의 점간의 거리를 계산하여, 거리 순으로 정렬하여 큐에 입력한다(19~22번 행).

나머지 타입인 경우는 실제 객체를 나타내므로 24번 행에서 객체를 출력하고 출력된 객체의 수를 증가하고 객체의 수가 k를 만족하면 알고리즘을 중단하고 만족하지 못한 경우는 위 과정을 반복한다.

```
1 : queue = CREATE_PRIORITY_QUEUE();

2 : for i=0 to 2d - 1 do
3 :     dist = MINDIST(q, dRi,); /* 정의 3.1 */
4 :     ENQUEUE(queue, dRi, dist);
```

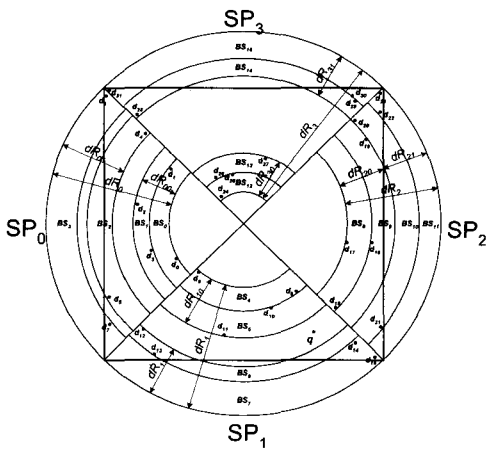
```
5 : end for

6 : while not IsEmpty(queue) do
7 :     Element = DEQUEUE(queue);

8 :     if Element is a doughnuts range then
9 :         for each Sub in a doughnuts range do
10 :            if Sub is a doughnuts range then
11 :                dist = MINDIST(q, dRl); /* 정의 3.2 */
12 :                ENQUEUE(queue, dRl, dist);
13 :            else /* bounding slice */
14 :                dist = MINDIST(q, BSl); /* 정의 3.2 */
15 :                ENQUEUE(queue, BSl, dist);
16 :            end if
17 :        end for
18 :    else if Element is a bounding slice then
19 :        for each object in a bounding slice do
20 :            dist = DIST_QUERY_TO_OBJ(q, object);
21 :            ENQUEUE(queue, object, dist);
22 :        end for
23 :    else /* Element is a object */
24 :        report element as the next nearest object
25 :    end if
26 : end while
```

(알고리즘 3.1) 점진적 최근접 질의 처리

3.1.3 PdR-트리 vs SPY-TEC : 최근접 질의 처리의 예

각 경계 슬라이스(BS)는 2개의 데이터 객체를 포함하고, 각 영역정보(dR)는 2개의 하위 요소를 포함한다고 가정하고, 질의 점 $\boldsymbol{q}$(0.75, 0.10)로부터 3개의 최근접 객체를 검색하는 예로 우선 순위 큐(Priority Queue)의 상태를 나열한다. 데이터 객체, 영역정보(dR) 그리고 경계 슬라이스(BS)로 구성된 PdR-트리의 상태는 (그림 3.5)와 같이 표시되며, 각 피라미드 SP_j, 각 영역정보는 dR_i, 각 경계 슬라이스 BS_l, 각 데이터 객체 d_i로 정의한다.

예제 데이터는 PdR-트리에서 제안한 최근접 질의 처리 알고리즘과 SPY-TEC에서 제안한 알고리즘 2가지 모두 적용하여 성능을 비교한다.



(그림 3.5) 최근접 질의 처리의 예(32 points)

예제로 사용한 데이터 객체는 <표 3.1>과 같으며, 각 피라미드, 영역정보, 경계 슬라이스 간의 거리는 <표 3.2>와 같다.

〈표 3.1〉 예제 데이터(32 points)

Point	X	Y	Dist.	Point	X	Y	Dist.
d_0	0.26	0.37	0.559	d_{16}	0.83	0.19	0.120
d_1	0.24	0.70	0.787	d_{17}	0.87	0.43	0.351
d_2	0.12	0.57	0.786	d_{18}	0.96	0.43	0.391
d_3	0.17	0.40	0.653	d_{19}	0.94	0.81	0.735
d_4	0.15	0.83	0.945	d_{20}	0.94	0.83	0.754
d_5	0.02	0.23	0.741	d_{21}	0.99	0.12	0.241
d_6	0.01	0.97	1.142	d_{22}	0.99	0.91	0.845
d_7	0.02	0.13	0.731	d_{23}	0.99	0.98	0.912
d_8	0.34	0.32	0.465	d_{24}	0.42	0.60	0.599
d_9	0.69	0.25	0.162	d_{25}	0.46	0.68	0.648
d_{10}	0.60	0.19	0.175	d_{26}	0.40	0.67	0.669
d_{11}	0.43	0.09	0.320	d_{27}	0.58	0.74	0.662
d_{12}	0.14	0.11	0.610	d_{28}	0.12	0.90	1.018
d_{13}	0.18	0.02	0.576	d_{29}	0.90	0.95	0.863
d_{14}	0.94	0.06	0.194	d_{30}	0.89	0.97	0.881
d_{15}	0.15	0.01	0.610	d_{31}	0.02	0.99	1.151

<표 3.2>에서 dR_0~dR_3는 SP_0~SP_3와 일대일 대응하므로 PdR-트리를 이용한 최근접 질의에서는 SPY-TEC에서 제안한 SP0~SP1까지의 최소 거리 대신 사용할 수 있다. SP_i 대신에 dR_i를 이용하는 것은 검색시 탐색 공간을 더욱 더 세분하므로 더 효율적이다.

〈표 3.2〉 예제 데이터(피라미드, 영역정보, 경계 슬라이스의 최소거리)

BS_l		dR_{ji}		dR_j		SP_i	
Label	MINDIST	Label	MINDIST	Label	MINDIST	Label	MINDIST
BS_0	0.489	dR_{00}	0.489	dR_0	0.489	SP_0	0.460
BS_1	0.518						
BS_2	0.593	dR_{01}	0.593				
BS_3	0.734						
BS_4	0.158	dR_{10}	0.056	dR_1	0.000	SP_1	0.000
BS_5	0.056						
BS_6	0.231	dR_{11}	0.231				
BS_7	0.132						
BS_8	0.154	dR_{20}	0.106	dR_2	0.106	SP_2	0.106
BS_9	0.106						
BS_{10}	0.139	dR_{21}	0.139				
BS_{11}	0.208						
BS_{12}	0.672	dR_{30}	0.672	dR_3	0.672	SP_3	0.472
BS_{13}	0.688						
BS_{14}	0.860	dR_{31}	0.860				
BS_{15}	0.899						

① SPY-TEC를 이용한 최근접 질의 처리의 예

먼저 SP0~SP3를 우선 순위 큐에 넣고 다음 단계를 따른다.

1. Enqueue $SP_0 \sim SP_3$. Queue : {[SP_1, 0.000], [SP_2, 0.106], [SP_0, 0.460], [SP_3, 0.472]}
2. Dequeue SP_1, enqueue BS_4, BS_5, BS_6, BS_7. Queue : {[BS_5, 0.056], [SP_2, 0.106], [BS_7, 0.132], [BS_4, 0.158], [BS_6, 0.231], [SP_0, 0.460], [SP_3, 0.472]}
3. Dequeue BS_5, enqueue d_{10}, d_{11}. Queue : {[SP_2, 0.106], [BS_7, 0.132], [BS_4, 0.158], [d_{10}, 0.175], [BS_6, 0.231], [d_{11}, 0.320], [SP_0, 0.460], [SP_3, 0.472]}
4. Dequeue SP_2, enqueue BS_8, BS_9, BS_{10}, BS_{11}. Queue : {[BS_9, 0.106], [BS_7, 0.132], [BS_{10}, 0.139], [BS_8, 0.154], [BS_4, 0.158], [d_{10}, 0.175], [BS_{11}, 0.208], [BS_6, 0.231], [d_{11}, 0.320], [SP_0, 0.460], [SP_3, 0.472]}
5. Dequeue BS_9, enqueue d_{18}, d_{19}. Queue : {[BS_7, 0.132], [BS_{10}, 0.139], [BS_8, 0.154], [BS_4, 0.158], [d_{10}, 0.175], [BS_{11}, 0.208], [BS_6, 0.231], [d_{11}, 0.320], [d_{18}, 0.391], [SP_0, 0.460], [SP_3, 0.472], [d_{19}, 0.735]}
6. Dequeue BS_7, enqueue d_{14}, d_{15}. Queue : {[BS_{10}, 0.139], [BS_8, 0.154], [BS_4, 0.158], [d_{10}, 0.175], [d_{14}, 0.194], [BS_{11}, 0.208], [BS_6, 0.231], [d_{11}, 0.320], [d_{18}, 0.391], [SP_0, 0.460], [SP_3, 0.472], [d_{15}, 0.610], [d_{19}, 0.735]}
7. Dequeue BS_{10}, enqueue d_{20}, d_{21}. Queue : {[BS_8, 0.154], [BS_4, 0.158], [d_{10}, 0.175], [d_{14}, 0.194], [BS_{11}, 0.208], [BS_6, 0.231], [d_{21}, 0.241], [d_{11}, 0.320], [d_{18}, 0.391], [SP_0, 0.460], [SP_3, 0.472], [d_{15}, 0.610], [d_{19}, 0.735], [d_{20}, 0.754]}
8. Dequeue BS_8, enqueue d_{16}, d_{17}. Queue : {[d_{16}, 0.120], [BS_4, 0.158], [d_{10}, 0.175], [d_{14}, 0.194], [BS_{11}, 0.208], [BS_6, 0.231], [d_{21}, 0.241], [d_{11}, 0.320], [d_{17}, 0.351], [d_{18}, 0.391], [SP_0, 0.460], [SP_3, 0.472], [d_{15}, 0.610], [d_{19}, 0.735], [d_{20}, 0.754]}
9. Dequeue d_{16}, report d_{16} as nearest neighbor. Queue : {[BS_4, 0.158], [d_{10}, 0.175], [d_{14}, 0.194], [BS_{11}, 0.208], [BS_6, 0.231], [d_{21}, 0.241], [d_{11}, 0.320], [d_{17}, 0.351], [d_{18}, 0.391], [SP_0, 0.460], [SP_3, 0.472], [d_{15}, 0.610], [d_{19}, 0.735], [d_{20}, 0.754]}
10. Dequeue BS_4, enqueue d_8, d_9. Queue : {[d_9, 0.162], [d_{10}, 0.175], [d_{14}, 0.194], [BS_{11}, 0.208], [BS_6, 0.231], [d_{21}, 0.241], [d_{11}, 0.320], [d_{17}, 0.351], [d_{18}, 0.391], [SP_0, 0.460], [d_8, 0.465], [SP_3, 0.472], [d_{15}, 0.610], [d_{19}, 0.735], [d_{20}, 0.754]}
11. Dequeue d_9, report d_9 as nearest neighbor. Queue : {[d_{10}, 0.175], [d_{14}, 0.194], [BS_{11}, 0.208], [BS_6, 0.231], [d_{21}, 0.241], [d_{11}, 0.320], [d_{17}, 0.351], [d_{18}, 0.391], [SP_0, 0.460], [d_8, 0.465], [SP_3, 0.472], [d_{15}, 0.610], [d_{19}, 0.735], [d_{20}, 0.754]}
12. Dequeue d_{10}, report d_{10} as nearest neighbor.

② PdR-트리를 이용한 최근접 질의 처리의 예

먼저 dR0~dR3를 우선 순위 큐에 넣고 다음 단계를 따른다.

1. Enqueue $dR_0 \sim dR_3$. Queue : {[dR_1, 0.000], [dR_2, 0.106], [dR_0, 0.489], [dR_3, 0.672]}
2. Dequee dR_1, enqueue dR_{10}, dR_{11}. Queue : {[dR_{10}, 0.056], [dR_2, 0.106], [dR_{11}, 0.231], [dR_0, 0.489], [dR_3, 0.672]}
3. Dequeue dR10, enqueue BS4, BS5. Queue : {[BS5, 0.056], [dR2, 0.106], [BS4, 0.158], [dR11, 0.231], [dR0, 0.489], [dR3, 0.672]}
4. Dequeue BS5, enqueue d10, d11. Queue : {[dR2, 0.106], [BS4, 0.158], [d10, 0.175], [dR11, 0.231], [d11, 0.320], [dR0, 0.489], [dR3, 0.672]}
5. Dequeue dR2, enqueue dR20, dR21. Queue : {[dR20, 0.106], [dR21, 0.139], [BS4, 0.158], [d10, 0.175], [dR11, 0.231], [d11, 0.320], [dR0, 0.489], [SP3, 0.672]}
6. Dequeue dR20, enqueue BS8, BS9. Queue : {[BS9, 0.106], [dR21, 0.139], [BS8, 0.154], [BS4, 0.158], [d10, 0.175], [dR11, 0.231], [d11, 0.320], [dR0, 0.489], [dR3, 0.672]}
7. Dequeue BS9, enqueue d18, d19. Queue : {[dR21, 0.139], [BS8, 0.154], [BS4, 0.158], [d10, 0.175], [dR11, 0.231], [d11, 0.320], [d18, 0.391], [dR0, 0.489], [dR3, 0.672], [d19, 0.735]}
8. Dequeue dR21, enqueue BS10, BS11. Queue : {[BS10, 0.139], [BS8, 0.154], [BS4, 0.158], [d10, 0.175], [BS11, 0.208], [dR11, 0.231], [d11, 0.320], [d18, 0.391], [dR0, 0.489], [dR3, 0.672], [d19, 0.735]}
9. Dequeue BS10, enqueue d20, d21. Queue : {[BS8, 0.154], [BS4, 0.158], [d10, 0.175], [BS11, 0.208], [dR11, 0.231], [d21, 0.241], [d11, 0.320], [d18, 0.391], [dR0, 0.489], [dR3, 0.672], [d19, 0.735], [d20, 0.754]}
10. Dequeue BS8, enqueue d16, d17. Queue : {[d16, 0.120], [BS4, 0.158], [d10, 0.175], [BS11, 0.208], [dR11, 0.231], [d21, 0.241], [d11, 0.320], [d17, 0.351], [d18, 0.391], [dR0, 0.489], [dR3, 0.672], [d19, 0.735], [d20, 0.754]}
11. Dequeue d16, report d16 as nearest neighbor. Queu

e : {[BS4, 0.158], [d10, 0.175], [BS11, 0.208], [dR11, 0.231], [d21, 0.241], [d11, 0.320], [d17, 0.351], [d18, 0.391], [dR0, 0.489], [dR3, 0.672], [d19, 0.735], [d20, 0.754]}

12. Dequeue BS4, enqueue d8, d9. Queue : {[d9, 0.162], [d10, 0.175], [BS11, 0.208], [dR11, 0.231], [d21, 0.241], [d11, 0.320], [d17, 0.351], [d18, 0.391], [dR0, 0.489], [d8, 0.465], [dR3, 0.672], [d19, 0.735], [d20, 0.754]}
13. Dequeue d9, report d9 as nearest neighbor. Queue : {[d10, 0.175], [BS11, 0.208], [dR11, 0.231], [d21, 0.241], [d11, 0.320], [d17, 0.351], [d18, 0.391], [dR0, 0.489], [d8, 0.465], [dR3, 0.672], [d19, 0.735], [d20, 0.754]}
14. Dequeue d10, report d10 as nearest neighbor.

③ 예제를 통한 PdR-트리와 SPY-TEC의 성능 분석

최근접 질의 처리의 예제에서 SPY-TEC가 PdR-트리보다 2단계 적은 단계로 주어진 개수의 최근접 객체를 찾아 더욱 효율적인 것처럼 보이지만 페이지 접근 회수를 고려한다면 SPY-TEC보다 PdR-트리가 더 효율적임을 알 수 있다. 대부분의 인덱스 구조에서 성능을 좌우하는 요소는 페이지 접근 회수이다.

최근접 질의 처리의 예제에서 BS를 순회하면서 페이지 접근이 일어난다. SPY-TEC의 경우는 BS_5, BS_9, BS_7, BS_{10}, BS_8, BS_4 순으로 페이지를 접근한 반면에 PdR-트리의 경우는 BS_5, BS_9, BS_{10}, BS_8, BS_4 순으로 접근하여 BS_7에 대한 접근을 제외했다는 사실을 알 수 있다.

PdR-트리에서 BS_7이 제외된 이유는 SPY-TEC는 해당 구형 피라미드에 속한 모든 BS를 큐에 삽입하고, PdR-트리는 영역정보(dR)를 기반으로 삽입하기 때문이다. 그러므로 각 피라미드에 속한 BS가 많은 경우에는 SPY-TEC보다 PdR-트리가 더 적은 페이지 접근을 행하게 되어 검색 속도를 향상하게 된다.

4. 실험 및 분석

이 논문에서는 최근접 질의 처리에서의 PdR-트리의 성능 및 유용성을 증명하기 위해 기존의 피라미드 기법의 동일한 조건과 환경에서 비교하여 실험하였다. 즉 비교 실험에 대한 공정성을 부과하기 위하여 실제 데이터(real data)와 모의 데이터(simulation data)을 이용한 실험을 하였고, 특히 모의 데이터를 이용한 실험에서는 고차원 공간상의 여러 분포(GU, LU, GG, LG)를 가지는 데이터를 생성하여 실험하였다. 각 실험에서 데이터 크기 변화에 따른 실험과 데이터 차원 크기 변화에 따른 페이지 접근 비율 및 CPU-사용시간을 비롯한 종합 경과 시간을 비교하여 실험하였다. 페이지 접근 비율은 페이지의 개수가 같지 않은 두 기법의 페이지 접근빈도를 측정하기 위해 전체 페이지 개수로 평균 페이지 접근 회수를 나눈 값으로 다음과 같다.

$$\text{페이지 접근 비율} = \frac{\text{평균 페이지 접근 회수}}{\text{전체 페이지 개수}}$$

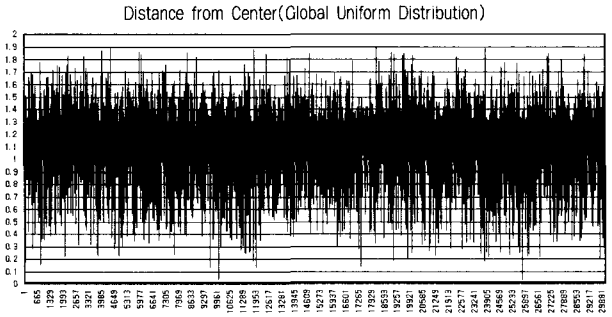
SPY-TEC과의 성능 비교는 상호 실험 환경이 다르기 때문에 직접적인 비교 실험은 할 수가 없었다. 다만 간접 비교의 수단으로 종합 경과 시간을 비교해보면 검색 성능에 대해 PdR-트리가 최근접 질의 처리에서도 우수함을 알 수 있었다. 프로그램은 C-언어를 이용하여 작성하였으며 gcc로 컴파일 하였다. 실험 환경은 SUN-Ultra II 워크스테이션으로 메인 메모리 256MB이며 12GB의 기억 용량을 가진 HDD를 사용하였다. 차원의 크기는 최소 4차원에서 최대 24차원이며 할당된 페이지 크기는 4KB로 설정하였고 k는 10으로 고정하여 반복 수행하였다.

4.1 모의 데이터(Simulation Data)

모의 데이터는 현실적인 데이터 분포를 고려하여 전역 균등 분포(GU : Global Uniform Distribution), 지역 균등 분포(LU : Local Uniform Distribution), 전역 가우스 분포(GG : Global Gauss Distribution), 지역 가우스 분포(LG : Local Gauss Distribution)를 가지는 데이터를 생성하여 실험하였다. 16차원 30,000개의 데이터를 각각의 분포로 생성한 다음 중심에서 데이터까지의 거리의 분포는 (그림 4.1)~(그림 4.4)와 같다.

4.1.1 전역 균등 분포(GU : Global Uniform Distribution)

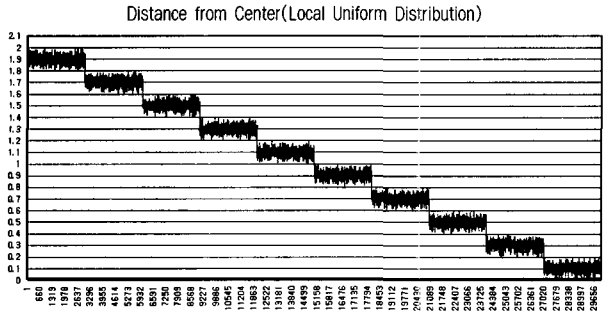
데이터가 전역 균등 분포하는 경우, (그림 4.1)에서 보는 바와 같이 중심에서 데이터까지의 거리는 특정 거리에 밀집하지 않는다. 중심에서 데이터까지의 거리를 기반으로 한 인덱스 구조에서는 노드에 균등하게 데이터가 분포하므로 최근접 객체를 찾는 검색 성능은 여러 데이터 분포 중 가장 우수하게 나타난다.



(그림 4.1) 전역 균등 분포(Global Uniform Distribution)

4.1.2 지역 균등 분포(LU : Local Uniform Distribution)

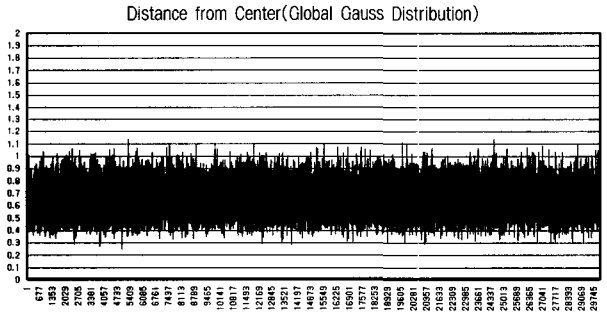
데이터가 지역 균등 분포하는 경우, (그림 4.2)에서 보는 바와 같이 중심에서 데이터까지의 거리는 구간별로 밀집한다. 구간별로 포함된 특정 노드에 데이터가 밀집하여 분포하므로 최근접 질의 객체를 찾는 검색 성능은 전역 균등 분포보다 저하한다.



(그림 4.2) 지역 균등 분포(Local Uniform Distribution)

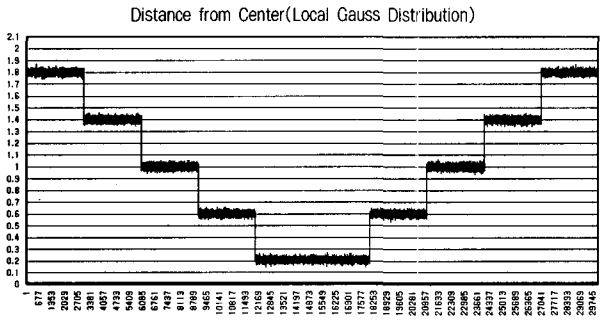
4.1.3 전역 가우스 분포(GG : Global Gauss Distribution)

데이터가 전역 가우스 분포하는 경우, (그림 4.3)에서 보는 바와 같이 중심에서 데이터까지의 거리는 특정 거리에 밀집한다. 이러한 데이터 집합에서는 최근접 객체를 검색하기 위해 거의 대부분 노드를 순회하게되어 노드 액세스에 대한 부담으로 인하여 오히려 순차 검색보다 좋지 않을 수 있다.



(그림 4.3) 전역 가우스 분포(Global Gauss Distribution)

4.1.4 지역 가우스 분포(LG : Local Gauss Distribution)



(그림 4.4) 지역 가우스 분포(Local Gauss Distribution)

데이터가 지역 가우스 분포하는 경우, (그림 4.4)에서 보는 바와 같이 중심에서 데이터까지의 거리는 구간별로 밀집한다. 지역 균등 분포하는 데이터 집합보다 구간별로 더 밀집하므로 더 많은 노드 액세스가 필요하나 전역 가우스 분포하는 데이터 집합보다는 성능이 좋다.

4.2 모의 데이터를 이용한 실험

4.2.1 데이터 개수에 따른 성능 비교

16차원 데이터의 개수를 20,000~100,000 변화시키면서 실험하였고, <표 4.1>은 데이터 개수 변화에 따른 검색 성능 비교를 수치로 나타내고, (그림 4.5)는 성능 비교에 대한 실험 결과를 그래프로 표현하였다.

〈표 4.1〉 데이터 개수 변화에 따른 성능 비교(차원=16, k=10)

Dist.	Data Size	Page Access Ratio		CPU Time(sec.)		Total Time(sec.)	
		Pyramid	PdR	Pyramid	PdR	Pyramid	PdR
Global Uniform	20,000	91 %	52 %	0.152	0.106	0.210	0.137
	40,000	89 %	48 %	0.296	0.194	0.397	0.254
	60,000	87 %	45 %	0.436	0.283	0.540	0.373
	80,000	85 %	42 %	0.567	0.340	0.735	0.511
	100,000	81 %	37 %	0.701	0.391	0.875	0.559
Local Uniform	20,000	97 %	66 %	0.169	0.133	0.257	0.167
	40,000	97 %	65 %	0.332	0.255	0.505	0.336
	60,000	97 %	64 %	0.498	0.384	0.698	0.512
	80,000	96 %	64 %	0.656	0.512	0.944	0.674
	100,000	96 %	64 %	0.829	0.639	1.222	0.846
Global Gauss	20,000	92 %	84 %	0.160	0.170	0.214	0.211
	40,000	91 %	82 %	0.305	0.329	0.424	0.410
	60,000	90 %	80 %	0.472	0.483	0.673	0.615
	80,000	89 %	79 %	0.608	0.628	0.901	0.799
	100,000	88 %	77 %	0.730	0.777	1.142	1.046
Local Gauss	20,000	96 %	67 %	0.166	0.135	0.224	0.171
	40,000	96 %	65 %	0.331	0.258	0.432	0.341
	60,000	96 %	64 %	0.490	0.393	0.714	0.508
	80,000	96 %	64 %	0.664	0.511	0.950	0.684
	100,000	96 %	63 %	0.810	0.644	1.284	0.859

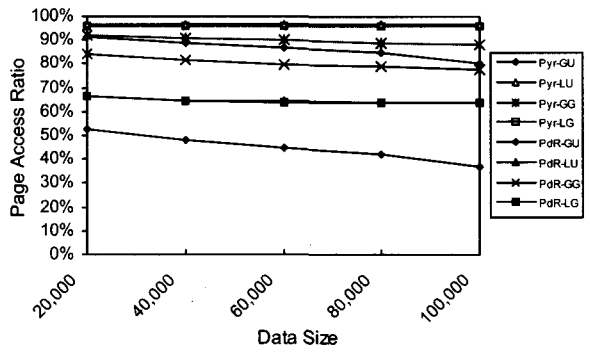
페이지 접근 비율(Page Access Ratio)은 (그림 4.5)(a)에서 보는 바와 같이 전역 균등 분포에서는 43%~54%, 지역 균등 분포에서는 32%~34%, 전역 가우스 분포에서는 9%~12%, 지역 가우스 분포에서는 31%~34% 감소하였다.

CPU 사용 시간은 (그림 4.5)(b)에서 보는 바와 같이 전역 균등 분포에서는 30%~44%, 지역 균등 분포에서는 21%~23%, 지역 가우스 분포에서는 19%~20% 감소하였지만 전역 가우스 분포에서는 2%~8% 증가하는 결과를 보였다.

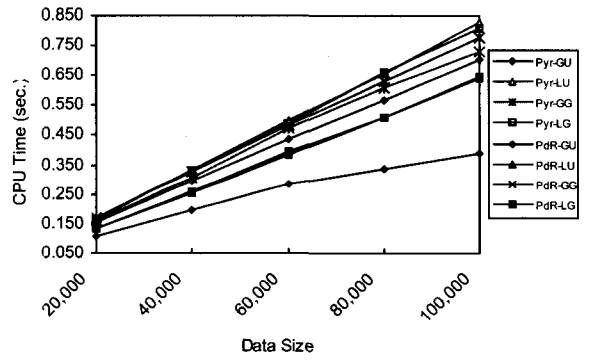
종합 경과 시간(Total time)은 (그림 4.5)(c)에서 보는 바와 같이 전역 균등 분포에서는 31%~36%, 지역 균등 분포

에서는 27%~35%, 전역 가우스 분포에서는 1%~11%, 지역 가우스 분포에서는 21%~33% 감소하였다.

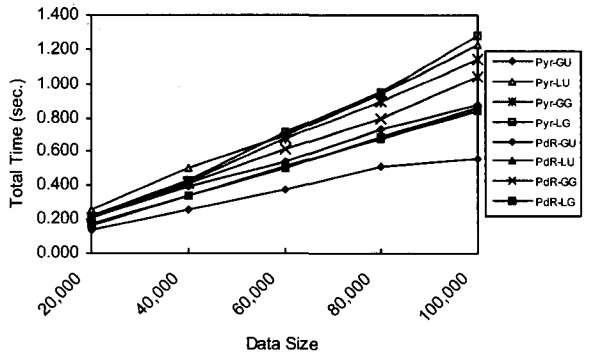
실험 결과, 종합적으로 PdR-트리의 성능이 피라미드 기법보다 우수하였고, 전역 가우스 분포의 경우에만 PdR-트리의 성능이 현저하게 향상되지 않았고, 이러한 이유는 데이터가 특정 거리에 밀집하여 다른 분포보다는 더 많은 페이지 접근이 필요하기 때문이다.



(a) Page Access Ratio



(b) CPU Time(sec.)



(c) Total Time(sec.)

(그림 4.5) 데이터 개수 변화에 따른 성능 비교

4.2.2 차원 크기의 변화에 따른 성능 비교

데이터의 개수를 100,000으로 고정하고, 차원을 4~24 변화시키면서 실험하였고, <표 4.2>는 차원 변화에 따른 검색 성능 비교를 수치로 나타내고, (그림 4.6)은 성능 비교에 대한 실험 결과를 그래프로 표현하였다.

〈표 4.2〉 차원 변화에 따른 성능 비교
(데이터 개수=100,000, k=10)

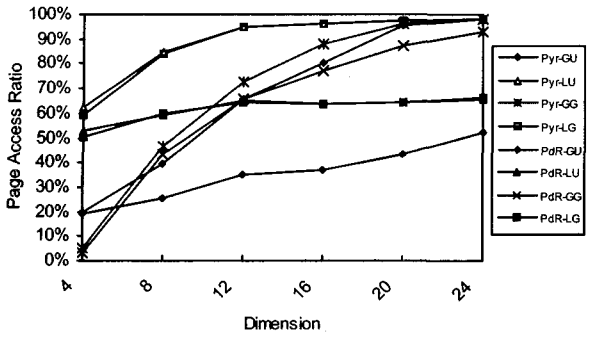
Dist.	Dim. Size	Pag6e Access Ratio		CPU Time(sec.)		Total Time(sec.)	
		Pyramid	PdR	Pyramid	PdR	Pyramid	PdR
Global Uniform	4	20 %	19 %	0.075	0.082	0.075	0.123
	8	39 %	25 %	0.203	0.151	0.203	0.253
	12	66 %	35 %	0.457	0.287	0.588	0.422
	16	81 %	37 %	0.701	0.391	0.879	0.559
	20	96 %	43 %	0.971	0.530	1.220	0.759
	24	98 %	52 %	1.148	0.751	1.408	1.044
Local Uniform	4	62 %	53 %	0.249	0.230	0.367	0.299
	8	85 %	59 %	0.447	0.361	0.657	0.497
	12	95 %	65 %	0.665	0.517	0.945	0.677
	16	96 %	64 %	0.829	0.639	1.222	0.846
	20	98 %	64 %	0.988	0.776	1.447	1.016
	24	98 %	66 %	1.206	0.916	1.729	1.197
Global Gauss	4	5 %	3 %	0.019	0.015	0.109	0.090
	8	47 %	44 %	0.247	0.273	0.507	0.485
	12	73 %	65 %	0.494	0.520	0.806	0.772
	16	88 %	77 %	0.730	0.777	1.142	1.046
	20	96 %	88 %	0.975	1.042	1.475	1.394
	24	98 %	93 %	1.158	1.303	1.789	1.779
Local Gauss	4	59 %	50 %	0.238	0.218	0.360	0.285
	8	84 %	60 %	0.454	0.363	0.659	0.499
	12	95 %	64 %	0.661	0.529	0.941	0.696
	16	96 %	63 %	0.810	0.644	1.284	0.859
	20	98 %	64 %	0.997	0.780	1.461	1.029
	24	98 %	66 %	1.169	0.922	1.735	1.232

페이지 접근 비율(Page Access Ratio)은 (그림 4.6)(a)에서 보는 바와 같이 전역 균등 분포에서는 4%~55%, 지역 균등 분포에서는 15%~34%, 전역 가우스 분포에서는 5%~32%, 지역 가우스 분포에서는 15%~34% 감소하였다.

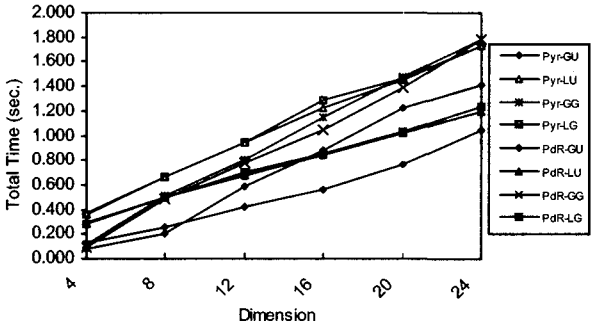
CPU 사용 시간은 (그림 4.6)(b)에서 보는 바와 같이 전역 균등 분포에서는 4차원인 경우 9% 증가하였고, 나머지 차원의 경우는 26~35% 감소하였다. 지역 균등 분포에서는 모든 차원에 대해 8%~24% 감소하였고, 전역 가우스 분포에서는 4차원의 경우만 21% 감소하였고, 나머지 차원에 대해서는 5%~13% 증가하였다. 지역 가우스 분포에서는 8%~22% 감소하였다.

종합 경과 시간(Total time)은 (그림 4.6)(c)에서 보는 바와 같이 전역 균등 분포에서는 4차원과 8차원의 경우 25%~65% 성능이 저하하였고, 전역 가우스 분포에서는 약간의 성능 향상을 나타냈고, 나머지 경우는 평균 20% 이상 성능

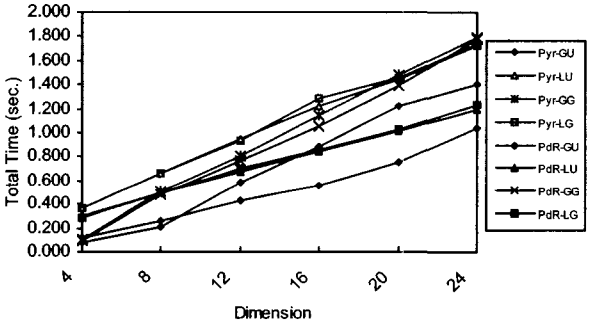
이 향상되었다.



(a) Page Access Ratio



(b) CPU Time(sec.)



(c) Total Time(sec.)

(그림 4.6) 차원 변화에 따른 성능 비교

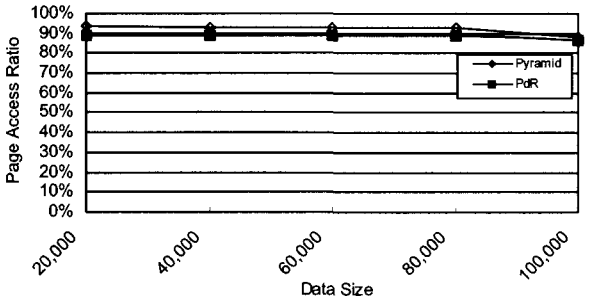
4.3 실제 데이터를 이용한 실험

실제 데이터를 이용한 실험에서는 내용기반 이미지 검색 시스템[22]에서 사용하였던 모양을 나타내는 12차원 Fourier 계수 데이터를 사용하였고, 개수를 20,000~100,000으로 변화하면서 k = 10인 최근접 객체를 검색하도록 수행하였다. <표 4.3>은 개수 변화에 따른 실제 데이터를 이용한 성능 비교를 수치로 나타내고, (그림 4.6)은 성능 비교에 대한 실험 결과를 그래프로 표현하였다.

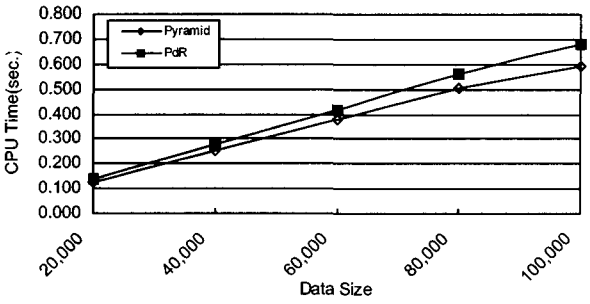
실험 결과, (그림 4.6)에서 보는 바와 같이 피라미드 기법보다 페이지 접근 비율(Page Access Ratio)이 감소하였고, CPU 사용시간은 증가하였으나, 종합 경과 시간은 최소 9%에서 최대 21% 감소하였다.

〈표 4.3〉 실제 데이터를 이용한 성능 비교(12차원, k=10)

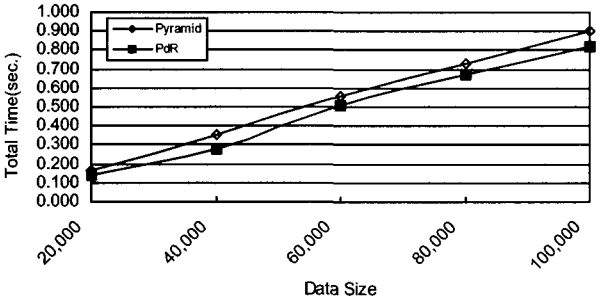
Data Size	Page Access Ratio		CPU Time		Total Time	
	Pyramid	PdR	Pyramid	PdR	Pyramid	PdR
20,000	93%	89%	0.124	0.138	0.164	0.138
40,000	93%	89%	0.249	0.275	0.349	0.275
60,000	93%	89%	0.378	0.416	0.559	0.510
80,000	93%	89%	0.503	0.558	0.726	0.674
100,000	89%	87%	0.594	0.680	0.900	0.819



(a) Page Access Ratio



(b) CPU Time(sec.)



(c) Total Time(sec.)

(그림 4.7) 실제 데이터를 이용한 실험

5. 결 론

이 논문에서는 고차원 데이터의 검색 성능 향상을 위해 제안된 인덱스 구조인 PdR-트리의 최근접 객체 검색 기법으로 확장 INN을 제안하고, 다양한 분포의 데이터를 이용한 여러 가지 실험을 통해 PdR-트리가 최근접 객체 검색에서도 뛰어난 성능을 보임을 증명하였다.

검색 성능 향상에 대한 이유는 PdR-트리가 영역정보를 저장하고 있기 때문에 전체 노드의 방문회수를 기존의 방법보다 감소시켜 전체적인 검색성능이 향상되었다고 분석된다. 종합경과시간의 경우 데이터 개수변화에 따른 실험에서 피라미드 기법에 비해 최대 35% 향상하였고, 구형 피라미드 기법에 비해 최대 15%의 향상을 이룰 수 있었다. 또한 차원변화에 따른 실험에서 피라미드 기법에 비해 최대 36%의 향상을 구형 피라미드 기법에 비해 최대 16% 검색속도가 향상되었다. 실제 데이터를 이용하여 질의범위변화에 따른 실험에서도 최대 21% 성능이 향상되었다.

또한 실험을 통해 데이터 분포 측면에서도 PdR-트리는 전역 균등 분포, 지역 균등 분포, 지역 가우스 분포, 전역 가우스 분포 순으로 성능이 우수함을 알 수 있었다. 전역 가우스 분포를 제외한 다른 분포들에서는 현저한 성능 향상을 보였으나, 특정 거리에 밀집되어 분포하는 전역 가우스 분포에서는 약간의 성능 향상만을 이룰 수 있었다. 전역 가우스 분포하는 데이터 집합에 대해서는 PdR-트리 뿐만 아니라 기존에 제안된 모든 인덱스 기법들이 모든 노드 및 리프를 접근하므로 노드에 대한 부담으로 인하여 오히려 순차검색보다 우수한 성능을 나타낼 수 없기 때문에 성능 향상을 기대할 수는 없다.

향후 최근접 객체를 검색하는데 단일 스레딩 기반이 아니라 다중 스레딩을 이용한 병렬 및 분산 처리 방법과 다중 스레딩에 적합하도록 효율적으로 분할된 인덱스 구조의 연구가 필요할 것이다. 또한 이 연구에서 약간의 성능 향상만을 나타내는 전역 가우스(GG) 분포하는 데이터 집합에 대한 해결 방안도 앞으로의 수행해야할 연구 과제이다.

참 고 문 헌

[1] A. Hinneburg, D. A. Keim, "Optimal Grid_Clustering : Towards Breaking the Curse of Dimensionality in High-Dimensional Clustering," proceedings of the 25th VLDB Conference, Edinburgh, Scotland, 1999.

[2] G. R. Hjaltason, H. Samet, "Distance Browsing in Spatial Databases," ACM Transaction on Database Systems, 24(2), pp.265-318, 1999.

[3] R. Weber, H. J. Schek, S. Blott, "A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces, 24th VLDB Conference," NY, USA, 1998.

[4] S. Berchtold, C. Bohm, H-P. Kriegel. "The Pyramid- Technique : Towards Breaking the Curse of Dimensionality," Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998.

[5] A. Henrich, "The LSDh-tree : An Access Structure for Feature Vectors," ICDE, 1998.

[6] P. Ciaccia, M. Patella, P. Zezula, "M-tree : An Efficient Access Method for Similarity Search in Metric Spaces," 23rd VLDB Conference, 1997.

[7] J. M. Hellerstein, E. Koutsoupias, C. H. Papadimitriou, "On the Analysis of Schemes," ACM PODS, pp 249-256, 1997.

[8] R. Weber, S. Blott, "An Approximation-Based Data Structure for Similarity Search," Esprit Project Hermes, technical report, Oct., 1997.

[9] N. Katayama, S. Satoh, "The SR-tree : An Index Structure for High-Dimensional Nearest Neighbor Queries," ACM SIGMOD, 1997.

[10] S. Berchtold, D. A. Keim, H. P. Kriegel, "The X-tree : An Index Structure for High-Dimension Data," 22nd VLDB Conference, 1996.

[11] A. D. Bimbo, P. Pala, S. Santini, "Image Retrieval by Elastic Matching of Shapes and Image Patterns," Proceeding of the itn'l conf. on multimedia computing and systems, Japan, 1996.

[12] D. A. White, R. Jain, "Similarity Indexing with the SS-tree," IEEE, 1995.

[13] L. I. Lin, H. V. Jagadish, C. Faloutsos, "The TV-tree-an index structure for high-dimensional data," VLDB, 1995.

[14] N. Roussopoulos, S. Kelley, F. Vincent, "Nearest Neighbor Querys", Proc. ACM SIGMOD Int. Conf., pp.71-79, 1995.

[15] K. Aizawa, H. Harashima, "Model based image coding," SPIE/IS, Electronic Imaging, Vol.4-1, pp.1-2, 1994.

[16] I. Kamel, C. Faloutsos, "Hilbert R-tree : An Improved R-tree using Fractals," VLDB, 1994.

[17] C. C. Chang, S. Y. Lee, "Retrieval of Similar Pictures on Pictorial Databases," Pattern Recognition, pp.24, 675-680, 1991.

[18] N. Beckermann, H. P. Kriegel, R. Schneider, B. Seeger, "The R*-tree : An Efficient and Robust Access Method for Points and Rectangles," ACM, 1990.

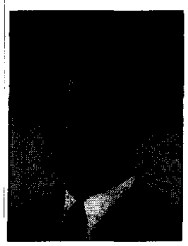
[19] A. Gutmann, "A Dynamic Index Structure for Spatial Searching," ACM, USA, 1984.

[20] 박영배, 조범석, "PdR-트리 : 고차원 데이터의 검색 성능 향상을 위한 효율적인 인덱스 기법", 정보처리학회논문지D, 제8-D권 제2호, 2001.

[21] 이동호, 정진완, 김형주, "고차원 데이터의 유사성 검색을 위한 효율적인 색인기법", 정보과학회논문지(B), 제26권 제11호, 1999.

[22] 이동호, 송용준, 김형주. "SCARLET : 웨이브릿 변환을 이용한 내용기반 이미지검색시스템의 설계 및 구현", 정보과학회논문지(C), 제3권 제4호, 1997.

[23] 이혜명, 임채명, 박영배, "시계열 패턴을 위한 dR-트리", 한국정보과학회 가을 학술발표논문집(A), 1996.



김 진 호

e-mail : solbong@mju.ac.kr

1994년 군산대학교 전자공학과(공학사)

2000년 명지대학교 일반대학원 컴퓨터공학과(공학석사)

2003년 명지대학교 일반대학원 컴퓨터공학과(박사수료)

관심분야 : Spatial DB, Multimedia DB, Web DB, XML DB, Data Warehousing & Data Mining 등

박 영 배

e-mail : parkyb@mju.ac.kr

1993년 서울대학교 컴퓨터공학과(공학박사)

1990년~1992년 명지대학교 전자계산소장

1997년~2001년 명지대학교 산업대학원장

1981년~현재 명지대학교 컴퓨터공학과 교수

관심분야 : Spatial DB, Multimedia DB, Web DB, XML DB, Large Fingerprint DB, Data Warehousing & Data Mining 등