Voronoi 다이어그램을 이용한 고속 최근접 검색 기법

권동섭 °, 최원익, 박명선, 이석호
{subby, styxii, mspark}@db.snu.ac.kr, shlee@comp.snu.ac.kr
서울대학교 컴퓨터공학과

Fast Nearest Neighbor Search using Voronoi Diagram

Dongseop Kwon °, Wonik Choi, Myungsun Park, Sukho Lee
Dept. of Computer Engineering, Seoul National University

요 약

최근접 검색(nearest neighbor search)을 위해서 대부분의 기존 기법들은 데이타를 특정한 공간 인덱스 구조를 이용하여 인덱싱하고 이 인덱스를 이용하여 질의를 수행하는 방법을 사용하였다. 본 연구에서는 이러한 데이타 자체를 인덱싱하는 방법과는 달리 미리 최근접 질의의 결과가 되는 Voronoi 다이어그램을 생성해두고, 이를 통하여 최근접 검색을 수행하는 VGrid(Voronoi diagram-Grid)기법을 제안한다. 이 방법은 미리 모든 데이타에 대한 Voronoi 다이어그램을 계산하고 그 결과를 격자(grid)를 이용하여 인덱싱한 다음 최근접 검색 질의가 주어지면 이 격자 인덱스를 이용하여 빠르게 결과를 찾아낸다. 이 방법을 이용하면 처음 인덱스를 생성할 때는 많은 계산 시간이 소모되지만, 일단 인덱스가 구성되고 나면 최근접 검색 질의 처리 시 디스크 접근 회수가 줄기 때문에 기존의 기법에 비해 빠르게 최근접 검색 질의를 수행할 수 있다.

1. 서론

최근 데이타베이스 응용에서 최근접 검색은 중요한 기능으로 부각되었다. 멀티미디어 데이타베이스에서의 내용 기반 검색이나 유사도 검색뿐 아니라 GIS, 데이타마이닝 등의 여러 가지 데이타베이스 응용에서 최근접 검색은 널리 사용되고 있다. 지금까지 최근접 검색 기법에 관한 연구들은 일반적으로 전체 데이타를 R-tree나 Grid 파일과 같은 특정한 인덱싱 구조에 저장하고 이 인덱스를 이용하여 최근접 검색을 수행하는 방법에 관한 것이었다. 하지만, 이러한 기법들은 데이타의 차원이 높아지고 크기가 커질수록 인덱스 구조의 효율성이 떨어지고, 이에 따라 최근접 검색 시 디스크 접근회수가 늘어나게 된다.

본 논문에서는 Voronoi 다이어그램을 이용하여 미리 최근접 검색의 해를 구해두고 이를 이용하여 최근접 질의를 빠른 시간에 처리할 수 있는 방법을 제시한다. 차원이 높지 않고, 데이타의 변화가 작은 경우 본 논문에서 제시한 방법을 이용하면 대규모의 데이타에 대해서도 기존 기법보다 빠르게 최근접 검색을 수행할 수가 있다.

본 논문의 구성은 다음과 같다. 2절에서는 관련 연구를 살펴보고, 3절에서 본 논문에서 제안하는 방법인 VGrid기법을 소개한다. 그리고, 4절에서는 실험을 통하여 VGrid기법과 다른 기법을 비교 분석하고, 마지막으로 5절에서 결론을 짓는다.

2. 관련 연구

2.1 Voronoi 다이어그램

기하학에서 하나의 객체에 대하여 다른 객체들 보다 가까이 있는 모든 점들로 이루어진 공간을 그 객체에 대한 Voronoi 셀(Cell)이라고 한다. 그리고, 주어진 객체의 집합에 대하여 각 객체의 Voronoi 셀로 구분된 공간을 그 객체 집합에 대한 Voronoi 다이어그램이라고 한다[1][2]. 예를 들어 2차원의 유클리디안(Euclidean) 공간에 10개의 점이 주어졌다고 하면 이 점들에 대하여 그림 1의 (a)와 같이 10개의 Voronoi 셀로 구분된 Voronoi 다이어그램을 구성할 수 있다. 그림에서 알 수 있듯이 Voronoi 다이어그램은 주어진 객체에 대한 최근접 검색의 해답 영역이 된다.

2.2 최근접 검색 기법

서론에서 살펴본 바와 같이 공간 인덱스를 이용한 다양한 최근접 검색 기법[3][4]이 이미 제안되어 있지만, 여러 가지 단점을 지니고 있다. 최근, 이러한 방법의 한계를 극복하기 위한 새로운 접근 방법으로서 데이타 자체를 인덱싱하지 않고 해답 영역을 인덱싱하는 방법이 제안되었다[5]. 이 방법은 우선 최근접 검색의 해답 영역이 되는 Voronoi 다이어그램을 그림 1의 (b)와 같이 MBR(Maximum Boundary Rectangle)로 근사화하여 이를 R-tree 계열의 인덱스에 저장한다. 이 인덱스는 최근접 질의만을 위한 전용 인덱스로서 최근접 검색 질의가 주어졌을 경우 이 인덱스에 최근접 질의 대신 점 질의를 수행하여 최근접 질의를 수행한다. 이 방법은 최근접 검색 대신 상대적으로 복잡도가 작은 점 질의를 수행하게 되기 때문에 질의 응답 시간을 줄여줄 수 있고 고차원의 데이타에 대해서도 효과적인 질의 처리를 할 수 있는 장점이 있다. 하지만, 겹치는 영역이

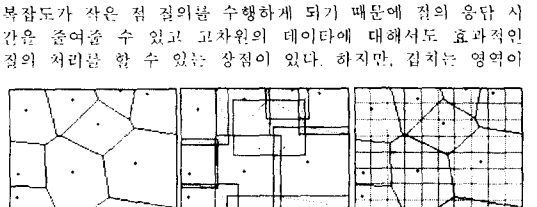

그림 1 Voronoi 다이어그램과 MBR 근사 기법, VGrid

전혀 없는 Voronoi 다이어그램을 MBR로 근사해야 하므로 그림 1의 (b)와 같이 겹치는 영역이 커지게 되어 점 질의를 수행할 때 후보가 검색될 수도 있다. 이런 이유로 저차원의 데이타에서는 오히려 이전의 연구들보다 오히려 성능이 떨어지는 경우도 발생하는 단점이 있다.

3. VGrid(Voronoi diagram-Grid) 기법

본 논문에서는 저차원 데이타에서 [5]의 기법이 가지는 위와 같은 단점을 보완하기 위하여 VGrid기법을 제안한다. VGrid 기법은 Voronoi 다이어그램을 [5]과 같이 MBR로 근사시키지 않고 미리 정의된 동일한 크기의 격자로 근사시키는 방법이다. 이러한 근사 방법은 공간 객체를 격자로 근사시키는 기법인 4CRS(4 colors raster signature) 기법[6]과 유사하다.

3.1 VGrid 기법의 구조

VGrid 기법은 전체 공간을 그림 1의 (c)와 같이 미리 정의된 격자에 의해 나누고 이에 의해 나누어진 한 격자를 하나의 노드에 저장한다. 그리고, Voronoi 다이어그램의 한 셀 R이 지나는 격자에 해당하는 노드들에 R의 정보를 기록하여 놓는다. 격자의 개수가 미리 정의되어 있기 때문에 격자에 해당하는 노드의 디스크 페이지에 한번에 접근할 수 있고, 따라서 최근접 검색 질의가 들어왔을 때 질의 점(query point)에 해당하는 격자의 노드만을 읽으면 최근접 검색 질의를 처리할 수 있다.

그림 2와 같이 하나의 격자는 하나의 노드에 저장되고, 하나의 노드는 하나의 디스크 페이지에 저장된다. 그리고, 각 노드는 디스크 상의 기본 데이타 영역에 순차적으로 저장된다. 따라서 각 격자에 해당하는 노드는 한번에 디스크에서 읽고 쓸 수 있다. 만일 하나의 격자를 지나는 Voronoi 셀이 많아져서 하나의 노드에 저장할 수 없으면 오버플로우가 발생한다. 오버플로우 처리법은 3.4절에서 설명하겠다. 오버플로우가 많이 발생하면 최근접 검색 질의를 처리할 때 성능이 떨어진다. 그러므로 가장 중요한 것은 격자의 개수를 선택하는 것이다. 격자의 개수가 커지면 하나의 격자에 포함되는 Voronoi 셀의 개수가 적으므로 오버플로우도 줄어들게 되고, 그러므로 질의 수행 속도가 증가하지만, 디스크 공간이 많이 필요하다. 반대로 격자의 개수가 적으면 전체 디스크 공간은 줄어들지만, 오버플로우가 발생할 확률도 높아지게 된다. 따라서 VGrid를 생성할 때 데이타 분포나 개수에 따라 적정한 격자 개수를 미리 정의해주어야 한다.

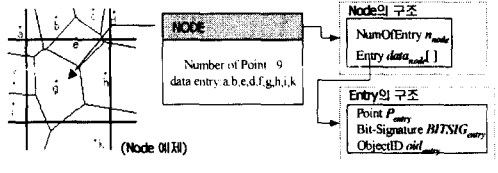

그림 2 VGrid 노드의 예와 구조

3.2 노드와 엔트리의 구성

그림 2와 같이 노드는 여러 개의 엔트리로 구성된다. 하나의 노드는 하나의 디스크 페이지에 저장될 수 있는 크기로 구성된다. 그리고, 엔트리는 하나의 Voronoi 셀을 나타낸다. 여기서 P_{entry}는 해당 Voronoi 셀에 해당하는 점 데이타이고, oid_{entry}는 이 점 데이타가 나타내는 객체의 식별자이다. Bit Signature $BITSIG_{entry}$는 해당 엔트리가 격자 안에서 어떤 구역을 지나는지에 대한 정보이다. 2차원의 경우 격자를 다시 4×4의 작은 격자로 나누어 각 격자를 지나는지 지나지 않는지를 그림 3과 같이 1비트로 표현하면 2바이트 값으로 Bit-Signature를 나타낼 수 있다. 이 Bit-Signature는 노드의

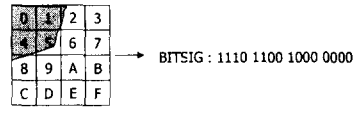

그림 3 Bit-Signature의 예

오버플로우를 처리할 때와 최근접 질의를 처리할 때 사용된다. 최근접 질의 수행 시 이 값을 먼저 비교하면 실제 거리 비교 연산의 회수를 줄일 수 있다. Bit-Signature의 비교는 비트 연산만으로 처리할 수 있으므로 거리 비교 연산보다 훨씬 빠르게 수행될 수 있다. 2차원의 경우 이러한 방식으로 엔트리를 구성하면 엔트리는 각 14바이트가 필요하고, 디스크 페이지 크기가 4KB라고 하면 하나의 노드는 292개의 엔트리를 가지게 된다.

3.3 VGrid의 생성 알고리즘

VGrid는 알고리즘 1과 같은 방법으로 생성할 수 있다.

```
Procedure BuildVGrid(PointsData) Begin
  // 주어진 Data로부터 Voronoi 다이어그램을 구한다.
  VoronoiDiagram ← GetVoronoiDiagram(PointsData)
  For each Voronoi Cell R in VoronoiDiagram
    MBR ← GetMBR(R)  //Voronoi셀의 MBR을 구한다.
    For each Grid G in MBR
      If Overlap(R,G) then
        // Grid G를 지나는 셀이면 Node에 삽입한다.
        Write R to GridNode(G) // Overflow처리 필요
      End If
    Next
  Next
End Procedure
```

알고리즘 1 VGrid 생성 알고리즘

3.4 오버플로우 처리 기법

Node에 더 이상 엔트리를 저장할 공간이 없으면 오버플로우 처리를 해야 한다. 오버플로우가 발생하면 다음과 같은 두 가지 방법을 이용하여 처리한다.

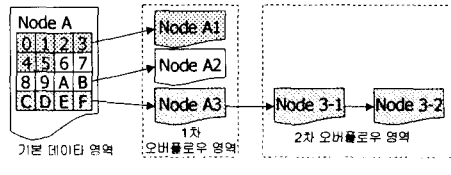

그림 4 오버플로우 처리

(1) Bit-Signature 기법

기본 데이타 영역의 노드에 오버플로우가 발생한 경우, 그림 4와 같이 오버플로우가 발생한 노드를 Bit-Signature에 따라 분할하여 1차 오버플로우 노드에 분할을 저장한다. 최대 한 비트가 하나의 노드를 연결할 수 있으므로 2차원의 경우 최대 16개의 노드를 연결할 수 있다. 이러한 방식으로 오버플로우를 처리하면 Bit-Signature를 비교하여 바로 오버플로우 노드를 찾을 수 있으므로 거리 비교를 해야하는 후보의 수가 줄어든다. 따라서, 질의를 수행할 때 성능 저하가 작다.

(2) 선형 연결 기법

1차 오버플로우 노드에서 다시 오버플로우가 발생한 경우, 1차 오버플로우 노드에 선형으로 2차 오버플로우 노드를 계속 연결한다. 이러한 연결노드가 늘어나면 질의 수행 시 디스크 접근 회수가 증가할 뿐 아니라, 연결된 노드의 모든 엔트리가 거리 비교의 후보가 되므로 질의 수행 성능이 떨어진다. 따라

서 오버플로우가 너무 많이 발생하면 격자를 더 잘게 나누어 인덱스를 재구성하는 것이 바람직하다.

3.5 최근접 검색 알고리즘

VGrid의 최근접 검색기법은 다음과 같다.

① 질의 점에 해당하는 격자를 찾는다.
② 격자에 해당하는 노드를 읽는다.
③ 오버플로우 노드가 아닌 경우, 노드의 데이타중 질의 점과 가장 가까운 데이타를 찾는다. 오버플로우 노드인 경우 오버플로우 처리 방법에 따른 다음 연결 노드를 읽고 ③을 반복한다.

기본적으로 오버플로우가 없다면 VGrid는 질의 점이 포함된 격자의 노드 하나만 읽어오면 되므로 한 번의 디스크 접근으로 최근접 검색을 처리할 수 있다. 최악의 경우 가장 길게 연결된 노드의 개수만큼 디스크 접근이 필요할 수도 있다. 하지만, VGrid를 생성할 때 격자의 개수를 적정한 수준으로 선택하면 오버플로우 노드의 연결 개수를 줄일 수 있으므로 평균적인 디스크 접근 회수는 최소화 할 수 있다. 따라서, VGrid 기법은 일반적인 공간 인덱스들의 최근접 검색 기법에 비해 월등히 적은 디스크 접근 회수만으로 최근접 질의를 처리할 수 있다. 이에 비하여 R-tree계열의 인덱스의 경우 최소한 트리의 깊이 이상 디스크 접근이 필요하고, 겹치는 영역이 많은 경우나 데이타가 많은 경우 여러 번의 트리 탐색이 필요할 수도 있다. [5]의 기법 역시 입력된 데이타 개수만큼의 Voronoi 셀을 MBR로 근사하여 트리에 저장해야 하므로 겹치는 영역이 증가하여 수행 성능을 떨어진다. 비록 점 질의를 수행한다고는 하지만, 최소한 트리의 깊이 이상, 트리의 겹치는 영역이 많다면 훨씬 더 많은 수의 디스크 접근이 필요하다. 뿐만 아니라 트리의 순회 시나 데이타 노드에서 데이타를 읽고 난 후 VGrid 보다 훨씬 많은 거리 비교 연산이 필요하다. 따라서 실제 CPU 연산시간도 늘어나게 된다.

VGrid의 경우 미리 오버플로우의 연결 개수를 n개 이하가 되도록 격자의 개수를 선택하여 VGrid 인덱스를 생성한다면 디스크 접근 회수는 반드시 n회 이하이므로 최근접 질의의 수행시간을 최대 한계를 미리 정의할 수 있다. 따라서, 최근접 질의의 응답시간이 매우 중요한 응용에서는 VGrid를 이용하면 최근접 질의에 대한 응답 시간을 보장할 수 있다.

4. 실험결과

본 연구에서는 2차원의 점 데이타를 대상으로 하여 VGrid 기법을 구현하고, Voronoi 셀을 MBR로 근사시키는 [5]의 기법, 그리고 R*-tree에 데이타를 인덱싱한 기법과 VGrid 기법을 비교 분석하였다. 디스크 페이지의 크기는 4KB로 가정하였다. R*-tree는 [5]의 저자인 S. Berchtold가 공개한 소스 코드를 이용하였고, Voronoi 다이어그램 생성은 qhull[7] 프로그램을 이용하였다. 실험은 512MB의 CPU와 18GB의 HDD를 가진 UltraSPARC-IIi 333MHz 기계에서 수행하였으며 사용 OS는 Solaris 7이다.

4.1 VGrid 생성 실험

임의로 생성한 10만개에서 100만개의 점을 가지고 VGrid를 생성하여 보았다. 표1은 100×100개의 격자를 사용한 경우 데이타의 개수에 따른 생성시간과 노드 당 평균 엔트리 수이다. 참고를 위하여 100만개의 점을 R*-tree에 저장한 경우의 생성시간과 트리 구성 정보와 비교해 보았다. 실험 결과, VGrid의 생성시간이 R*-tree의 생성 시간보다 6배정도 더 걸렸다. 비록, 생성 시간은 많이 걸렸지만 100×100개의 격자를 이용한 경우 100만개의 점에서도 전혀 오버플로우가 발생하지 않았다. 그리고, 이 경우 40MB정도의 디스크 공간이 필요하였다.

표 1 VGrid와 R*-tree의 생성 결과

	점 개수	생성시간	노드 당 평균 엔트리 수
VGrid (100×100)	10만개	17분	19.45개
	30만개	32분	45.28개
	50만개	49분	69.30개
	100만개	92분	126.12개
R*-tree	점 개수	생성시간	트리 구성
	100만개	16분	높이 3, 노드 개수 4400개

4.2 최근접 질의 수행

4.1에서 생성한 VGrid 인덱스와 R*-tree, [5]의 MBR 근사 기법에 대해 임의로 생성한 10000개의 점을 가지고 최근접 질의를 수행하여 보았다. 그림 5는 이 때의 평균 최근접 검색 질의응답 시간이다. VGrid는 오버플로우가 발생하지 않았기 때문에 각 경우에 모두 한 번의 디스크 접근으로 최근접 질의를 수행할 수 있다. 따라서 질의 응답 시간은 거의 동일하였다. R*-tree의 경우 트리의 높이는 3이었고, 실험 결과 평균 4~5회의 디스크 접근이 필요하였다. MBR-근사 방법 역시 R*-tree와 거의 유사한 응답 시간을 보여 주었다. 실험 결과 VGrid 기법은 다른 방법의 1/5정도의 수행시간에 최근접 검색을 수행할 수 있었다.

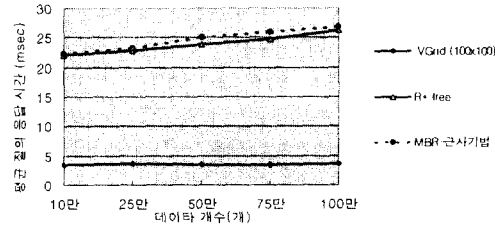

그림 5 최근접 질의 응답 시간 비교

5. 결론

본 논문에서는 최근접 검색 알고리즘에 대한 새로운 접근 방법으로서 최근접 검색의 해인 Voronoi 다이어그램을 이용하여 해당 공간 자체를 인덱싱하는 방법인 VGrid 기법을 제안하였다. 최근접 검색의 결과를 미리 계산해두는 방법이므로 데이타의 변화가 작은 경우 본 기법을 이용하면 처음 VGrid 인덱스를 생성할 때는 더 많은 계산 시간이 필요하지만, 실제 최근접 검색을 수행할 때는 기존 기법보다 빠르게 최근접 검색의 결과를 구할 수 있다. 따라서 최근접 질의의 응답 시간이 중요한 응용에서는 효과적으로 사용될 수 있다.

참고 문헌

[1] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf, "Computational Geometry, Algoruthms and Applications", Springer, 7:145-161, 1997

[2] F. Aurenhammer, "Voronoi Diagrams-A Survey of a Fundamental Geometric Data Structure", ACM Computing Surveys, Vol. 23, No. 3, September 1991

[3] T. Seidl, H. P. Kriegel, "Optimal Multi-Step k-Nearest Neighbor Search.", SIGMOD, 1998

[4] F. Korn, N. Sidiropoulos, C. Faloutsos, E. Siegel, Z. Protopapas, "Fast Nearest Neighbor Search in Medical Image Databases", VLDB, 1996

[5] S. Berchtold, B. Ertl, D. A. Keim, H. P. Kriegel, T. Seidl, "Fast Nearest Neighbor Search in High-dimensional Space", ICDE, 1998

[6] G. Zimbrao, J. M. de Souza, "A Raster Approximation for the Processing of Spatial Joins", VLDB, 1998

[7] Qhull, http://www.geom.umn.edu/locate/qhull