예외자가 공간 색인에 미치는 영향에 관한 분석

김시완 정성훈 이기준
부 산 대 학 교 전 산 학 과
{ swkim, shjung }@quantos.cs.pusan.ac.kr, lik@hyowon.cc.pusan.ac.kr

An Analysis of Outlier Effect on Spatial Indices

Si-Wan Kim Sung-Hoon Jung Ki-Joune Li
Dept. of Computer Science, Pusan National University

요 약

예외자는 공간색인의 성능을 떨어뜨리는 요소가 될 수 있다. 예를 들어, R-tree 계열의 색인 방법은 예외자 때문에 MBR의 넓이나, MBR사이의 겹치는 공간이 넓어져 성능이 떨어진다. 따라서, 공간색인을 구축할 때, 적절하게 예외자를 처리하면 성능을 향상시킬 수 있다. 본 논문에서는 예외자와 공간색인의 성능과의 관계를 관찰하고 적절한 방법으로 예외자를 처리하여 공간색인의 성능을 향상시키는 방법을 확인한다. 실험의 결과에 따르면, 예외자를 적절하게 처리할 경우 성능을 평균적으로 15% 향상시킬 수 있다.

1. 서론

공간색인의 성능은 데이터의 분포에 많은 영향을 받는다. 그러나 공간객체의 분포를 특성화 하는 것은 많은 어려움이 있다. 지리정보시스템이나 공간데이터베이스와 같이 실세계의 공간객체는 여러 인위적인 요인으로 인해 수학적으로 서술되기 힘든 특성을 가지고 있다. 특히, 실세계의 공간객체에는 많은 예외자가 존재한다.

예외자는 일반적으로 다른 공간적인 객체와 상당히 고립되어 있는 객체를 말한다. 이러한 예외자는 공간색인의 성능에 많은 영향을 준다. 예외자로 인하여 R-tree[4] 계열의 공간색인에서 MBR의 넓이는 불필요하게 넓어지고, 겹침 현상은 더욱 심해져 성능이 떨어진다.

따라서, 예외자를 적절하게 처리하면, MBR의 넓이를 상당히 줄일 수 있고, 또한 불필요한 겹침을 줄이며 결과적으로 공간색인의 성능을 향상시킬 수 있다. 본 논문에서는 우선, 예외자가 공간색인에 주는 영향을 분석하여, 예외자를 처리할 경우, 얼마만큼의 성능을 향상시킬 수 있는지를 살펴본다. 또한 기존의 공간색인 방법의 성능을 향상시킬 수 있도록,적절한 예외자의 처리 방법을 제안한다. 실험 결과에 따르면, 본 논문에서 제안된 예외자 처리방법으로 기존의 공간색인의 성능을 평균 15%정도 향상할 수 있었다.

본 논문은 다음과 같이 구성되어 있다. 2장에서는 지금까지의 관련연구를 살펴본다. 3장에서는 예외자가 R-tree의 성능에 미치는 영향에 대해 알아보고 4장에서는 예외자를 조정하여 R-tree의 성능을 향상시키는 방법을 제안한다. 5장에서는 제안된 방법의 성능 향상에 대한 실험 결과를 살펴보고, 6장에서 결론을 맺는다.

2. 관련연구

공간객체의 특성과 공간색인의 성능간의 관계를 분석하는 것은 공간색인의 성능에 영향을 주는 요소를 발견하여, 공간색인의 성능을 향상시키는 방법을 개발하는데 매우 중요한 역할을 한다. R-tree 계열의 공간색인 방법의 성능은 MBR의 수, 크기, 모양에 따라 공간색인의 성능이 결정된다. MBR의 특성은 MBR의 분할 방법에 따라 주로 좌우되는데, 분할의 효율성은 주로 공간객체의 분포에 많은 영향을 받는다.

지금까지의 연구는 공간객체의 일반적인 위치적 분포나, 크기의 분포 등을 중심으로 공간색인의 성능이 어떻게 달라지는가에 대한 연구가 대부분이었다. 그러나 MBR을 이용하는 R-tree 계열의 공간색인 방법은 예외자가 성능에 매우 중요한 영향을 준다. 따라서, 본 논문에서는 공간객체의 예외자와 공간색인의 성능 사이의 관계를 살펴본다. 이를 위해서 우선 공간객체에서 예외자를 정의해야 한다. 또한 예외자를 효과적으로 발견하는 방법도 필요하다. 최근에 예외자에 대한 주목할 만한 몇몇의 연구가 이루어지기 시작하였다. 각 연구에서는 약간씩 다른 예외자에 대한 정의를 내리고 예외자를 효과적으로 발견하는 방법을 제시하였다. [1]은 예외자를 " 데이터 집합 T에서 어떤 객체 O가 있을 때 객체 O로부터 거리 D보다 더 멀리 떨어져 있고 최소한 fraction p를 가지는 객체들" 로 정의하였다. 이 정의는 매개변수 p와 D를 사용자가 주어야 한다. [2]는 이 정의를 사용하면서, 하나의 공간객체가 주위의 공간객체로부터 얼마만큼 고립되어 있는가를 나타내는 LOF(Local Outlier Factor)를 이용하여 예외자를 찾는 방법을 제시하였

다. 그러나 [3]은 예외자의 정의를 약간 수정하여 예외자를 " k와 n이 주어졌을 때 객체 p의 k번째 가까운 이웃 객체의 거리가 멀리 떨어져 있는 순으로 n번째 이내에 있는 것" 으로 정의하였다. 이 정의는 k-nearest를 이용하여 예외자를 정의한 것이다. 이것의 장점은 위의 두 방법과 달리 특정한 매개변수가 필요 없다는 것이다. 또한 예외자에 대한 순위를 부여할 수 있다는 장점도 있다.

본 논문에서는 [3]의 정의가 본 논문에서 살펴 보고자 하는 예외자의 개념과 가장 유사하기 때문에 [3]에서 제안된 예외자의 정의와 예외자 발견 방법을 이용할 것이다.

본 논문에서 다루는 공간색인 방법은 R-tree 계열의 공간색인 방법이다. R-tree 계열, 특히 R*-tree[5]는 다른 공간색인 방법에 비하여 일반적으로 좋은 성능을 보여 가장 널리 사용되는 방법이다.

3. 예외자와 R-tree의 성능

R-tree의 성능에 영향을 주는 요소는 여러 가지가 있다. 그 중에서 중요한 것은 공간객체의 분산에 관련된 것이다. 본 논문에서는 그 중에서도 예외자와 R-tree 계열의 성능사이의 관계를 살펴보고자 한다. 여기서 다루는 것은 2차원 공간으로 전체 공간을 $[0,1)^2$로 가정한다.

3.1 R-tree의 성능

R-tree의 성능은 대부분 노드 MBR에 의해 결정된다. 즉, MBR의 크기, 모양, MBR 사이의 겹침 등이 R-tree의 성능에 많은 영향을 준다. 좋은 성능을 보이기 위해서는 MBR이 다음의 세가지 조건을 만족해야 한다.

조건 1: MBR의 넓이가 가능한 작을 것
조건 2: MBR 사이의 겹침이 가능한 작을 것
조건 3: MBR이 가능한 정사각형일 것

3.2 예외자와 MBR

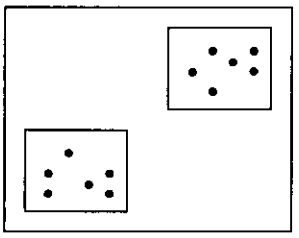

그림 1. 예외자 없는 MBR

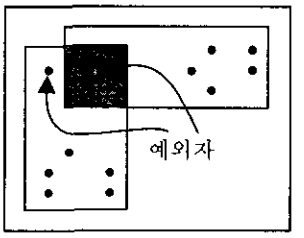

그림 2. 예외자 있는 MBR

그림 1과 그림 2는 같은 수의 객체를 R-tree에 삽입했을 때 나타난 모양이다. 그림 1은 예외자가 없기 때문에 최적화된 형태로 나타난다. 그러나 그림 2는 예외자로 인해 그림 1의 경우 보다 최적화 되지 못했다.

먼저 그림 2는 그림 1보다 MBR의 넓이가 많이 늘어났다. 그리고 MBR 사이의 겹침이 늘어 났으며 MBR의 모양이 정사각형에서 더 멀어졌다. 이와 같이 예외자는 3.1에서 보인 조건 3가지를 모두 위배하는 결과를 초래한다. 그러므로 예외자로 인해 R-tree의 성능이 줄어들 것이라는 것을 예측 할 수 있다.

4장에서는 이러한 예외자를 처리하는 방법에 대해서 소개할 것이다.

4. 예외자의 제어

본 장에서는 R-tree의 성능을 향상시키기 위하여, 예외자를 제어하는 방법을 제안한다. 예외자는 R-tree의 성능에 영향을 주기 때문에 적절하게 처리하면, R-tree의 성능을 향상시킬 수 있다. 본 논문에서 제안하는 예외자 제어 방법의 기본적인 개념은 예외자를 찾아내어 다른 공간객체와 분리하여 관리하는 것이다. 즉, 다음과 같은 두 가지 단계로 나누어 예외자를 처리한다.

단계 1 : p개의 예외자의 OID를 찾는다. C(p)
단계 2 : C1(=C-C(p))을 위한 R-tree를 구축하고, C(p)는 별도의 공간에 저장한다.

위의 방법은 원래의 데이터 집합에서 예외자에 해당되는 객체를 찾아내어 제거하고, 나머지 객체만으로 R-tree를 구축하는 것이다. 예외자로 제거된 객체는 예외자 공간에 별도로 저장된다. 그런데 예외자를 저장하는 공간은 주기억장치에 유지하는 것이 효율적이다.

4.1 예외자의 발견

본 논문에서 제안하는 예외자를 제어하는 방법을 적용하기 위해서는 먼저 예외자를 발견하는 방법이 필요하다. 앞의 2장에서 예외자를 발견하는 방법에 대하여 설명하였다. 그런데, 여러 가지 방법 중에서 하나를 선택할 때, 고려해야 하는 사항은 예외자의 개수를 제어할 수 있는가 하는 것이다. 즉, 주어진 데이터 집합에서 상대적으로 제한된 주기억장치에 저장할 수 있는 정도의 예외자만 선택할 수 있어야 한다.

앞 장에 열거된 방법 중에서 이와 같은 조건을 가장 적절하게 만족시키는 것은 [3]에서 제안된 방법이다. 이 방법은 우선 p의 k-nearest 객체의 거리 $D_k(p)$가 큰 n개의 객체를 예외자의 집합으로 정의한다. 이 정의의 장점은 사용자가 원하는 n개의 예외자를 선택할 수 있다는 사실이다. LOF와 같은 다른 예외자의 정의를 통하여서는 예외자의 개수를 사용자가 임의로 조정할 수 없다.

이와 같은 사실을 바탕으로, 본 논문에서 사용하는 예외자의 정의는 [3]에서 의하여 제안된 방법을 이용한다.

4.2 예외자의 저장

발견된 예외자는 주기억장치에 따로 저장하여 관리한다. 만일 예외자 공간의 크기를 적당한 크기, 예를 들어 1K bytes 또는 4K bytes로 제한하면, 시스템에 거의 부담을 주지 않는다. 따라서, 예외자의 개수는 주기억장치에서 사용할 수 있는 예외자 공간의 크기에 의하여 결정된다. 하나의 예외자를 표현하기 위해서 (ID, x, y)의 데이터가 필요하고, ID, x, y가 각 4 bytes로 표현되며 예외자 공간의 크기가 4K bytes라면, 350개 정도의 예외자를 저장할 수 있다.

다음으로 예외자를 제거한 나머지 객체를 이용하여 R-tree를 구축한다. 본 논문에서는 R-tree 중에서도 가장 안정적이고 성능이 좋은 R*-tree를 사용하였다. 그런데, 전체 객체의 수는 매우 크기 때문에 예외자를 제외하여도 R*-tree의 크기나 전체 노드의 수에는 크게 영향을 받지 않는다.

공간질의가 주어지면, 우선 예외자 공간에 있는 공간객체를 검색한다. 그와 동시에 R*-tree를 이용하여 공간연산을 수행하여, 원하는 공간객체를 검색하게 된다.

5. 실험

본 장에서는 앞장에서 제안된 방법을 실세계 데이터와 합성된 데이터에 적용하여 성능이 실제로 얼마만큼 향상되는가를 살펴본다.

실험을 위하여 준비된 실세계 데이터는 서울의 주요 건물을 나타내는 68736개의 점으로 구성된 데이터와 Tiger 파일에서 추출된 Long Beach County의 36548개의 점으로 구성된 데이터 2가지이다. 이들 데이터는 다음 그림과 같다.

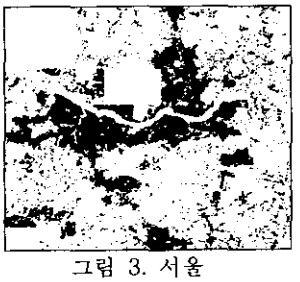

그림 3. 서울

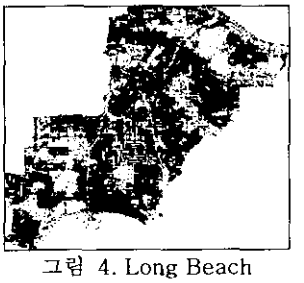

그림 4. Long Beach

그림 3과 그림 4의 데이터를 이용하여 여러 가지 실험을 하였다. 그림 5는 R-tree 단말 노드 MBR의 넓이와 예외자의 관계를 나타낸 것이다. 여기서 x축은 예외자의 개수이고 y축은 단말 노드 MBR의 넓이의 평균을 의미한다. 예외자의 수가 증가되면 단말 노드 MBR의 넓이가 작아지므로 성능이 향상됨을 그림 5에서 확인할 수 있다. 그러나 이 경우 예외자의 수가 적절한 수를 넘어서면 성능의 향상이 미미하여 진다.

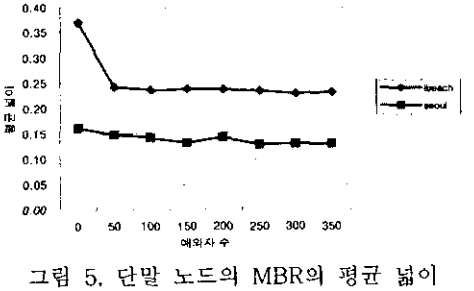

그림 5. 단말 노드의 MBR의 평균 넓이

실제로 R-tree의 성능은 MBR의 넓이 뿐만 아니라 MBR 사이의 겹침, 정사각형에 가까운 정도 등에 영 향을 받아 디스크 액세스의 수로 나타난다. 예외자의 개수가 변함에 따라 단말 노드의 디스크 액세스 수가 어떻게 달라지는지를 실험한 결과를 표 1에서 보여준다.

여기서 행은 데이터의 종류를 열은 예외자의 수를 나타낸다. s는 서울 데이터를 b는 Long Beach를 나타내는 것이고 0.1, 0.01은 질의의 종류를 나타내는 것이다. 표에 나타난 값은 질의를 1000번 주었을 때 각 디스크 액세스 수를 모두 합한 값이다.

	s 0.1	b 0.1	s 0.01	b 0.01
0	26297	17897	1502	1036
50	26117	17642	1401	939
100	25577	17561	1402	900
150	25450	17474	1377	912
200	25936	17422	1341	915
250	25373	17428	1275	898
300	25430	17493	1291	912
350	25372	17307	1310	878

표 1. 단말 노드의 디스크 액세스 수

표 1의 모든 경우에 단말 노드의 디스크 액세스 수는 예외자를 제거하기 전 보다 최고 20%정도 줄어드는 것을 알 수 있다. 평균적으로는 15%정도 줄어들었다. 그러나 여기서도 그림 2와 같이 적절한 수를 넘어서면 성능의 향상이 미미하여 진다.

6. 결론

본 논문에서는 R-tree와 같은 공간색인의 성능에 예외자가 어떤 영향을 미치는가를 알아 보았다.

또한, 예외자를 처리하기 위한 방법을 제시하고 그에 대한 실험을 통해 얼마나 성능이 향상되는가를 알아보았다. 예외자를 처리하였을 경우 디스크 액세스 회수가 최고 20% 줄었고 평균 15%의 디스크 액세스가 줄어들었음을 실험으로 확인할 수 있었다.

7. 참고 문헌

[1] E. Knorr and R. Ng, Algorithms for mining distance-based outliers in large datasets, *Proc. VLDB Conf*, 392-403, 1998

[2] M. M. Breunig, H.-P. Kriegel, Raymond T. Ng and J. Sander, LOF: Identifying Density-Based Local Outliers, *Proc. SIGMOD Conf*, 93-104, 2000

[3] S. Ramaswamy and R. Rastogi and K. Shim, Efficient Algorithms for Mining Outliers from Large Data Sets, Proc. SIGMOD Conf, 427-438, 2000

[4] A. Guttman, R-Trees A Dynamic Index Structure For Spatial Searching, Proc. SIGMOD Conf, 47-57, 1984

[5] N. Bechmann, H.-P. Kriegel, R. Schneider and B. Seeger, The R*-tree: An Efficient and Robust Access Method for Points and Rectangles, Proc. SIGMOD Conf, 322-331, 1990

[6] B. U. Pagel, H-W. Six, H. Toben, P. Widmayer, Towards an Analysis of Range Query Performancein Spatial Data Structures, Proc. PODS Conf, 214- 221, 1990