STR-Tree : 계층 공간 분할을 이용한 다차원 정적 데이터 색인

최미나° 문정욱 이기준
부 산 대 학 교 전 자 계 산 학 과
{ mnchoi, jwmoon }@quantos.cs.pusan.ac.kr, lik@hyowon.cc.pusan.ac.kr

STR-Tree : A Multidimensional Index Structure for Static Data using a Hierarchical STR

Mina Choi° Jung-Wook Moon Ki-Joune Li
Dept. of Computer Science, Pusan National University

요 약

최근 다차원 공간색인 방법의 성능 향상을 위해 근사법을 사용하여 노드의 팬아웃을 증가시키려는 시도가 많이 행해졌다. 하지만 이러한 방법은 색인 구조의 정확성이 떨어져 불필요한 노드를 방문할 확률을 높다는 단점이 있다. 본 논문에서는 정적 데이터에 대하여 노드의 팬아웃을 증가시키기 위해 하향식 STR 공간분할방법을 사용한 새로운 색인 방법을 제안한다. 제안한 방법은 공간분할방법을 사용하므로 근사법을 이용한 방법에 비해 정확성이 높을 뿐 아니라 하향식 계층 STR을 제안하여 STR 공간분할방법을 효율적으로 트리 구조에 적용할 수 있도록 하였다. 이 외에도 이중분할 방법을 제안하여 점 데이터 및 사각형 데이터의 색인을 가능하게 할 뿐 아니라 사장 공간을 줄여 불필요한 노드의 방문을 막아 성능을 향상시켰다.

1. 서론

대부분의 공간 데이터는 대용량의 데이터이다. 이러한 대용량 데이터를 이용하여 효율적인 질의 처리를 하기 위해서는 공간 데이터를 위한 적절한 색인이 필요하다.

공간 데이터의 색인 방법에는 여러 가지가 있다. 그 가운데 가장 널리 사용되는 방법이 R-tree[1] 계열의 데이터분배 방법을 이용한 공간 색인이다. R-tree 계열의 색인에서 성능을 향상하기 위한 연구가 계속되어 왔다. 특히, 디스크 효율을 높이기 위해 R^*-tree[3]와 같이 노드의 최소경계사각형의 넓이와 둘레를 최소화하여 디스크 참조를 줄이기 위한 연구 등이 있었고, Hilbert R-tree[5]나 STR[2] 방법을 이용한 패킹 R-tree와 같이 디스크 공간의 효율성을 극대화하여 디스크 참조를 줄이기 위한 연구가 있었다. 또한 근사법을 이용하여 노드의 팬아웃을 증가시킨 A-tree[6]와 같은 연구도 있었다. R-tree외에도 공간을 분할하여 데이터를 저장하는 방식의 색인도 연구되어 왔다[3]. 본 논문에서는 공간 분할 방법을 응용한 정적 공간 데이터의 색인을 제안한다. 제안한 방법에서는 디스크 참조 수를 줄이기 위해 노드의 팬아웃을 증가시킨다. 팬 아웃을 증가시키기 위해 하향식 계층 STR 공간분할 방법을 적용한다.

본 논문은 다음과 같이 구성되어 있다. 2장에서는 제안하는 STR-tree의 전체적인 특징과 공간 분할 방법을 설명하고 있다. 3장에서는 제안한 방법을 확장하여 사각형 데이터에 적용한 방법을 살펴본 후, 4장에서는 제안한 색인과 R-tree[1]계열인 R^*-tree[3]와의 성능을 비교한다. 끝으로 5장에서는 결론과 향후 연구에 대해서 살펴본다.

2. STR tree

본 논문에서 제안한 STR-tree에서는 R-tree 패킹에 사용되는 STR[2] 방법을 공간 분할에 적용한다. STR[2] 공간분할 방법은 주어진 영역에 대하여 데이터의 개수를 일정하게 분배하며 효과적으로 공간을 분할하는 방법이지만, 이를 그대로 트리 구조에 사용하기에는 적합하지 않다. 왜냐하면 전체 트리에 적용할 경우 상위 노드의 영역들이 서로 겹칠 가능성이 높기 때문이다. 이런 문제를 해결하기 위해 본 논문에서는 STR 공간분할 방법을 하향식(top-down) 계층적으로 적용한다. 그림 1은 하향식으로 STR 공간분할을 적용한 그림이다. 그림 1의 왼쪽 그림은 공간을 x축 2개, y축 2개로 분할한 것이다. 왼쪽 노드를 다시 하향식으로 분할한 것이 오른쪽 그림이다. 오른쪽의 공간 분할은 대칭추가분할 방법이다.

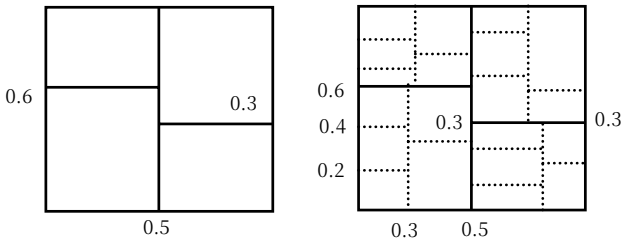

그림 1. 하향식 계층 STR 분할(높이=3)

하향식으로 STR-tree를 구축할 때 트리의 전체 높이를 추정하여 적절하게 공간을 분할하는 것이 가장 중요하다

2.1. 트리의 높이 계산

아래의 식 1은 STR-tree의 전체 트리의 높이를 추정한 식이다. N은 데이터의 수이고 $Bf_{non-leaf}$, Bf_{leaf}는 각각 중간노드와 단말노드의 팬아웃이다. STR-tree는 전체 트리의 높이를 정한 후에 루트 노드부터 공간분할을 시작한다. 중간 노드의 팬아웃을 이용하여 다시 하위 노드의 공간분할 적용하게 된다. 상위 노드부터 하위노드로 순환적으로 해당노드의 팬아웃을 이용한 공간분할을 적용한다.

$$tree\_depth(N) = \left\lceil \frac{MAX(\log N - \log Bf_{leaf}, 0)}{\log Bf_{non-leaf}} \right\rceil + 1 \quad \text{...식1}$$

2.2 공간 분할 방법

STR-tree는 하향식 공간분할을 이용한 색인으로 공간을 분

할하는 방법에 의해 색인의 성능에 큰 영향을 미치게 된다. 본 절에서는 STR-tree에서 사용하는 공간분할방법을 개량한 새로운 공간분할을 제안한다.

STR-tree에서 사용하는 공간분할 방법은 크게 다음의 세 가지로 분류할 수 있다. C는 축별로 분할한 공간 분할 수를 의미한다.

1) 대칭 분할 방법 — c^2
2) 비대칭 분할 방법 — $c_1 \times c_2$
3) 대칭추가 분할 방법 — $c^2+\alpha$

대칭 분할 방법은 공간을 대칭으로 분할하는 것으로 예를 들어 필요한 분할 수가 13일 때, x축과 y축에 대해 모두 4개의 영역으로 공간분할이 이루어진다. 이 경우 3개의 셀은 실제 사용되지 않으므로 낭비되고 있다. 같은 경우 비대칭 분할 방법을 적용하면 x축으로 7, y축으로 2의 공간으로 분할하게 되면 1개의 셀이 낭비되게 된다. 반면 대칭 추가 분할 방법을 적용하여 분할할 경우 x축과 y축 모두 3개의 셀이 분할되고 추가로 4개의 셀이 더 나눠지게 된다. 이 경우에는 아무런 셀도 낭비되지 않아 디스크 참조 수를 줄이게 되므로 가장 효율적이다. 따라서 대칭 분할 방법과 비대칭 분할 방법은 중간노드의 최대 팬아웃을 사용할 수 없는 단점이 있다. 하지만, 대칭추가 분할 방법은 대칭 분할을 한 후 중간 노드의 최대 팬아웃을 모두 활용할 수 있어 트리의 높이를 줄이는데 큰 역할을 한다. 그러므로 본 연구에서는 공간분할방법으로 대칭추가 분할 방법을 사용한다.

대칭추가 분할 방법은 필요한 분할을 할 때, 일단 대칭으로 분할한 후 부족한 분할을 추가로 분할하는 방법이다. 아래의 그림 2와 그림 3은 점 데이터에 기존의 STR[2]의 공간 분할 방법과 STR-tree의 공간 분할 방법인 대칭추가 분할 방법의 차이를 나타낸 것이다. 그림 2에서 회색 격자는 그림 3의 회색 격자보다 둘레길이가 큼을 알 수 있다.

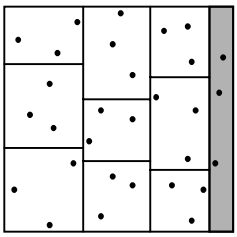

그림 2. 기존의 STR 공간분할

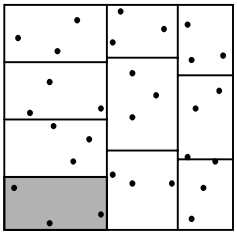

그림 3. 대칭추가분할 방법

STR-tree는 각 노드별로 STR 분할을 적용하여 색인 구조로 만든 것이다. STR-tree에 영역 질의가 주어질 경우, 방문한 노드의 STR 분할에 대하여 영역 질의와 겹치는 격자에 대하여 자식 노드를 방문하게 되는데, 한 레벨에서 STR-tree 색인 방법의 디스크 참조 수는 식 2[7]과 같다.

$$DiskAccess(R^2) = \sum_{i=1}^{n} (w_i + R)\cdot(h_i + R)$$

$$= 1 + p\cdot R + n\cdot R^2 \qquad \left(단, p = \sum_{i=1}^{n} (w_i + h_i) \right) \quad \cdots \quad 식\ 2$$

식 2에서 w_i와 h_i는 i번째 격자의 너비와 높이를 의미하고, R은 영역질의의 한 변의 길이를 의미한다. p는 STR 분할로 생긴 모든 격자들의 둘레길이 1/2을 의미하고, n은 격자의 개수를 의미하는데, 디스크 참조를 최소화하기 위해서는 격자의 개수를 최소화해야 하고, 또한 격자들의 둘레길이를 줄여야 한다. 둘레길이가 큰 노드가 작은 노드에 비해 디스크 참조 횟수에 약점을 가지게 됨을 식2에서 알 수 있다. 따라서 그림 2의 노드보다 그림 3의 노드가 디스크 참조를 줄인다는 사실을 알 수 있다.

2.3. 트리 구조

STR-tree의 노드는 크게 단말 노드와 중간 노드의 두 가지로 구분된다. 또 각각의 노드는 단일 분할일 경우와 이중 분할인 경우 저장하는 정보의 양이 달라진다.

STR-tree의 중간 노드는 셀 분할 정보와 자식 노드로의 포인터를 가지고 있다. 중간 노드를 R-tree와 비교할 때, 유지하는 정보의 양이 공간 분할 정보만 가지게 되므로 최소경계사각형을 가지는 경우보다 더 높은 팬아웃을 가지게 된다. 그림 4는 STR-tree의 중간 노드 구조이다. c는 각 축의 분할 수이며, α는 대칭추가분할에서 사용하는 추가되는 셀의 수로 $0 \le \alpha \le 2c$을 조건을 만족한다.

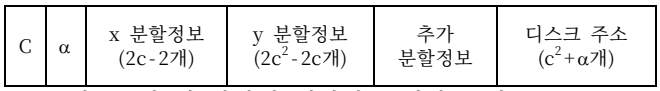

그림 4. 점 및 사각형 데이터를 위한 중간노드 구조

그림 4은 x축에 대하여 c개, y축에 대하여 c개로 영역을 이중분할할 경우 x 분할정보는 2c-2개 필요하며, y 분할정보는 $2c^2$-2c개 필요하다. 이때 α개의 추가분할정보가 필요할 경우 추가분할정보는 추가 분할 정보영역에 기록하게 된다. 그림 5에서의 추가분할정보는 $0 \le \alpha \le c$ 일 때 y분할정보 α개로 이루어지며, $c < \alpha \le 2c$ 일 때 x분할정보 1개와 y분할정보 α-1개로 이루어진다.

STR-tree의 단말노드 구조는 R-tree의 노드 구조와 같다. 즉, 데이터와 데이터의 디스크 주소를 저장하고 있다. STR-tree에서 특징적인 구조는 중간노드의 구조이다. 이 경우 단일분할과 이중 분할에서 저장하는 정보의 양이 다르다. 아래의 식 2와 식 3은 STR-tree와 R-tree의 중간노드 팬아웃을 식으로 표현한 것이다.

$$fanout_{STR-tree} = \frac{BlockSize - NodeHeadSize + FloatSize}{FloatSize + DiskAddressSize} \quad \cdots \ 식\ 2$$

$$fanout_{R-tree} = \frac{BlockSize - NodeHeadSize}{2\cdot FloatSize\cdot Dimension + DiskAddressSize} \quad \cdots \ 식\ 3$$

FloatSize=4, *DiskAddressSize*=4 일 때 STR-tree의 팬아웃은 R-tree의 팬아웃의 2.5배이다. STR-tree의 중간노드는 분할정보를 가지고 있는데 이로 인해 팬아웃의 증가를 가져오게 된다. 특히, 식 2에서 주목할 점은 STR-tree의 팬아웃이 차원에 영향을 받지 않는다는 점이다.

아래의 그림 5는 그림 1의 STR-tree를 도식화한 것이다. 높이 3인 트리는 중간 노드와 단말 노드의 두 가지 노드가 있으며, 각각의 노드에는 공간 분할 정보와 디스크 주소를 저장하고 있다. 데이터는 0에서 1사이로 정규화 되어있다. 루트 노드는 x축으로 한 개의 분할 정보, y축으로 2개의 분할 정보를 가지고 있다. 또한 루트의 하위 노드는 다시 2^2+1의 대칭추가분할이 이루어져 있다.

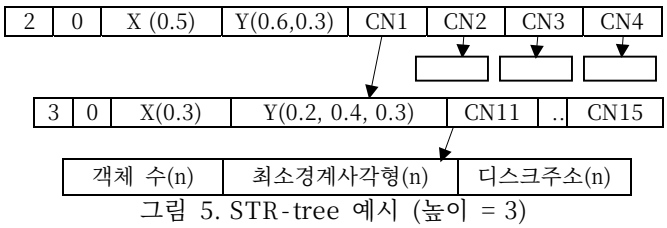

그림 5. STR-tree 예시 (높이 = 3)

3. 사각형 데이터의 처리

STR-tree에서는 단순분할과 이중분할의 두 가지 셀 분할 정보 저장 구조를 가진다. 점 데이터일 경우에는 그림 3와 같은 형태로 셀을 분할한다. 사각형 데이터의 경우에는 그림 6과 같이 데이터의 최소 값과 최대 값의 경계 값을 저장하는 이중 분할 저장구조를 사용한다.

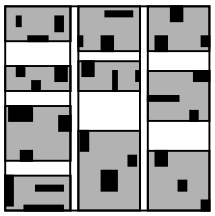

그림 6. 사각형 데이터의 셀 이중분할 정보(3^2 +1분할)

이중 분할인 경우 단순 분할 정보를 저장할 경우보다 저장해야 하는 셀 분할 정보가 두 배가 되지만 사장공간을 줄여 디스크 참조 수를 줄일 수 있다. 중간 노드의 팬아웃을 비교할 때, R-tree[1] 계열에서 사용하는 최소경계사각형에서 사용하는 x의 최소값과 최대값과 y의 최소값과 최대값을 저장하는 방법에 비해 절반의 정보를 저장하므로 상대적으로 노드의 저장 효율이 높다.

4. 실험

본 장에서는 STR-tree의 색인 방법과 R*-tree[3]의 방법을 비교하여 성능 향상 정도를 알아 본다. 실험에 사용한 데이터는 그림 7의 서울 지역 점 데이터(68,736개)와 그림 8의 미국 몽고메리카운티 지역의 점 데이터(27,282개)이다. 사용한 디스크 블록의 크기는 4Kbytes이며 이중공간분할 방법을 사용하였다.

그림 7. 서울 데이터

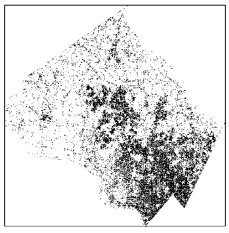

그림 8. 몽고메리카운티 데이터

표 1과 표 2는 R^*-tree와 STR-tree의 노드 접근 수를 비교하여 두 색인간의 성능을 비교한 것이다.

표 1. 몽고메리카운티의 R^*-tree STR-tree의 노드 참조 수

질의크기 (한 변의 길이)	데이터분포		균일분포	
	R*-tree	STR-tree	R*-tree	STR-tree
1/10	81056	21228	34,737	12,709
1/100	16546	5917	7,343	3,345
1/1000	5869	2994	5,902	3,047

표 2. 서울데이터의 R^*-tree STR-tree의 노드 참조 수

질의크기 (한 변의 길이)	데이터분포		균일분포	
	R*-tree	STR-tree	R*-tree	STR-tree
1/10	115,659	62,311	61,892	39,806
1/100	25,788	19,674	11,065	12,422
1/1000	8,308	9,964	4,879	9,731

표 1과 표 2에서 보는 바와 같이 질의 크기가 클 경우 월등한 성능 차이를 보인다. 그러나 질의 크기가 작아질 경우 성능이 떨어짐을 알 수 있다. 이는 R*-tree의 경우 최소경계사각형을 사용하므로 중간노드에서 불필요한 노드의 방문을 제한함으로 디스크 참조를 더 이상 하지 않기 때문에 비슷한 성능을 보이는 것이다.

STR-tree는 크게 두 가지의 장점을 갖는다. 첫째, 팬아웃을 늘려 트리의 높이를 줄여 디스크 참조 수를 줄인다는 것이다. 또, 식 2에서 알 수 있듯이 차원에 관계 없이 중간 노드의 팬아웃이 일정하다는 것이다. 아래의 표 3는 차원이 증가함에 따라 STR-tree와 R-tree의 중간노드 팬아웃을 비교한 것이다.

표 3. STR-tree와 R-tree의 중간노드 팬아웃 비교

(블록크기=1024bytes)

차원	R-tree	STR-tree	비율
2	51	128	2.510
4	28	128	4.571
6	19	128	6.737
8	15	128	8.533
10	12	128	10.667
12	10	128	12.800

5. 결론

본 논문에서는 STR-tree라는 정적 데이터를 위한 효율적인 공간색인을 제안하였다. 사용한 공간분할 방법은 대칭 추가분할 방법을 사용하였으며, 점 데이터와 정적 사각형 데이터 모두 적용 가능하다. 또한, 제시한 방법과 R^*-tree와의 성능을 비교하여 노드의 저장 효율 향상을 알아보았다.

본 논문에서는 2차원 데이터로 데이터 수가 10만개 이하의 데이터를 이용하여 실험하였다. 본 색인을 좀더 범용적으로 사용하기 위해서는 대용량의 다차원 데이터에 적용한 연구가 필요할 것이다. 또, 색인의 성능을 보다 향상시킬 수 있는 요소에 관한 추가적인 구현이 필요하다.

6. 참고 문헌

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

[2] S. T. Leutenegger, J. M. Edgington, M. A. Lopez, 'STR: A Simple and Efficient Algorithm for R-Tree Packing.' Proc. ICDE Conf, pp. 497-506, 1997

[3] N. Beckmann, H. P. Kriegel and R. Schneider , B. Seeger , 'The R*-tree: An Efficient and Robust Access Method for Points and Rectangles', Proc. ACM SIGMOD, pp. 322-331, 1990

[4] J. Nievergelt and H. Hinterberger, 'The Grid Files: An Adaptive, Symmetric Multikey File Structure', ACM TODS, vol. 9, No. 1, pp. 38-71, 1984

[5] I. Kamel, C. Faloutsos, 'Hilbert R-tree: An Improved R-tree using Fractals'. VLDB, pp. 500-509, 1994

[6] Y.i Sakurai, M. Yoshikawa, S. Uemura, H. Kojima, 'The A-tree: An Index Structure for High-Dimensional Spaces Using Relative Approximation', VLDB, pp. 516-526, 2000

[7] B.-U. Pagel, H.-W. Six, H. Toben, and P. Widmayer, 'Towards an Analysis of Range Query Performance in Spatial Data Structures', Pro. ACM PODS, pp. 214-221, 1993