TPR 트리에서 경계사각형 재구성 기법의 설계 및 구현

The Design and Implementation of Reorganization Schemes for Bounding Rectangles in TPR trees

김동현*, 홍봉희**
Dong-Hyun Kim, Bong-Hee Hong

요 약 TPR-tree의 각 노드는 이동체를 색인하기 위하여 시간 함수를 기반으로 한 경계사각형을 이용한다. 경계사각형의 각 축을 계산하기 위하여 시간함수를 사용하므로 시간이 흐름에 따라 노드의 경계사각형은 확장된다. 따라서 이웃하는 노드간의 중첩(overlap) 영역이 커지기 때문에 영역질의의 성능이 점차적으로 떨어지는 문제가 있다.
이 논문에서는 이동체 삽입과 삭제 시 노드의 경계사각형을 재구성하기 위한 기법들을 제시한다. 이동체를 삽입할 때 노드간의 중첩을 줄이기 위하여 중첩이 심한 두 개의 단말 노드를 강제 합병하고 재분할하는 강제 합병 기법을 사용한다. 그리고 이동체를 삭제할 때 다른 이동체도 재삽입하는 강제 재삽입 기법을 이용한다. 강제 재삽입 기법은 삭제 노드 강제 재삽입 기법과 중첩 노드 강제 재삽입 기법으로 분류된다. 실험 결과에서 중첩 노드 강제 재삽입 기법이 다른 두 기법에 비하여 우수함을 알 수 있다.

Abstract The TPR-tree exploits bounding rectangles based on the function of time in order to index moving objects. As time passes on, each edge of a BR expands with the fastest velocity vector. Since the expansion of the BR results in a serious overlaps between neighboring nodes, the performance of range query is getting worse.
In this paper, we propose schemes to reorganize bounding rectangles of nodes. When inserting a moving object, we exploit a forced merging scheme to merge two overlapped nodes and re-split it. When deleting a moving object, we used forced reinsertion schemes to reinsert other objects of a node into a tree. The forced reinsertion schemes are classified into a deleted node reinsertion scheme and an overlapped nodes reinsertion scheme. The overlapped nodes reinsertion scheme outperforms the forced merging scheme and the deleted node reinsertion scheme in all experiments.

주요어 : TPR 트리, 강제 합병, 강제 재삽입, 이동체 색인, 이동체 데이터베이스
Keywords : TPR-tree, forced merging, forced reinsertion, moving object index, moving object databases

1. 서 론

최근 무선 이동통신 기술의 발달로 휴대폰, 이동단말기(PDA)등의 무선이동기기가 보편화되면서 위치기반서비스(LBS, Location-Based Service) 응용이 증가하고 있다. 대표적인 응용의 예로 친구 찾기, 카네비게이션등이 있다[1][2][3]. 이러한 응용에서 처리하는 대표적인 질의 중 하나는: 다음 예와 같은 이동체 현재 위치 질의이다[1][2][3]"현재부터 10분 이내에 시청 앞을 지나갈 차량을 검색하라." 따라서 이동체의 현재 위치 데이터를 관리하고 현재 위치 질의를 효과적으로 처리하기 위한 이동체 색인에 관한 연구가 필요하다.

이동체 현재 위치를 위한 색인으로 메인 메모리에 상주하는 R-tree 계열의 색인을 주로 사용한다[1][2]. 왜냐하면 첫째, 최종 보고한 위치 정보를 이용하여 이동체의 현재 및 미래의 위치를 계산할 수 있기 때문에 현재 데이터만 색인하는 현재 위치 색인의 데이터 크기는 작다. 둘째, 시간 도메인은 동적으로 증가하기 때문에 만약 공간 분할 색인을 사용하면 시간 도메인을 관리하기가 어렵다.

이동체는 시간에 따라 객체의 위치 및 모양이 연속적으로 변경되는 데이터다[1][2][5]. 이동체에 대한 현재 및 미래 위치의 시공간 질의를 처리하기 위한 대표적인 색인으로 TPR-tree[1]와 TPR*-tree[2]가 있다. TPR-tree는 시간 함수를 사용하는 경계사각형

* 동서대학교 소프트웨어 전문대학원 pusrover@pusan.ac.kr
** 부산대학교 컴퓨터공학과 교수 bhhong@pusan.ac.kr

(BR, Bounding Rectangle)을 기반으로 이동체를 색인하였다. BR은 노드의 모든 이동체를 포함하는 경계사각형으로 질의 처리시 각 축의 방향으로 확장된다. TPR*-tree는 TPR-tree를 기반으로 노드간의 중첩 영역에 삽입되는 이동체를 위한 삽입 알고리즘을 제시하였다. 또한 갱신된 노드의 조상 노드들에 해당하는 BR들을 축소하는 BR 최적화(tightening) 방법을 제시하였다. 그러나 TPR-tree와 TPR*-tree는 시간이 흐름에 따라 주변 BR 즉, 노드간의 중첩 영역이 확대된다.

이동체의 현재 및 미래 위치 질의를 처리하기 위해서 TPR-tree는 질의 시간(tq)을 매개변수로 하는 시간함수를 사용하여 BR을 확장한다. BR의 확장된 축을 계산할 때 시간함수는 BR에 포함된 이동체 중에서 가장 빠른 속도로 이동하는 이동체의 속도 값과 질의 시간을 이용한다. 따라서 BR의 생성 후 질의까지의 시간이 증가될수록 BR의 영역도 그에 비례하여 확장되기 때문에 BR간의 즉, 주변 노드간의 중첩 영역의 크기도 시간이 지남에 따라 증가된다. 그러므로 시간이 지날수록 영역 질의의 성능이 점차적으로 저하된다.

이 논문에서는 시간이 지남에 따라 심화되는 노드간의 중첩 영역을 줄이고 영역 질의의 성능을 개선하기 위하여 이동체의 삽입과 삭제 시 노드의 BR을 재설정하기 위한 재구성기법들을 제시한다. 첫 번째 기법은 강제 합병 기법이다. 이 기법은 이동체가 삽입될 때 만약 삽입된 노드와 이웃 노드 간의 중첩이 심하면 두 노드를 강제 합병하고 재분할한다. 두 번째 기법은 삭제 노드 강제 재삽입 기법이다. 이 기법은 하나의 이동체를 삭제할 때 삭제가 발생한 노드의 모든 이동체를 강제로 색인에 재삽입한다. 세 번째 기법은 중첩 노드 강제 재삽입 기법이다. 이 기법은 삭제가 발생한 노드와 중첩되는 모든 노드의 이동체를 강제로 재삽입한다. 그리고 실험을 통하여 각 기법들을 사용하였을 때 영역 질의의 성능 개선도를 보여준다. 이 논문에서 제시한 재구성 기법들은 TPR-tree의 구조를 동적으로 최적화하는 장점을 가진다.

이 논문의 구성은 다음과 같다. 먼저 2장에서 관련 연구를 그리고 3장에서 문제정의를 기술한다. 4장에서 강제 합병 기법을 제시하고 5장에서는 두 가지 강제 재삽입 기법에 대하여 제시한다. 6장에서는 제안한 기법들을 사용한 실험 결과에 대하여 기술한다. 그리고 7장에서 결론 및 향후 연구를 기술한다.

2. 관련 연구

이동체의 현재 및 미래 위치에 대한 질의를 처리하기 위한 색인은 크게 공간 분할 방식에 기반한 색인 기법[6][7]과 데이터 분할 방식에 기반한 색인 기법[1][2]으로 분류될 수 있다.

공간 분할 기법에 기반한 연구로 해쉬 기반 색인[6]과 PMR-Quadtree[7]를 이용한 연구가 있다. 이동체 이동에 따른 갱신 비용을 줄이기 위하여 [6]에서는 이동체가 셀을 벗어나지 않으면 갱신 연산이 없다는 점을 이용하여 해쉬 기반 색인을 제시하였다. 그리고 이동체를 2차원 공간상의 점으로 모델링하였다. 그러나 해쉬 기반 색인에서 이동체 분포가 불균등 분포인 경우 셀 오버플로우 문제가 발생한다. [7]에서는 현재 및 미래 위치 질의를 처리하기 위하여 이동체의 이동 방향을 이용한 PMR-Quadtree 기반 색인을 제시하였다. 이 색인은 보고 위치에서 미래 위치까지의 궤적을 선분으로 모델링하고 PMR-Quadtree에 저장한다. 그러나 이 기법은 색인을 일정한 시간 간격마다 재생성해야 하는 문제가 있으며 특히, 색인의 특성상 중복 저장의 문제를 가진다.

데이터 분할 기법 중 하나인 R*-Tree를 기반한 연구로 TPR-Tree[1]가 있다. 이동체는 위치 보고 후에도 계속 이동하고 있기 때문에 시간, 방향, 속도등의 매개 변수들을 이용하여 이동체의 위치를 예측할 수 있다. TPR-tree는 질의 시간을 시간 함수의 입력 매개 변수로 사용하는 색인 기법을 제시하였다. 이 기법에서 각 이동체의 현재 및 미래 위치는 노드의 모든 엔트리들 즉, 이동체들을 포함하는 시간 함수 기반의 경계 사각형인 BR을 이용하여 색인된다. 이를 위하여 BR의 각 축은 이동체들의 이동 방향 별로 가장 빠른 속도로 이동하는 이동체의 속도 값을 이용하여 확장한다. 그러나 이 기법은 시간이 지남에 따라 BR이 점차적으로 확장되기 때문에 노드간의 중첩 영역이 증가하는 문제가 있다.

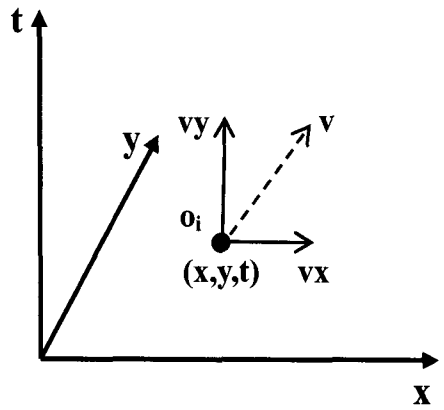
TPR*-tree[2]는 TPR-tree를 기반으로 개선된 삽입 알고리즘을 제안하고 있다. 제안된 알고리즘은 TPR-tree에서 노드간의 중첩 영역에 이동체가 삽입될 때 하위 노드의 BR이 커지는 문제를 해결하기 위하여 노드의 하위 노드를 고려한다. 즉, 삽입할 단말노드 탐색 시 탐색된 노드의 하위 노드 크기가 작

은 쪽으로 이동체를 삽입함으로써 전체적인 BR의 크기를 감소시킨다. 그리고 이동체 삭제 시 삭제가 발생한 노드의 모든 조상 노드들에 해당하는 BR들까지 최적화하는 기법을 제시하였다. 그러나 이 기법도 시간에 비례하여 BR이 계속 확장되기 때문에 노드간의 중첩 영역이 증가되어 영역 질의의 성능이 떨어지는 문제가 있다.

3. 문제정의

3.1 경계 사각형(BR)

이동체의 현재 및 미래 위치를 계산하기 위하여 이동체의 이동 속성을 시간 함수로써 정의할 수 있다. 즉, 이동체의 가장 최근에 보고된 위치 정보와 속도와 시간의 항으로 구성된 시간 함수를 이용하면 일정 시점의 미래 위치를 계산할 수 있다. 따라서 이동체 oi는 <그림 1>과 같이 가장 최근에 보고된 시점의 위치와 속도로 표현될 수 있다.

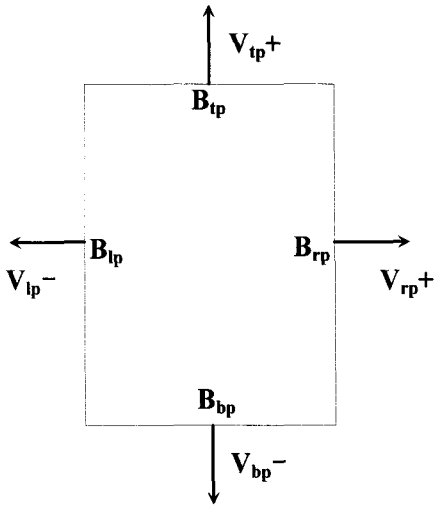


<그림 1> 이동체 oi의 표현 예

이 때 tj 시간 때 oi의 위치는 다음과 같이 정의할 수 있다.

정의 1: ti 시간 때 oi의 위치가 (xi, yi) 이고 속도가 (vxi, vyi)이라 할 때, tj 시간 때 oi의 위치 (xj, yj)는 다음과 같다: xj = xi + vxi * (tj - ti), yj = yi + vyi * (tj - ti) 단, ti ≤ tj

색인 트리의 각 노드는 포함된 모든 엔트리 즉, 이동체의 위치를 색인하기 위하여 경계사각형(BR)을 사용한다. BR은 노드의 모든 이동체를 포함하는 경계사각형으로 질의를 처리할 때 색인 생성 후 질의 시점까지 이동한 이동체를 포함한다. 따라서 그림 2와 같이 속도와 시간의 항으로 정의된 시간함수를 이용하여 각 축의 방향으로 확장되며, 확장된 각 축은 BR에 포함된 이동체 중 가장 빠른 속도로 이동하는 이동체의 속도 값을 이용하여 계산한다.



<그림 2> 경계사각형 (BR)

특정 노드의 모든 이동체를 포함하는 BR는 다음과 같이 정의된다.

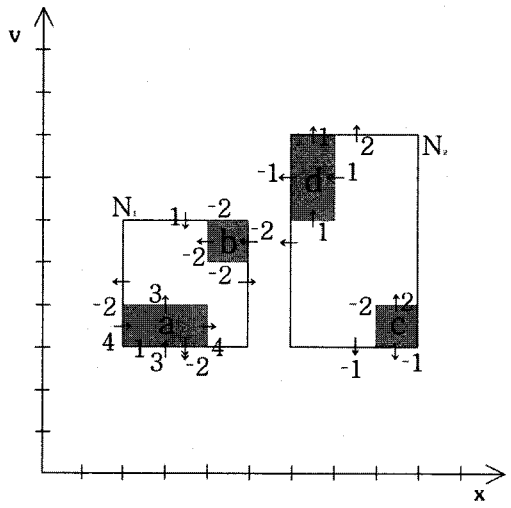
정의 2: BR = <(Blp, Brp, Btp, Bbp, Vlp- ,Vrp+ ,Vtp- ,Vbp+ , tc)>

- □ Blp, Brp, Btp, Bbp: BR 생성 또는 최적화 수행 후 축의 위치
- □ Vlp-, Vrp+, Vtp-, Vbp+: 각 축의 방향에 따른 이동체 또는 하위 노드들의 최대 속도

3.2 시간의 흐름에 따른 BR간의 중첩

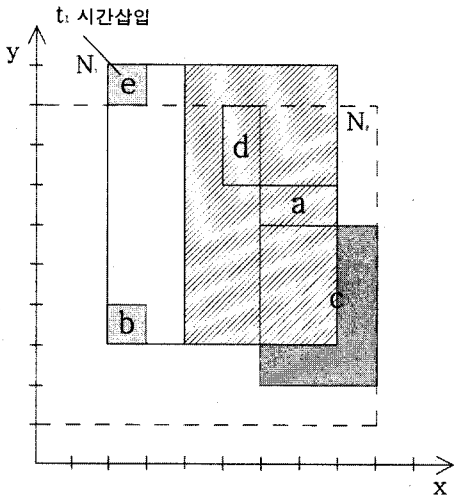
TPR-tree는 현재 및 미래 위치 질의를 처리할 때 질의 시간(tq)을 입력 매개변수로 사용하여 각 노드의 BR을 계산한다. BR을 계산할 때 정의 2에 따라 각 축의 방향으로 이동체의 최대 속도를 나타낸 Vlp- ,Vrp+ ,Vtp- 그리고 Vbp+의 값을 이용한다. 따라서 BR 생성 또는 최적화 수행 후 시간이 흐름에 따라 BR의 영역은 점차적으로 커지게 된다. 시간의

흐름에 따른 BR의 증가는 노드 간의 중첩 영역의 크기를 커지게 만든다. 따라서 영역 질의의 성능은 시간이 흐를수록 점차적으로 저하되는 문제가 발생한다.



<그림 3> t_0 시간의 BR의 예

<그림 3>은 t_0시간의 BR의 예를 보여준다. N_1 노드는 단말 노드인 a와 b를 포함하고 N_2는 c와 d를 포함한다. 이 때 BR의 각 축은 포함하고 있는 이동체 또는 하위 노드들의 최대 속도 값으로 확장된다.



<그림 4> BR의 확장에 따른 중첩의 증가

<그림 4>는 t_1 시간에 e를 삽입할 때 각 BR의 확장 상태를 보여준다. t_1 시간에 e를 삽입하기 위하여 각 노드의 BR들은 확장된다. <그림 4>에서 N_1, N_2 노드의 BR이 확장되는 것을 볼 수 있다. e이 N_1에 삽입되기 위하여 N_1의 BR이 확장된 후에 N_1 노드의 BR은 최적화된다. 점선으로 표시된 부분이 최적화되기 전의 BR 형태이다. 그러나 <그림 4>에서 보듯이 e의 삽입으로 인하여 N_1 노드가 최적화되더라도 이웃하는 노드인 N_2와의 중첩 영역이 발생하고 이전 시간인 t_0 시간과 비교할 때 중첩 영역의 크기가 확장되는 것을 알 수 있다. 따라서 시간이 흐름에 따라 N_1과 N_2간의 중첩 영역이 확대되기 때문에 영역 질의의 효율이 저하된다.

4. 강제합병기법

이 장에서는 BR의 확장으로 인하여 발생하는 노드 간의 중첩 영역을 줄이기 위하여 이동체 삽입 시 두 노드를 합병하는 강제 합병 기법에 대하여 기술한다. 강제 합병 기법은 중첩 영역이 일정 비율 이상인 두 단말 노드를 강제로 합병함으로써 노드 간의 중첩을 제거한다.

BR은 시간이 흐름에 따라 점차 확장되므로 이웃한 노드의 BR간에 중첩이 발생한다. 노드 Ni의 BR을 Ni.BR이라 할 때 노드간의 중첩 비율은 다음과 같다.

정리 1: 두 노드 Ni와 Nj가 있고 area(Ni.BR)이 Ni.BR의 면적을 나타낼 때 Ni와 Nj간 중첩 비율, OverlapRate(Ni, Nj),은 다음과 같이 계산된다.

□ OverlapRate(Ni, Nj) = area(Ni.BR ∩ Nj.BR) / min(area(Ni.BR), area(Nj.BR))

심화중복(Large Overlap Rage, LOR)는 노드간의 중첩이 심화되어 영역 질의의 성능이 저하되기 시작하는 두 단말 노드간의 중첩 비율을 나타낸다. LOR을 30%, 40%, 50%... 등으로 실험한 결과에 따르면 40%에서 영역 질의의 성능이 가장 우수하였다.

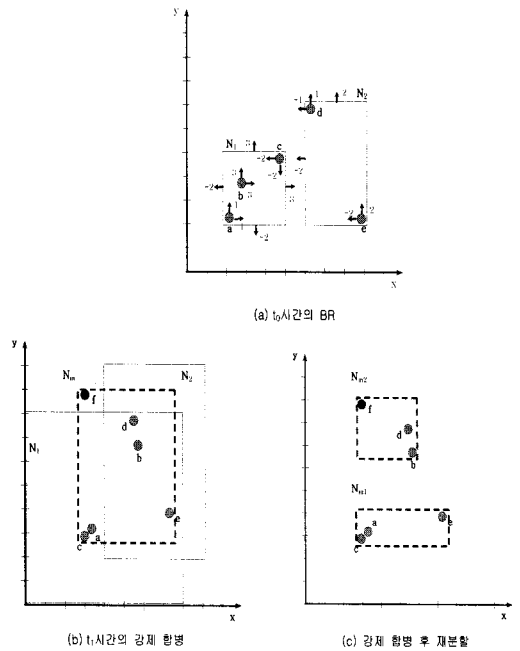
이 논문에서는 강제 합병의 시기를 두 노드간의 중첩 비율이 LOR보다 클 때 해당 노드들을 강제 합병한다.

정리 2: Ni를 삽입이 발생한 단말 노드라 하고 Nj를 Ni.BR와 중첩 비율이 최대인 단말 노드라 할 때 OverlapRate(Ni, Nj) > LOR이면 Ni와 Nj를 강제 합

병한다.

만약 강제 합병을 수행한 후 합병된 노드에서 오버플로우가 발생하면 기존의 재분할 알고리즘에 따라 재분할을 수행한다.

<그림 5>의 (a)는 t_0 시간의 BR 상태를 보여준다. <그림 5>의 (b)는 t_1 시간에 이동체 f가 삽입될 때 확장된 각 BR을 보여주고 있다. t_1 시간에 f를 삽입하기 위하여 N_1과 N_2 노드의 BR은 확장된다. f가 N_1에 삽입된다고 가정하자. f는 N_1에 삽입되고 N1의 BR은 최적화된다. 만약 최적화된 N_1과 이웃노드 N_2간의 중첩 비율이 LOR 이상이면 N_1과 N_2를 강제 합병하여 Nm을 생성한다. <그림 5>(b)에서 점선으로 된 BR은 강제합병으로 생성된 Nm의 BR을 보여준다.



<그림 5> 강제 합병과 재분할의 예

각 노드의 팬아웃(fan-out)이 4라고 가정하자. 강제 합병된 Nm의 이동체 수가 6이기 때문에 Nm에서 오버플로우가 발생한다. 따라서 Nm에 대하여 재분할을 수행해야 한다. <그림 5>의 (c)는 오버플로우로 인하여 Nm을 재분할 결과를 보여준다. <그림 5>의 (c)에서 보듯이 재분할을 수행한 후 분할된 노드의 BR이 이동체의 현 분포에 따라 최적화되었음을 알 수 있다.

강제 합병은 LOR값 이상으로 중첩 비율이 큰 두 노드를 강제로 합병함으로써 두 노드간에 존재하던 중첩 영역을 제거한다. 따라서 영역 질의의 성능을 향상시킬 수 있다. 또한 합병된 노드에서 오버플로우가 발생할 때 재분할을 수행하기 때문에 이동체의 현 분포에 따라 트리 구조를 동적으로 최적화시키고 이동체의 삽입 순서에 따른 종속성 문제를 해결한다.

```
Let Ni be tightened the best leaf node to insert new vector of point object
Algorithm Check_Large_Overlap(Ni)
Find all the entries N1, N2, ··· Nn that were overlapped with Ni.BR
Choose Nj ← maximum(OverlapRate(Ni, Nj), for 1 <= i <= n IF(Nj exists and is largely overlapped over LOR)
{   Merge two node(Ni, Nj) to Nm, Nm is tightened
    IF Nm is overflow {
        Split Nm to Nm1, Nm2;
        Ni ← Nm1; Nj ← Nm2;
        AdjustTree(Nj);}
    ELSE {DeleteNode(Nj); Ni ← Nm;}
```

알고리즘 1 강제 합병 알고리즘

알고리즘 1은 이동체의 삽입 시 강제 합병을 수행하기 위한 알고리즘을 보여준다. 삽입이 발생한 노드와 중첩되는 이웃 노드들을 검색한 후에 이웃 노드간의 중첩 비율을 계산한다. 이 때 중첩 비율이 최대인 이웃 노드와 비율이 LOR값 이상이면 삽입이 발생한 노드와 해당 노드를 합병한다. 만약 합병된 노드에서 오버플로우가 발생하면 해당 노드를 재분할한다.

5. 강제 재삽입 기법

이 장에서는 노드 간의 중첩 영역을 줄이기 위하여 한 이동체의 삭제 시 다른 이동체도 동반 삭제하여 재삽입하는 강제 재삽입 기법에 대하여 기술한다. 강제 재삽입 기법은 삭제가 발생한 노드는 물론 그 노드와 중첩되는 이웃 노드들의 이동체들도 강제 재삽입 함으로써 중첩 영역을 감소시킨다.

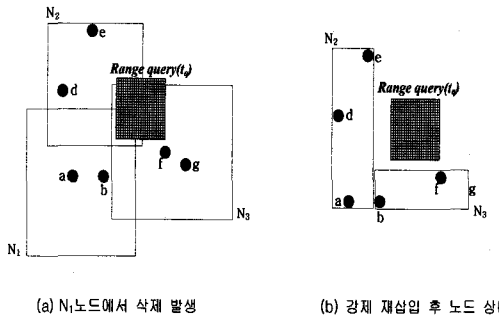
강제 재삽입 기법은 크게 두 가지로 나눌 수 있다. 첫 번째는 삭제가 발생한 노드의 이동체만을 재삽입

하는 삭제 노드 강제 재삽입 기법이다. 두 번째는 삭제가 발생한 노드와 중복되는 모든 노드의 이동체도 재삽입하는 중첩 노드 강제 재삽입 기법이다..

5.1 삭제 노드 강제 재삽입 기법

새로운 이동체를 삽입하기 위해서는 이전 위치의 삭제가 필요하기 때문에 이동체의 삽입 연산은 삭제 연산을 동반하여 발생시킨다. 삭제 노드 강제 재삽입 기법은 특정 단말 노드에서 이동체가 삭제될 때 삭제가 발생한 노드에 포함되는 모든 이동체를 삭제한 후에 재삽입한다.

정리 3: 노드 Ni에 포함되는 이동체의 집합을 Si라 하자. oi ∈ Si인 oi를 Si에서 삭제할 때 oj ∈ Si - { oi }인 모든 oj도 삭제하여 재삽입한다.



(a) N_1노드에서 삭제 발생 (b) 강제 재삽입 후 노드 상태

<그림 6> 삭제 노드 강제 재삽입의 예

<그림 6>의 (a)에서 이동체 c의 새로운 위치 보고로 인하여 c의 삭제가 발생한다고 가정하자. 삭제 발생 노드의 강제 재삽입 기법에 의해 노드 N_1의 이동체들은 강제 재삽입되기 때문에 N_1의 a와 b는 삭제 후 재삽입된다. 그림 6의 (b)는 이전 N_1 노드의 모든 이동체들이 재삽입되고 난 후의 상태를 보여 주고 있다. 노드 N_1의 이동체들의 재삽입은 (b)에서 보는 것과 같이 주변 노드의 BR을 이동체의 분포에 따라 최적화시킨다. 그러나 삭제 노드 강제 재삽입을 수행하지 않을 경우 N_1과 N_1의 부모 노드만 최적화된다. 따라서 만약 영역질의 tq가 주어 졌을 때 그림 6의 (a)에서 보듯이 재삽입 연산을 수행하지 않으면 N_2와 N_3에 접근해야만 한다. 그러나 삭제 노드 강제 재삽입 기법을 사용한 그림 6의 (b)에서는 N_2와 N_3 노드에 접근할 필요가 없다.

```
re-insertedi ← flase for all levels 1<= i <= h-1 (h is the tree height)
Initialize an empty re-insertion list Lreinsert
Ni is the entry to be deleted

Algorithm Forced_Reinsert(Ni)
N ← FindDeleteNode(Ni, S);  // to find the leaf node N to delete Ni,
                            // Stack S ← retrieves node all the level
DeleteData(N, Ni);
condenseTree(S, Lreinsert); //Adjust and reinsertion
While(!Lreinsert.empty())
    Insert(Lreinsert);

Algorithm condenseTree(S, Lreinsert)
IF(isRoot())
        return;
P ← S;                      // each retrieves the level
deleteNode(N);
Nj ← find the entry in the parent, that points to this node
IF(P is underflow || this→getLevel() == 0 ) // all entries of the N
        P→DeleteData(Nj);
        Lreinsert ← True;   // all entries of the P
                               Else
        AdjustTree(P);      // the entry in 'p' to contain the
                            // new bounding box of this node
```

알고리즘 2 삭제노드 강제 재삽입 알고리즘

알고리즘 2는 삭제 노드 강제 재삽입 기법을 위한 알고리즘을 보여준다. 삭제시킬 이동체가 포함된 노드를 검색한 후 삭제시킬 이동체를 먼저 삭제한다. 그 후에 해당 노드의 모든 이동체들을 재삽입하기 위해 재삽입될 이동체들을 스택에 넣는다. 그리고 스택의 모든 이동체를 재삽입한다. 이때 condenseTree는 상위 노드의 언더플로우 발생시 재삽입될 이동체들을 검사한다.

5.2 중첩 노드 강제 재삽입 기법

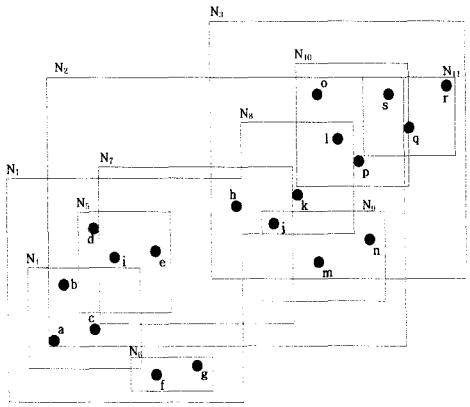
두 번째 기법은 삭제가 발생한 노드와 중첩되는 모든 노드의 이동체들을 강제 재삽입하는 중첩 노드

강제 재삽입 기법이다.

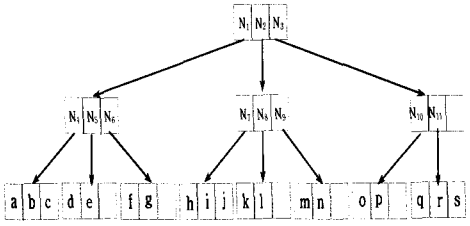
정리 4: Ni.BR ^ Nk.BR = Ø인 형제 노드 Nk가 있고 Ni와 Nk에 포함되는 이동체의 집합을 각각 Si와 Sk라 하자. oi ∈ Si인 oi를 Si에서 삭제할 때 oj ∈ Si - { oi }인 모든 oj와 ok ∈ Sk인 모든 ok를 삭제하여 재삽입한다.

노드의 BR은 시간이 지남에 따라 점차 확장되기 때문에 최악의 경우에 거의 모든 노드가 삭제가 발생한 노드와 중첩이 있을 수 있다. 따라서 이러한 경우 모든 노드의 이동체를 재삽입해야 하는 문제가 발생할 수 있다. 이 논문에서는 이 문제를 해결하기 위하여 삭제가 발생한 노드의 형제 노드만을 중첩 검사를 위한 대상으로 한다.

다음 그림은 중첩 노드 강제 재삽입의 예를 보여준다. <그림 7>에서 j 이동체의 새로운 위치 보고로 인해 j의 이전 위치 데이터가 삭제된다. <그림 8>은 <그림 7>의 트리 구조를 보여준다. 중첩 노드 강제 재삽입 기법에 따라 삭제가 발생한 노드 N_7의 이동체인 h와 i와 함께 중첩되는 형제 노드 N_8와 N_9의 이동체인 k, l, m 그리고 n도 삭제 후 재 삽입된다.

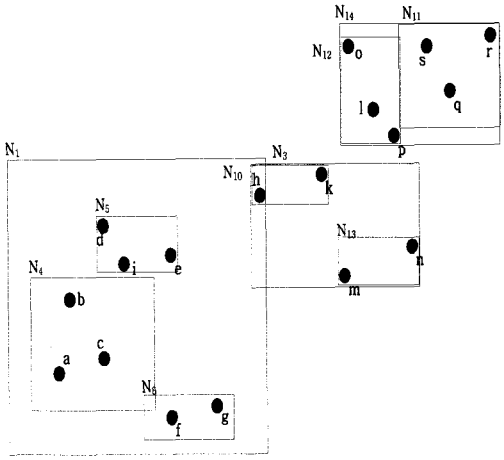


<그림 7> N_7 노드내의 j 삭제 발생

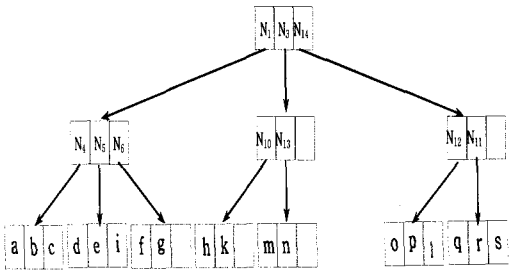


<그림 8> <그림 7>의 트리 구조

<그림 9>는 삭제가 발생한 N_7과 중첩되는 형제 노드인 N_8, N_9에 포함되는 이동체를 재삽입 한 후의 상태를 보여준다. <그림 10>은 <그림 9>의 트리 구조를 나타낸다. <그림 7>과 <그림 9>를 비교하면 재삽입 후 노드간의 중첩 영역이 감소하였음을 알 수 있다.



<그림 9> j 삭제 발생 후 형제 노드 재삽입



<그림 10> <그림 9>의 트리 구조

알고리즘 3은 삭제가 발생한 노드와 중첩되는 노드의 모든 이동체들을 강제 재삽입하는 알고리즘이다. 삭제 발생 노드와 중첩되는 형제 노드들을 검사하여 중첩되는 노드의 이동체들을 강제 재삽입한다. 삭제 후 언더플로우 발생시 삭제 발생 노드의 공간을 삭제한다. condenseTree는 상위 노드의 언더플로우로 발생시 재삽입 될 이동체들을 검사한다.

```
re-insertedi ← flase for all levels 1<= i <= h-1 (h is the
tree height)
Initialize an empty re-insertion list LreinsertNi is the entry
to be deleted
Algorithm Forced_Reinsert(Ni)
N ← FindDeleteNode(Ni, S); // to find the leaf node N to
                           delete Ni,
                           // Stack S ← retrieves node
                           all the level
DeleteData(N, Ni);
Lreinsert);              //Adjust and reinsertion
While(!Lreinsert.econdenseTree(S, mpty())
      Insert(Lreinsert);

Algorithm condenseTree(S, Lreinsert)
IF(isRoot())
           return;
P ←  S;   // point of parent node at each retrieves the
           level
N ← this;  // point of current node
Lreinsert ← all entris that be deleted the leaf node
Nj ← find the entry in the parent, that points to this node
IF(LS+1 > P.level) {
           For exist each entry of P {
           IF(P != N) {  //LS is overlap check restriction
              O ← Check_Overlap(the node of  be deleted
              entry, sibling node of P);
              IF(O == True) Lreinsert ←  all entries at
              sibling node of P
           }
           }
}
IF(P is underflow // this→getLevel == 0)
           P→DeleteData(Nj);
           Lreinsert ← all entries of the P;
Else
           AdjustTree(P); // the entry in 'p' to contain
                          the
                         //new bounding box of this node
P→condeseTree(S, Lreinsert);
```

알고리즘 3 중첩 노드 강제 재삽입 알고리즘

6. 성능 평가

6.1 성능 평가 항목

이 논문에서 제안한 3가지 재구성 기법의 성능을 검사하기 위하여 기존의 TPR 트리에서 제안한 기법을 사용하지 않은 상태와 비교한다. 재구성 기법들은 메인 메모리 환경에서 현재 및 미래질의 처리 시 성능 개선을 목적으로 한 기법이다. 이를 위하여 기존의 Disk 기반의 TPR-tree를 메인 메모리 환경에서 운용되도록 변환하였다. 그리고 재구성 기법들을 적용하지 않은 경우와 적용한 경우의 성능을 실험을 통하여 비교하였다. 대상 질의는 영역 질의이고 성능 비교 항목은 영역 질의 처리 시간과 영역질의 처리 시 비교하는 엔트리의 수이다.

성능 평가 실험을 위한 평가 인자는 다음의 표 1과 같다.

<표 1> 성능 평가 인자

성능평가 인자	설 명
MO	이동체의 수
RQ	영역질의 크기
LOR	심한 중복 비율(Large Overlap Rate)
ST	검색 시간(초)
SCE	검색 수행 동안의 엔트리 비교 횟수
Dereinsert	삭제노드 강제 재삽입
Ovreinsert	중복노드 강제 재삽입

영역 질의의 크기에 따른 성능 평가는 이동체에 대한 질의 시 색인의 특성과 성능을 평가하는 요소가 된다. 이 논문에서는 전체 영역에 대한 영역 질의의 크기(RQ)를 1%, 5%, 10%로 하였다. 그리고 생성된 색인에 대하여 타임 스탬프를 0에서 시작해서 질의 간격마다 질의를 1000회 발생시켜서 검색 시간(ST)과 검색 시 비교하는 엔트리의 수(SCE)를 측정한다.

색인의 성능 평가에서 사용될 데이터 집합은 GSTD 알고리즘을 이용하여 다음과 같이 생성하였다.

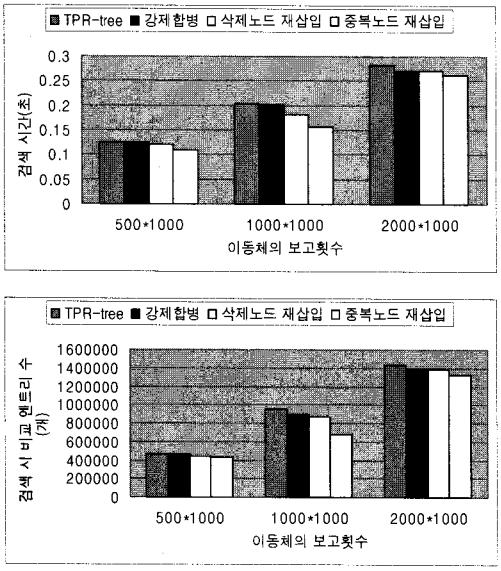
- □ 이동체 수 - 500개, 1000개, 2000개
  - □ 각 이동체의 개수에 따라 1000번 보고
- □ 초기 분포 - 가우시안 분포(평균: 0.5, 분산: 0.1)
- □ 위치 보고 간격 - 가우시안 분포(최소: 0, 최대: 0.001)
- □ 중심 이동 간격: 최소값(-0.02, -0.02), 최대값(0.02, 0.02)

이 실험을 위하여 TPR-tree는 C++ 언어로 구현하

였고, WinXP Pro 운영체제를 기반으로 RDRAM 1GB, CPU Intel 3.06Ghz를 장착한 서버를 이용하였다.

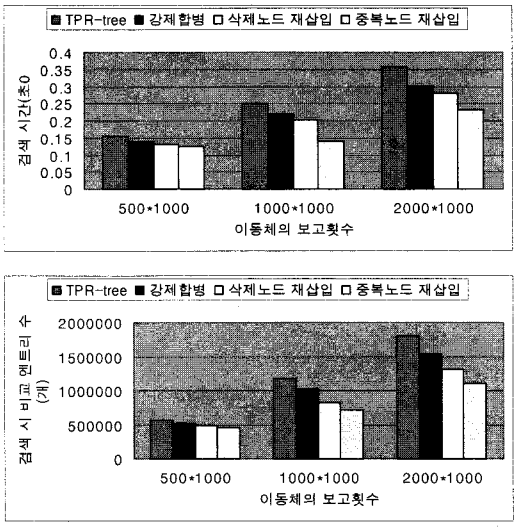
6.2 실험 결과

<그림 11>은 질의 영역을 전체 영역의 1%로 설정하였을 때 실험 결과를 보여준다. <그림 11>에서 보듯이 강제 합병 기법은 이동체의 수가 작은 경우에 기존의 기법에 비하여 성능 개선 효과가 매우 작다. 이는 대상 이동체 분포를 가우시안 분포로 사용하고 검색하기 위한 BR의 수가 작기 때문에 이동체들이 밀집하여 있는 경우 두 노드간의 강제 합병 후에 재분할을 수행하더라도 BR의 재구성 효과가 작기 때문이다. 그러나 강제 재삽입 기법들은 강제 합병 기법에 비하여 상대적으로 성능 개선 효과가 크다.



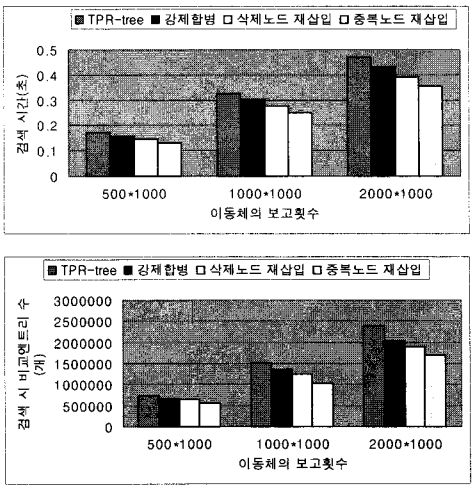
<그림 11 1% 영역에 대한 영역 질의 시 실험 결과>

<그림 12>는 전체 영역의 5%영역에 대하여 질의할 경우의 실험 결과를 보여준다. <그림 12>에서 보듯이 강제 합병 기법도 성능 개선 효과가 나타남을 알 수 있다. 이는 질의 영역이 커짐에 따라 검색해야 할 BR의 개수도 증가하므로 밀집된 이동체간에도 강제 합병으로 인하여 BR의 구성이 최적화되기 때문이다.



<그림 12> 5% 영역에 대한 영역 질의 시 실험 결과

<그림 13>은 10%영역에 대한 영역질의를 수행할 때의 실험 결과를 보여준다. <그림 13>에서 보듯이 모든 재구성 기법들이 성능 개선 효과를 나타내며 특히 검색 대상인 이동체의 수가 많을 경우 개선 효과가 뚜렷하다. 이유는 BR내에 포함되는 이동체의 수가 증가할수록 BR간의 중첩이 심해지며 재구성 기법들을 사용할 경우 BR이 최적화되어 BR간의 중첩이 최소화되기 때문이다. 따라서 검색 시 비교 엔트리의 수도 줄어든다.



<그림 13> 10% 영역에 대한 영역 질의 시 실험 결과

<그림 11>, <그림 12> 그리고 <그림 13>의 실험 결과에서 보듯이 약 7%에서 15%의 성능 개선 효과가 있다. 왜냐하면 가우시안 분포의 경우에 이동체간의 밀집도가 높은 편이기 때문에 BR간의 중첩 영역도 큰 경향이 있다. 따라서 재구성 기법을 사용하지 않을 경우 BR간의 중첩이 더욱 커진다. 그러나 재구성 기법들을 사용하면 BR간의 중첩 영역을 줄이기 때문에 영역 질의 시 비교 엔트리의 수도 줄일 수 있으며 검색 시간도 감소된다.

실험 결과에서 보면 강제 합병 기법보다 재삽입 기법들이 우수한 성능을 보이고 있다. 이는 강제 합병 기법은 두 노드간에만 강제 합병 후 재분할하기 때문에 두 개의 BR만이 최적화된다. 그러나 재삽입 기법들은 두 개 이상의 노드에 대하여 BR간의 중첩을 줄여주기 때문에 많은 수의 노드들이 최적화되기 때문이다.

7. 결론 및 향후 연구

이동체의 현재 및 미래 위치 질의를 처리하기 위한 색인 중 하나인 TPR-tree에서 이동체를 색인하는 각 노드의 BR은 시간이 흐름에 따라 확장된다. 따라서 노드간의 중첩 영역이 확대되고 영역 질의 성능이 저하되는 문제가 발생한다. 이 논문에서는 중첩 영역을 줄이기 위하여 이동체의 삽입과 삭제시 노드를 재설정하는 재구성 기법들인 강제 합병 기법, 삭제 노드 강제 재삽입 기법 그리고 중첩 노드 강제 재삽입 기법을 제시하였다. 그리고 성능 평가 실험을 통하여 제시한 기법들을 사용할 때 영역 질의에 대하여 약 7%에서 15%의 성능 개선 효과가 있음을 보였다.

이 논문에서 제시한 재구성 기법들은 삽입 또는 삭제 연산이 발생할 때 이동체의 분포에 따라 연산이 발생한 노드와 주변 노드의 BR을 이동체의 분포에 따라 최적화시킨다. 따라서 TPR-tree의 BR들을 점진적으로 재구성하기 때문에 TPR-tree의 구조적 문제점이었던 시간의 흐름에 따른 질의 성능 저하 문제를 개선하는 장점을 가진다.

그러나 강제 재삽입 기법이 영역 질의 처리에 대하여 가장 월등한 성능 개선 효과를 보이지만 그에 비례하여 삽입 비용이 증가하는 단점이 있다. 따라서 향후 연구로는 삽입 비용의 증가 문제를 해결하기 위한 비용 감소 방법에 대한 연구가 필요하다.

참고문헌

[1] S. Saltenis, C. S. Jensen, S.T. Leutenegger, and M. A. Lopez, "Indexing the positions of continuously moving objects," Proc. Of the ACM SIGMOD Int'l Conf. on Management of Data, pp. 331-342, 2000.

[2] Y. Tao, D. Papadias, and J. Sun, "The tpr*-tree: an optimized spatiotemporal access method for predictive queries," Proceedings of 29th International Conference on Very Large Data Bases, Berlin, Germany, September 9-12, 2003.

[3] J. Green, D. Betti, and J. Davison, Mobile Location Services: Market Strategies, Ovum Ltd, 2000.

[4] N. Beckmann and H. P. Kriegel "The R*-tree: An efficient and robust access method for points and rectangles," Proc. Of the ACM SIGMOD Int'l Conf. on Management of Data, pp. 332-331, 1990.

[5] R. H. Guting, M. H. Bohlen, M. Erwig, C. S. Jensen, N. A. Lorentzos, M. Schneider, and M. Vazirgiannis, "A foundation for representing and querying moving objects," ACM Transactions on Database Systems, vol. 25, no. 1, pp. 1-42, 2000.

[6] Z. Song and N. Roussopoulos, "Hashing moving object," Int'l. Conf. on Mobile Data Management, pp. 161-172, 2001.

[7] J. Tayeb, O. Ulusoy, and O. Wolfson. "A quadtree based dynamic attribute indexing method," The Computer Journal, vol. 41, no. 3, pp. 185-200, 1998.

[8] P. K. Agarwal, L. Arge, and J. Erickson, "Indexing moving points," Proc. of the ACM Symposium on Principles of Database Systems, pp. 175-186, 2000.

[9] S. Berchtold, D. A. Keim, and H. P. Kriegel, "The X-tree : An index structure for

high-dimensional data," Proc. Of Int'l Conf. on Very Large Data Bases, pp. 28-39, 1996.

[10] L. Forlizzi, R. H. Guting, E. Nardelli, and M. Schneider, "A data model and data structures for moving objects databases," Proc. Of the ACM SIGMOD Int'l Conf. on Management of Data, pp. 319-330, 2000.

[11] D. Pfoser, C. S. Jensen, and Y. Theodoridis, "Novel approaches in query processing for moving objects," Proc. Of Int'l Conf. on Very Large Data Bases, pp. 395-406, 2000.

김동현

1995년 부산대학교 컴퓨터공학과 졸업 (학사)

1997년 부산대학교 대학원 컴퓨터공학과(석사)

2003년 부산대학교 대학원 컴퓨터공학과(박사)

2004년 ~ 현재 동서대학교 소프트웨어전문대학원 전임강사

관심분야: 이동체 색인, 모바일 GIS, 지리정보시스템, 이동체 데이타베이스, 공간 데이터베이스.

홍봉희

1982년 서울대학교 컴퓨터공학과 졸업 (학사)

1984년 서울대학교 대학원 컴퓨터공학과(석사)

1988년 서울대학교 대학원 컴퓨터공학과(박사)

1987년 4월 ~ 현재 부산대학교 공과대학 컴퓨터공학과 교수

관심분야: 데이터베이스, 공간 데이타베이스, 이동체 데이타베이스, 이동체 색인, 트랜잭션