이동체의 색인을 위한 시간 기반 R-트리의 설계 및 구현

(Design and implementation of a time-based R-tree for indexing moving objects)

전 봉 기 † 홍 봉 희 ††
(Bong Gi Jun) (Bong Hee Hong)

요 약 위치 기반 서비스는 이동체의 위치에 따라 종속적인 결과를 얻는 위치 기반 질의를 필요로 하게 하였다. 연속적으로 이동하는 이동체의 위치를 추적하는 것은 위치 기반 서비스에서는 중요한 응용 중의 하나이다. 효과적인 질의 처리를 위해 이동체 데이터베이스는 연속적으로 위치를 변경하는 이동체의 이동을 관리하는 3차원 색인을 필요로 한다. 2차원 R-tree의 확장으로 시간 도메인을 포함하는 3DR-tree와 같은 이동체 색인은 노드 간의 높은 중복과 사장 공간으로 인하여 낮은 공간 활용도와 검색 성능이 저하되는 문제점이 있다. 이 논문에서 제시하는 TR-tree는 R-tree 기반의 색인으로서, 시간 도메인의 성장을 고려하여 시간 축 분할 시 비균등 분할 정책을 사용하여 공간 활용도를 높였다. 노드간의 중복과 사장 공간을 최소화 하기 위하여 강제 합병 정책을 사용하여 중복이 심한 노드를 강제 합병 시킨다. 또한 오버플로우 노드의 분할 시에 노드간의 중복을 심하게 하는 원인이 되는 긴 선분을 절단 정책을 사용하여 2개의 선분으로 절단하여 분할 노드 간의 중복을 제거한다. 실험 평가 결과에서 TR-tree는 3DR-tree와 TB-tree 보다 성능이 우수하였으며, 특히 R-tree와 R*-tree보다 색인의 크기가 작다.

키워드 : 이동체, 이동체 데이터베이스, 지리정보시스템, 공간 색인

Abstract Location-Based Services(LBS) give rise to location-based queries of which results depend on the locations of moving objects. One of important applications of LBS is to examine tracks of continuously moving objects. Moving objects databases need to provide 3-dimensional indexing for efficiently processing range queries on the movement of continuously changing positions. An extension of the 2-dimensional R-tree to include time dimension shows low space utilization and poor search performance, because of high overlap of index nodes and their dead space. To solve these problems, we propose a new R-tree based indexing technique, namely TR-tree. To increase storage utilization, we assign more entries to the past node by using the unbalanced splitting policy. If two nodes are highly overlapped, these nodes are forcibly merged. It is the forced merging policy that reduces the dead space and the overlap of nodes. Since big line segments can also affect the overlap of index nodes to be increased, big line segments should be clipped by the clipping policy when splitting overfull nodes. The TR-tree outperforms the 3DR-tree and TB-tree in all experiments. Particularly, the storage utilization of the TR-tree is higher than the R-tree and R*-tree.

Key words : Moving Objects, Moving Object Database, GIS, Spatial Indexing

1. 서 론

위치 기반 서비스(Location-Based Service)는 무선 통신 서비스의 핵심 응용으로 대두되고 있다. 이동체의 연속적인 이동을 고려하는 위치 기반 질의를 처리하기 위해서는 이동체의 위치 정보를 관리하는 이동체 데이터베이스 기술 개발이 선행되어야 한다. 특히 이동체의 위치를 기반으로 하는 응용인 물류 및 수송관리, 교통정보 서비스, 디지털 전장과 같은 응용 서비스의 개발을

· 본 연구는 부산대학교 연구(보조)비 (4년과제)에 의한 연구임.
† 학생회원 : 부산대학교 컴퓨터공학과
bgjun@pusan.ac.kr
†† 종신회원 : 부산대학교 컴퓨터공학과 교수
bhhong@pusan.ac.kr
논문접수 : 2002년 10월 29일
심사완료 : 2003년 2월 13일

위해서는 시간의 흐름에 따라 증가하는 이동체의 위치 정보를 효과적으로 관리하고, 빠른 검색을 제공하는 이동체 색인의 개발이 필요하다.

이동체는 시간에 따라 연속적으로 위치 데이터가 갱신되는 객체를 말한다. 이동체는 이동 점과 이동 영역으로 나눌 수 있으며, 이동 점은 위치 정보만이 갱신되는 반면 이동 영역은 시간에 따라 위치와 모양이 변하는 객체이다. 예를 들면 이동 점은 자동차, 동물, 비행기 등이 있고, 이동 영역은 폭풍의 영향권, 암세포의 상태 등이다. 이 논문에서는 이동 점에 관한 연구로서 이동체를 3 차원의 점으로 표현 한다.

이동체의 모델링은 이동체의 위치가 시간에 따라 연속적으로 변하기 때문에 시공간 모델링에서 이를 연속적 모델과 이산적 모델로 구분한다. 연속적 모델은 이동체를 무한개의 점 집합으로 표현한다. 이를 3차원의 연속적 곡선으로 표현할 수 있으며 벡터를 기반으로 한다. 반면 이산적 모델은 이동체를 유한개의 점 집합으로 표현하는 것으로 3차원 도메인에서 다중선(polyline)으로 표현한다. 다중선은 선분들의 집합이다. 이 논문은 이산적 모델로 이동체를 모델링하고, 이동체의 궤적(trajectory)을 선분들의 집합으로 정의한다.

이동 데이터베이스의 질의 종류는 크게 영역 질의, 타임스탬프(timestamp) 질의, 궤적 질의, 복합 질의로 나누어진다. 영역 질의는 주어진 시간 간격 동안에 공간 윈도우(spatial window)에 속하는 이동체들을 검색하는 3차원 질의이다. 타임스탬프 질의는 특정 시간에 주어진 공간 윈도우에 속하는 이동체들을 검색하는 질의이다. 궤적 질의는 특정한 이동체의 궤적을 검색하는 질의이다. "복합 질의는 특정 시간에 주어진 영역을 지나간 객체의 궤적을 검색하라"와 같이 시공간 도메인의 영역 질의와 궤적 질의가 복합된 질의이다. 이전 연구에서 모든 이동체 질의에서 뛰어난 성능을 보인 색인은 없었다. 이 논문은 가장 일반적인 질의인 영역 질의의 성능 향상을 목적으로 한다.

기존의 색인 구조 연구에서는 다음과 같은 이동체의 이동 특징을 고려하고 있지 않는 문제가 있다. 첫째, 이동체들의 궤적은 중복이 심하고, 사장 공간(dead space)가 크다는 문제가 있다. 색인에서 중복은 다중 경로를 유발하여 검색 성능을 저하시킨다. 둘째, 선형적으로 증가하는 시간 도메인에서 데이터의 삽입 순서를 고려하고 있지 않다. 이동체의 위치 정보는 현재 시간을 기준으로 보고하기 때문에, 저장된 데이터 보다 큰 시간 값을 가진다. 시간 축으로 분할된 과거 노드는 추가 삽입이 없다. 셋째, 이동체의 분포는 시간에 따라 동적으로 변한다. 이동체 위치 데이터의 삽입 패턴은 시간에 따라 변하기 때문에, 동적인 색인 구조가 필요하다.

이 논문에서 제안하는 TR-tree는 이동체의 이동 특징을 고려하여 다음과 같은 정책을 이용하여 질의 성능을 개선 하였다. 첫째, 이동체들의 색인은 중복을 최소화하게 하는 구조를 가져야 한다. 노드간의 중복을 조사하여 중복이 심한 노드들을 심한 중복 노드로 규정하고, 강제 합병 후 재분할(resplit)을 수행하여 중복을 최소화 하였다. 둘째, 노드간의 심한 중복을 유발하는 선분을 긴 선분(Big Line Segments)으로 규정하여 절단하여 중복을 최소화 하였다. 셋째, 시간 축 분할을 비균등 분할하여 추가 삽입이 없는 과거 노드의 공간 활용도를 높였다.

이 논문의 구성은 다음과 같다. 2장에서는 관련 연구를 소개하고, 3장에서는 이동체 색인들의 문제를 정의하였다. 4장에서는 강제 합병 정책을 소개하고, 노드간의 심한 중복을 규정한다. 5장에서는 분할 축 결정 방법과 시간 축 분할 방법을 제안한다. 6장에서는 긴 선분을 정의하고 긴 선분의 절단 방법을 제안한다. 7장에서는 실험을 수행하여 TR-tree의 성능을 비교하였다. 마지막으로 8장에서 결론 및 향후 연구를 기술한다.

2. 관련 연구

기존의 공간 색인은 Quad-tree[1], KDB-tree[2]와 같은 공간-분할 방법과 R-tree 계열[3, 4, 5]의 데이터-분할 방법으로 나뉜다. 이동체의 분포는 시간에 따라 변하기 때문에, 공간-분할 방법은 적합하지 않다. 또한 시간 도메인은 동적으로 증가하기 때문에, 공간-분할 방법에서 시간 도메인을 관리하기가 어렵다는 문제가 있다. 대부분의 이전 연구에서도 이와 같은 문제로 인하여 R-tree의 변형(variants)을 사용한다.

R-tree[3]는 데이터-분할 방법의 대표적인 공간 색인으로서 공간 객체를 최소 경계 사각형(Minimum Bounding Rectangle)을 사용하여 표현하는 저장하는 균형 트리 구조의 색인이다. 공간 객체를 색인에 삽입 시에 사용되는 R-tree의 ChooseLeaf 알고리즘은 최소 영역 증가 정책(Least Area Enlargement Policy)를 사용하여 삽입 노드를 선택한다. R*-tree는 R-tree가 삽입 시에 영역만을 고려하는 단점을 보완하기 위하여 중복(overlap)과 가장자리(margin)을 추가적으로 고려한 색인 구조를 제시하였다. R*-tree는 단말 노드를 선택 시에 최소 중복 증가 정책(Least Overlap Enlargement Policy)를 사용하며, 오버플로우가 발생한 노드를 분할 시에 분할 축을 선택하기 위하여 가장자리 값과 중복

값을 고려한 새로운 분할 기법을 제시한다. 또한 재삽입 정책을 사용하여 R-tree의 공간 활용도가 낮은 문제점을 보완하였다.

이동체의 색인에 관련된 연구는 크게 3가지 부분으로 나누어 볼 수가 있다. 첫째, 현재와 미래 위치를 검색하기 위한 색인에 관련된 연구들로서 R-tree, 해쉉, Quad-tree를 기반으로 하는 색인이 있다. 둘째, 이동체의 궤적과 같은 과거 이력 색인에 대한 연구로서 STR-tree, TB-tree가 있다. 셋째, 시공간 색인에 대한 연구로서 3DR-tree, HR-tree, MV3R-tree가 있다.

이동체의 현재 위치를 저장하는 색인에 관한 연구에는 갱신 횟수에 관한 연구[6]와 미래 위치를 예측하는 연구들[7, 8]로 나누어진다. Tayeb[8]은 PMR-quadtree를 이용하여 미래 궤적을 선분으로 저장한다. 하지만 PMR-quadtree는 시간 간격 마다 색인을 재생성해야 하는 문제가 있다. 해쉉 기법을 이용한 관련 연구[6]은 해쉬 기반 색인에서 이동체가 셀을 벗어나지 않으면 색인 갱신이 없다는 아이디어를 사용하여 해쉉 함수를 사용하여 물리적인 갱신의 수를 줄였다. TPR-tree[7]는 이동체들의 속도와 방향을 R*-Tree[4]에 적용하였다. 시간에 대한 함수를 파라미터로 사용하여, 이동체의 현재와 가까운 미래의 위치를 좀 더 빠르게 검색할 수 있게 하는 방법을 제시하였다.

STR-tree[9]는 이동체들의 궤적을 선분의 집합으로 나타내고, R-tree의 최소경계박스를 시간으로 확장하여 표현함으로써 이동체의 궤적 정보를 저장한다. STR-tree는 보존 파라미터(preservation parameter)를 사용하여 같은 객체의 궤적을 근접한 페이지에 저장하도록 유도 하였다. TB-tree[10]는 이동체의 궤적을 3 차원 최소경계박스에 저장하고, 이 최소경계박스의 연결로 이동체의 모든 궤적을 표현한다. 단말 노드에서 이동체의 궤적은 양방향 연결 리스트(linked list)로 구성되어 있으므로 궤적에 대한 질의를 처리할 때는 뛰어난 성능을 보인다. 그러나 TB-Tree는 궤적에 대한 질의 성능은 좋은 반면 시간 간격에 대한 질의나 공간 간격에 대한 질의에는 좋지 않은 성능을 보인다.

이동체는 시간에 따라 위치가 변하기 때문에 시공간 객체로 볼 수 있다. 시공간 색인 중에서 이동체를 실험한 색인에는 3DR-tree [11], HR-tree[12], MV3R-tree[13]가 있다. 3DR-tree는 기존의 다차원 색인인 R-tree의 3차원 버전으로서, 현재 또는 UC(until changed)에 대해 처리할 수 없는 문제가 있기 때문에 객체의 모든 이동을 시간에 대하여 닫힘(closed)되어 있다고 가정한다. 즉 닫힌-선(closed- line)들만이 삽입 될 수 있다. HR-tree는 연속적인 상태를 표현하기 위하여 R-tree에 중복 개념을 추가한 것이다. HR-tree는 타임스탬프 마다 다른 색인 인스턴스로 구성된다. HR-tree는 이동체들의 이동이 빈번하지 않은 경우에는 효율적이지만, 이동이 많이 발생하는 경우 단말 노드 및 비단말 노드를 새로 생성해야 하고, 특히 영역 질의의 성능이 저하되는 문제를 가진다. MV3R-tree는 MVB-tree(multi-version B-tree)의 개념을 이용한 것으로 MVR-tree와 단말 노드들로 구성되는 보조적인 3DR-tree를 결합하여 타임스탬프 질의와 영역 질의를 효율적으로 처리할 수 있다. HR-tree는 타임스탬프 질의에 효율적이고 3DR-tree는 영역 질의에 효율적인 반면에 MV3R-tree는 타임스탬프 질의나 영역 질의에 모두 효율적으로 다룰 수 있다. 또한 HR-tree에 비해 저장공간을 적게 필요로 한다. 하지만, MV3R-tree는 버전 분할에서 발생하는 중복되는 데이터 때문에, MV3R-tree의 크기는 3DR-tree보다 대략 1.5배 정도 크다.

노드 간의 중복을 줄여서 색인의 성능을 향상 시키는 기존의 공간 색인에 연구로서는 R*-tree[4], R+-tree [5], X-tree[14]가 있다. R*-tree는 개체를 삽입하기 위한 단말 노드 선택 시, 단말 노드간의 최소 중복 증가 정책(Least Overlap Enlargement Policy)을 사용한다. 또한 오버플로우 노드를 분할 시에 중복이 최소화되는 그룹으로 분할을 수행한다. 하지만 R*-tree의 분할 알고리즘을 사용하더라도 심한 중복의 단말 노드를 분할 시에는 분할된 노드들이 중복이 심하다는 문제가 있다. 이는 R*-tree가 단말 노드 자체의 중복을 줄이는 정책을 고려하지는 않았기 때문이다. 이 논문에서는 절단 방법을 사용하여 단말 노드 간의 중복을 최소화하였다. R+-tree는 절단-기반 방법(clipping-based scheme)을 사용하는 색인으로서 노드간의 중복을 허용하지 않는다. 하지만 Greene[15]은 R+-tree는 CPU 비용이 증가하고 절단으로 인한 색인의 크기가 증가하는 문제점으로 인하여 R*- tree 보다 검색 성능이 나쁘다는 성능 평가 결과를 제시하였다. X-tree는 심한 중복시에 발생하는 R*-tree의 문제점과 절단 정책을 사용하는 R+-tree의 문제점을 고려하여 공간 색인에서의 노드간의 중복을 최소화하는 색인 방법을 제시한다. X-tree에서 노드 분할 시에 노드간의 중복 비율을 고려하여 허용치(MAX_OVERLAP)을 초과하는 분할을 허용하지 않음으로서 노드 간의 중복을 회피하는 정책을 사용한다. 즉, 중복이 심한 노드의 분할 시에 분할을 수행하지 않고 수퍼 노드(SuperNode)를 사용하여 노드의 최대 엔트리 수를 증가시킨다. 하지만 X-tree는 수퍼 노드로

인하여 노드간의 중복을 줄였지만, 4차원 이하의 색인에서 R-tree와 같은 검색 성능을 보인다. 이것은 수퍼 노드가 페이지 접근 횟수가 일반 노드 보다 많기 때문이다.

이 논문은 이동체의 과거 궤적에 대한 효과적인 질의 처리를 위한 색인 구조에 관한 연구이다. 앞에서 언급한 R-tree의 변형을 이용한 색인 연구에서 시간의 흐름에 따라 증가하는 이동체의 궤적으로 인한 이동체 색인의 문제점인 노드 간의 중복을 줄이는 방법에 관한 연구는 없었다. 이 논문에서는 이동체의 과거 궤적에 관한 영역 질의의 성능을 향상시키기 위하여, 노드의 사장 공간을 최소화 하고, 노드간의 중복을 줄이는 방법을 연구하는 것을 목적으로 하였다.

3. 문제 정의

이 장에서는 3DR-tree를 이동체 색인으로 이용할 경우에 발생하는 2가지 문제점을 소개한다. 첫째, 최소경계박스(MBB)들 간의 심한 중복이 발생한다. 둘째, 공간 활용도가 낮다. 최소경계박스 들 간의 심한 중복은 이동체의 이동 패턴과 이동체의 분포에 종속적이다. 삽입되는 이동체의 시간 값이 저장된 객체들의 시간 값 보다 높다. 낮은 공간 활용도는 시간 축으로 균등 분할을 수행할 경우에 분할된 과거 노드에 더 이상 삽입이 되지 않기 때문이다.

정의 1. 최소 경계 박스(Minimum Bounding Box)

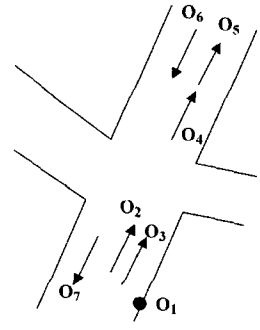
노드의 최소 경계 박스(Minimum Bounding Box)는 노드의 모든 엔터리들을 포함하는 가장 작은 크기의 엔터리의 차원과 같은 차원의 박스를 의미한다. 이 논문에서는 약자로 MBB로 표기하며, 3차원 박스이다. 이동체의 이동 궤적인 선분은 색인의 단말 노드에서 최소경계박스(MBB)로 저장되며 선분의 최소경계박스(MBB) 또는 궤적의 MBB로 표기한다. □

3.1 이동체 궤적의 중복 문제

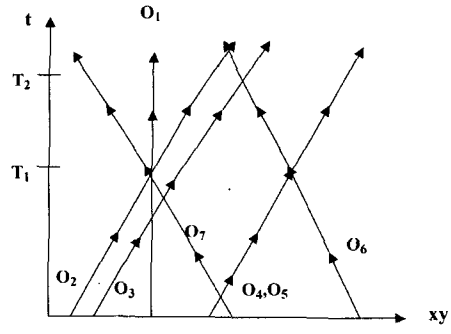
이 논문에서는 R-tree의 중복을 줄이기 위한 노드 분할 정책과 선분 절단 정책을 제안한다. 이동체의 궤적은 3차원 선분이지만, R-tree 색인에서는 단말노드에 최소경계박스(MBB) 형태로 표현된다. 이동체들의 궤적에 대한 최소경계박스 간에 중복이 심한 것은 이동체들의 분포, 이동 패턴에 매우 종속적인 것을 알 수 있다.

이 논문에서는 3-차원 공간에서의 이동 중복 문제를 단순화하기 위하여 x, y 축을 하나의 도메인으로 매핑한 2-차원 공간으로 기술한다. 그림 1(b)는 그림 1(a)의 실제 도로에서 이동하는 이동체의 궤적을 2-차원 공간으로 매핑하여 기술한 것이다. 이동체들의 분포는 도로망에 종속적이다. 이동체의 이동 패턴은 다음과 같이 나타난다.

- 정지 객체(O_1)
  공간적인 위치 변화 없이 시간 도메인으로만 긴 선분을 가진다. 이웃 차선를 이동하는 이동체와 중복 최소경계박스를 발생시킨다.
- 이웃 차선에서 이동하는 2개의 차량(O_2, O_3)
  공간적으로 근접되어 있으며 중복 최소경계박스(MBB)를 발생시킨다.
- 같은 차선을 이동하는 2개의 차량(O_4, O_5)
  보고 시간만 다르고 이동 궤적은 같다. 심하게 중복된 최소경계박스(MBB)가 일어난다.
- 반대 차선의 차량(O_6, O_7)



(a) 도로상의 이동체



(b) 시간 도메인을 고려한 이동체의 2차원 매핑

그림 1 이동체의 이동 궤적의 특징

반대 차선에서 이동하는 차량의 이동 궤적은 시간이 지남에 따라 현재 차선의 이동체 궤적과 중복된다. 예를 들면, O_6은 T_1에서 O_5와 O_4 객체와 중복 최소경계박스(MBB)가 발생하고, T_2에서 O_2와 O_3와 중복 최소경계박스(MBB)를 발생시킨다.

이동체의 이동 속도와 현재 위치를 보고하는 시간 간격에 따라 긴 선분이 만들어진다. 표 1과 같이 시간 간격이 길거나 이동 속도가 빠른 경우에는 긴 선분이 만들어진다. 긴 선분을 포함하는 노드의 최소경계박스(MBB)는 매우 커지며 다른 노드와의 중복이 증가하는 문제를 일으킨다.

표 1 이동 패턴

	빠른 속도	느린 속도
긴 보고 간격	매우 긴 선분	긴 선분
짧은 보고 간격	긴 선분	짧은 선분

3.2 최소경계박스(MBB) 확장과 중복 문제

이동체의 이동 궤적은 시공간 도메인에서 (x_1, y_1, t_1)-(x_2, y_2, t_2)의 선분으로 저장 관리된다. 이때 t_2는 트랜잭션(transaction) 시간에서 현재에 해당되며, R-tree에서 단말노드의 최소경계박스(MBB)는 시간 축으로 확장된다. 또한 이동체는 특정한 방향으로 시간에 따라 이동하기 때문에, 이동체의 이동 정보인 선분이 삽입되면 단말 노드의 최소경계박스(MBB) 확장에 따른 중복 문제가 발생하게 된다.

그림 2(a)(b)와 같이 $line_1$과 $line_2$를 삽입한다면 R-tree의 ChooseSubtree 알고리즘에 의하면 (MBB_2의 확장 영역 > MBB_1의 확장 영역)이기 때문에 최소 영역 확장 정책에 의해 MBB_1을 선택하게 된다. 하지만 삽입으로 인한 MBB_1의 확장은 $line_3$를 삽입할 때, MBB_1과 MBB_2가 심한 중복이 발생하는 문제점이 발생한다. 이러한 문제는 기존의 R-tree의 삽입 알고리즘이 시간 도메인의 확장을 고려하지 않기 때문에 발생하는 것이다.

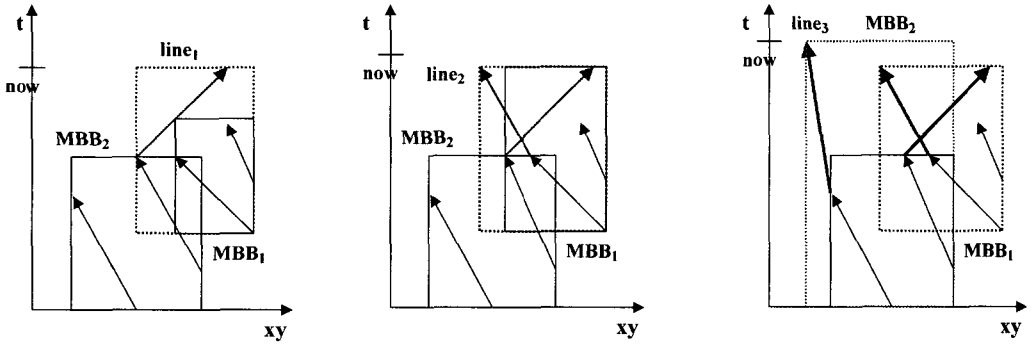
이 논문에서는 이러한 문제점을 해결하기 위해서 MBB간의 중복 비율을 정의하고, 삽입 시에 중복이 심한 MBB들을 강제 합병하는 정책을 사용하여 노드 간의 중복을 최소화 하였다.

3.3 균등 분할의 문제점

R-tree에서 노드가 포함하는 엔트리의 최대수(upper bound)가 M이면, 최소수(lower bound)는 m <= M/2이다. 오버플로우가 발생한 노드(M+1)을 분할하면, 분할된 2개의 노드는 적어도 m개의 엔트리를 가져야 한다. 그림 3에서 M = 10, m = 5라고 하면 새로운 선분의 삽입으로 오버플로우가 발생되어 분할을 수행하게 되고, 그림 3(a)처럼 균등 분할의 결과가 얻어진다. 그림 3(a)에서 MBB2는 과거 노드로서 더 이상 삽입이 없다. 그러므로 시간 축의 균등 분할은 시간 도메인을 고려할 때 공간 활용도를 저하시키는 문제가 있다. 이 논문에서는 시간 축으로의 비균등 분할을 수행하여 더 이상 증가하지 않는 과거 노드의 공간 활용도를 높이는 아이디어를 사용한다.

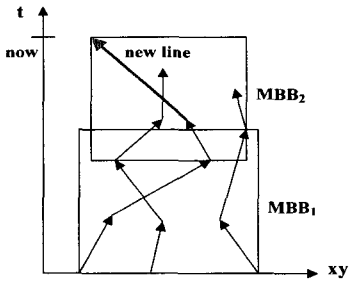
4. 심한 중복 경계박스(HOMBB)간의 강제 합병 정책

이동체의 이동 궤적은 시간에 따라 연속적으로 증가하므로 근접한 노드 간의 중복이 시간이 경과함에 따라 더 심해진다. 이동체들의 이동 궤적을 삽입하면 삽입 노드의 최소경계박스는 확장되며, 근접한 노드의 최소경계박스와 중복이 발생한다. R-tree에서는 근접한 노드 간의 중복이 증가하여 검색 비용이 증가하는 문제는 고려하지 않는다. TR-tree에서는 최소경계박스 확장에 의해 발생하는 중복을 줄이는 정책으로 강제 합병 정책을 제안한다.

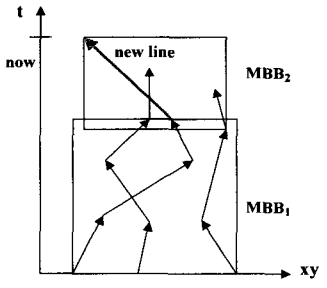


(a) $line_1$ 삽입 후 MBB_1의 확장 (b) $line_2$ 삽입 후 MBB_1의 확장 (c) $line_3$ 삽입 후 MBB 간의 중복 증가

그림 2 새로운 선분의 삽입으로 인한 MBB 확장 문제



(a) R-tree에서의 시간축 균등 분할



(b) TR-tree에서의 시간축 비균등 분할

그림 3 R-tree에서 시간 축 균등 분할의 문제

정의 2. 단말 노드 간의 중복 비율

인접한 2개의 단말 노드를 E_i, E_j라 한다. E_i와 E_j의 최소경계박스 간의 중복이 있다면, E_i, E_j의 중복 비율은 다음과 같이 정의한다.

OverlapRate(E_i, E_j) ← area(E_i. MBB∩E_j. MBB) /min(area(E_i. MBB), area(E_j. MBB)) □

정의 3. HRO(High Rate of Overlap)

2개의 단말 노드 E_i와 E_j가 중복이 있을 때, E_i, E_j의 중복 비율이 HRO 이상이면, E_i는 E_j와 심한 중복이 발생한 것이며, 강제합병 정책에 의해 합병된다. HRO은 30%, 35%, 40%, 45%, 50%으로 실험한 결과, 40%에서 검색 성능이 가장 우수하였다. □

정의 4. 심한 중복 경계박스: HOMBB(Highly Overlapped MBB)

삽입되는 선분을 LS라 하고, LS가 삽입되는 단말 노드를 Ei라 한다. LS의 삽입에 의해 Ei의 최소경계박스가 다른 노드와 심한 중복이 발생된다면 Ei의 최소경계박스는 심한중복 경계박스(HOMBB)이다.

IF OverlapRate(E_i, E_j) > HRO, E_i는 E_j와 심한 중복이 발생하였으며, E_i는 HOMBB를 가진다. □

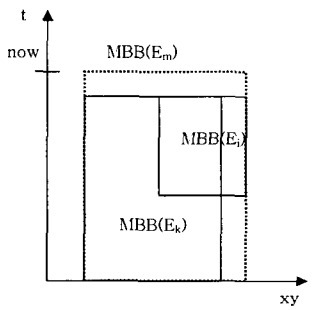
정의 5. 심한 중복 경계박스(HOMBB)의 연쇄(Cascading)

초기에 E_i, E_j와 E_k는 서로 심한 중복관계가 아니라고 가정한다. 새로운 선분 LS가 E_k에 삽입되어 E_k와 E_i가 심한 중복이 관계가 되었다. 강제합병 정책에 의해 두 노드(E_i, E_k)를 합병한 노드를 E_m이라 하면, E_m와 E_j가 심한 중복 관계일 수 있다. 이러한 조건이 만족되는 경우를 HOMBB의 연쇄라 한다. □

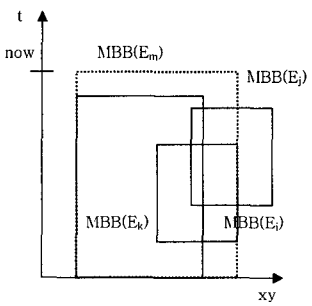
강제 합병으로 생성된 노드를 E_m이라 하였을 때, E_m의 최소경계박스는 합병 전의 두 노드의 최소경계박스보다 크다. 최소경계박스의 확장은 그림 4와 같이 새로운 HOMBB를 유도할 수 있다. HOMBB가 아닌 MBB(E_j)가 E_m에 의해 HOMBB가 되는 경우를 HOMBB의 연쇄라 한다. HOMBB의 연쇄가 발생하는 경우는 단말 노드간에 중복이 심할 때, 공간 축으로 발생할 수 있으나(그림 4(b)), 시간 도메인으로 분할 된 최소경계박스들 중에 하나가 HOMBB인 경우에도 HOMBB의 연쇄가 일어난다.(그림 4(c))

강제 합병 정책은 다음과 같은 규칙을 가진다.

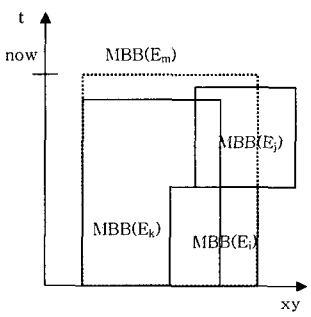
- 2개의 단말 노드가 심한 중복 관계이면 강제 합병한다.
- 심한 중복 관계인 노드가 2개 이상이면, 중복 비율이 최대인 노드와 합병한다.



(a) 단순한 강제 합병



(b) 공간 축으로 HOMbb의 연쇄



(c) 시간 축으로 HOMbb의 연쇄

그림 4 HOMBB의 연쇄

- 심한 중복 관계인 2개의 노드를 강제 합병하였을 때, 오버플로우가 발생하면 재분할한다.

- HOMBB의 연쇄인 경우에, 재합병을 수행한다.

강제 합병 정책은 단말 노드간의 중복을 줄이기 때문에 영역 질의의 성능을 향상시킨다. 또한 이동체의 분포에 상관없이 트리 구조를 동적으로 만드는 장점이 있다. 하지만 I/O 비용이 증가하기 때문에 삽입 비용은 증가한다. 그림 5는 선분의 삽입으로 발생할 수 있는 HOMBB를 검사하는 알고리즘이다.

Let E_k be the best leaf node to insert new line segment
Algorithm Check_High_Overlap(E_k)
Find all the entries $E_1, E_2, .., E_n$ that were overlapped with $E_k.MBB$
Choose $E_i \leftarrow maxmum(OverlapRate(E_k, E_i))$, for 1<=i<=n
IF (E_i exist and Highly overlapped over HRO)
  Merge two nodes(E_i, E_k) to E_m
  IF E_m is overflow **THEN** Split E_m to E_{m1}, E_{m2}
  ELSE Invoke *Check_High_Overlap(E_m)* to check cascading of HOMBB

그림 5 HOMBB의 검사 알고리즘

5. 분할 정책

이 장에서는 더 이상 삽입이 되지 않는 과거 노드의 공간 활용도를 높이는 새로운 분할 정책을 제안한다. 좋은 분할 결과를 얻기 위해 노드의 최적의 크기를 정의할 필요가 있다. 하지만, 시간 단위는 공간 거리와 직접적으로 비교할 수 없다. 이 논문에서는 이동체의 수와 이동체의 보고 횟수를 이용한 새로운 기준을 제안한다.

5.1 시간 도메인과 공간 도메인의 크기 비교

이동체 데이터베이스는 3차원 공간의 시간 도메인과 공간 도메인에 정의된 값의 집합으로 구성된다. 단말 노드의 오버플로우 시에 노드를 분할할 때 R-tree에서는 정사각형 최소경계박스(MBB)가 가장 최적이다고 한다. 하지만 시공간 도메인에서 시간 도메인(t축)의 시간 단위와 공간 도메인(x, y축)의 공간 거리를 직접적으로 비교 하기가 어렵다. 예를 들면, 10 km와 1시간 중 어느 것이 크다고 할 수가 없다

R*-tree에서는 오버플로우가 발생한 노드를 분할할 때, 가장자리 값(margin value)을 이용하여 분할 축을 선정한다. 하지만 시간 도메인의 시간 단위 값이 공간 도메인의 공간 거리 보다 클 경우에는 시간 축 분할이 자주 발생하고, 반대의 경우에는 공간 분할이 자주 발생한다. 시간 도메인은 시간에 따라 연속적으로 성장하므로 시간 도메인과 공간 도메인을 고정 비율로 비교할 수도 없다. 이 논문에서는 이러한 문제를 해결하기 위해, 시간 도메인은 보고 빈도로 측정하고 공간 도메인은 이동체의 수로 측정하는 방법을 제안한다.

표 2 시공간 도메인의 크기 비교 모델을 위한 기호와 의미

기 호	의 미
Nls	모든 선분의 수
Nln	색인에서의 단말 노드의 수
Cnode	노드의 용량(capacity)
Nmo	모든 이동체의 수
Tsampling	이동체의 샘플링(sampling) 횟수
X, Y, T	도메인의 길이
x_i, y_i, t_i	단말 노드 LN_i 의 크기
Nmo_i	단말 노드 LN_i 의 이동체 수
Ts_i	단말 노드 LN_i 의 이동체 샘플링 횟수

공간 축의 이동체 수와 시간 축에서의 이동체 샘플링 횟수의 관계를 분석하기 위해, 다음과 같은 수식을 유도한다. 다음의 가정은 분할 축을 선택하기 위한 조건을 찾기 위함이다.

가정 1) 데이터 집합은 균등 분포(Uniform Distribution)이며, 단말 노드는 같은 크기이고 중복이 없다고 가정한다.

$Nln = Nls/Cnode$ (수식 1)

$Nln = (X*Y*T)/(x_i*y_i*t_i)$ (수식 2)

$Nls/Cnode=(X*Y*T)/(x_i*y_i*t_i)$ (수식1과 수식2에 의해)

$Nls*(x_i*y_i*t_i)=(X*Y*T)*Cnode$ (수식 3)

가정 2) 이동체의 수는 변화가 없고, 같은 시간 간격마다 보고 한다고 가정한다.

$Nls=Nmo*Tsampling$ (수식 4)

$Nmo*Tsampling*(x_i*y_i*t_i)=(X*Y*T)*Cnode$ (수식 3, 4에 의해)

$Nmo*(x_i*y_i)*Ts_i=(X*Y)*Cnode$, ($Tsampling/Ts_i=T/t_i$이므로)

$Nmo*Ts_i=(X*Y)/(x_i*y_i)*Cnode$ (수식 5)

가정 3) 시간에 관계없이 모든 이동체들이 모든 단말 노드에 균등하게 분포되었다고 가정하면 단말 노드(LN_i)의 이동체 수(Nmo_i)는 다음 수식이 성립한다.

$Nmo_i=Nmo*(x_i*y_i)/(X*Y)$ (수식 6)

$Nmo*Ts_i=Nmo/Nmo_i*Cnode$, (수식 5, 6에 의해)

$Ts_i*Nmo_i=Cnode$ (수식 7)

수식 7은 앞의 가정을 모두 만족하였을 때, 가장 최적인 R-tree의 단말 노드 조건이다.

정의 6. 최적의 최소경계박스를 위한 Nmo_i와 Ts_i의 관계

x_i, y_i, t_i는 단말 노드(LN_i)의 각 도메인 크기이다.

LS_j를 LN_i의 선분이라 하고, $LS_j.x$, $LS_j.y$, $LS_j.t$는 선분 LS를 x, y, t축으로 투영(projection)한 것이다. 이동체들이 주어진 단말 노드의 공간에서 균등분포(Uniform Distribution)와 중복이 없다고 가정하므로 $|LS_1.x| = |LS_2.x| = \cdots = |LS_{Cnode}.x|$이다. 이동체의 샘플링 횟수가 Ts_i이고, 이동체 수가 Nmo_i개 이므로, x 축으로 투영하면 각 도메인의 크기는 다음과 같다.

$x_i=Nmo_i*|LS_j.x|$, $y_i=Nmo_i*|LS_j.y|$, $t_i=Ts_i*|LS_j.t|$ (수식 8)

$x_i=y_i=t_i$인 조건을 최적이라 하다면 수식 7과 수식 8에 의해 다음 식을 유도할 수 있다.

$Nmo_i*|LS_j.x|=Nmo_i*|LS_j.y|=Cnode/Nmo_i*|LS_j.t|$ (수식 9)

$|LS_j.x|=|LS_j.y|=|LS_j.t|$라고 가정하면 수식 9는 다음 식을 유도할 수 있다.

$Nmo_i=Cnode^{1/2}$ (수식 10)

수식 10은 최적의 최소경계박스가 생성하기 위한 조건이다. 선분 LS_j의 x, y, t축의 측정 단위는 거리(cm), 시간(sec)으로 다르지만 이동 거리와 시간 단위가 같다고 가정하면 시간 도메인은 샘플링 횟수, 공간 도메인은 이동체 수로 모델링한다. □

5.2 분할 축의 선정

분할 축을 선택할 때, 오버플로우 발생한 노드 LN_i의 이동체의 수(Nmoi)와 이동체의 샘플링 횟수(Ts_i)를 기준으로 한 분할 축 선정 방법을 제안한다. 다음은 분할 축을 선정하는 조건이다.

정의 7. 분할 축을 선정하는 조건

IF $Ts_i >= Nmo_i$ then t 축으로 분할 //조건(1)
ELSE IF $Nmo_i >= 2 * Cnode^{1/2}$
THEN x, y축으로 분할 //조건(2)
ELSE IF $\sqrt{2}*Cnode^{1/2} <= Nmoi < 2 * Cnode^{1/2}$ and $1/2*Cnode^{1/2}<Ts_i<=1/\sqrt{2}* Cnode^{1/2}$
THEN x, y축으로 분할 //조건(3)
ELSE t축으로 분할 □

최소경계박스가 정사각형일 때 최적이라고 가정하는 것은 t축의 Ts_i과 x, y축의 Nmo_i의 값이 같을 때 최적이라고 보는 것이다. 검색 성능을 최대화하는 분할은 분할된 노드가 $Nmo_i = Ts_i$인 최적의 최소경계박스가 되도록 하는 것이다. $Ts_i >= Nmoi$ (조건 1)인 조건에서, t 축으로 분할하여 Ts_i의 값을 줄여야 한다. 오버플로우가 발생한 노드를 E_i라고 하고 분할된 노드를 E_{i1}, E_{i2}라고 한다. 시간 축 분할에서 E_{i1}를 과거 노드, E_{i2}를 새 노드라고 하면 $E_{i1}.t < E_{i2}.t$이다.

조건 2)에서와 같이 $Nmo_i >= 2*Cnode^{1/2}$이면, $Ts_i<1/2*Cnode^{1/2}$이다. x, y축 분할은 균등 분할되었다면, 이동체의 수는 분할 후 이등분 되므로 Nmo_{i1}, $Nmo_{i2} > \fallingdotseq Cnode^{1/2}$ 근사값을 가진다. 공간 분할은 시간 도메인의 변화가 없으므로 Ts_{i1}, $Ts_{i2}<1/2*Cnode^{1/2}$ 이다.

조건 3)에서 조건1과 조건2를 만족하지 않는 조건은 $Ts_i<Nmo_i$와 $Nmo_i<2*Cnode^{1/2}$이다. $Ts_i=Cnode/Nmo_i$ 이므로 다음과 같다.

$Cnode^{1/2}<Nmo_i<2*Cnode^{1/2}$와 $1/2*Cnode^{1/2}<Ts_i<Cnode^{1/2}$ (수식 11)

x, y 축 분할을 수행할 때, Ts_i의 조건이 변화가 없다고 가정하면, 공간 분할된 $Nmo_i \fallingdotseq 1/2*Nmo_i$이 된다. 분할된 결과가 최적조건($Ts_i = Nmo_i$)이 되기 위해서 공간 분할 결과가 $Ts_{i1}>Nmo_{i1}$인 경우에는 공간 분할을 하지 않고 시간 분할을 한다. 공간 분할을 위한 조건은 $Ts_{i1}<=Nmo_{i1}$이므로 $Ts_i<=1/2*Nmo_i$이다. $Nmo_i*Ts_i=Cnode$를 치환하면 $Cnode/Nmo_i<=1/2*Nmo_i$이므로, $\sqrt{2}*Cnode^{1/2}<= Nmo_i$이다. 수식 11의 조건을 만족하는 공간 분할의 조건은 다음과 같다.

$\sqrt{2}*Cnode^{1/2} <= Nmon<2*Cnode^{1/2}$ (수식 12)

5.3 시간 축 분할 정책

이동체의 위치 보고는 현재의 위치를 보고하는 것이다. 이것은 시간 공간의 확장을 의미하며, 결과적으로 노드의 오버플로우를 가져오며 이를 해결하기 위해 분할 정책이 필요하다.

정의 8. UC 선분(UC line)

UC(until changed) 변수는 이동체가 가장 최근에 보고한 위치를 의미한다. 이동체가 보고한 위치는 (x, y, t)의 3차원 점으로 표현된다. 새로운 이동 궤적을 저장하기 위하여, 현재 보고된 위치인 UC(x_2, y_2, t_2)와 가장 최근에 보고된 위치(x_1, y_1, t_1)를 선분으로 저장한다. UC를 포함한 선분을 이동체 O_i의 UC 선분(UC line)이라 한다. □

정의 9. UC 최소경계박스(UC minimum bounding box)

이동체 O_i의 UC 선분을 $UCLine_i$이라 할 때, $UCLine_i$를 하나 이상 포함하는 노드의 최소경계박스를 UC 최소경계박스이라 한다. UC 최소경계박스가 아닌 최소경계박스를 고정 최소경계박스(fixed MBB)라 하며 $UCLine_i$를 포함하지 않는다. □

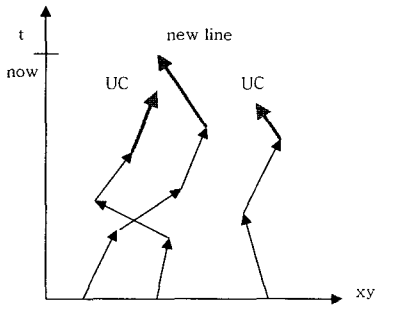
시간 축 분할은 UC 선분을 포함하는 UC 최소경계박스와 UC 선분을 포함하지 않는 고정 최소경계박스로 비균등 분할한다(그림 6). 새로운 선분은 항상 UC 선분이다. 새로운 선분과 노드의 UC 선분을 포함하는 최소

경계박스가 UC 최소경계박스가 된다. 고정 최소경계박스는 대부분 추가적인 삽입이 없는 최소경계박스이므로 공간 활용도 증가를 위해 (Cnode - UC 선분의 수)를 저장한다. 하지만 고정 최소경계박스는 이웃 노드의 선분으로 인해 추가적으로 삽입될 수 있다. 새로운 노드인 UC 최소경계박스는 UC 선분만을 포함하고 있으며, 추가적인 삽입이 가능하다.

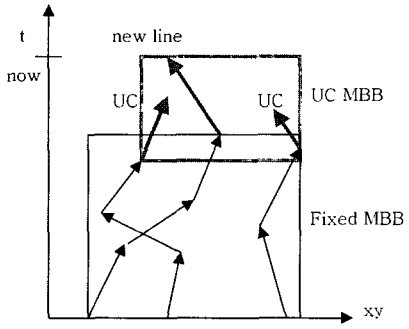
오버플로우가 발생한 노드를 시간 축으로 UC 최소경계박스와 고정 최소경계박스로 분할하였다면, 고정 최소경계박스는 Cnode/2보다 커야 한다. 이때, Cnode/2는 시간 축 분할에서 고정 최소경계박스의 최소 엔트리 수이다. 만약 고정 최소경계박스가 Cnode/2보다 작으면, 언더플로우이므로 시간 분할을 수행하지 않고 공간 분할을 수행한다.

5.4 삽입 알고리즘

TR-tree는 3가지 새로운 삽입 방법을 제안한다. 삽입으로 인하여 확장되는 노드 간의 중복을 줄이기 위하여 4장에서 강제합병 정책을 소개하였고, 시간 축 분할 시에 공간 활용도를 증가시키는 시간 축 비균등 분할 방법을 5장에서 제안하였다. 또한 5.2절에서는 새로운 분할 축 선정 방법을 소개하였다. 6장에서는 오버플로우 노드의 분할 시 분할되는 노드간의 중복을 최소화하고, 노드의 사장 공간을 줄이는 긴 선분 절단 방법을 제안한다. 이 절에서는 새로운 선분의 삽입 시에 이 3가지 새로운 삽입 방법의 연결성을 보이기 위해 그림 7의 삽입 알고리즘을 소개한다.



(a) 새로운 선분의 삽입으로 오버플로우 발생



(b) 시간 축의 비균등 분할

그림 6 시간 축의 비균등 분할

```
Algorithm  Insert(LineSegment LSi)
Choose Ek to be the best leaf node for inserting LSi
IF (Ek has room for inserting LSi)  {
  Ek.AddData(LSi);
  Ek.Check_High_Overlap();
  AdjustTree(Ek);
} ELSE  {          // Ek is overflow
  SplitAxis = ChooseSplitAxis(Ek, LSi);
  IF (SplitAxis == TimeAxis)  {
    Enew = SplitByT_Axis(Ek, LSi);
    Ek.Check_High_Overlap();
  } ELSE  {
    Enew = SplitNode_SpatialAxis(Ek, LSi, SplitAxis);
    Ek.Check_High_Overlap();  Enew.Check_High_Overlap();
  }
  AdjustTree(Ek, Enew);
}
```

그림 7 삽입 알고리즘

TR-tree는 새로운 선분을 삽입하기 위한 단말 노드를 선택하는 방법은 R*-tree의 ChooseSubtree() 알고리즘을 사용한다. 만약 선택된 단말노드가 새로운 선분을 삽입할 공간이 있는 경우에는 선택된 단말 노드에 새로운 선분을 삽입한 후 4장에서 소개한 Check_High_Overlap() 알고리즘을 사용하여 새로운 선분의 삽입으로 확장된 노드간의 중복을 검사하여 심한 중복이 발생한 경우에는 합병을 수행한다.

새로운 선분의 삽입으로 단말 노드가 오버플로우가 발생하는 경우에는 분할을 수행한다. TR-tree에서는 5.2절에서와 같이 이동체 수(Nmo)와 보고 횟수(Tsample)를 이용하여 분할 축을 결정한다. 공간 축 내에서 x, y 축을 결정하는 것은 margin-values를 이용하는 R*-tree의 ChooseSplitAxis 알고리즘을 이용한다. 만약 선택된 분할 축이 시간 축인 경우에는 5.3절에서 소개한 시간 축 비균등 분할을 수행한다. 이때 새롭게 생성되는 단말 노드는 UC 최소경계박스(MBB)로서 다른 노드에 비해 상대적인 크기가 작기 때문에, 심한 중복 검사를 수행하지 않는다. 6장에서 소개할 긴 선분의 절단 방법은 분할 된 노드 간의 크기와 선분의 크기에

의해 상대적으로 결정이 되기 때문에, SplitNode_TimeAxis()와 SplitNode_SpatialAxis() 내에서 긴 선분이 결정되고, 긴 선분은 절단된다.

6. 긴 선분의 절단 정책

노드 간의 심한 중복은 긴 선분의 매우 큰 최소경계박스(MBB)가 원인이 된다고 할 수 있다. 기존의 분할 정책들은 이용하더라도 긴 선분에 의한 심한 중복 문제를 해결할 수 없다. 이 장에서는 긴 선분의 문제점을 소개하고, 긴 선분을 결정하고 절단하는 방법을 제안한다.

6.1 긴 선분의 문제점

이 논문에서 제안하는 절단 방법은 R+-tree와 같은 기존의 절단 방법과 다음과 같은 차이점이 있다. 첫째, 절단은 단말 노드에서만 수행된다. 둘째, 분할된 2개의 노드는 중복의 최대 허용치인 HRO 보다 작은 중복을 허용한다. 반면에 R+-tree는 노드 간의 중복을 허용하지 않는다. R+-tree[13]는 절단 방법을 사용하여 중복을 삭제하는 색인 구조이다. 하지만 기존 연구[15]에서 R+-tree가 CPU 비용이 많이 들고, 절단으로 인한 색인의 크기가 증가하여 R-tree보다 나쁜 결과를 보였다.

그림 8에서 오버플로우가 일어난 단말 노드의 MBB_1을 분할하였을 때, 분할된 MBB들 (MBB_1', MBB_2')이 또 다시 HOMBB가 되었다. TR-tree에서 HOMBB들은 강제 합병되며, 합병 결과가 다시 오버플로우이면 다시 분할해야 하는 무한 반복이 발생한다. 이러한 문제는 이들 MBB들 중에 긴 선분을 가지고 있기 때문이다.

정의 10. 노드 분할에서의 긴 선분

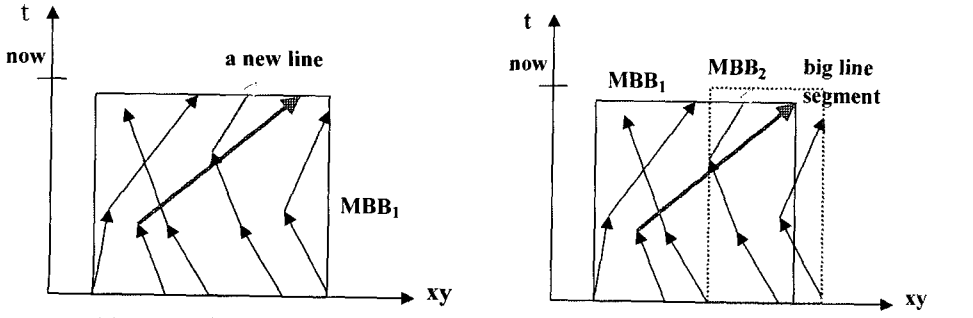
오버플로우가 발생한 노드를 E_m이라 하고, 삽입되는 선분을 LS_{new}라 하자. E_m와 LS_{new}를 분할한 결과를 E_1, E_2라 한다. 선분 LS_i는 Em의 엔터리인 선분들과 LS_{new}이다. LS_i를 E_1 또는 E_2에 삽입 시에 E_1.MBB와 E_2.MBB가 HOMBB가 되면, LS_i는 긴 선분이라 한다. 긴 선분은 2개의 선분으로 절단되어 E_1과 E_2에 각각 삽입된다. □

긴 선분은 선분의 절대 크기에 의해서 결정되는 것이 아니고, 오버플로우 노드를 분할할 때 심한 중복을 야기시키는 긴 선분과 분할되는 2개 노드의 최소경계박스(MBB) 간의 상대 크기에 의해 결정된다. 긴 선분을 분할된 노드 E_1, E_2에 할당하기 전에 $\max((E_1 \cap E_2)/E_1, (E_1 \cap E_2)/E_2) <= HRO$이라고 가정하면, $E_1 = E_1 \cup LS_i$하고 할 때, $\max((E_1 \cap E_2)/E_1, (E_1 \cap E_2)/E_2) > HRO$이면 LS_i는 긴 선분이다.

6.2 공간 도메인에서의 노드 분할과 긴 선분

긴 선분은 오버플로우가 발생한 노드를 분할할 때 결정된다. TR-tree에서 공간 축과 시간 축의 분할 정책이 다르기 때문에, 긴 선분을 결정하는 알고리즘도 다르다. 이것은 공간 축은 균등 분할 정책을 사용하고, 시간 축은 비균등 분할 정책하기 때문이다. 시간 축 분할에서의 긴 선분은 6.3장에서 소개한다.

SplitNode_SpatialAxis 알고리즘은 긴 선분을 결정하는 것을 제외하면, R*-tree의 분할 알고리즘과 유사하다. 먼저 분할 축을 결정하고, line.MBB의 low 값과 high 값을 정렬하여 visited_queue에 삽입한다. 노드를 분할할 때, 첫번째 단계에서 deque의 양쪽 끝에서 선택하여 low 점 집합(low_pointset)과 high 점 집합(high_pointset)을 만든다. 선분의 양쪽 점이 모두 같은 점 집합에 있으면, 점 집합에서 삭제하여 선 집합(lineset)에 추가한다. deque의 모든 점을 방문하면 두개의 선분 집합(low_lineset이 E_1, high_lineset이 E_2)을 만들고, E_1과 E_2에 중복되는 선분은 두개의 점 집합에 남아 있게 된다. 점 집합에 남아 있는 선분이 후보 긴 선분(candidate of big line segment)이며, 두번째 단계에서 이것을 분할된 노드 E_1, E_2에 추가 할 때 HOMBB가 되는 지를 조사한다. 긴 선분의 결정은 현재의 E_1, E_2의



(a) MBB1에서 오버플로우 발생 (b) 2개로 분할된 MBB들이 HOMbb인 경우

그림 8 노드 분할에서 긴 선분의 예

```
E_m is an overfull node. Let LS_i be line segments in E_m
Let LS_i.mbr be a space-projected MBR for each LS_i (ignoring time)
PROCEDURE  SplitNode_SpatialAxis (E_m, LS_i.SplitAxis)
visited_deque ← sort all the lower-ends and higher ends of LS_i.mbr
Initialize empty to low_pointset, high_pointset, low_lineset, high_lineset.
// 1st step: Split line segments to two groups along the chosen axis
WHILE (visited_deque is NOT empty)   {
  low_end = visited_deque.getfront();        high_end = visited_deque.getrear();
  IF ( low_end.line# exists in low_pointset.line# )
  THEN { Add low_end.line# to low_lineset;     Delete low_end from low_pointset; }
  ELSE  Add low_end to low_pointset;
  IF ( high_end.line# exists in high_pointset.line#)
  THEN { Add high_end line# to high_lineset;   Delete high_end from high_pointset; }
  ELSE  Add high_end to high_pointset;
}
//  2nd Step . Detecting and Clipping BigLineSegment
WHILE (low_pointset is NOT empty)
  IF ( distance(low_lineset.MBB, lower_point(high_pointset)) <=
        distance(high_lineset.MBB, higher_point(low_pointset)) )
  THEN { candidate_big_ls ← choose line# of lower point from high_pointset;
      IF is_Not_big_line_segment(candiate_big_ls)
      THEN  Add candidate_big_ls to low_lineset
      ELSE  Add clipping_line(candidate_big_ls) to low_lineset, high_lineset;
  ELSE { candidate_big_ls ← choose line# of higher point from low_pointset;
      IF is_NOT_big_line_segement(candiate_big_ls)
      THEN  Add candidate_big_ls to high_lineset
      ELSE  Add clipping_line(candidate_big_ls) to low_lineset, high_lineset;
  }
```

그림 9 공간 도메인의 분할 알고리즘

크기와 선분의 크기에 따라 동적으로 결정된다.

그림 8(a)의 오버플로우가 발생한 노드(MBB_1)를 SplitNode_SpatialAxis 프로시저로 분할하면, 첫번째 단계의 결과는 그림 10(a)과 같다. $line_1$, $line_2$, $line_3$는 후보 긴 선분으로 2개의 점 집합에 남아 있다. 두번째 단계에서 $line_3$, $line_1$, $line_2$의 순서로 삽입된다. $line_3$는 low_lineset에 삽입되고, line1는 high_lineset에 삽입된다. $line_2$가 low_lineset에 삽입되면, low_lineset.MBB와 high_lineset.MBB는 HOMBB가 되므로 $line_2$는 절단되어 2개의 선분으로 삽입된다.

긴 선분을 2개의 선분으로 절단 할 때, 그림 10(c)와 같이 선분이 교차하는 두개의 최소경계박스(MBB) 간의 교차점의 중점으로 나눈다. 분할되는 노드간의 중복이 있는 경우에는 중점으로 긴 선분을 절단하여, 절단된 선분의 삽입으로 인한 최소경계박스(MBB)의 확장이 발생하지 않는다. 분할되는 노드간의 중복이 없는 경우에는 중점으로 절단하여 삽입하는 것이 같은 크기의 최소경계박스(MBB) 확장이 발생한다.

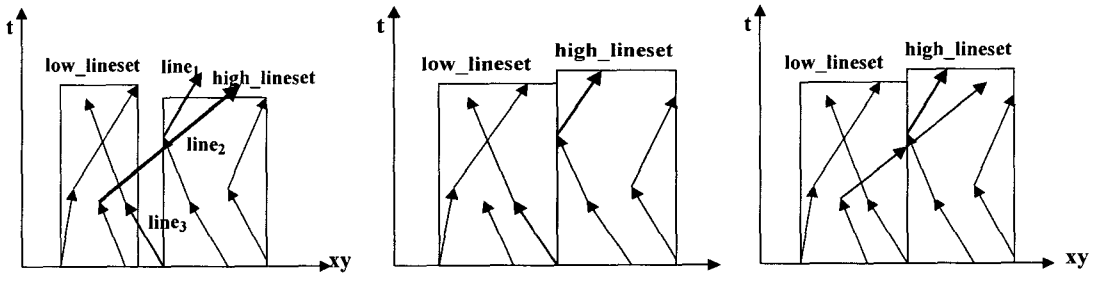
6.3 긴 선분의 시간 축 분할

시간 축의 비균등 분할된 결과는 UC 최소경계박스와 고정 최소경계박스이다. 대부분의 UC 최소경계박스는 고정 최소경계박스에 비해 작은 크기의 영역을 가지기 때문에 OverlapRate(UC_MBB)=area(UC_MBBFixed_MBB)/area(UC_MBB)에 의하면, 작은 중복에도 OverlapRate(UC_MBB) > HRO인 경우가 발생한다. 하지만 UC 최소경계박스는 시간 축으로 증가하기 때문에 시간 축 분할에서는 OverlapRate(UC_MBB) > HRO인 경우는 예외로 하여 HOMBB를 적용하지 않는다.

시간 축 분할에서 OverlapRate(Fixed_MBB)> HRO인 경우에는 HOMBB로 간주한다. 그림 11와 같이 오버플로우 노드의 최소경계박스에 비해 상대적으로 긴 시간 간격의 선분이 삽입되는 경우(그림 11(a))에 시간 축으로 분할하면 HOMBB가 된다. 긴 선분은 그림 11(b)와 같이 고정 최소경계박스의 최고 시간 값으로 분할된다. 시간 축 분할 알고리즘은 그림 12와 같다.

```
E_m is an overfull node, Let LS_new be a new line segment.
PROCEDURE  SplitByT_Axis(E_m, LS_new)
line_set = the collection of all line segments within E_m
UC_MBB ← Find UC_lines in E_m
Non_UC_line_set ← Em.line_set   UC_lines
Let LS_i be line segments in Non_UC_line_set.
Sort Non_UC_line_set by LS_i.t_high
While( Non_UC_Line_set is not empty )  {
  LS_i = Non_UC_line_set.getfront();
  IF (UC_MBB.OverlapRate(MBB(Fixed_MBB+LS_i)) < HRO)
    Fixed_MBB.AddData(LS_i);
  ELSE  Add clipping_line(LS_i) to UC_MBB, Fixed_MBB ;
}
IF (MBB(UC_MBB+ LS_new).OverlapRate(Fixed_MBB) < HRO)
  UC_MBB.AddData(LS_new);
ELSE  Add clipping_line(LS_new) to UC_MBB, Fixed_MBB ;
```

그림 12 시간 분할 알고리즘



(a) 첫 번째 단계의 실행 결과 (b) $line_3$, $line_1$의 삽입 결과 (c) $line_2$를 2개의 선분으로 절단하여 삽입

그림 10 오버플로우 노드의 분할과 긴 선분의 절단의 예

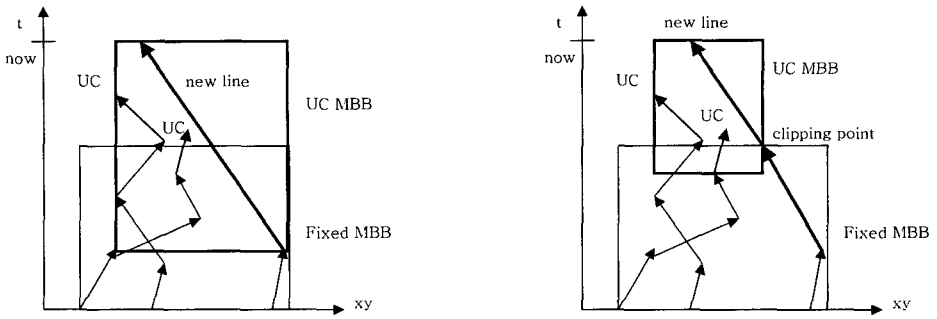


그림 11 시간 축 분할에서의 긴 선분의 예

7. 실 험

이 장에서 TR-tree의 성능을 평가하기 위해 이동체의 색인 중에서 3DR-tree, TB-tree와 성능을 비교하고자 한다. 이동체 색인에 관련된 이전 연구에서, 이동체의 과거 궤적에 대한 영역 질의 성능이 가장 우수한 색인은 3DR-tree와 TB-tree이다. 시공간 색인인 HR-tree는 타임스탬프 질의는 우수하나 영역 질의 성능이 나쁘다. MV3DR-tree는 이중 구조로 구성되어 있으며, 영역 질의를 하부 구조인 보조적인 3DR- tree를 이용하기 때문에 영역 질의의 성능은 3DR-tree와 같다. 궤적 질의 처리를 목적으로 하는 TB-tree는 이동체 수가 작거나, 영역 질의의 크기가 큰 경우에 3DR-tree 보다 성능이 우수하다. TR-tree, 3DR-tree, TB-tree는 각각 C++ 언어로 구현하였고, 윈도우 2000에서 256MB 메인 메모리, CPU Pentium IV 1.7 Ghz를 장착한 PC를 이용하여 실험하였다.

현실 세계의 이동체 데이터 집합에 관한 공개된 데이터가 없기 때문에, GSTD 생성기[16]을 사용하여 데이터 집합을 생성하였다. GSTD 생성기는 시간 간격 마다 이동체 위치인 점 좌표를 생성하기 때문에, 이전 위치를 참조하여 선분을 생성하여 색인에 저장하였다. 이동체 위치 변경 데이터를 생성하기 위한 매개변수로, 이동체의 초기 분포는 가우시안 분포이며 이동체의 이동은 랜덤(random)이다. 이동체의 위치 보고를 위한 타임스탬프(timestamp)는 100 K이며, 이동체의 공간 이동 매개변수는 0.02이다.

색인의 성능을 평가하기 위해 색인의 공간 활용도와 영역 질의 성능을 평가하였다. 영역 질의의 성능은 노드 접근 횟수로 측정한다. 영역 질의의 크기(RQS)는 각 도메인 크기의 5%, 10%, 20%이다. RQS 값은 각 도메인에 대한 비율로서 RQS 값이 5%이면 전체 영역의 0.0125%가 된다. RQS 값이 20%이면 전체 영역의 0.8%가 된다.

표 3 성능 평가 인자

성능평가 인자	설 명	사용 예
N_{MO}	이동체 수	500,1000,1500,2500
Tsample	이동체의 샘플링 횟수	500,1000,2000,5000
RQS	영역 질의 크기	5%, 10%, 20%
HRO	심함 중복 비율 (High Rate of Overlab)	30%, 35%, 40%, 45%, 50%

7.1 공간 활용도의 비교

높은 공간 활용도는 일반적으로 트리의 높이와 같은 질의 비용을 감소시킨다. R-tree는 증가하는 시간 도메인을 고려하지 않았기 때문에 공간 활용도가 대략 57%이다. R*-tree는 재삽입 연산에 의해 63%의 결과를 얻었다. R*-tree는 공간 활용도가 높았지만, R-tree와 영역 질의 성능을 평가를 해보았으나, 성능 개선이 없어 실험 평가에서 제외하였다. TB-tree는 같은 객체의 궤적을 모아서 데이터 페이지에 저장하기 때문에, 90%이상의 공간 활용도를 가진다. 하지만, TB-tree는 단말 노드 간의 중복이 심하기 때문에, 7.2장에서와 같이 영역 질의의 성능이 떨어진다. TR-tree는 시간 축의 비균등 분할의 효과로 77%~68%의 공간 활용도를 가진다.

색인의 크기는 공간 활용도가 제일 높은 TB-tree가 가장 작다. 이동체 수가 2500개 이고, 보고 횟수가 2000번인 실험 데이터 집합에서 삽입되는 선분의 개수는 5백만개이다. 삽입 후 3DR-tree의 크기는 248 MB이며, TB-tree는 155 MB, TR-tree는 212 MB이다.

그림 13은 이동체 수와 이동체 보고 횟수의 변화에 따른 공간 활용도 변화를 보여 준다. 그림 13(a)의 데이터집합은 보고 횟수가 2000으로 동일한 상태에서 이동체 수를 증가 시켰다. 그림 13(a)에서 TR-tree의 공간 활용도가 이동체 수가 많을수록 낮은 이유는 이동체 수가 보고 횟수 보다 크기 때문에, 공간 분할이 많이 발생하고 시간 축 분할 후 작은 엔터리를 가진 UC 최소경계박스(UC MBB)가 많기 때문이다. 그림 13(b)는 이동

체 수가 1000으로 동일한 상태에서 이동체의 보고 횟수가 증가 시켰다. TR-tree는 이동체의 보고 횟수가 증가할수록 공간 활용도가 높아진다.

7.2 이동체 수에 따른 영역 질의 성능 평가

공간 도메인의 크기가 같다면 이동체 수가 많을 수록, 이동체의 궤적은 중복이 심해진다. 그림 14의 데이터 집합은 보고 횟수가 2000으로 동일하다. 영역 질의의 성능은 RQS 값에 따라 3가지 종류의 영역 질의를 각각 1000개 생성하여 수행하였다. 그림 14에서 노드 방문 횟수는 1000번의 영역 질의를 수행하는 동안 방문한 노드의 총 횟수이다.

TR-tree는 강제 합병 정책과 긴 선분의 절단 정책을 사용하여 노드간의 중복을 최소화하기 때문에 성능이 우수하다. 그림 14(c)에서 TB-tree가 R-tree와 비슷한 영역 질의 성능을 보인 이유는 TB-tree의 공간 활용도가 R-tree 보다 월등히 우수하기 때문에 색인의 높이가 작기 때문이다. 특히, 20%와 같이 비교적 질의 영역의 크기가 큰 경우에는 공간 활용도가 영역 질의 성능에 중요한 요소가 된다.

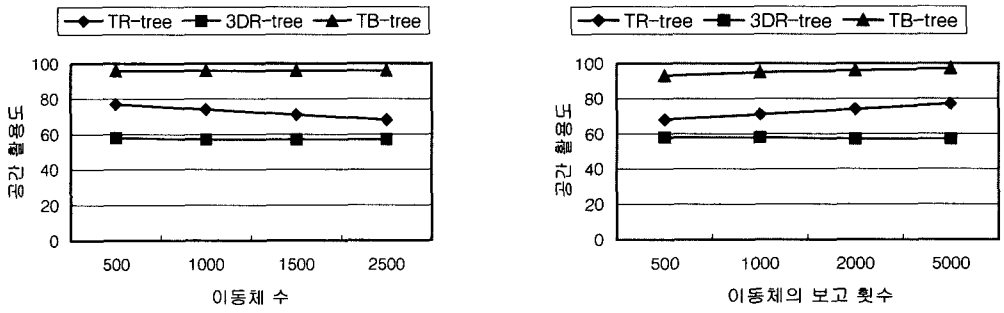
7.3 이동체 보고 횟수에 따른 성능 평가

보고 횟수가 증가하면, 시간 도메인의 크기도 증가한다. 또한 같은 이동체의 수에서 이동체의 궤적인 선분의 수도 증가한다. 그림 15에서 이동체의 수는 1000으로 동일하다. 그림 15에서 TR-tree는 시간 도메인의 크기를 고려하지 않고, 보고 횟수와 이동체 수를 이용하여 분할 축을 선택하기 때문에, 보고 횟수에 선형적으로 질의 비용이 증가한다.

질의 영역이 작은 경우에는 TB-tree는 노드간의 중복으로 인해 질의 성능이 낮지만, 질의 영역이 큰 경우에는 공간 활용도가 높기 때문에 질의 성능이 우수하다. TR-tree는 보고횟수가 500인 경우에만 3DR-tree와 10%의 성능을 향상을 보였으며, 다른 경우에는 20%이상의 성능 향상이 있었다.

7.4 삽입 성능의 비교

TB-tree는 R-tree와 다른 삽입 알고리즘을 사용한다. TB-tree는 같은 이동체의 궤적을 같은 단말 노드에 저장하기 하기 때문에, 새로운 선분이 삽입될 단말노드를 찾기 위해 다중 경로로 색인을 탐색한다. 이러한 이유 때문에 TB-tree의 삽입 비용의 70~80%를 삽입될 단말 노드를 검색하는데 소모하며, 3DR-tree와 TR-tree



(a) 이동체 수의 증가에 따른 공간 활용도 (b) 이동체 보고횟수의 증가에 따른 공간 활용도

그림 13 이동체 수와 보고 횟수에 따른 공간 활용도의 변화

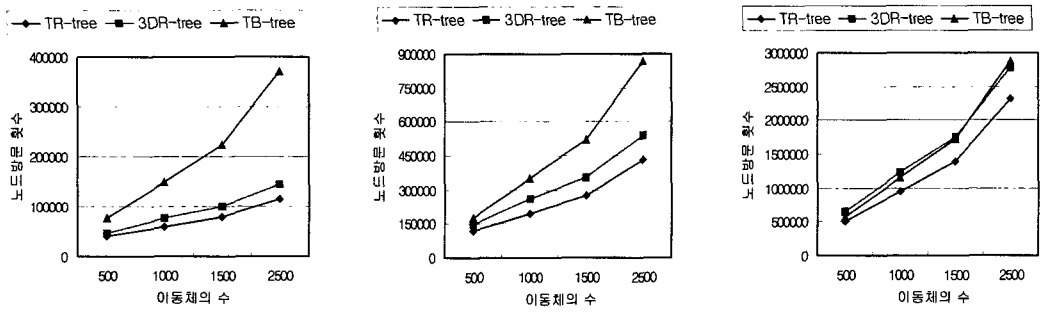


그림 14 이동체 수에 따른 영역 질의 성능 비교

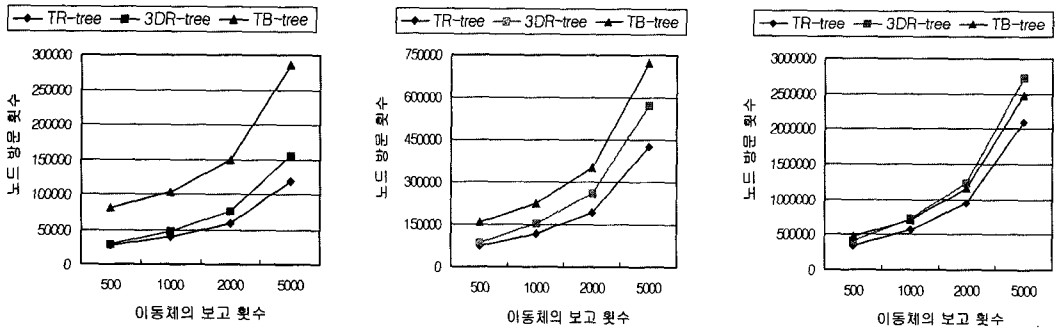
보다 삽입 성능이 매우 낮아 삽입 성능 비교에서는 제외한다.

삽입 성능은 CPU 연산 비용과 삽입에 필요한 I/O 비용으로 나눌 수 있다. TR-tree는 분할 축 선정 방법과 분할 방법이 3DR-tree에 비해 간단하기 때문에, 매우 낮다. 그림 16(a)와 같이 실험에 의하며 삽입 시간이 3배 정도 빠르다. 하지만 TR-tree는 강제 합병과 강제 합병 후 재분할 연산으로 인하여 3DR-tree에 비해 I/O 비용이 증가한다. 증가하는 I/O 횟수는 강제합병의 발생 횟수이다. 강제 합병의 발생 횟수는 HRO의 값에 따라 전체 삽입의 0.25%~0.1%이다. 하지만 이동체는 현재를 기준으로 삽입이 진행되기 때문에, 전체 색인에서 일부만이 삽입이 진행된다. 이러한 특징을 이용하여 TR-tree는 버퍼를 사용하여 I/O 횟수를 줄였다. 그림 16은 TR-tree와 3DR-tree 모두 2000개의 노드 버퍼를 사용하여 삽입 성능을 비교한 결과로서, TR-tree는 HRO이 40%이며, 강제합병 횟수는 전체 삽입의 0.18%이다. 그림 16(b)에서와 같이 TR-tree는 3DR-tree에 비해 색인의 노드 수가 작기 때문에, 버퍼를 사용하는 경우에 I/O 횟수가 3DR-tree에 비해 작다.

7.5 HRO(High Rate of Overlap)의 효과

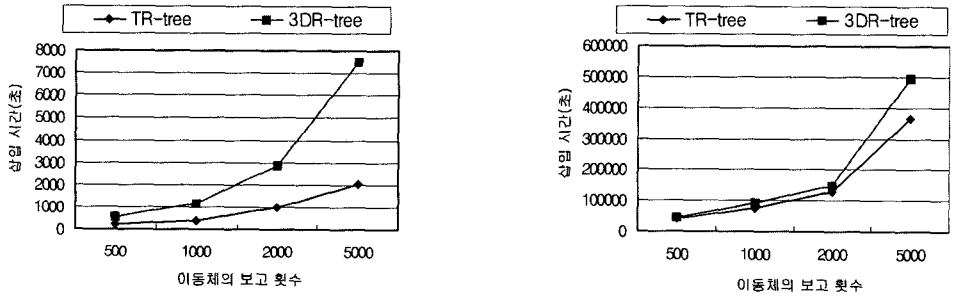
제안하는 강제 합병 정책과 긴 선분 절단 정책은 HRO 값을 기준으로 한다. TR-tree에서 HRO 값은 노드 간의 허용된 중복을 의미하며, HRO 값이 높은 경우에 노드 간의 허용된 중복도 증가한다. 하지만 HRO 값이 낮은 경우에서는 노드 간의 중복을 HRO 값 미만으로 낮추기 위해 많은 선분의 절단이 수행되어야 하며 이는 색인의 크기를 증가시키고, 강제 합병 후 재분할 시에 사장 공간(deadspace)이 큰 분할이 생길 수 있다.

HRO 값에 따른 영역 질의 결과는 HRO 값이 30%, 35%, 40%인 경우가 다른 경우 보다 항상 우수하다. 40%인 경우가 가장 많은 경우에 영역 질의 성능이 우수하다. 30% 미만의 낮은 HRO 값에서 질의 성능이 나쁜 이유는 빈번한 강제 합병 후 분할로 인하여 긴 선분의 절단이 많이 발생하여 단말 노드의 수가 증가하였기 때문이다. 45% 이상의 높은 HRO 값에서는 강제 합병의 횟수가 작고, 긴 선분의 발생 빈도가 낮기 때문에 노드간의 중복 제거가 작았다. 실험을 통해서 강제 합병되는 2개의 노드를 분석한 결과로 이동체의 궤적은 시간 축으로 증가되는 특징이 있기 때문에, 30%~40%의



(a) RQS=5% (b) RQS=10% (c) RQS=20%

그림 15 이동체 보고 횟수에 따른 영역 질의 성능 비교



(a) 삽입 시간 (b) I/O 횟수

그림 16 삽입 성능의 비교

HRO 값은 2개의 노드가 공간적으로 40% 이상의 중복이 발생하고 시간 축으로 80% 이상의 중복이 발생하는 경우를 의미한다. HRO 값이 30% 미만인 경우에, 공간적으로 40% 이하의 중복을 가진 2개의 노드를 합병 후 재분할을 수행하면 분할 된 노드의 사장 공간은 증가하게 된다. 이는 검색 성능을 낮게 하는 원인이 된다.

그림 17은 1000개의 이동체 수와 2000번의 보고 횟수를 가진 데이터 집합으로 실험한 결과이다. 그림 17(a)에서 5%의 영역 질의 성능은 HRO이 40%에서 가장 우수하였다. 그림 17(b)에서와 같이 HRO의 값에 따라 강제 합병 횟수는 감소한다.

8. 결론 및 향후 연구

이 논문에서는 이동체의 성장하는 궤적으로 인한 중복 문제와 긴 선분의 문제를 설명하였다. 이동체 색인은 연속적으로 성장하는 시간 축을 고려한 색인 구조를 고려하여야 한다. 제안하는 TR-tree는 시간 축 비균등 분할 정책을 사용하여 공간 활용도를 높였으며, 노드 간의 중복을 심한 중복 최소경계박스(HOMBB)를 정의하여 삽입으로 인해 중복이 심화되는 문제를 강제 합병 후 재분할 정책을 사용하여 해결하였다. 또한 분할 후에도 심한 중복을 야기하는 긴 선분을 정의하였으며, 절단 정책을 사용하여 제거 함으로서 노드간의 중복을 최소화하였다.

TB-tree는 공간 활용도가 좋으나, 단말 노드간의 중복이 심하여 작은 크기의 영역 질의 성능이 3DR-tree와 TR-tree 보다 나쁘다. 제안하는 TR-tree는 3DR-tree 보다 공간 활용도가 좋고, 단말 노드간의 중복이 작기 때문에 영역 질의의 성능이 뛰어 나다.

향후 연구로서 TB-tree와 같은 높은 공간 활용도를 가진 색인을 개발하고자 한다. 이동체 색인에서는 시간이 증가할수록 이동체의 위치 보고가 증가하기 때문에 대량의 위치 궤적 정보를 저장해야 한다.

참 고 문 헌

[1] H. Samet, The design and analysis of spatial data structures, Reading, MA: Addision-Wesley, 1989.

[2] J. T. Robinson, The K-D-B-tree: A search structure for large multidimensional dynamic indexes, ACM SIGMOD Conference, p10-18, 1981.

[3] A. Guttman, R-trees: A dynamic index structure for spatial searching, ACM SIGMOD Conference, p47-54, 1984.

[4] N. Beckmann and H. P. Kriegel, "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles", In Proc. ACM SIGMOD, p332-331, 1990.

[5] T. K. Sellis, N. Roussopoulos and C. Faloutsos, "The R+-Tree: A Dynamic Index for Multi-Dimensional Objects", Proceedings of the 13th VLDB Conference, p507-518, 1987.

[6] Z. Song and N. Roussopoulos, Hashing Moving Object, Intl. Conf. on Mobile Data Management, p161-172, 2001.

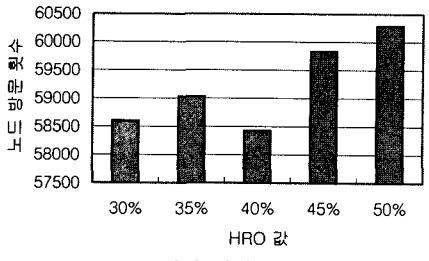
[7] S. Saltenis, C. S. Jensen, S.T. Leutenegger, and M. A. Lopez, Indexing the Positions of Continuously Moving Objects, In Proc. ACM SIGMOD on Management of data, p331-342, 2000.

[8] J. Tayeb, O. Ulusoy, and O. Wolfson. A Quadtree Based Dynamic Attribute Indexing Method, The Computer Journal, 41(3), p185-200, 1998.

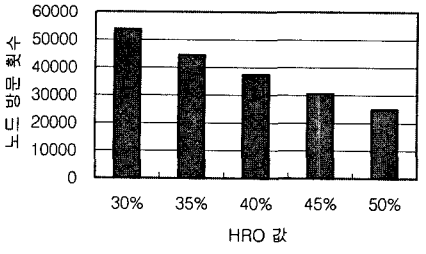
[9] D. Pfoser, Y. Theodoridis, and C.S. Jensen, Indexing Trajectories in Query Processing for Moving Objects, CHROochronos Technical Report, CH-99-3, October, 1999.

[10] D. Pfoser, C.S. Jensen, and Y. Theodoridis, Novel Approaches in Query Processing for Moving Objects, In Proc. of the VLDB Conference, p395-406, 2000.

[11] Y. Theodoridis, M. Vazirgiannis, and T. K. Sellis, Spatio-Temporal Indexing for Large Multimedia Applications, IEEE International Conference on Multimedia Computing and Systems, p441-448,



(a) 영역 질의 성능



(b) 강제 합병 횟수

그림 17 HRO 값에 따른 영역 질의 성능과 강제 합병 횟수

1996.
[12] M. A. Nascimento and J. R. O. Silva, Towards historical R-trees, ACM symposium on Applied Computing, p235-240, 1998.
[13] Y. Tao and D. Papadias, MV3R-Tree: A Spatio-Temporal Access Method for Timestamp and Interval Queries, Proceedings of International Conference on VLDB, p431-440, 2001.
[14] S. Berchtold, D. A. Keim, H. P. Kriegel, The X-tree: An Index Structure for High-Dimensional Data, International Conference on Very Large Data Bases, p28-39, 1996.
[15] D. Greene, An Implementation and Performance Analysis of Spatial Data Access Methods, Proceedings of International Conference on Data Engineering, IEEE, p606-615, 1989.
[16] Y. Theodoridis, J. R. O Silva and M.A Nascimento, On the Generation of Spatiotemporal Datasets, SSD, LNCS 1651, p147-164, 1999.

전 봉 기

1991년 부산대학교 컴퓨터공학과(공학사). 1993년 부산대학교 컴퓨터공학과(공학석사). 1993년~1998년 한국통신 연구소 전임연구원. 1998년~현재 부산대학교 컴퓨터공학과 박사과정 재학중. 관심분야는 데이타베이스, 지리정보시스템, 공간 객체데이타베이스

홍 봉 희

982년 서울대학교 전자계산기공학과졸업(공학사). 1984년 서울대학교 대학원 전자계산기공학과 졸업(공학사). 1988년 서울대학교 대학원 전자계산기공학과졸업(공학박사). 현재 부산대학교 공과대학 컴퓨터공학과 정교수/부산대학교 컴퓨터및정보통신 연구소 연구원. 관심분야는 병렬공간 DB, 분산 공간 DB, 개방형 GIS