이동체 데이타베이스에서 복합 질의를 위한 궤적 분할 트리의 설계 및 구현

(Design and Implementation of Trajectory Riving Tree for Combined Queries in Moving Object Databases)

임 덕 성 † 전 봉 기 †† 홍 봉 희 ††† 조 대 수 ††††
(Duksung Lim) (Bonggi Jun) (Bonghee Hong) (Dae-Soo Cho)

요 약 이동체는 시간에 따라 위치를 변경하는 특성과 이동체의 경로는 궤적으로 표현되는 특성을 가진다. 이동체 궤적 데이타에 대한 저장 및 검색을 처리하는 이동체 데이타베이스 시스템에서는 효율적인 데이타 접근 방법이 필요하다. 특히 궤적 검색을 위한 대표적인 질의 유형인 복합 질의는 영역내의 궤적 검색과 궤적의 일부분을 추출하는 과정을 포함한다. 그러나, 영역 질의에 우수한 색인 방법은 부분 궤적을 추출하기 위한 비용이 높은 단점을 가진다. 반면, 궤적 질의를 위한 색인 방법의 경우 노드 간의 중첩이 매우 높아 영역내의 궤적 검색 비용이 높은 단점이 있다.

이 논문에서는 이동체 데이타베이스에서 복합 질의를 효율적으로 처리하기 위해 TR-tree를 제시한다. TR-tree는 궤적 질의를 위해 궤적 보존 및 단말 노드의 용량을 증가시키고, 영역 질의 처리를 위해 사장 영역과 MBB(Minimum Bounding Box)의 중첩을 감소시키는 논리적 궤적 분할을 지원하는 특징을 가진다. 실험 평가에서 TR-tree는 STR-tree, TB-tree의 복합 질의 성능 비교에서 평균 25%의 노드 접근 회수를 감소시킨다.

키워드 : 이동체 데이타베이스, 이동체 색인, 영역 질의, 궤적 질의

Abstract Moving objects have characteristics that they change continuously their positions over time. The movement of moving objects should be stored on trajectories for processing past queries. Moving objects databases need to provide spatio-temporal index for handling moving objects queries like combined queries. Combined queries consist of a range query selecting trajectories within a specific range and a trajectory query extracting to parts of the whole trajectory. Access methods showing good performance in range queries have a shortcoming that the cost of processing trajectory queries is high. On the other hand, trajectory-based index schemes like the TB-tree are not suitable for range queries because of high overlaps between index nodes.

This paper proposes new TR(Trajectory Riving)-tree which is revised for efficiently processing the combined queries. This index scheme has several features like the trajectory preservation, the increase of the capacity of leaf nodes, and the logical trajectory riving in order to reduce dead space and high overlap between bounding boxes of nodes. In our performance study, the number of node access for combined queries in TR-tree is about 25% less than the STR-tree and the TB-tree.

Key words : Moving Objects Databases, Moving Objects Indexing, Range Query, Trajectory Query

· 본 연구는 한국전자통신연구원의 공간정보(4S)연계기술 지원사업에서 지원함

† 비 회 원 : 영진전문대학 컴퓨터정보기술계열 교수
junsung@yjc.ac.kr
†† 비 회 원 : 신라대학교 컴퓨터정보공학부 교수
bgjun@silla.ac.kr
††† 종신회원 : 부산대학교 컴퓨터공학과 교수
bhhong@pusan.ac.kr
†††† 정 회 원 : 한국전자통신연구원 LBS 연구팀 연구원
junest@etri.re.kr

논문접수 : 2003년 4월 17일
심사완료 : 2003년 11월 25일

1. 서 론

위치 기반 서비스(Location-Based Service)는 무선 통신에 기반 한 서비스로서 최근 그 중요성이 증대되고 있다. 이동체의 연속적인 이동으로 생성되는 이동체 궤적에 대한 위치 기반 질의를 처리하기 위해서는 이동체의 궤적 정보를 관리하는 이동체 데이타베이스 기술 개발이 선행되어야 한다. 이동체 데이타베이스는 물류 시

스템, 항해 시스템, 기상 정보 시스템, 항공 교통 통제 시스템, 디지털 전장 등과 같은 많은 응용 서비스를 위한 기반 기술로서 이동체의 궤적 정보를 효과적으로 관리하고, 빠른 검색을 제공하는 이동체 색인 방법이 필요하다.

궤적 검색을 위한 질의는 크게 위상 질의(topological query), 항해 질의(navigational query)와 복합 질의(combined query)로 분류된다[2]. 복합 질의는 1)궤적의 선택과 2)궤적의 부분을 추출하는 질의로서 "오전 7:00시에서 8:00시 사이에 시청 앞을 지나간 차량의 다음 1시간내의 궤적을 구하라"라는 질의는 "오전 7:00시에서 8:00시 사이에 시청 앞을 지나간 차량"과 같이 궤적을 선택하는 부분과 "다음 1시간내의 궤적"과 같이 궤적의 일부를 추출하는 부분으로 구성된다.

궤적에 대한 복합 질의는 영역내의 궤적 선택과 궤적의 일부분을 추출하는 과정을 포함하기 때문에 영역 질의에 우수한 색인 방법은 궤적의 일부를 추출하기 위한 비용이 높은 단점을 가진다. 반면, 궤적 질의를 위한 색인 방법의 경우 노드 간의 중첩이 매우 높아 영역내의 궤적 검색 비용이 높은 단점이 있다. 따라서 시공간 도메인의 동적 확장을 지원하고 대용량 궤적 정보에 대한 복합 질의를 효율적으로 처리하기 위한 이동체 궤적 색인이 필요하다.

복합 질의를 처리하기 색인은 궤적 보존을 지원하는 색인 구조로서 STR-tree, TB-tree[2]가 있다. STR-tree는 궤적을 보존하기 위해 동일 궤적을 최대한 가까운 위치에 저장함으로서 검색 성능을 향상 시키는 방법이다. TB-tree는 단말 노드에 동일한 궤적만을 저장함으로서 궤적 추출 비용을 감소시킨 색인 방법으로 STR-tree 보다 복합 질의 성능이 우수하다.

이 논문에서는 궤적 질의 성능은 우수하지만 공간적 지역성을 고려되지 않은 TB-tree를 사용하여 영역질의 및 복합 질의에 성능이 우수한 색인구조를 제시한다. TB-tree의 경우에는 다음과 같은 단점이 있다. 첫째, 궤적 보존을 위해 단말 노드에 하나의 궤적만을 저장함으로써 사장 영역을 증가시켜 비단말 노드간의 중첩이 심한 특성을 가진다. 둘째, 오버플로우 발생시 시간에 따라 분할하여 처리하기 때문에 공간적 지역성을 고려하지 않는다. 셋째, 단말 노드 저장 방법에서 데이타 중복을 포함한다.

영역 질의 및 복합질의를 효율적으로 처리하기 위한 색인 구조는 시공간적 지역성을 고려하여 사장 영역 및 비단말 노드간의 중첩을 줄이고, 저장 방법에서 데이타 중복이 없어야 한다. 이 논문에서는 효율적 궤적 질의 처리를 위해 단말 노드에 저장되는 연속된 궤적 선분의 MBB들의 데이타 중복을 제거하여 단말 노드의 용량을 증가시키고, 사장 영역 및 비단말 노드간의 중첩을 줄여 영역 질의를 효율적으로 처리하기 위한 방법으로 궤적을 논리적으로 분할하는 색인 구조인 TR(Trajectory Riving)-tree를 제시한다.

이 논문에서 제시한 TR-tree는 노드간의 중첩을 줄여 궤적 선택시 비용을 줄이고, 단말 노드 저장 구조의 중복을 줄여 궤적 추출 성능을 개선한 색인으로서 STR-tree, TB-tree보다 이동체의 복합 질의 성능이 우수한 것을 실험을 통해 검증한다.

이 논문의 구성은 다음과 같다. 2장에서는 궤적 색인을 위한 관련 연구를 기술하고, 3장에서는 궤적 질의를 효과적으로 처리할 수 있는 TR-tree의 색인 구조를 제시한다. 4장에서는 삽입 및 분할 알고리즘을 제시하고, 5장에서는 궤적 색인의 성능을 비교하고, 마지막으로 6장에서는 결론과 향후 과제를 기술한다.

2. 관련연구

이동체의 과거 데이타 검색을 위한 색인은 질의에 따라 크게 영역 질의를 위한 색인, 타임슬라이스 질의를 위한 색인, 궤적 질의을 위한 색인으로 나누어 볼 수가 있다. 영역 질의를 위한 색인은 이동체의 과거 시간 뿐만 아니라 공간에 대한 영역을 검색하는 것으로 R-tree를 기반으로 한다. 3DR-Tree[5]는 시간을 2차원 공간 외의 또 다른 차원으로 간주하여 데이타를 3차원 MBB 형태로 표현한 색인으로 3차원 영역 질의에 효율적이지만 궤적 질의를 시공간 조인으로 처리해야 하는 단점이 있다.

타임슬라이스 질의를 위한 색인인 HR-tree[4]는 R-tree에 시간 개념을 추가하여 이력 정보를 표현하는 구조로서 각 타임 스탬프마다 R-tree인스턴스를 생성한다. 타임슬라이스 질의에 좋은 성능을 나타내지만 이동체의 궤적에 대한 영역 검색의 경우 추가적 비용을 수반해야 하고, 궤적 질의에는 성능이 매우 낮다. MV3R-tree[6]는 타임슬라이스 질의와 영역 질의에 좋은 성능을 나타내는 색인으로 HR-tree의 장점과 3DR-tree장점을 통합한 구조이지만, 삽입 비용이 크고, 궤적 질의를 지원하지 않는 구조를 가진다. SETI[11]는 시공간에서 공간적 지역성과 시간의 연속된 증가를 고려하여 공간적으로 분할된 각 셀에 시간 색인을 두는 복합 색인 구조로서 셀 경계에 있는 객체를 중복 저장함으로서 색인의 크기가 증가하고, 다수의 시간 색인 관리 비용이 높다는 단점이 있다.

궤적 질의를 위한 색인인 STR-tree[2]는 궤적 보존을 위한 색인 구조이지만 TB-tree[2]에 비해 영역 질의 및 궤적 질의에서 높은 검색 비용을 가지는 색인 구조이다. TB-tree는 이동체의 궤적을 3 차원의 방향성을

가진 MBB로 저장하고, 이 MBB의 연결로 이동체의 궤적을 표현할 수 있는 구조이다. 단말 노드에서 이동체의 궤적은 양방향 연결 리스트(linked list)로 구성되어 있으므로 궤적 추출에 대한 질의를 처리할 때는 뛰어난 성능을 보인다. 그러나 TB-Tree는 궤적에 대한 질의 성능은 좋은 반면 이동체의 개수가 증가함에 따라 시간 간격에 대한 질의와 공간 간격에 대한 영역 질의에는 좋지 않은 성능을 보인다. TB-tree는 이 논문의 기반 색인 구조이지만 이 논문에서 제시한 TR-tree는 구조적인 면에서 궤적 분할을 지원하고 단말 노드의 저장구조를 개선시킨 구조이고, 알고리즘면에서 시공간 지역성을 고려하여 노드간의 중첩을 줄이는 분할 정책을 제시한 점에서 TB-tree와 차이점을 가진다.

사장 영역의 중첩을 줄여 색인의 성능을 향상하기 위한 연구에는 최소 영역 확장 정책(Least Area Enlargement Policy)[8]이 있다. 그러나 이 방법은 삽입시 영역 질의의 비용을 줄이기 위해 삽입될 노드를 찾는 방법으로 궤적 보존과 상반되는 방법이다. 노드간의 중복을 줄이기 위한 연구는 노드간의 중첩을 허용하는 R*-tree[12], X-tree[13], 그리고 노드간의 중첩을 허용하지 않는 R+-tree[14]가 있다. R*-tree는 단말 노드간에 중복을 최소화하기 위해 최소 중복 확장 정책(Least Overlap Enlargement Policy)[12]을 분할 정책으로 사용하고, X-tree는 노드 분할시 중복 비율에 따라 수퍼 노드를 생성함으로서 노드간 중복을 회피하는 방법을 사용하고, R+-tree는 노드간의 중첩을 허용하지 않게 하기 위해 클리핑 기법을 사용한다. 그러나, 이 방법들은 단말 노드에 인접한 객체들을 저장할 경우 발생하는 중복을 개선하기 위한 연구로서 궤적 보존 방법의 적용이 어려운 문제가 있다.

3. 색인 구조

이 장에서는 복합 질의를 효율적으로 처리하기 위한 TR-tree의 구조로서 궤적 클러스터링을 위한 단말 노드의 구조와 사장 영역 및 비단말 노드간의 중첩을 줄이기 위해 색인 구조를 기술한다.

3.1 궤적 클러스터링

이동체의 복합 질의에서 궤적 추출을 효율적으로 처리하기 위해 궤적의 보존이 필요하다. 궤적 보존은 하나의 궤적을 구성하는 각 선분을 인접한 위치에 저장하여 궤적 검색시 비용을 줄이는 방법으로 STR-tree에서 제시한 방법은 보존 매개 변수가 증가할수록 MBB간의 중첩을 증가시켜 검색 성능을 저하시키는 단점을 가진다. 궤적 보존의 가장 효율적 방법은 이동체의 궤적을 동일 블록에 저장하는 궤적 클러스터링 방법이다. 이 논문에서는 단말 노드의 궤적 클러스터링을 위해 단말 노드에 저장되는 궤적의 수를 Clustering Factor(CF)로 정의하고, 단말 노드에 저장될 수 있는 최대 궤적수를 Max Clustering Factor(MCF)로 정의한다. 이 논문에서 사용되는 용어는 그림 1과 같다.

정의 1) Clustering Factor(CF), Max Clustering Factor(MCF)

$$S = \{N_i\}$$
$$N_i^0 = \{E_1, E_2, \cdots, E_m\}$$
$$CF(N_i^0) = Card(\{x \mid x = Traj(E_i),\ E_i \in N_i^0\})$$
$$MCF = Max(\{x \mid x = CF(N_i^0), N_i^0 \in S\})$$

그림 2(a)는 시각 T_1에서 T_0까지 증가하면서 이동체의 공간 위치의 변경을 표현한 궤적 τ_1, τ_2이다. 그림 2(b)와 같이 MCF를 1로 할 경우 단말 노드에 하나의 궤적을 저장하는 TB-tree의 구조와 유사하지만, MCF의 값을 증가함에 따라 R-tree와 같이 단말 노드에 공간적으로 근접한 다수의 궤적을 저장함으로서 공간적 지역성을 높게 할 수 있다. 그림 2(b)에서 궤적 τ_1의 T_1에서 T_6까지의 궤적이 저장되는 단말 노드의 MBB R_1^{16}은 하나의 노드에 하나의 궤적만을 저장하므로 궤적간의 공간적인 지역성보다 궤적 보존에 초점을 맞추고 있다. 그림 2(c)와 같이 MCF가 2일 경우 단말 노드

S	: 노드 집합	τ_i	: 궤적 식별자 i의 궤적
N	: 노드	$\overline{\tau_i}$	: 단말노드에서 정규화된 궤적
E	: 엔트리	P^i	: 레벨 i의 노드의 총수
$m, \overline{m}$	: 단말, 비단말 노드의 용량	N^i	: 레벨 i의 노드 (단말 노드: N^0)
$Traj(E)$	: 엔트리에서 궤적 식별자를 리턴하는 함수		
$Max(x)/Min(x)$	: 집합x에서 최대값/최소값을 리턴하는 함수		
$Card(x)$	: 집합 x에서 카디널리티를 리턴하는 함수		

그림 1 용어 정리

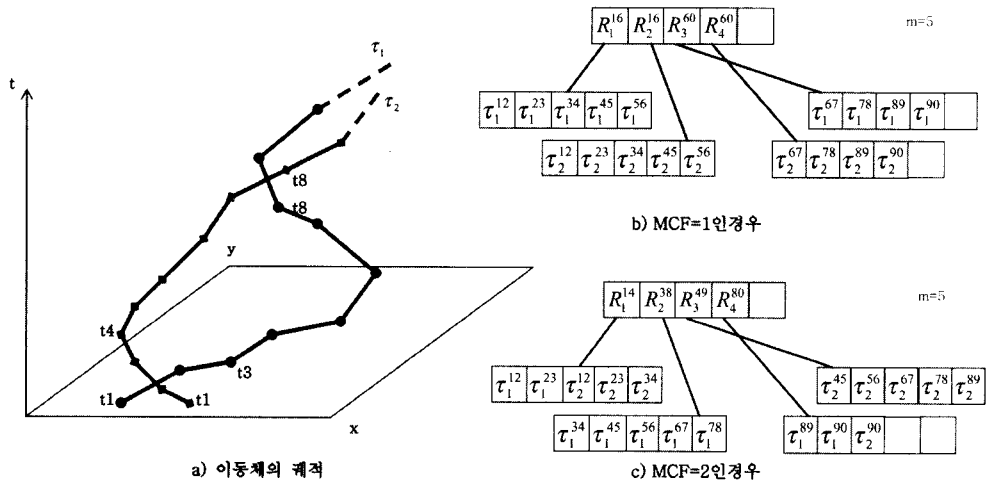


그림 2 이동체의 궤적과 표현

는 궤적을 2개까지 저장할수 있으므로 하나의 궤적이 동일한 보고 주기를 가질 경우 최대 T_4까지의 궤적을 저장할 수 있다. 그림 2(c)의 R_1^{14}가 가리키는 노드에는 τ_1, τ_2의 각 선분들이 각각 2개, 3개씩 저장되어 있다. 즉, MCF가 2인 경우에는 단말 노드에 저장되는 궤적당 선분수가 줄어드는 대신 인접한 다른 궤적을 저장할 수 있기 때문에 단말 노드의 MBB는 MCF가 1인 경우 보다 시간적으로 축소될 뿐만 아니라 공간적 지역성을 가지게 된다.

3.2 단말 노드의 데이타 중복

단말 노드에 궤적을 저장하기 위해 MBB와 방향성을 유지하는 저장방법[2]은 중복을 포함한다. 궤적이 선분의 집합으로 모델링될 경우 선분은 하나의 객체로서 색인에 삽입된다. 삽입된 객체는 단말노드의 엔트리에서 <id, trajectory#,MBB,orientation>와 같은 구조로 저장된다. 이 경우 단말 노드에 저장되는 연속된 두 객체의 경우 후위 객체의 시작점은 전위 객체의 끝점과 중복이 발생한다. 예를 들어 동일 궤적 ID을 가진 시각 t_2에 삽입된 객체 τ_{12}와 시각 t_3에 삽입된 τ_{23}선분을 동일한 단말 노드에 저장할 경우 시각 t_2에서의 위치 데이타는 단말 노드의 두 개의 엔트리에 중복되어 저장된다.

데이타 중복에 의해 낮은 용량을 가지는 색인은 색인의 크기와 검색 성능에서 비효율적이기 때문에 중복을 제거하여 단말 노드의 용량을 확장할 필요가 있다. 이 논문에서는 단말노드에 삽입되는 객체의 마지막 점만을 저장하여 궤적을 점들의 배열인 다중선으로 저장한다. 이 경우 노드의 팬아웃은 최대 2배까지 확장될 수 있다. 그러나, 끝점만을 저장하여 구성된 다중선의 경우 다중선 내의 검색은 가능하지만 다중선의 시작 선분의 검색은 불가능하다. 이를 보완하기 위해 다중선의 첫 선분은 시작점을 포함하여 두개의 점으로 저장한다.

3.3 사장 영역 감소를 위한 논리적 궤적 분할

단말 노드에 하나의 궤적만을 저장할 경우 궤적의 증가에 따른 사장 영역이 증가하는 문제가 있다. 사장 영역의 증가는 검색 시 불필요한 노드의 접근을 유발하여 I/O비용의 증가를 초래한다. 또한 사장 영역의 증가는 비단말 노드의 중첩을 증가시키는 문제점이 있다. 그림 3은 두 이동체의 궤적에 대한 MBB와 질의 영역 Q_1을 나타낸다. 이 경우 궤적 τ_A는 Q_1과 중첩 되지 않지만, 궤적 τ_A의 MBB인 R_{A1}과 R_{A2}는 Q_1과 서로 중첩이 되므로, 검색 시 불필요한 노드 접근을 필요로 한다. 또한 서로 다른 두 이동체의 궤적 τ_A와 τ_B는 시공간상의 두점에서 서로 교차되지만 단말 노드에 하나의 궤적만을 저장하는 경우 서로 다른 두 궤적의 MBB가 중첩되

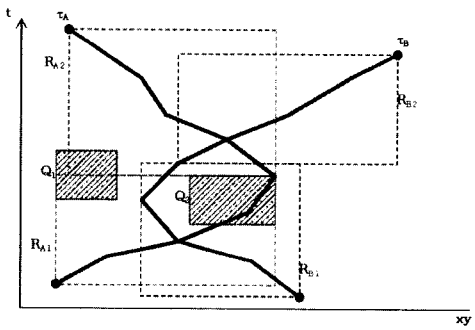
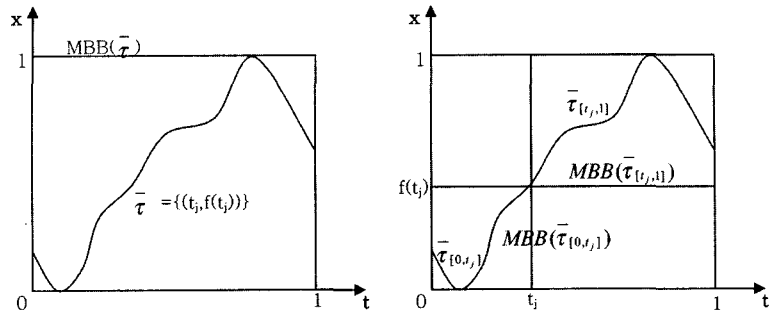


그림 3 궤적의 중첩



a) 정규화된 궤적 $\bar{\tau}$ b) $\bar{\tau}$ 의 분할

그림 4 단위 공간에서의 궤적 표현

는 영역이 매우 커지게 된다.

사장 영역 및 노드간의 중첩을 줄이기 위한 방법으로 단말 노드 MBB를 분할하는 방법이 있다. 단말 노드의 분할은 사장 영역을 줄이고 감소된 사장 영역으로 인해 비단말의 MBB간의 중첩도 줄어들게 된다.

정리 1. 한 궤적의 MBB는 두 개 이상으로 분할된 궤적의 MBB의 합보다 작지 않다.

증명. 시간을 포함한 d차원의 궤적 τ의 MBB를 단위 영역 $(x_1,x_2,...,x_n)$에 매핑하는 함수를 이용하면 정규화된 궤적 $\bar{\tau}$를 얻을 수 있다. 2차원의 경우 정규화된 궤적은 그림 4(a)와 같이 표현될 수 있고, 시간 t에 대한 전사 함수로서 다음과 같이 정의된다.

$$\bar{\tau}: t \rightarrow f(t)(t \in [0,1], f(t) \in [0,1])$$

그림 4(b)와 같이 궤적 $\bar{\tau}$의 임의의 한 점 $(t_j, f(t_j))$를 기준으로 궤적을 분할할 경우 두개의 부분 궤적 $\bar{\tau}_{[0,t_j]}$, $\bar{\tau}_{[t_j,1]}$로 나뉜다. 궤적 $\bar{\tau}$이 구성하는 MBB의 넓이(S)와 부분 궤적 $\bar{\tau}_{[0,t_j]}$, $\bar{\tau}_{[t_j,1]}$이 각각 구성하는 MBB의 총넓이의 차이(Diff)는 아래와 같다. 단, 부분 궤적의 MBB의 넓이는 MBB의 최대값과 최소값의 차이와 분할 되는 시점의 시각의 곱으로 계산되고, $X_{\max}(MBB(\bar{\tau}[0,t_j])) - X_{\min}(MBB(\bar{\tau}[0,t_j]))$를 a로, $X_{\max}(MBB(\bar{\tau}[t_j,1])) - X_{\min}(MBB(\bar{\tau}[t_j,1]))$를 b로 대치한다.

$$\begin{aligned}
Diff &= S(MBB(\bar{\tau}[0,1])) - (S(MBB(\bar{\tau}[0,t_j])) + S(MBB(\bar{\tau}[t_j,1]))) \\
&= 1 - (X_{\max}(MBB(\bar{\tau}[0,t_j])) - X_{\min}(MBB(\bar{\tau}[0,t_j]))) * t_j \\
&\quad - (X_{\max}(MBB(\bar{\tau}[t_j,1])) - X_{\min}(MBB(\bar{\tau}[t_j,1]))) * (1-t_j)) \\
&= 1 - at_j - b(1-t_j) \\
&= 1 - (a-b)t_j + b \\
&= (b-a)t_j + 1 - b
\end{aligned}$$

분할된 MBB에서 각 축의 최대값과 최소값의 차이는 0과 1사이이므로 $0 \le a \le 1$, $0 \le b \le 1$, $0 \le t_j \le 1$이 성립한다. 이와 같은 조건에서 t에 대한 두 넓이의 차이는 그림 5와 같이 $0 \le (b-a)t_j + 1 - b \le 1$의 범위를 가진다.

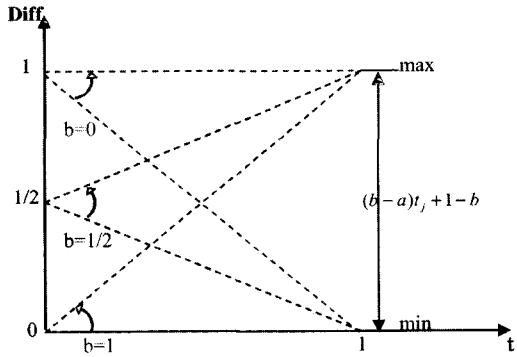


그림 5 궤적의 전체 영역과 분할 영역의 관계

따라서 $S(MBB(\bar{\tau}_{[0,1]})) \ge S(MBB(\bar{\tau}_{[0,t]})) + S(MBB(\bar{\tau}_{[t,1]}))$ 와 같은 관계가 성립하므로 궤적의 MBB는 궤적의 두 개 이상의 분할된 MBB의 합보다 작지 않다. 동일한 방법으로 d>2인 차원 에서도 적용 가능하다. □

정리 2. 단말 노드 팬아웃 증가율(σ)과 단말 노드당 MBB의 개수 증가율(ρ)이 동일한 경우 색인의 높이는 동일하다.

증명. 단말 노드에 저장되는 궤적을 다중선으로 표현할 경우 저장되는 엔트리의 개수를 m'이라고 할 때 단말 노드 팬아웃 증가율(σ)은 $\sigma = m'/m$로 정의된다. 단말 노드당 MBB 개수 증가율(ρ)은 단말 노드의 MBB의 분할로 인해 생성되는 상위 노드의 엔트리의 개수로서 다중 레벨 색인(multi-level index)에서 단말 노드의 MBB는 상위 노드에서 1개의 엔트리로 표현되는 반면 MBB의 분할로 인해 1개이상의 엔트리가 존재할 경우 이에 대한 증가율을 의미한다. 색인 S_j에 삽입되는 모든

데이타의 개수가 n이고 단말 노드의 공간 활용도가 1인 경우 단말 노드 N^0의 개수는 $P^0 = Card(\{N_i^0\}) = \lceil n/\sigma m \rceil$, $(N_i^0 \in S_j)$와 같다. 단말 노드의 MBB 분할이 발생할 경우 상위 노드 N'의 엔트리 개수는 $Card(\{E_k^1\}) = \rho P^0 = \rho\lceil n/\sigma m \rceil$와 같다.

단말 노드에 궤적을 저장할 경우 궤적의 각 선분을 MBB로 저장하고, 단말 노드의 MBB가 상위 노드에서 하나의 엔트리로 표현되는 색인 S_1과 단말 노드에 다중선을 저장하고 분할을 허용하는 색인 S_2가 있을 경우 두 색인의 높이(Height)의 차는 다음과 같다.

$$Height(S_2) - Height(S_1) = \log_{\overline{m}}(\lceil n/\sigma m \rceil \rho) + 1 - (\log_{\overline{m}}(\lceil n/m \rceil) + 1)$$

$$= \log_{\overline{m}} \frac{\lceil n/\sigma m \rceil \rho}{\lceil n/m \rceil}$$

$$\cong \log_{\overline{m}} \frac{n\rho/\sigma m}{n/m} = \log_{\overline{m}} \frac{\rho}{\sigma}$$

레벨 1의 엔트리 개수를 $\overline{m}$으로 나눈후 나머지의 엔트리 수는 색인에 삽입되는 데이타의 개수가 클수록 높이에 영향을 거의 주지 않기 때문에 이 차이를 무시할 경우 색인의 높이의 차이는 식과 같이 $\log_{\overline{m}} \rho/\sigma$ 와 같다. 이 경우 단말 노드 팬아웃 증가율(σ)과 단말 노드당 MBB 개수 증가율(ρ)이 동일할 경우 두 색인의 높이의 차이는 $\log_{\overline{m}} 1 = 0$과 같다. □

정리 1,2와 같이 궤적 보존을 지원하는 색인에서 단말 노드의 MBB의 분할로 인한 사장 공간은 최대 100%로 축소가능한 반면 색인의 높이는 $\log_{\overline{m}} \rho/\sigma$에 따라 증가하게 된다. 그러므로 분할로 인한 색인의 크기는 팬아웃 증가율에 반비례하므로 단말노드의 MBB를 분할하여 사장 영역을 감소시키는 것이 필요하다.

단말 노드MBB의 분할을 지원할 경우 그림 3의 R_{B1}은 그림 6의 MBB R_{B1}과 R_{B2}로 분할 될 수 있다. 이와 같은 색인 구조는 첫째, 사장 영역의 크기를 축소하여 검색시 불필요한 노드의 접근을 줄일 수 있다. 둘째, 사장 영역의 축소의 효과로 MBB간들의 중첩이 작아지게 되어 Q_1과의 MBB의 중첩도 발생하지 않게 된다.

그림 3의 경우 질의 영역 Q_1과 궤적 τ_A의 MBB R_{A1}과 R_{A2}간의 중첩이 발생되었으나, 그림 6의 경우 궤적 τ_A의 분할된 MBB는 질의 영역 Q_1과 교차하지 않으므로 검색시 노드 접근을 줄이게 된다. 또한, 그림 3에서 질의 영역 Q_2를 검색시 궤적 τ_A, τ_B의 단말 노드를 모두 검색해야 하지만, 그림 6의 경우 MBB간의 중첩이 작아져 검색시 궤적 τ_A의 단말 노드만을 검색하여 검색 비용을 줄일 수 있다.

3.4 색인 구조

궤적의 클러스터링과 클러스터링된 노드간의 사장 영역의 축소와 노드간의 교차를 줄이는 TR-tree의 색인 구조는 그림 7과 같다. 그림 7의 색인 구조는 R-tree의 색인 구조와는 상이하다. 즉, 단말 노드에 궤적별 클러스터링을 지원하고, 궤적의 연결성을 위해 단말 노드간의 이중 연결 구조를 가진다. 또한 단말 노드의 부모 노드의 엔트리는 단말노드의 MBB를 두개 이상 분할된 형태로 가질 수 있는 구조로서 TB-tree와 구별된다.

다수의 궤적을 저장하고, 중복을 제거하기 위한 TR-tree의 단말 노드의 구조는 그림 8과 같이 궤적 엔트리(TEntry)와 선분 엔트리(SEntry)로 구성된다. TEntry는 단말 노드에 저장된 궤적의 연결 정보 및 부가 정보를 저장하기 위한 요소로서 전체 궤적 ID, 궤적의 성장 유무를 판별하기 위한 정보(GM: 4.1참조), 시간에 따른 궤적의 연결 정보로서 이후 시간의 궤적을 저장하는 노드의 위치를 Right, 이전 시간을 위해서 Left정보를 저장하고, 다중선의 시작점을 저장한다. SEntry는 궤적의 부분인 다중선을 구성하기 위한 기본 요소를 표현하기

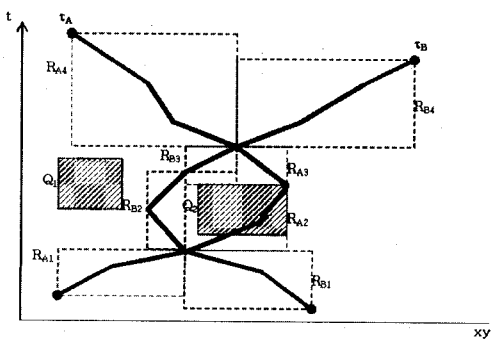


그림 6 궤적의 분할

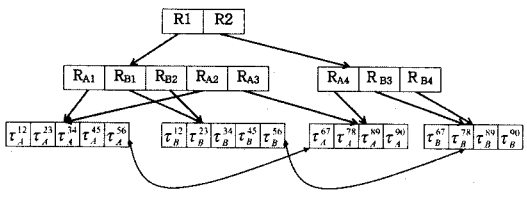


그림 7 TR-tree 색인 구조

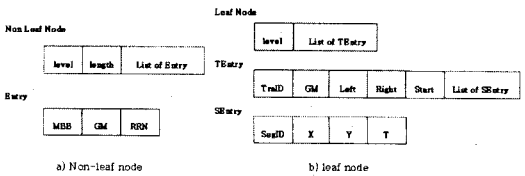


그림 8 단말 노드와 비단말 노드의 구조

위한 구조로서 삽입시 선분의 ID, 끝점을 저장한다.

4. 삽입 및 분할 알고리즘

이 장에서는 궤적 증가에 따른 단말 노드에서 MBB 분할 방법 및 노드 분할 방법에 대한 개념을 정의하고, 이를 기반으로 TR-tree의 삽입 알고리즘 및 분할 알고리즘을 제시한다.

4.1 궤적 증가에 따른 분할

이동체의 궤적은 시간에 따라 성장하는 특성을 가지므로 그림 6에서 이후에 삽입되는 이동체의 궤적 정보는 MBB R_{A4}와 R_{B4}의 영역을 증가하게 된다. 이와 같이 시간 T에 따라 점진적으로 증가하는 MBB를 Growing MBB(GM)로 정의한다.

정의 2) Growing MBB(GM), Non Growing MBB (GM^-)

$\overline{E}$: 종료되지 않은 궤적의 가장 최근 보고된 데이타를 포함하는 엔트리 GM(E_j^i) : E_j^i를 루트로 하는 하위 트리가 $\overline{E}$을 포함하는 경우 참을 반환하는 함수라고 할 때,

$GM = MBB(N^i)$ iff $\overline{E} \in N^i, i = 0$

$GM^- = MBB(N^i)$ iff $\overline{E} \notin N^i, i = 0$

$GM = MBB(N^i)$ iff $\exists E_j^i(GM(E_j^i)), E_j^i \in N^i$ and $i > 1$

$GM^- = MBB(N^i)$ iff $\forall E_j^i(\neg GM(E_j^i)), E_j^i \in N_n$ and $i > 1$

이동체의 현재 위치 보고에 따라 GM이 확장되므로 GM과 다른 MBB간의 중첩을 유발시킨다. 따라서 중첩을 최소화 하기 위해 GM의 분할이 필요하다. GM분할 여부 선정은 단말 노드와 비단말 노드에서 다른 방법을 사용한다. 단말 노드간의 분할 여부 선정을 위해 중첩율(Overlap Ratio)과 궤적증가율(Trajectory increment Ratio)을 정의한다.

정의 3) Overlap Ratio(OR), Trajectory increment Ratio(TR)

$$N = \{E_k\}, 1 \le k \le p \le m, Card(N) = p$$

$$overlap(E_k) = \sum_{i=1, i \neq k}^{p} S(MBB(E_k) \cap MBB(E_i))$$

$$OR(E_k) = overlap(E_k^{t+1}) / overlap(E_k^t)$$

$$TR(N) = Card(Traj(N)) / MCF$$

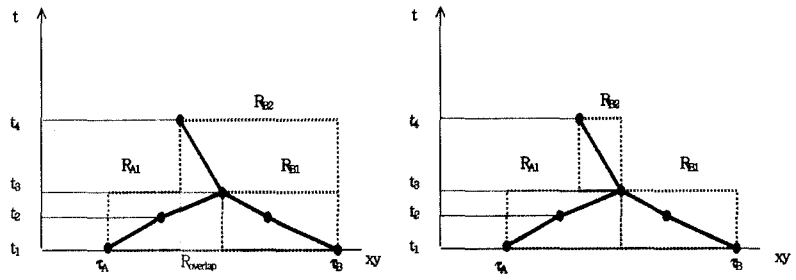
중첩율(OR)은 t시간에 엔트리간의 중첩과 t+1시간에 궤적 삽입에 의해 증가된 MBB에 의한 중첩의 비율로서 정의된다. 궤적증가율 TR은 노드가 포함하는 궤적수와 단말 노드가 포함할 수 있는 최대 궤적 수(MCF)의 비율로써 정의된다.

삽입시 두 인자를 이용하여 단말 노드간의 GM의 분할 여부 선정은 표 1과 같다. 단말 노드의 궤적수 CF가 MCF보다 작고 중첩이 발생하지 않으면 GM은 분할되지 않는다. 그러나 중첩이 발생할 경우에는 중첩율에 따라 GM을 분할할 필요가 있다. 단말 노드 MBB간의 최대 허용 중첩율을 MOR이라 할 때, 최대 허용 중첩율을 초과하는 경우($OR > MOR$)에는 분할하고 그렇지 않은 경우에는 궤적 증가율에 따라 분할 여부를 결정한다. 비단말 노드간의 분할은 오버플로우 시에만 적용한다.

그림 9(a)에서는 그림 6의 이동체 A, B가 순서적으로 보고할 경우 궤적 τ_A가 t_4시간에 보고한 궤적 정보의 삽입으로 궤적 τ_B의 GM이 R_{B1}에서 R_{B2}로 증가하게 되고 이 GM은 궤적 τ_A의 GM인 R_{A1}와의 중첩 영역 $R_{overlap}$을 생성시킨다. 이 경우 MOR=1일 경우 중첩의 발생으로 GM의 분할이 발생한다. 그림 9(b)는 GM의 분할 후 상태로서 MBB간 중첩이 발생하지 않게 된다.

표 1 분할 여부 선정

		OR≤MOR	OR>MOR
TR	≤1	Insert	Split MBB
	>1	Split Node	Split Node



a) 두 이동체 궤적의 중첩 b) 이동체 궤적의 분할

그림 9 궤적의 중첩 및 분할

4.2 삽입 알고리즘

제시한 색인 구조의 삽입 알고리즘은 그림 10과 같이 선분이 삽입될 단말 노드를 검색하는 FindNode()를 사용하는 TB-tree와 ChooseSubtree()와 같이 새로운 궤적이 삽입될 경우 삽입될 노드를 찾는 R-tree색인을 기반으로 한다. 이 때, 삽입할 단말 노드의 검색 경로는 스택에 저장된다.

```
Algorithm Insert(N,E)
Invoke FindNode(N,E)
IF node N' is not found,
        ChooseSubtree(N,E)
Compute OR using entries E' in parent node of N'
  IF N' has space
     IF TR >1
        Invoke Split(N')
     ELSE IF OR > MOR
        Invoke SplitGM(N')
     ELSE
        Insert E
   ELSE Invoke Split(N')
AdjustTree(N')
```

그림 10 삽입 알고리즘

삽입될 노드의 여유가 없는 경우나 궤적 수의 증가가 발생하여 MCF를 초과할 경우에는 삽입된 선분을 포함한 노드N'의 분할 작업이 이루어진다. 그러나 삽입될 노드에 여유가 있고, 삽입되는 엔트리와 동일한 궤적을 저장하지만, 삽입으로 인해 발생되는 중첩율이 높은 경우(OR>MOR)에는 삽입되는 엔트리는 단말노드에 삽입되는 동시에 삽입되는 단말 노드의 MBB는 분할된다(SplitGM). 이외의 경우에는 검색되는 노드에 선분을 삽입한다. 삽입 또는 분할이 이루어진 후 MBB의 변경을 반영하기 위해 AdjustTree()를 호출한다.

4.3 분할 알고리즘

궤적 선분의 삽입으로 인해 노드가 오버플로우가 되면 노드 분할이 필요하다. TR-tree의 분할의 주된 목표는 분할후 예상되는 노드간의 중첩을 줄이는 동시에 높은 공간활용도를 달성하는 것이다. 분할 방법은 단말 노드와 비단말 노드를 구분하여 처리한다. 단말 노드의 노드 분할은 TR > 1인 경우와 오버플로우시에 발생한다. 두 경우 모두 가장 최근에 삽입된 선분과 나머지 선분들의 두 그룹으로 분할한다. 이 방법은 단말 노드의 시간에 대한 분할로 궤적 보존 및 공간 활용도를 향상시킨다.

비단말 노드의 오버플로우가 발생할 경우에는 GM분할이 발생한다. GM분할의 기본 정책은 분할 이후 중첩의 최소화와 공간 활용도의 최대화이다. 중첩의 최소화 방법은 R*-tree의 분할 방법이 가장 효과적이다. 그러나, 이 방법의 경우 분할 후 삽입되는 궤적에 의해 분할된 노드의 MBB가 확장하여 노드간 중첩을 증가시키는 문제가 있다. TR-tree에서는 추가적인 삽입에 의해 증가되는 GM을 이용한 분할 방법인 GMTimeSplit, GMSpaceSplit과 공간 활용도 향상을 위한 TimeSplit 방법을 제시한다.

GMTimeSplitCheck()의 경우 시간 축 선정을 위해 오버플로우가 발생한 노드의 GM들과 GM^-들의 두 그룹간의 중첩을 검사한다. GM과 GM^-의 최대 허용 중첩율을 $\overline{MOR}$이라 할 때, 중첩율이 $\overline{MOR}$ 이하인 경우 노드를 시간축으로 분할한다. GM^-로 분할된 노드는 더 이상 노드의 MBB가 확장되지 않으므로 이후 삽입되는 궤적에 의한 다른 노드와의 중첩을 최소화 할 수 있다. 이때, 최소한의 공간 활용도를 유지하기 위해 GM^-로 분할되는 노드의 엔트리 수는 m/2보다 커야 한다. 이 방법은 그림 11(a)와 같이 GM_3의 삽입으로 분할이 발생할 경우 GM_4와 GM^-_5로 분할하는 방법이다. 그러나 GM^-와 GM간의 중첩이 높은 경우에는 시간 축에 대한 분할보다 공간 축에 대한 분할을 고려하여야 한다.

공간 축 선정을 위한 GMSpaceSplitCheck()는 R-tree의 분할 방법과 유사하다. 그러나 R-tree와 달리 Seed선택시 거리가 가장 먼 두 개의 GM 엔트리를 선정하여 두 그룹으로 나눈다. 따라서 GM이 아닌 엔트리

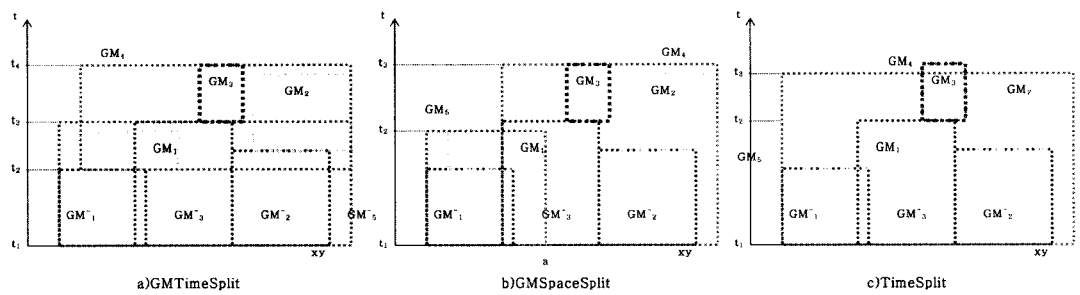


그림 11 GM에 따른 분할 방법

는 Seed가 될 수 없다. 분할된 두 그룹간의 중첩율이 $\overline{MOR}$ 이하인 경우 분할 축을 공간축으로 선정한다. 즉, 공간축 분할의 경우 시간 축으로 중첩이 높기 때문에 궤적의 삽입으로 인해 증가하는 두개의 GM을 공간적으로 분리되게 노드를 분할하는 방법으로 그림 11(b)와 같이 GM_3의 삽입으로 공간적 분할이 발생할 경우 GM을 Seed로 하여 GM_4와 GM_5로 분할하는 방법이다. 이 경우에도 중첩율은 GM_4와 GM_5의 전체 영역에 대한 교집합의 비율을 의미한다.

시간축과 공간축 선정에 의해 조건이 만족하면 해당 분할 방법으로 분할하고 두 조건을 모두 만족하지 않는 경우에는 공간 활용도를 최대화하기 위해 가장 최근 삽입된 엔트리와 나머지 엔트리들의 두 개의 그룹으로 분할하는 TimeSplit()을 이용한다. 그림 11(c)와 같이 GM_3의 삽입으로 비단말 노드의 오버플로우가 발생할 경우 새로운 엔트리로 구성되는 GM_4와 삽입전에 구성된 노드인 GM_5의 두 노드로 분할한다. 이 경우 GM_5는 공간 활용도가 100%이므로 추가적으로 삽입되는 노드와 시간축에서 정렬된 효과를 가진다. 분할 방법을 위한 알고리즘은 그림 12와 같다.

```
Algorithm SplitGM(N)
    Split the GM of N into GM' and GM"(fixed MBB) by least
     overlap enlargment

Algorithm  Split(N)
If node is a non-leafNode(N)
    Invoke SplitNon-leafNode(N)
ELSE
    Invoke SplitLeafNode(N)

Algorithm SplitLeafNode(N)
        Put new node entry into a new node

Algorithm SplitNon-leafNode(N)
        If TimeSplitCheck(N)
          Split the MBB of N into GM and GM"
        ELSE IF SpaceSplitCheck(N)
        Split the MBB of N into two GMs
        ELSE TimeSplit(N)
```

그림 12 분할 알고리즘

5. 성능 평가

이 장에서는 STR-tree, TB-tree, TR-tree의 3가지 색인에 대하여 삽입 및 검색 성능을 평가한다. 삽입의 경우 생성 시간과 인덱스의 크기를 비교하고, 검색은 타임 스탬프 질의, 영역 질의 및 복합질의에 대한 노드 접근 횟수를 비교한다. 각 색인의 비단말 노드와 단말 노드를 위한 페이지 크기는 1024byte를 사용한다. 이 페이지 크기에서 STR-tree, TB-tree의 경우 비단말과 단말 노드의 용량 $m = \overline{m}$=19, TR-tree의 경우 m=29, $\overline{m}$ =19이다. 이론적으로 STR-tree와 TB-tree의 단말 노드의 pan-out은 19개 보다 작지만, 이 실험에서는 STR-tree와 TB-tree의 단말 노드와 비단말 노드를 동일한 크기로서 성능을 비교한다.

5.1 실험 데이타셋

일반적으로 이동체 데이타는 시공간의 궤적 정보를 가지는 데이타로서 이 논문에서는 다양한 분포의 시공간 데이타셋을 생성할 수 있는 GSTD[6]를 이용하여 생성된 데이타셋을 기반으로 한다. GSTD에 의해 생성된 데이타셋의 경우 시공간상의 점으로서 이동체의 궤적 정보가 저장된다. 성능 평가에 적용하기 위해 이동체의 궤적은 시공간의 선분으로서 색인에 삽입되어야 하므로 GSTD에 의해 생성된 데이타셋에서 동일 궤적 ID를 가진 시간 순서화 된 두 점을 하나의 선분으로서 사용한다. 사용된 데이타는 표 2와 같다. 분포의 경우 LR은 이동체들이 왼쪽에서 오른쪽으로 이동하는 분포, P의 경우에는 초기 분포가 가우시안 분포에서 북동쪽으로 퍼지는 분포, UA1의 경우 초기 가우시안 분포에서 느린 속도로 전체 영역으로 퍼지는 분포, UA2의 경우 UA1과 동일하지만 빠른 속도로 이동하는 분포를 가진다.

표 2 데이타셋의 종류

분포	이동체 수	보고 회수	데이타셋명
LR	100/1000	1000	S101/S110
P	100/1000	1000	S102/S120
UA1	100/1000	1000	S103/S130
UA2	100/1000	1000	S104/S140

5.2 Overlap Rate에 따른 성능 비교

비단말 노드간의 중첩의 최소화와 궤적 보존을 위해 OR과 MCF의 두 인자를 사용하여 이를 비교 분석한다. MCF의 증가에 따라 궤적 질의는 평균 n/(m/MCF)의 노드 접근수를 가진다(n: 궤적의 엔트리 개수). 실험 결과 MCF의 증가에 따라 영역 질의 및 타임슬라이스 질의에서 성능이 선형적으로 성능 향상이 이루어지지 않고, 단지 타임슬라이스 질의의 경우 부분적인 성능 개선이 있었다. 따라서 실험은 MCF=1인 경우에 대해 비교한다. 그림 13은 최대 허용 중첩률에 따른 성능 비교로서 단말 노드 MBB간의 중첩률(OR)은 누적 중첩률로서 1보다 클 수 있지만 비단말 노드간의 최대 허용 중첩률($\overline{MOR}$)은 두 개의 노드간의 중첩으로서 중첩률이 1이하이다.

그림 13은 단말 노드와 비단말 노드간의 중첩률의 변화에 따른 타임슬라이스 질의의 결과를 비교한 그래프이다. 중첩률을 비단말 노드의 경우 3가지와 단말 노드의 경우 5가지의 변수로서 비교하고, 타임슬라이스 질의는 특정 시간에 대한 공간의 비율로서 측정된다. 이 경

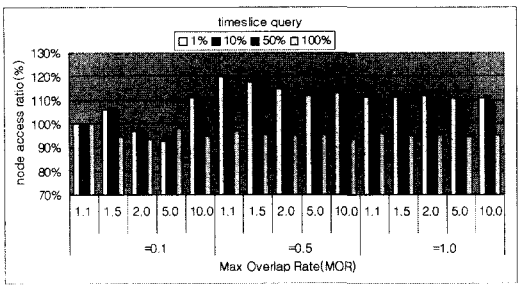


그림 13 중첩률에 따른 타임슬라이스 질의 결과

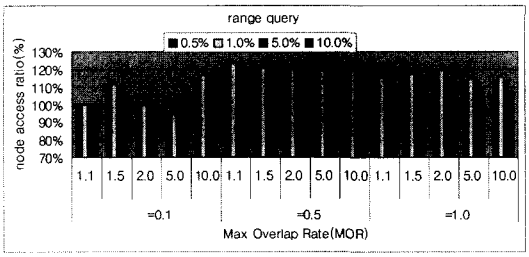


그림 14 중첩률에 따른 영역 질의 결과

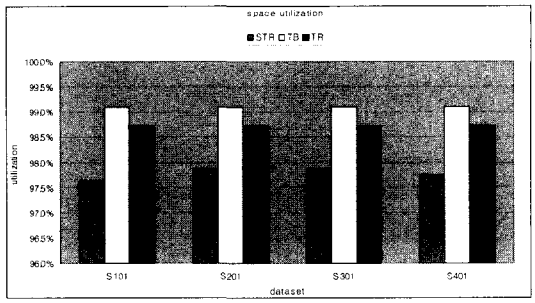


그림 15 Space Utilization

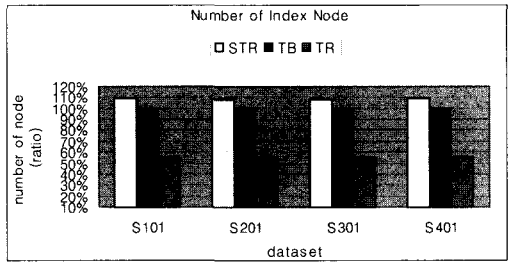


그림 16 Index Size

우 $\overline{MOR}$=1.0과 MOR=1.1을 기준으로 상대적인 노드 접근 회수를 비교한다. 평균적으로 MOR이 2와 5일 때 성능이 우수하다. 그림 14는 영역 질의의 비율로서 이 경우에는 MOR=5.0과 비단말 노드 중첩률 $\overline{MOR}$=0.1일 경우가 가장 성능이 좋다. 실험 결과 비단말 노드의 중첩이 적을수록 성능이 우수한 반면, 단말 노드의 경우 단말 노드간의 중첩이 너무 낮거나, 너무 높은 경우에는 성능이 좋지 않다. 이것은 MOR이 너무 낮을 경우 단말 노드의 분할이 빈번해지므로 비단말 노드를 증가시켜 색인의 높이를 증가시키는 원인을 제공한다. 반면 MOR이 큰 경우 공간적으로 먼 단말 노드를 하나의 노드에 저장함으로서 MBB를 크게하여 사장 공간의 증가를 초래한다. 따라서 이후의 실험은 비단말 노드의 중첩률($\overline{MOR}$)을 0.1, 단말 노드의 중첩률(MOR)을 5.0으로 실험한다.

5.3 공간 활용도 및 색인 크기

그림 15는 색인별 공간 활용도를 데이타셋별로 비교한 그래프이다. 비교되는 3가지 색인은 궤적 질의를 위한 색인으로 기존 R*-tree의 평균 공간 활용도가 67%에 비해 시간에 따라 궤적을 축척하는 구조이므로 3가지 색인 모두 높은 공간 활용도를 나타낸다. 이것은 3가지 색인이 노드의 용량을 100%로 채우면서 성장하는 구조를 가지기 때문이다.

그림 16은 색인별 생성 노드 수를 비교한 그래프이다. 색인의 노드 수는 STR-tree의 크기가 가장 크고, TB-tree, TR-tree순으로 측정된다. TR-tree가 TB-tree보다 45% 정도 크기가 작은 이유는 TB-tree의 단말 노드 구조가 MBB와 Orientation을 저장하여 단말 노드에서 T1에서의 끝점과 T2에서의 시작점을 중복하고 있는 반면, TR-tree는 점의 중복을 제거하여 단말노드에 궤적을 다중선으로 저장하는 구조를 가지고 있기 때문이다.

5.4 영역 질의

영역 질의는 X,Y의 공간 축과 T의 시간 축당 0.5, 1, 5, 10% 비율로 검색한 질의 결과와 영역에 대한 비율별 질의로서 성능을 비교한다. 축당 질의의 경우 작은 영역 질의에 해당되고 영역별 질의의 경우 큰 영역 질의에 해당한다. 그림 17은 10^6개의 궤적 정보를 가진 데이타셀에 대한 축별 영역 질의 결과이고, 그림 18은 영역별 질의 결과를 나타낸다. 질의 영역은 균등 분포로서 1000개의 임의 영역을 생성하여 이를 각 색인에 적용했을 경우 노드 접근 회수를 구한다. 그림에서는 TB-tree의 노드 접근 회수를 100%로 할 경우 각 색인별 비율을 도시한다.

STR-tree는 작은 영역 질의에서는4가지 데이타셋에서 빠른 속도로 이동하는 궤적 데이타셋에서 가장 나쁜 성능을 보인 반면 TR-tree는 TB-tree에 비해 평균 20~30% 빠른 검색 성능을 보인다. 큰 영역 질의에서는 STR-tree의 경우 왼쪽에서 오른쪽으로 이동하는 데이타셋이 다른 색인에 비해 성능이 안 좋은 반면, 질의 영

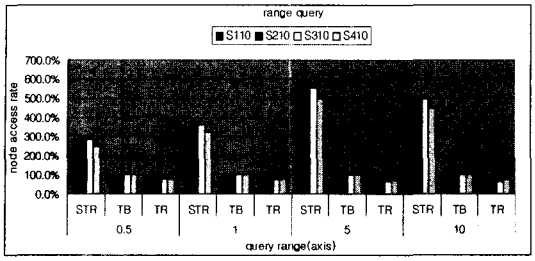


그림 17 Range Query(Axis)

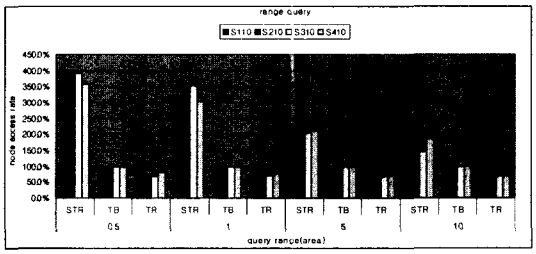


그림 18 Range Query(Area)

역이 증가함에 따라 줄어드는 성향을 나타낸다. 그러나 10% 영역 질의(축당 질의의 경우 약 25%질의)에서도 TB-tree에 비해 50% 이상의 추가 검색 비용이 필요하다. 반면, TR-tree의 경우 작은 영역 질의에서와 마찬가지로 큰 영역 질의에서도 20~30% 정도의 효율적인 성능을 나타낸다.

5.5 타임슬라이스 질의

타임슬라이스 질의의 유형은 임의의 시간 t에 대해 영역이 1%, 10%, 50%, 100%인 경우를 비교한다. 그림 19는 데이타셋별 타임슬라이스 질의에 대한 결과로서 TB-tree를 기준으로 한다. 타임슬라이스 질의에서 특이한 결과로서 100%영역에 대한 질의에서 STR-tree의 성능이 최고 좋은 것으로 나타난다. 이 이유는 TB-tree와 TR-tree의 경우 시간 간격 단위로 색인을 구성하는 구조인 반면 STR-tree는 공간적 지역성에 초점을 맞춘 색인이므로 특정 시간대의 공간 도메인을 모두 포함하는 질의의 경우 TB-tree, TR-tree에서 이동체 개수 n만큼의 단말 노드의 접근이 필요한 반면, STR-tree는 지역적으로 그룹화된 상태이므로 최악의 경우 n개의 단

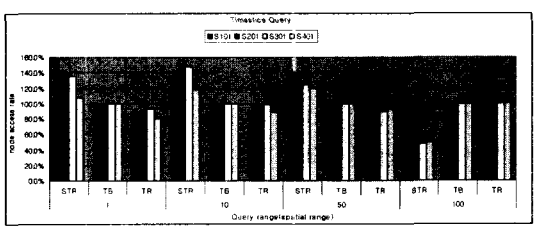


그림 19 Time slice Query

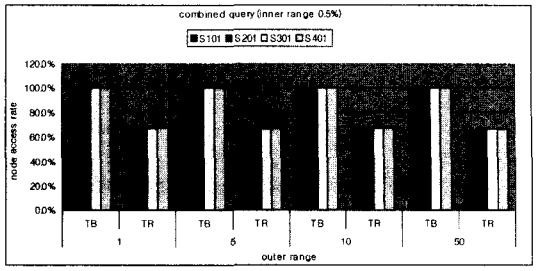


그림 20 Combined Query(inner range 0.5%)

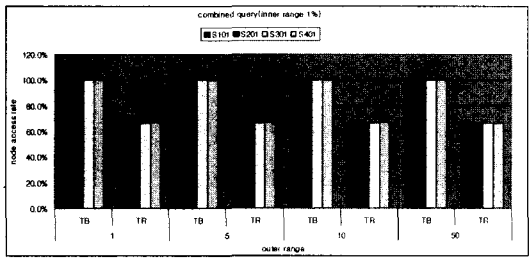


그림 21 Combined Query(inner range 1.0%)

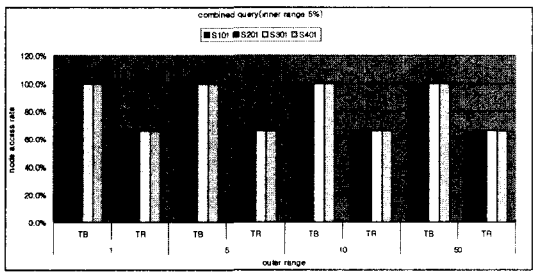


그림 22 Combined Query(inner range 5.0%)

말 노드 접근이 필요하기 때문이다.

5.6 복합 질의

복합 질의는 내부 영역(inner range)에 포함되는 궤적의 외부 영역(outer range)에서의 궤적 정보를 획득하는 질의로서 실험에서는 0.5%, 1%, 5%의 내부 영역과 1%, 5%, 10%, 50%의 외부 영역에 대한 성능을 평가한다. STR-tree의 경우 궤적을 위한 색인이지만 TB-tree에 비해 영역이 클수록 최소 200%이상의 비효율적인 성능을 가지기 때문에 실험 결과는 TB-tree와 TR-tree의 두 색인에 대한 상대적인 노드 접근 회수를 비교한다. 실험 결과 영역 질의에서와 마찬가지로 TR-tree가 평균 30% 정도의 성능 향상의 효과를 가진다. 즉, 내부 영역을 위한 질의에서 시간 순서로 된 TB-tree와는 달리 시간에 대한 공간 지역성을 고려한 분할 전략으로 성능을 향상시키고, 외부 영역을 위한 구조에서 TB- tree와 TR-tree는 링크를 통한 궤적의 정보 유지하는 점에서 동일하지만, 단말 노드의 용량을 향

상 시킴으로서 TB-tree보다 적은 노드 접근을 통해 복합 질의를 수행할 수 있는 장점을 가진다.

6. 결론 및 향후 연구

이 논문에서는 궤적 보존, 사장 영역 축소 및 비단말 노드간의 중첩 최소화를 위해 단말 노드의 MBB를 분할하는 TR-tree의 색인 구조를 제시하고, 제시된 색인에서의 삽입 및 분할 알고리즘을 제시하였다. 특히, 분할 알고리즘은 삽입에 의해 MBB의 증가가 예상되는 GM을 기반으로 GM시간 분할, GM공간 분할 및 시간 분할 방법으로 분류하였다. 또한 STR-tree, TB-tree및 TR-tree를 구현하여 성능을 평가하였다. TR-tree는 기존의 궤적 기반 색인보다 사장 영역 및 비단말 노드의 중첩을 줄이고, 단말 노드의 용량을 증가시킴으로서 영역 질의, 복합 질의에서 20~30%정도의 성능 향상이 있음을 실험을 통하여 검증하였다.

궤적 색인의 경우 기존의 공간 색인 및 시공간 색인에 비해 대용량의 데이타를 처리하기 때문에 높은 검색 비용을 가지므로 LBS와 같은 시스템에 직접 적용이 어려운 문제가 있다. 또한 많은 궤적 질의가 올 경우 재접근이 되는 노드의 수가 많으므로 색인의 캐쉬를 적용할 경우 삽입과 검색시 적중율에 대한 연구 및 FindNode의 높은 삽입 비용의 절감과 객체별 궤적 검색 성능 향상을 위해 객체 색인을 추가한 구조 및 대용량의 색인을 관리하기 위한 기법에 대한 추가 연구가 필요하다.

참 고 문 헌

[1] Martin Erwig, Markus Schneider: Spatio-Temporal Predicates. TKDE 14(4): 881-901: 2002.

[2] Dieter Pfoser, Christian S. Jensen, Yannis Theodoridis: Novel Approaches in Query Processing for Moving Object Trectories. VLDB 2000: 395-406.

[3] Volker Gaede, Oliver Günther, Multidimensional access methods, ACM Computing Surveys (CSUR) June 1998.

[4] Mario A. Nascimento, Jefferson R. O. Silva, Towards historical R-trees, Proceedings of the 1998 ACM symposium on Applied Computing February 1998.

[5] Theoderidis, Y.; Vazirgiannis, M.; Sellis, T.Spatio-temporal indexing for large multimedia applications, Multimedia Computing and Systems, 1996. Proceedings of the Third IEEE International Conference on, 1996 Page(s): 441-448.

[6] Y. Tao and D. Papadias, "MV3R-Tree: A Spatio-Temporal Access Method for Timestamp and Interval Queries," Proceedings of International Conference on VLDB, pp.431-440, 2001.

[7] Theodoridis, Y., Silva, R., and Nascimento, M.: On the Generation of Spatiotemporal Datasets. In Proc. of the 6th Int'l Symposium on Spatial Databases, pp.147-164, 1999.

[8] A. Guttman, R-trees: A dynamic index structure for spatial searching, ACM SIGMOD Conference, pp.47-54, 1984.

[9] N. Beckmann and H. P. Kriegel, The R*-tree: An Efficient and Robust Access Method for Points and Rectangles, In Proc. ACM SIGMOD, pp.332-331, 1990.

[10] J. Tayeb, O. Ulusoy, and O. Wolfson. A Quadtree Based Dynamic Attribute Indexing Method, The Computer Journal, 41(3), pp.185-200, 1998.

[11] V. Prasad Chakka, Adam Everspaugh, and Jignesh M. Patel, Indexing Large Trajectory Data Sets with SETI, CIDR 2003.

[12] N. Beckmann and H. P. Kriegel, The R*-tree: An Efficient and Robust Access Method for Points and Rectangles, In Proc. ACM SIGMOD, pp.332-331, 1990.

[13] 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, pp.28-39, 1996.

[14] T. K. Sellis, N. Roussopoulos and C. Faloutsos, The R+-Tree: A Dynamic Index for Multi-Dimensional Objects, Proceedings of the 13th VLDB Conference, pp.507-518, 1987.

임 덕 성

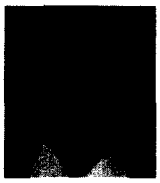
1998년 동아대학교 컴퓨터공학과 졸업(학사). 2000년 부산대학교 컴퓨터공학과 졸업(공학석사). 2002년 부산대학교 컴퓨터공학과 박사수료. 2003년~현재 영진전문대학 컴퓨터정보기술계열 전임강사. 관심분야는 지리정보시스템(GIS), 이동체 데이타베이스, 이동체 색인

전 봉 기

1991년 부산대학교 컴퓨터공학과(학사) 1993년 부산대학교 컴퓨터공학과(공학석사). 1993년~1998년 한국통신 연구소 전임연구원. 2003년 부산대학교 컴퓨터공학과(공학박사). 2003년~현재 신라대학교 컴퓨터정보공학부 전임강사. 관심분야는 데이타베이스, 공간 데이타베이스, 이동체 데이타베이스

홍 봉 희

1982년 서울대학교 전자계산기공학과 졸업(학사). 1984년 서울대학교 전자계산기공학과 졸업(석사). 1988년 서울대학교 전자계산기공학과 졸업(박사). 1987년~현재 부산대학교 공과대학 컴퓨터공학과 교수/부산대학교 컴퓨터 및 정보통신연구소 책임연구원. 관심분야는 이동체 데이타베이스, 모바일 데이타베이스, 공간 데이타베이스



조 대 수

1995년 부산대학교 컴퓨터공학과(학사) 1997년 부산대학교 컴퓨터공학과(공학석사). 2001년 부산대학교 컴퓨터공학과(공학박사). 2001년~현재 한국전자통신연구원 LBS 연구팀 선임연구원. 관심분야는 GIS, 공간DB, LBS, 이동체DB