# 공간 네트워크상의 이동객체를 위한 궤적기반 색인구조의 설계 및 구현

## (Design and Implementation of a Trajectory-based Index Structure for Moving Objects on a Spatial Network)

엄 정 호 † 장 재 우 ††
(Jung-Ho Um) (Jae-Woo Chang)

**요 약** 대부분의 이동객체들은 공간 네트워크상을 움직이기 때문에, 그들의 궤적을 효과적으로 색인 검색할 수 있는 궤적 기반 색인 구조가 필요하다. 하지만 도로와 같은 공간 네트워크상의 궤적 기반 색인 구조에 대한 연구는 FNR-트리나 MON-트리와 같은 연구가 진행되었을 뿐 연구가 많이 진행되어 있지 않다. 하지만, FNR-트리나 MON-트리 또한 이동객체의 세그먼트만을 저장할 뿐 전체 궤적을 유지하지 못하여, 궤적 질의에 대해 비효율적이다. 따라서 본 논문에서는 공간 네트워크상의 이동객체를 위한 궤적 기반 색인 구조인 TMN -Tree(Trajectory of Moving objects on Network-Tree)를 제안한다. 이를 위해, 이동객체를 공간과 시간 특성으로 분류하고, 궤적을 유지함으로써 영역질의와 궤적질의를 동시에 처리할 수 있는 색인 구조를 설계한다. 아울러, 사용자 질의를 시공간영역 내 궤적 질의, 시간영역 내 유사궤적 질의, k-최근접 질의로 분류하고, 이들을 처리하기 위한 질의 처리 알고리즘을 제안한다. 마지막으로 본 논문에서 제안한 궤적 기반 색인 구조가 기존의 색인구조인 FNR-Tree, MON-Tree보다 성능이 향상되었음을 보여준다.

**키워드** : 공간 네트워크데이타베이스, 궤적, 색인구조

***Abstract*** Because moving objects usually move on spatial networks, efficient trajectory index structures are required to achieve good retrieval performance on their trajectories. However, there has been little research on trajectory index structures for spatial networks such as FNR-tree and MON-tree. But, because FNR-tree and MON-tree are stored by the unit of the moving object's segment, they can't support the whole moving objects' trajectory. In this paper, we propose an efficient trajectory index structure, named Trajectory of Moving objects on Network Tree(TMN-Tree), for moving objects. For this, we divide moving object data into spatial and temporal attribute, and preserve moving objects' trajectory. Then, we design index structure which supports not only range query but trajectory query. In addition, we divide user queries into spatio-temporal area based trajectory query, similar-trajectory query, and k-nearest neighbor query. We propose query processing algorithms to support them. Finally, we show that our trajectory index structure outperforms existing tree structures like FNR-Tree and MON-Tree.

**Key words** : Spatial Network database, trajectory, Index Structure

· 본 과제(결과물)는 교육인적자원부, 산업자원부, 노동부의 출연금으로 수행한 최우수실험실지원사업의 연구결과입니다.

† 학생회원 : 전북대학교 컴퓨터공학과
jhum@dblab.chonbuk.ac.kr
†† 종신회원 : 전북대학교 컴퓨터공학과 교수
jwchang@chonbuk.ac.kr
(Corresponding author임)

논문접수 : 2006년 5월 2일
심사완료 : 2008년 1월 3일

Copyright@2008 한국정보과학회 : 개인 목적이나 교육 목적인 경우, 이 저작물의 전체 또는 일부에 대한 복사본 혹은 디지털 사본의 제작을 허가합니다. 이때, 사본은 상업적 수단으로 사용할 수 없으며 첫 페이지에 본 문구와 출처를 반드시 명시해야 합니다. 이 외의 목적으로 복제, 배포, 출판, 전송 등 모든 유형의 사용행위를 하는 경우에 대하여는 사전에 허가를 얻고 비용을 지불해야 합니다.

정보과학회논문지: 데이타베이스 제35권 제2호(2008.4)

## 1. 서 론

기존의 공간 데이타베이스에서 이동객체를 위한 색인 구조 연구가 많이 진행되어 왔다[1-3]. 대표적인 연구로는 3DR-Tree[1], TB-Tree[2]가 있다. 3DR-Tree는 2DR-Tree[4]의 시간 축을 확장하여 3차원의 공간상에서 움직이는 이동객체를 표현 하였다. TB-Tree는 3차원의 공간상에서 움직이는 이동객체를 MBR 대신 하나의 라인 세그먼트 형식으로 표현하여 저장하였다. 여기서, 3DR-Tree와 TB-Tree의 연구는 유클리디언(Euclidean) 공간상의 이동객체를 색인하는 공통점을 가지고 있다. 그러나 LBS (Location- Based Service) 및 텔레매틱스와 같은 실제 응용 서비스에서는, 도로, 철도와 같이 유클리디언 공간보다는 네트워크 구조를 가지는 공간 네트워크상에서 이동객체를 색인해야 한다. 따라서 이를 고려한 이동객체를 색인할 수 있는 구조가 근래에 연구되고 있다[5-9]. 이 중에서 대표적인 색인구조로 아테네 국립대학에서 연구한 FNR-Tree[7]와 독일 하겐 대학에서 연구한 MON-Tree[9]가 있다.

유클리디언 공간상을 움직이는 이동객체는 객체가 시간에 따라 이동 할 때 공간의 제약이 없다. 반면에 공간 네트워크상에서 움직이는 이동객체는 공간의 제약을 가진다. 이로 인하여, 유클리디언 공간을 움직이는 이동객체를 색인하는 구조는 공간 네트워크상을 움직이는 이동객체를 색인하는 경우 공간상의 중복이 많이 발생하는 문제점을 지닌다. 따라서 공간 네트워크상에 움직이는 이동객체는 공간 제약 특성을 고려하여 공간과 시간을 분리, 색인해야 한다. 공간 데이타는 데이타를 한 번 삽입하면 거의 삽입이 이루어지지 않는 특성을 가지고 있다. 이와 달리 시간데이타의 경우 시간에 따라 데이타가 축적되는 특성이 있기 때문에 공간 데이타에 비해 삽입이 빈번히 일어난다. 따라서 본 논문에서 제안한 색인 구조의 특징은 다음과 같다. 첫째, 공간 제약 특성을 고려하여, 공간과 시간을 분리, 저장한다. 둘째, 이동객체의 궤적을 유지함과 아울러 삽입된 시간을 기준으로 시간 데이타에 대하여 색인한다. 마지막으로, k-최근접 질의와 같은 다양한 공간 질의를 가능하게 하기 위해 공간 네트워크 구조를 유지한다. 본 논문에서는 앞에서 언급한 특성을 고려하여 이동객체 궤적을 위한 색인구조를 설계한다. 아울러, 실세계의 LBS나 텔레매틱스 응용에 적용할 수 있는 질의 처리 알고리즘을 설계한다. 그리고 이를 구현하여 기존의 색인구조인 FNR-Tree[7], MON-Tree[9]와 본 논문에서 제안하는 궤적기반 색인구조에 대해 성능평가를 수행한다.

본 논문의 구성은 다음과 같다. 2장에서는 공간 네트워크상을 움직이는 이동객체를 위한 색인구조에 대한 관련 연구와 공간 네트워크상에서의 질의 처리 알고리즘에 대한 관련 연구를 알아본다. 3장에서는 궤적기반 색인구조를 제안하고, 4장에서는 질의 처리 알고리즘을 제안한다. 아울러, 5장에서는 궤적기반 색인구조의 구현 및 성능평가를 수행한다. 마지막으로 6장에서는 결론 및 향후연구를 제시한다.

## 2. 관련 연구

공간 네트워크 데이타베이스(SNDB: Spatial Network DataBase)에 대한 연구는 크게 공간 네트워크상의 이동객체를 저장/색인하기 위한 연구[7,9]와 공간 네트워크 자체를 저장/색인하고 공간 네트워크상에서의 질의처리를 하기 위한 연구[10]가 진행되어 왔다. 먼저, FNR-Tree는 2003년 E. Frentzos에 의해 연구되었다[7]. FNR-Tree는 고정된 공간 네트워크를 2DR-Tree로 색인한다. 따라서 2DR-Tree의 리프노드에는 네트워크를 이루는 단위인 에지가 삽입된다. 이동객체에 대해서는 이동객체가 지나간 에지별로 1DR-Tree를 생성한다. 이는 이동객체의 궤적에 대해 공간 데이타와 시간 데이타를 분리했기 때문이다. FNR-Tree는 3DR-Tree보다 영역질의 측면에서 더 향상된 성능을 보이고 있다. 하지만 FNR-Tree는 삽입이나 질의 처리 시 2DR-Tree를 검색하고, 다시 이동객체의 궤적을 색인하는 1DR-Tree를 검색해야 하는 오버헤드가 있다. 둘째, MON-Tree는 2005년 *Ralf Hartmut Güting et al.*에 의해 연구되었다[9]. MON-Tree는 FNR-Tree에서 삽입이나 질의 처리 시 2DR-Tree와 1DR-Tree 두 개의 트리를 탐색해야 하는 오버헤드를 보완하고, 아울러 공간 네트워크에 대한 새로운 모델을 적용하여 공간 네트워크상에서 움직이는 이동객체를 색인하고자 하였다. MON-Tree의 전체 구조는 그림 1과 같다.

MON-Tree의 기본 구조는 FNR-Tree와 유사하다. 먼저 top R-Tree는 FNR-Tree의 2DR-Tree처럼 공간 네트워크를 저장한다. FNR-Tree와 다른 점은 리프노드가 에지가 될 수도 있지만 MON-Tree에서 제안하는 공간 네트워크 모델인 경로(Route)가 될 수 있다는 점이다. bottom R-Tree는 에지나 경로를 지나는 이동객체를 색인하는 R-Tree이다. FNR-Tree와 같이 1DR-Tree로 색인한다. 또한 MON-Tree는 해시 테이블 구조를 유지하여, 이동객체 궤적의 삽입 시 해시 테이블을 통해 삽입한다. MON-Tree는 해시 테이블을 이용함으로써 FNR-Tree보다 트리 접근 횟수를 줄여, 삽입성능을 향상시켰다. 마지막으로, MON- Tree는 경로 모델을 제안하였다. 경로 모델은 예를 들어 고속도로의 번호와 같이 실제적으로 도로가 분류되는 기준을 사용하여 이를 공간 네트워크의 기본단위로 사용하는 것이다. 셋

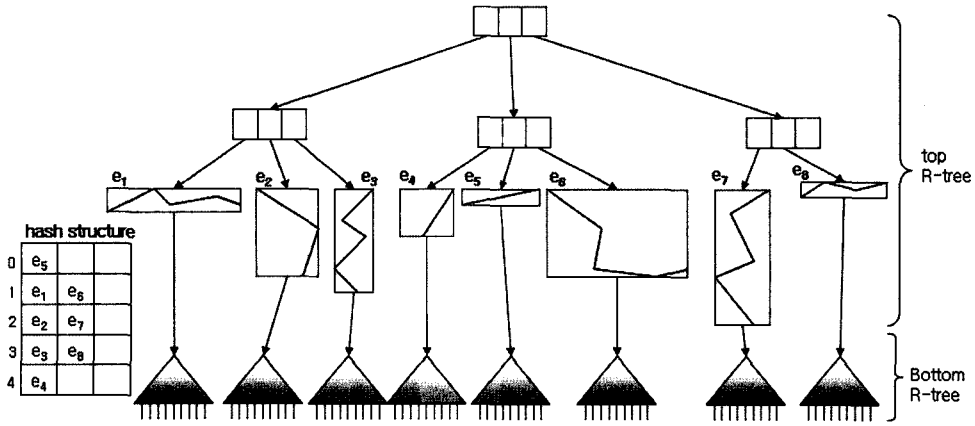


그림 1 MON-Tree의 전체 구조

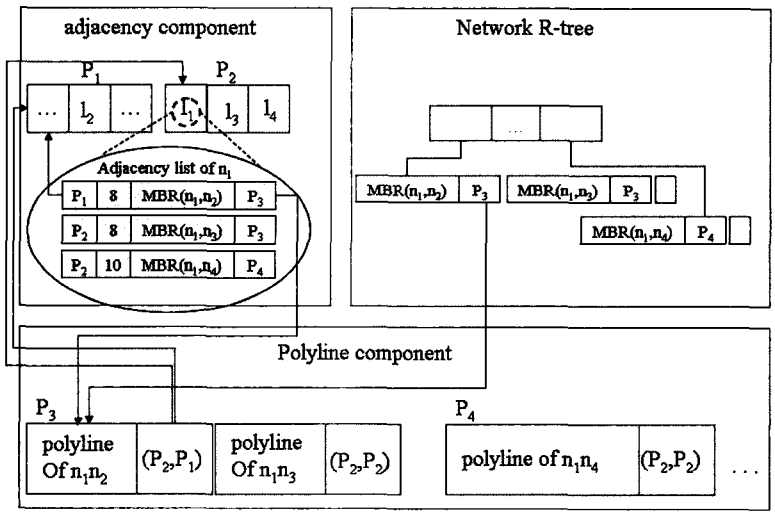


그림 2 공간 네트워크 저장/색인 전체 구조

제, D. Papadias et al.은 공간 네트워크 자체 데이타를 위한 저장/색인 구조와 공간 네트워크상에서 POI(Point of Interest)를 찾는 질의 처리 알고리즘에 대하여 연구를 진행시켜 왔다[10]. 이 연구에서 공간 네트워크를 저장/색인하기 위한 구조는 그림 2와 같다.

저장/색인 구조는 노드간의 연결에 따른 클러스터를 형성하여 노드들의 인접리스트로 저장한다. 단, 지역적 클러스터링 효과를 높이기 위해 힐버트 오더를 사용하여 노드 식별자를 부여한다. 이 저장/색인 구조는 크게 3개의 컴포넌트들로 구성되어 있다. 먼저 Adjacency component는 공간상에서 근접한 노드들의 리스트 정보를 가지고 있는 인접 리스트를 포함하고 있다. 다음으로 Poly-line component는 각 에지들의 poly-line정보를 상세히 표현하고, end point의 인접 리스트를 포함하는 디스크 페이지들에 대한 포인터 쌍을 포함하고 있다. Network R-tree는 poly-lines의 MBR들의 집합으로 네트워크상에서 영역 질의를 지원한다. 각 리프노드는 디스크 페이지에 대한 포인터를 가지고 있다. 마지막으로 엔티티 즉 POI를 위한 R-Tree를 사용한다. 이 연구에서는 앞에서 설명한 저장 구조를 가지고 노드간의 연결리스트를 사용한 네트워크 확장방법으로 영역질의(RNE(Range Network Expansion) 기법), k개의 POI를 찾는 k-최근접 질의(INE(Incremental Network Expansion)등을 제안하였다. 마지막으로 Cyrus Shahabi et al은 2004년에 공간 네트워크상에서 k-최근접 질의 처리 알고리즘을 제안하였다[11]. 이 연구에서는 네트워크 보로노이 다이어그램을(Voronoi Diagram)사용하여 POI의 서비스 지역 즉 POI가 최근접점이 될 수 있는 거리를 미리 계산하여 저장한다. 이 논문에서는 성능평가를 통하여 기존의 INE 방법보다 k-최근접 질의의 성

능이 향상되었음을 보여주고 있다.

## 3. 궤적 기반 색인 구조

### 3.1 연구 동기

FNR-트리나 MON-트리와 같은 공간 네트워크상의 이동객체를 위한 색인 구조는 크게 두 가지의 문제점을 갖는다. 첫째, 이동객체를 공간 네트워크의 세그먼트 단위로 저장을 하기 때문에 이동객체의 전체궤적을 유지하지 못한다. 이로 인하여 이동객체의 궤적 질의 처리에 있어 취약점을 가지고 있다. 둘째, 공간 네트워크를 R-트리로 저장하고 있지만, 영역 질의만 가능할 뿐 k-최근접 질의와 같은 대표적인 질의를 처리 하지 못한다. 따라서 공간 네트워크상의 이동객체에 대한 질의를 정의하고, 이를 모두 처리할 수 있는 색인 구조를 설계하는 것이 필요하다. 표 1은 본 논문에서 정의한 이동객체 질의를 분류한 표이다.

표 1과 같이 분류한 이유는 다음과 같다. 기존의 공간 네트워크 데이타베이스 분야에서 대표적인 질의는 시공간영역질의와 k-최근접 질의이다. 그리고 기존의 이동객체에 대한 대표적인 질의는 이동객체 궤적을 찾는 질의이다. 이 두 종류의 질의를 결합하면, 첫째, 영역을 검색하고 이동객체 궤적을 찾는 질의, 둘째, 궤적을 검색하고 특정한 시간영역 안에 있는 이동객체를 찾는 질의, 셋째, 이동객체가 계속 이동하면서 k-최근접 질의를 수행하는 질의로 분류할 수 있다. 분류한 질의 중 연속적 k-최근접 질의를 처리하기 위해서는 k-최근접 질의가 선행되어야 하기 때문에, 본 논문에서는 k-최근접 질의 처리를 목표로 하고, 연속적 k-최근접 질의는 향후 연구로 남긴다. 시공간영역 내 궤적질의와 시간영역 내 유사궤적질의를 효율적으로 처리하기 위해서, 공간 네트워크상의 이동객체를 위한 색인구조는 이동객체의 궤적을 유지해야 한다. 하지만 기존 공간 네트워크상의 이동객체를 처리하는 저장/색인 구조[7,9]는 이동객체의 궤적을 유지하고 있지 못하기 때문에, 시공간영역 내 궤적질의와 시간영역 내 유사궤적질의 처리를 효율적으로 할 수 없다. 아울러, k-최근접 질의는 MON-Tree[9]에서 향후 과제로 남겨두어 공간 네트워크 데이타베이스분야에서 k-최근접 질의를 지원하는 색인 구조는 연구 초기 단계에 있다. 즉, 표 1에서 분류한 질의를 모두 효율적으로 처리할 수 있는 색인 구조는 아직 연구된 적이 없다. 따라서 본 논문에서는 이동객체의 궤적과 네트워크 정보를 유지함으로써 표 1에서 분류한 질의를 효율적으로 처리할 수 있는 색인 구조인 TMN-tree(Trajectory of Moving objects on Network-Tree)를 제안한다.

### 3.2 TMN-Tree의 전체구조

본 논문에서 제시하는 TMN-Tree의 전체 구조는 공간 네트워크를 저장하기 위한 2DR*- Tree[12], 공간 네트워크를 움직이는 이동객체를 위한 B+-Tree, 공간 네트워크 정보를 유지하는 에지 인접리스트 테이블과 궤적을 유지하는 궤적레코드로 구성되어 있다(그림 3). TMN-tree는 다음과 같은 특징을 지닌다. 첫째, TMN-tree는 기존 MON-tree와 FNR-tree에서 처리하지 못하였던 k-최근접 질의를 처리할 수 있다. 이를 위해서 에지 연결리스트 테이블을 네트워크 구조에 추가한다. 이는 기존의 MON-tree의 에지 테이블과 달리, 에지 테이블에 에지 인접 리스트

정보를 유지하고 있어 이를 기반으로 네트워크 확장을 통해 k-최근접 질의를 처리 할 수 있기 때문이다. 둘째, TMN-tree는 MON-tree와 FNR-tree에 비해 궤적 질의를 보다 효율적으로 처리한다. 이는 MON-tree와 FNR-tree의 경우 이동객체 궤적을 세그먼트 단위로 분리, 저장하여 궤적을 유지할 수 없지만, TMN-tree는 Temporal B+-tree가 이동객체의 시간만을 색인하고, 이동객체 궤적 레코드와 연결되어 궤적을 유지할 수 있기 때문이다. 셋째, TMN-tree는 MON-tree와 FNR-tree에 비해 이동객체의 궤적에 대한 시공간 영역 질의를 보다 효율적으로 처리한다. 이를 위해 이동객체의 궤적을 타임스탬프 단위로 저장하고 이를 Temporal B+-tree로 색인한다. 이는 MON-tree나 FNR-tree처럼 1DR-tree로 시간을 색인하는 것보다 효율적이기 때문이다. 1DR-tree에 비해 B+-트리로 색인하여 얻을 수 있는 장점은 다음과 같다. 첫째, 1DR-tree가 시간 영역 질의 처리를 위해 재귀 호출하는 반면 B+-tree는 순차적으로 검색하므로 질의 처리 비용이 감소된다. 둘째, 1DR-Tree의 키는 라인 세그먼트인 반면 B+-Tree의 키는 하나의 점과 같기 때문에 색인구조의 크기가 작아진다.

3.2.1 Spatial 2DR*-Tree

공간 네트워크를 저장하기 위해 2DR*-Tree를 사용한다. 2DR*-Tree에 공간 네트워크의 삽입은 이동객체가 삽입되기 전에 미리 삽입이 되어 있는 것을 가정한

표 1 질의 타입의 분류

| 질의 타입 | 설 명 |
|---|---|
| 시공간영역 내 궤적질의 | 어떤 이동객체가 주어진 시공간 영역을 지나면서 어떤 궤적을 남겼는지를 찾는 질의 |
| 시간영역 내 유사궤적질의 | 어떤 이동객체가 주어진 시간 영역에서 주어진 궤적과 유사한 궤적을 지났는지를 찾는 질의 |
| 연속적 k-최근접 질의 | 이동객체가 움직이면서 자신과 가까운 k개의 POI를 찾는 질의 |

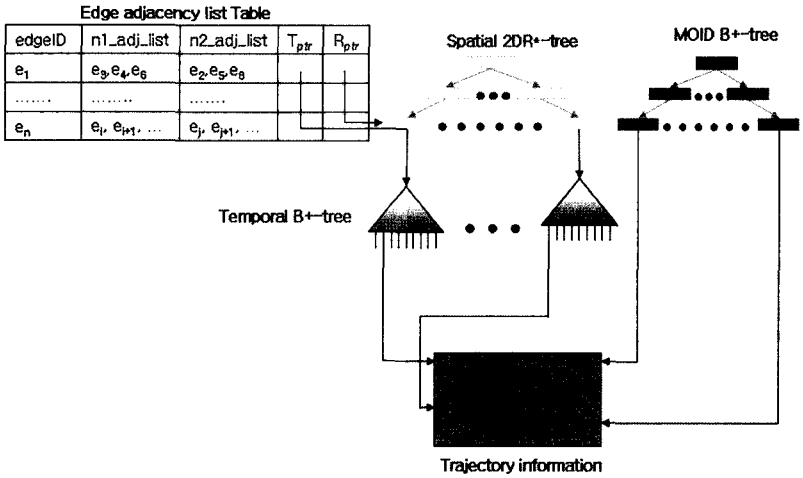


그림 3 TMN-Tree의 전체구조

다. 2DR*-Tree의 비단말 노드의 엔트리 구조는 <MBR, childptr> 와 같다. MBR은 현재 노드의 자식 노드가 가지고 있는 영역을 의미한다. childptr은 현재 노드의 자식 노드를 연결한다. 2DR*-Tree의 단말노드는 <MBR, Tptr, Edge_info>와 같다. MBR은 공간 네트워크상의 에지를 포함하는 영역을 의미하고, Tptr은 만일 이동객체가 이 에지를 지났을 경우 이동객체를 위한 B+-Tree가 생성되는데, 이 때 이 트리를 연결하는 포인터가 Tptr이다. Edge_info는 에지 정보를 담고 있는 레코드이며 Edge_info는 <{point1, point2, ...}, poi_info>과 같이 구성되어 있다. point의 집합은 에지를 보간 한 점들의 집합이다. poi_info는 에지 상에 있는 POI의 정보를 가지고 있으며, poi_info는 <poi_id, poi_position, description>으로 구성되어 있다. poi_id는 POI의 아이디를 의미하며, poi_position은 에지상의 POI의 위치 정보를 나타낸다. description은 POI의 상세 설명 정보를 가지고 있다.

3.2.2 Temporal B+-Tree

Temporal B+-Tree는 이동객체의 궤적이 삽입되면 이동객체가 이동한 시간을 키로 하여 어떤 시간에 어떤 이동객체가 움직였는가를 색인한다. Tmporal B+-Tree의 리프노드 구조는 <time, offset, trajectory_ ptr>와 같다. time은 이동객체가 움직인 시간을 나타내며 Temporal B+-Tree에서 키로 쓰인다. time을 키로 사용할 경우 키가 중복될 수 있기 때문에 Temporal B+-Tree에서는 키의 중복을 허용한다. offset은 궤적 레코드 상에서 몇 번째 엔트리에 삽입이 되어 있는지를 나타내는 값이다. trajectory_ptr은 이동객체의 궤적을 저장하는 궤적 레코드를 연결하는 포인터이다.

3.2.3 에지 연결리스트 테이블

에지 연결리스트 테이블은 MON-Tree의 해시 테이블과 같은 역할을 한다. 이동객체의 궤적이 삽입이 될 때 이동객체가 어떤 Temporal B+-Tree에 삽입이 되어야 할지 바로 찾을 수 있도록 Temporal B+-Tree의 포인터를 가지고 있다. 그리고 Spatial R*-Tree의 리프노드와 연결되는 포인터를 가지고 있어서 테이블에서 에지 정보를 얻어올 수가 있다. 또한 k-최근접 질의를 제공하기 위하여 에지의 연결리스트를 테이블에서 유지한다. 에지 테이블을 배열 형태로 사용하기 위해 노드 단위로 노드에 연결된 에지 리스트를 만들고 에지 연결리스트 테이블에서 이를 포인터로 연결하여 에지 연결리스트를 유지한다. 에지 연결 리스트 테이블을 배열 형태로 유지하는 이유는 MON-Tree의 해시 테이블과 같이 한 번의 접근을 통하여 바로 Temporal B+-Tree를 접근하기 위함이다. 아울러 빠른 접근을 위해 에지 연결리스트 테이블은 메모리에서 유지한다. 에지 연결리스트 테이블은 <edge_id, n1_adj_list, n2_adj_list, Tptr, Rptr>로 구성된다. edge_id는 에지의 아이디이며 n1_adj_list와 n2_adj_list는 에지를 이루는 두 개의 노드와 연결된 에지의 연결리스트를 연결하는 포인터이다. Tptr은 Temporal B+-Tree를 연결하는 포인터이며, Rptr은 spatial 2DR*-tree의 리프 노드를 연결하는 포인터이다. 노드의 에지 연결 리스트 구조는 <node_id, {e1, e2, e3, ...}> 로 구성되어 있다. node_id는 노드의 아이디이고, {e1, e2, e3, ...}는 노드와 연결된 에지가 에지 연결 리스트 테이블 상에서 어떤 위치에 있는지를 나타낸다.

3.2.4 이동객체의 궤적 레코드

이동객체의 궤적 레코드는 이동객체의 궤적을 유지하며 이동객체의 실제 궤적 정보를 저장한다. Temporal B+-Tree와 MOID B+-Tree가 이 궤적 레코드와 연결

되어 있으며 이동객체에 대한 질의 처리는 궤적 레코드와 연결된 트리를 통해서 수행한다. 이동객체의 궤적 레코드는 삽입되는 이동객체의 궤적 세그먼트들을 일정한 수(N)만큼 가지고 있으며 그 수를 넘어가면 새로운 궤적 레코드를 만들고 이동객체의 궤적을 저장한다. 이동객체의 궤적 레코드는 <MOID, traj_num, next_ptr, prev_ptr, {segment1, segment2, ...} >와 같이 구성되어 있다. MOID는 이동객체의 아이디를 나타낸다. traj_num은 현재 이동객체의 궤적 레코드가 가지고 있는 궤적 세그먼트의 개수를 의미한다. next_ptr과 prev_ptr은 같은 이동객체의 아이디를 가지는 궤적 레코드를 서로 연결하는 포인터이다. next_ptr은 이동객체의 현재 레코드의 궤적보다 미래의 시간을 prev_ptr은 과거의 시간을 갖는 궤적이 저장되어 있다. 이러한 포인터를 사용하는 이유는 같은 이동객체의 아이디를 갖는 궤적을 전체적으로 유지하기 위해서이다. {segment1, segment2, ...}는 이동객체의 궤적 세그먼트 집합을 나타낸다. 세그먼트는 <t1, e1, x, y, direction>와 같이 구성되어 있다. t1은 이동객체 세그먼트의 시간을 나타내고 e1은 이동객체가 지나간 에지 아이디를 나타내며, x, y는 이동객체의 좌표를 의미한다. 이동객체가 지난 에지 아이디와 이동객체의 좌표를 함께 유지하는 것은 사용자의 질의를 수행할 때, 이동객체의 에지 아이디나 좌표 중 어떤 것을 요구할지 모르기 때문이다. direction은 이동객체가 움직이고 있는 방향을 나타낸다. 이동객체의 방향을 저장하는 이유는 방향성에 따라서 이동객체의 미래 궤적을 예측할 수 있기 때문이다.

3.2.5 MOID B+-Tree

MOID B+-Tree는 이동객체의 아이디를 색인한다. MOID B+-Tree의 리프노드 구조는 <MOID, trajectory_ptr>와 같이 구성되어 있다. MOID는 이동객체의 아이디이며 trajectory_ptr은 이동객체의 궤적 레코드를 연결하는 포인터이다.

### 3.3 삽입 알고리즘

TMN-Tree의 삽입은 크게 세 가지로 나눌 수 있다. 첫째, 공간 네트워크 색인을 위해 공간 네트워크를 구성하고 있는 에지들을 삽입하는 것과, 둘째, k-최근접 질의를 위해 POI를 삽입하는 것, 그리고 마지막으로 공간 네트워크상을 움직이는 이동객체를 삽입하는 것이다. (1)절에서는 공간 네트워크의 삽입 알고리즘을 서술하며, (2)절에서는 이동객체의 궤적 삽입 알고리즘을 서술한다.

3.3.1 공간 네트워크 삽입 알고리즘

공간 네트워크 삽입은 공간 네트워크의 구성요소인 에지를 spatial 2DR*-Tree와 에지 연결 리스트 테이블에 삽입을 하는 것으로 이루어져 있다. 공간 네트워크의

```
Algorithm InsertNetwork()
1.  create Node Adjacency list Table
2.  create Edge Adjacency list Table
3.  for each node{
4.        Read the node information and set Node
5.        Insert to Node Adjacency list Table
6.  }
7.  for each edge{
8.        Read the edge information
9.        Find Node1, Node2
10.       Add Edge to Node1, Node2 Edge Adjacency List
11.   Set MBR using Node1,Node2 coordinate
12.   Insert MBR to RstarTree
13.   Set Rptr to Edge data record
14. }
End InsertNetwork
```

그림 4 공간 네트워크 삽입 알고리즘

삽입은 이동객체의 삽입 이전에 이루어지는 삽입이므로 본 논문에서는 공간 네트워크의 삽입이 전처리 작업에 의해 미리 저장/색인구조에 삽입이 되어 있는 것을 가정한다. 에지 연결 리스트 테이블을 구성하기 위해서는 앞 3.1절의 (3)에서 언급한 것과 같이 노드단위로 에지의 연결 리스트를 구성한다. 따라서 공간 네트워크의 노드와 에지를 저장/색인 구조에 삽입한다. 그림 4는 공간 네트워크 삽입 알고리즘을 나타낸다. 공간 네트워크의 삽입을 위해 먼저, 에지의 개수와 노드의 개수만큼 노드 테이블과 에지 연결 리스트 테이블을 생성한다. 노드 정보 파일에는 해당 노드의 좌표와 노드 아이디가 구성되어 있다. 노드 정보를 파일로부터 읽어서 노드 테이블에 노드를 삽입한다. 그런 다음 에지 정보를 파일로부터 읽어, 에지를 구성하는 노드를 가지고 이미 구성된 노드 중에 어떤 노드가 에지를 구성하는지를 찾는다. 노드를 찾게 되면 노드의 에지 연결리스트에 현재 에지를 추가한다. 에지를 구성하는 두 개의 노드를 노드 테이블에서 찾아 노드의 에지 인접 리스트에 삽입되는 에지를 저장한다. 그 후 두 개의 노드에 구성된 좌표정보를 읽어 MBR을 구성하고, 이를 Spatial 2DR*-Tree에 삽입한다. 마지막으로 Spatial 2DR*-Tree에 삽입을 한 후 에지를 저장한 레코드의 위치 정보를 가져와 Rptr을 설정한다.

3.3.2 이동객체의 궤적 삽입 알고리즘

먼저, 이동객체의 궤적을 삽입할 때 삽입이 될 이동객체의 궤적 세그먼트는 이동객체가 지나간 에지의 아이디를 가지고 있다고 가정한다. 이동객체의 삽입은 크게 두 가지의 작업 순서를 가진다. 먼저 MOID B+-Tree를 통해 이동객체가 삽입될 궤적 레코드를 얻어온 후 이동객체의 궤적을 색인할 적당한 Temporal B+-Tree

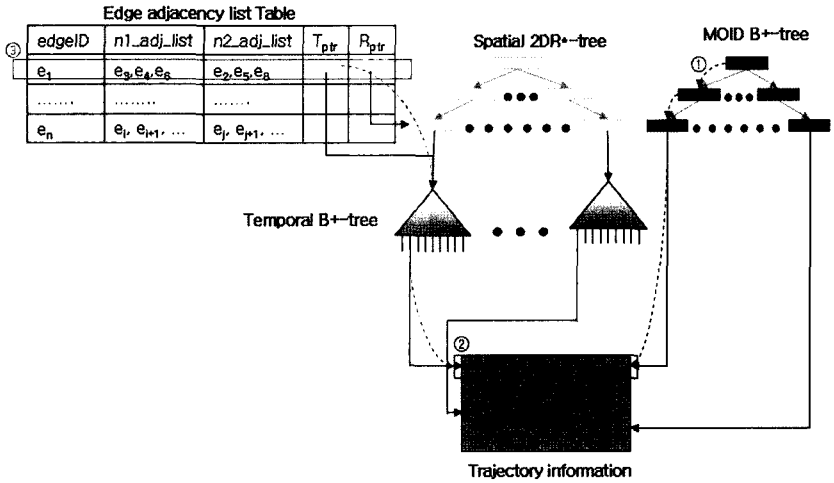


그림 5 이동객체의 궤적 삽입 알고리즘

를 찾아 궤적 레코드와 연결하는 것이다. 그림 5는 이동객체의 궤적 삽입 알고리즘의 과정을 보여준다. 이동객체의 삽입의 자세한 과정은 다음과 같다. 먼저, 그림 5의 ①과 같이 이동객체의 식별자로 MOID B+-Tree를 탐색한다. 만일 이동객체의 식별자가 발견되면 그림 5의 ②와 같이 MOID B+-Tree 단말노드의 레코드 식별자(trajectory_ptr)로 궤적 레코드를 찾는다. 궤적 레코드를 검색하여 궤적 레코드에 세그먼트를 삽입할 수 있으면 궤적 레코드에 세그먼트를 삽입한 후 레코드의 위치 정보를 유지한다. 만일 세그먼트를 삽입할 수 없는 경우 새로운 레코드를 생성하고 기존의 레코드와 연결한 후, 새로 생성된 레코드의 위치 정보를 MOID B+-Tree가 유지한다. MOID B+-Tree에서 이동객체를 검색하지 못했을 경우 MOID B+-Tree에 이동객체의 식별자를 삽입한 후 새로운 레코드를 생성하고 새로 생성된 레코드와 MOID B+-Tree의 레코드 식별자를 연결한 후 새로 생성된 레코드의 위치 정보를 유지한다. 이후로, 그림 5의 ③과 같이 이동객체가 지나간 에지 아이디를 가지고 에지의 연결 리스트 테이블을 검색한다. 에지 연결 리스트 테이블에서 Temporal B+-Tree 식별자(Tptr)로 연결된 Temporal B+-Tree가 있으면 찾은 Temporal B+-Tree에 삽입할 이동객체 세그먼트의 시간과 궤적 레코드의 위치를 삽입한다. 만일 Temporal B+-Tree가 존재하지 않으면 Temporal B+-Tree를 생성하고 Spatial 2DR*-Tree 리프노드의 Temporal B+-Tree 식별자를 새로 생성한 Temporal B+-Tree와 연결한다. 그 다음에 생성된 Temporal B+-Tree에 삽입할 이동객체 세그먼트의 시간과 궤적 레코드의 위치를 삽입한다. 그림 6은 이동객체의 궤적 삽입 알고리즘을 나타낸다.

```
Algorithm InsertMovingObject(int Edge_id, TrajSegment)
/* TrajSegment contains informations which are moving object id,
   sequence number of the moving object's trajectory, x y t coordinates,
   direction of the moving object */
1.  Find MOID from MOID B+-Tree
2.  if(ret≠NULL){//ret is a pointer which indicate trajectory record
3.          Get a trajectory record by using ret
4.  }else{
5.          Create trajectory record
6.          Set the MOID B+-Tree leaf node
7.          Insert leaf node to MOID B+-Tree
8.  }
9.  Insert TrajSegment to trajectory record
10. Find edge with Edge_id
11. if(edge_info.Tptr≠NULL) Get a Temporal B+-Tree
    handle by using Tptr
13. else                    Create Temporal B+-Tree
14. Set the Temporal B+-Tree leaf node
15. Insert leaf node to Temporal B+-Tree
End InsertMovingObject
```

그림 6 이동객체의 궤적 삽입 알고리즘

## 4. 질의 처리 알고리즘의 설계

본 장에서는 3장에서 공간 네트워크상의 이동객체의 특성을 고려하여 분류한 질의에 대해, 어떻게 LBS나 텔레매틱스의 응용서비스에 적용할 수 있는지 예제를 제시하고, 분류한 질의들에 적합한 질의 처리 알고리즘을 설계한다. 3장에서 공간 네트워크상에서 움직이는 이동객체에 대한 질의를 이동객체의 큰 특징인 궤적에 초점을 맞추어, 기존의 공간 데이타베이스에서 연구한 기존의 질의와 이동객체의 궤적을 찾는 질의를 결합하여 질의를 시공간영역 내 궤적질의, 시간영역 내 유사궤적질의, 연속적 k-최근접 질의로 분류하였다. 먼저, 시공간

영역 내 궤적질의는 주어진 시공간영역을 지난 이동객체가 어떤 궤적을 남겼는지를 찾는 질의이다. 이와 같은 질의는 실세계에서 택배 시스템이나 화물 시스템에서 많이 쓰일 수 있다. 택배 시스템이나 화물 시스템은 특정한 지역(시공간 영역)에서 출발하고, 운반 차량의 경유지(이동객체의 궤적)가 중요하기 때문에 시공간영역 내 궤적질의가 많이 쓰일 수 있다. 시간영역 내 유사궤적질의는 어떤 이동객체의 궤적 또는 공간 네트워크의 에지 세그먼트 셋이 주어지고, 주어진 시간영역에 어떤 이동객체가 주어진 궤적과 유사한 궤적을 가지는지를 찾는 질의이다. 실세계에서 도로 관리 시스템과 같이 도로를 지나간 차량이 얼마만큼 되는지 알고 싶은 경우에 시간영역 내 유사궤적질의가 많이 쓰일 수 있다. 연속적 k-최근접 질의는 이동객체가 이동하면서 자신과 가까운 POI를 찾는 질의이다. 이동객체는 계속 움직이므로 한 번의 k-최근접 질의로 정확한 k개의 POI를 찾을 수 없으므로 연속적으로 k-최근접 질의를 수행한다. 이와 같은 질의는 전자 상거래 서비스에서 사용이 빈번히 이루어질 수 있다. 이렇게 분류한 질의들에 대한 질의 처리 알고리즘은 본 논문에서 제시한 TMN-Tree상에서 이루어지며, 연속적 k-최근접 질의는 k-최근접 질의 처리가 선행되어야 질의 처리를 할 수 있으므로 본 논문에서는 k-최근접 질의 처리 알고리즘을 제시하고, 연속적 k-최근접 질의는 향후 연구로 남긴다. 4.1절에서는 시공간영역 내 궤적 질의 처리 알고리즘에 대하여 자세히 언급하며, 4.2절에서는 시간영역 내 유사궤적 질의 처리 알고리즘에 대하여 서술한다. 4.3절에서는 k-최근접 질의 처리 알고리즘에 대하여 서술한다.

### 4.1 시공간영역 내 궤적질의 처리 알고리즘

시공간영역 내 궤적질의는 먼저 시공간영역 안에 있었던 이동객체들이 지금까지의 시간동안 또는 어떤 특정한 시간동안에 움직인 궤적을 찾는 질의이다. 시공간영역 내 궤적질의를 위해서는 시공간영역과 궤적의 시간영역이 주어져야 한다. 궤적의 시간영역이 주어지지 않을 경우에는 이동객체의 모든 궤적을 찾는 것으로 가정한다. 시공간영역 내 궤적질의의 예제는 그림 7과 같다. 그림 7에서 보는 것과 같이 먼저 점선의 공간 영역에 대한 질의를 수행하여 해당하는 에지를 얻어온다. 그 다음, 해당 에지를 지난 이동객체에 대해 시간영역 질의를 수행한다. 최종적으로 시간과 공간 영역을 만족하는 이동객체의 궤적을 찾을 수 있다. 그림 7의 경우 MO 1~3이 공간 영역을 만족하므로 이들의 시간 영역을 탐색하여 해당하는 이동객체에 대하여 궤적을 검색한다. TMN-Tree에서 시공간영역 내 궤적 질의를 처리하기 위한 알고리즘은 다음과 같다. 첫째, Spatial 2DR*-Tree를 통해서 주어진 시공간영역의 공간영역을 지나는

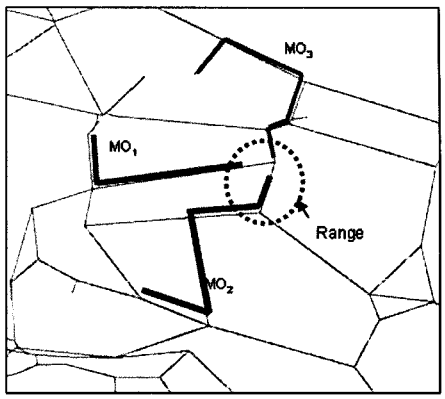


그림 7 시공간영역 내 궤적질의의 예제

```
Algorithm CombinedSpatialTemporalRangeAndTrajectoryQuery
(SpatioRange, TemporalRange,TrajectoryTimeRange)
1. Search RstarTree for spatio_result_set with SaptioRange
2. for each spatio_result_set{
3.     Get a Temporal B+-Tree by Tptr of result set
4.     if(Temporal B+-Tree≠NULL){
5.         Search Temporal B+-Tree for temporal_
           result_set with TemporalRange
6.         for each temporal_result_set{
7.             Get a trajectory record by
               trajectory_ptr of result set
8.             Search the trajectory segment with
               TrajectoryTimeRange
9.         }//of for
10.    }//of if
11. }//of for
End CombinedSpatialTemporalRangeAndTrajectoryQuery
```

그림 8 시공간영역 내 궤적질의 처리 알고리즘

공간 네트워크상의 에지를 찾는다. 둘째, 각각의 찾은 에지에 대하여 Temporal B+-Tree가 존재하면 주어진 시간영역으로 Temporal B+-Tree를 검색한다. 주어진 시간영역에 단말노드가 존재하면, 단말노드의 레코드 식별자(trajectory_ptr)을 이용하여 이동객체의 궤적 레코드를 얻어온다. 마지막으로 이동객체의 궤적 레코드에서 주어진 궤적의 시간영역에 부합하는 궤적 세그먼트를 반환한다. 시공간영역 내 궤적 질의 처리 알고리즘은 그림 8과 같다.

### 4.2 시간영역 내 유사궤적질의 처리 알고리즘

시간영역 내 유사궤적질의는 특정한 에지들을 주어진 시간영역 동안에 주어진 유사궤적을 지난 이동객체 또는 그들의 궤적을 찾는 질의이다. 그림 9는 특정한 에지 3개가 주어지고 이들을 지난 이동객체들의 궤적을 찾는 시간영역 내 유사궤적질의의 예제를 보여준다. 그림 9에서 보듯이 MO1과 MO2가 질의 궤적을 지나고 있고,

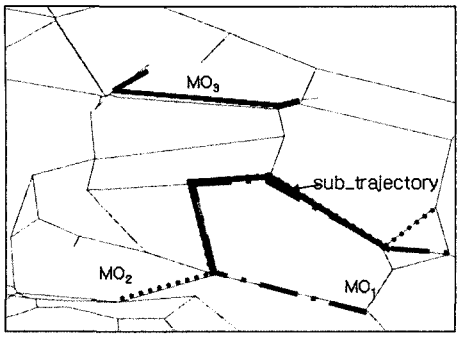


그림 9 시간영역 내 유사 궤적질의의 예제

```
Algorithm CombinedSimilarTrajectoryAndTemporalRangeQuery
(Edge_list, TimeRange, TrajectoryTimeRange)
 /* candiate_set contains trajectory pointer satisfying temporalrange query */
1. for each edge of the Edge_list{
2.        Find edge with Edge_list in edge table
3.        if(edge_table.Tptr≠NULL){
4.              Get a Temporal B+-Tree by edge_table.Tptr
5.              Search Temoral B+-Tree by TimeRange
6.              Add trajectory_ptr to candidate_set
7.      }
8. }
9. Remove duplicated pointer in candidate_set
10. for each candidate_set{
11.       Get a trajectory record by candidate_set
12.       Search trajectory segment with TrajectoryTimeRange
13. }
End CombinedSimilarTrajectoryAndTemporalRangeQuery
```

그림 10 시간영역 내 유사궤적질의 처리 알고리즘

MO3는 질의 궤적을 지나고 있지 않다. 따라서 MO1과 MO2가 질의 궤적의 유사궤적이 될 수 있다. 이 두 궤적의 시간 영역을 검색하여 시간 영역을 만족하는 이동객체의 궤적을 찾을 수 있다, TMN-Tree에서 시간영역 내 유사궤적질의 알고리즘은 다음과 같다. 첫째, 주어진 에지들의 리스트를 사용하여 에지 연결리스트 테이블에 있는 에지를 얻어온다. 둘째, 찾은 에지에 Temporal B+-Tree로 연결하는 포인터(Tptr)가 NULL이 아니라면 Temporal B+-Tree를 얻어서 주어진 시간 영역으로 Temporal B+ -Tree를 탐색한다. 셋째, 주어진 시간영역을 지나는 Temporal B+-Tree 단말노드의 레코드 식별자(trajectory_ptr)를 후보 집합에 저장한다. 넷째, 주어진 에지에 대하여 모든 검색이 완료되면 후보 집합의 중복 데이타를 제거한 후 궤적 레코드를 얻어 주어진 궤적 시간영역에 부합하는 궤적 세그먼트를 반환한다. 시간영역 내 유사궤적질의 처리 알고리즘은 그림 10과 같다.

### 4.3 k-최근접 질의 처리 알고리즘

k-최근접 질의는 공간 네트워크상의 이동객체가 움직

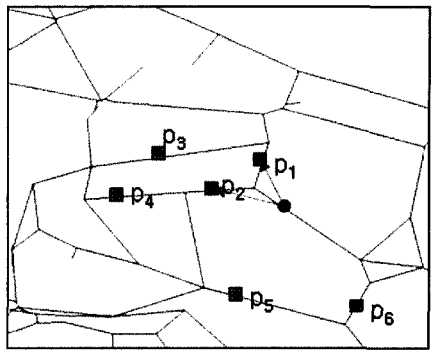


그림 11 k-최근접 질의의 예제

이면서 자신과 가장 가까운 k개의 POI를 찾는 질의이다. k-최근접 질의에 대한 질의 예제는 그림 11과 같다.

본 논문에서 제안하는 k-최근접 질의는 D. Papadias et al.이 제안한 INE 방법[10]과 같이 네트워크 확장을 기반으로 이동객체가 움직이는 방향성을 고려하여 k-최근접 질의를 수행한다. 이는 향후 연속적 k-최근접 질의 처리 알고리즘을 위하여 이동객체가 움직이는 방향성을 고려하였다. 그림 11은 k가 2일 때의 k-최근접 질의의 예제를 보여준다. 그림 11에서 보듯이, p1과 p2가 공간 네트워크에서 가장 가까우므로 결과 POI가 된다. 본 논문에서 제시하는 k-최근접 질의 처리 알고리즘은 에지 연결 리스트 테이블을 메모리에서 유지하기 때문에 네트워크 확장이 모두 메모리에서 이루어진다는 특징이 있다. 방향성을 고려한 k-최근접 질의 처리 알고리즘은 다음과 같다. 먼저 노드를 탐색했는지를 체크하는 플래그와, 에지를 탐색했는지를 체크하는 플래그, 그리고 노드의 네트워크 거리를 유지하고 있는 테이블을 초기화 한다. 그리고 POI의 결과 집합과 방문할 노드를 저장할 스택을 초기화 한다. 초기화가 모두 끝나면, 주어진 질의 지점을 가지고 spatial 2DR*-Tree를 검색하여 주어진 질의 지점을 포함하는 에지를 찾는다. 만일 이 에지에 POI를 가지고 있다면 질의 지점에서 POI까지의 거리를 계산하여 POI의 결과 집합에 삽입한다. 만일 결과 집합이 찾고자 하는 k개와 동일하다면 결과 집합을 반환하고 그렇지 않다면 검색한 에지의 노드를 스택에 삽입한다. 이 때 만일 방향성 정보가 있다면 이동객체가 움직이고 있는 방향 쪽에 있는 노드만 삽입을 하고, 그렇지 않다면 에지와 연결된 노드 모두 삽입한다. 노드의 스택은 질의 지점부터 노드까지의 거리로 정렬이 되도록 세팅되어 있다. 따라서 노드의 스택에서 노드를 꺼내면 그 노드는 질의 지점에서 가장 가까운 노드가 된다. 노드의 스택에서 노드를 하나 꺼낸 후, 그 노드의 연결 에지 리스트를 검색하여 연결된 에지의 다른 노드가 검색한 노드라면 노드의 네트워크 거리를 얻

```
Algorithm kNeareastNeighborQuery(int k, int direction, QueryPoint)
 /* Tables are isSearchedNodeTable, node distanceTable, isSearchedEdgeTable */
1.  Create and allocate Tables
2.  Find edge with QueryPoint
5.  if(edge has a POI){
6.          Calculate network distance from QueryPoint to POI
7.          Insert poi to poi result_set
8.          if(the number of result set equal to k){
9.                  return result set
10.         }
11. }
12.  if(direction ≠NULL){
13.         Push the node to which moving object will move to node stack
14.  }else{
15.         Push two nodes which is connected by edge to node stack
16.  }
17.  Set the Tables
18.  while until finding k nearest POI result{
19.         Pop from node_stack
20.         for each adjacency edge {
21.                 if(edge is not searched ){
22.                         if(node connected the other node,  is not    searched )
23.                                 Insert the POI of the edge to result set
24.                         else {
25.                                 Compare current node distance with already
                                        searched node distance
26.                                 Update distance or insert POI of the edge to
                                      result set

27.                         }
28.             }//of if
29.    }//of for
30.  }//of while
End kNeareastNeighborQuery
```

그림 12 k-최근접 질의 처리 알고리즘

어와 질의 지점부터 POI까지의 거리가 짧은 쪽을 택하여 결과 집합에 삽입한다. 만일 그렇지 않다면 확장한 노드의 네트워크 거리를 계산하여 질의 지점부터 POI까지의 거리를 얻어서 결과 집합에 삽입한다. 이와 같은 방식으로 POI의 결과 집합이 k가 될 때까지 확장을 하며 k가 되면 결과 집합을 사용자에게 반환한다. 그림 12는 k-최근접 질의 처리 알고리즘을 나타낸다.

## 5. 성능평가

본 장에서는 본 논문에서 설계한 TMN-Tree와 기존의 연구인 FNR-Tree, MON-Tree와 성능평가를 수행한다. TMN-Tree를 구현한 환경은 Visual C로 작성하였으며, Intel Xeon 3.0GHz CPU와 메인 메모리 2GB, 윈도우 2003 상에서 구현하였다. 성능평가에 사용한 데이타는 Brinkhoff가 제안한 알고리즘[5]으로 이동객체를 생성하였으며, 공간 네트워크 데이타는 Brinkhoff가 제안한 알고리즘에서 제공하는 공간 네트워크 데이타중의 하나인 샌프란시스코 만 데이타를 사용하였다. 샌프란시스코 만 공간 네트워크의 에지의 수는 약 220,000개이며, 노드의 수는 약 170,000개를 가지고 있다. POI의 개수는 10,000개를 사용하였다. 이동객체는 표 2와 같이 구성하여 생성하였다. 이동객체의 궤적 세그먼트는 이동객체의 수는 1,000, 2,000, 5,000, 10,000개로 주었으며, 시간은 1000으로 설정하여 1,000,000, 2,000,000, 5,000,000, 10,000,000 개의 이동객체의 궤적을 생성하였다. 각각의 데이타는 D1, D2, D3, D4로 명명한다.

표 2 생성된 이동객체의 궤적 세그먼트

| | 세그먼트 개수 | 이동객체 개수 | 타임스탬프 개수 |
|---|---|---|---|
| D1 | 1,000,000 | 1000 | 1000 |
| D2 | 2,000,000 | 2000 | 1000 |
| D3 | 5,000,000 | 5000 | 1000 |
| D4 | 10,000,000 | 10000 | 1000 |

사용한 R*-Tree와 B+-Tree의 디스크 페이지 크기는 1KB 이며, 궤적 레코드의 디스크 페이지 크기 역시 1KB로 설정하였다. 성능평가 항목은 각각 트리의 삽입 시간, 영역질의 응답시간, 유사궤적 질의 응답시간, k-최근접 질의 응답시간을 측정한다. 5.1절에서는 삽입 성능평가, 5.2절에서는 영역질의 성능평가, 5.3절에서는 유사궤적 질의 성능평가, 그리고 5.4절에서는 k-최근접 질의 성능평가에 대해 논의한다.

### 5.1 삽입 성능평가

본 절에서는 FNR-Tree, MON-Tree, TMN-Tree의 삽입 성능평가를 수행한다. MON-Tree는 에지 모델과 경로 모델 중에 에지 모델을 선택하여 삽입 성능평가를 한다. 그 이유는 TMN-Tree 역시 k-최근접 질의를 제공하기 위해 에지 모델을 사용하였기 때문이다. 삽입에 사용한 데이타는 앞에서 언급했던 네 가지의 데이타를 가지고 삽입 성능평가를 수행하였다. 그림 13은 삽입 성능을 비교한 그래프이다. 그림 13에서 보듯이 삽입 시간은 FNR-Tree, MON-Tree, TMN-Tree순으로 성능이 향상됨을 알 수 있다. 데이타가 D1일 경우에 삽입시간이 FNR-Tree는 약 700초이고 MON-Tree는 약 482초, TMN-Tree는 약 163초로 측정된다. 데이타가 D4일 경우에는 FNR-Tree가 약9200초, MON-Tree가 약 8500초 TMN-Tree가 약 2600초로 측정된다. D1에서 D4까지의 데이타 삽입 성능을 비교하였을 때 TMN-Tree가 MON-Tree에 비해 약 3배, FNR-Tree에 비해서는 약 4배의 삽입시간 성능이 향상되었다. 이는 FNR-Tree는 이동객체의 궤적이 삽입되면 그 이동객체

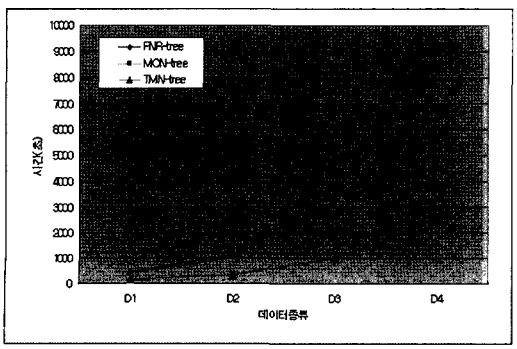


그림 13 삽입 성능평가

의 궤적이 어떤 에지를 지나는지 2DR-Tree를 검색해야 하며, 1DR-Tree로 삽입을 하기 위해 다시 1DR-Tree를 접근해야 하는 오버헤드가 있다. 즉 트리를 두 번 접근해야 하는 오버헤드로 인해 삽입 성능이 다른 트리에 비해 저하된다. MON-Tree와 TMN-Tree의 차이점은 이동객체의 궤적을 색인하는 구조가 MON-Tree는 1DR-Tree인 반면 TMN-Tree는 B+Tree로 색인을 한다. 1DR-Tree는 색인 구조를 구성하는 키가 1차원인 반면 B+-Tree는 차원이 없는 이동객체의 타임스탬프로 구성하기 때문에 삽입 성능의 차이가 나는 것을 알 수 있다.

### 5.2 시공간 영역 내 궤적질의 성능평가

본 절에서는 FNR-Tree, MON-Tree, TMN-Tree에 대한 영역질의 성능평가를 수행한다. 본 논문에서 제시한 시공간 영역 내 궤적질의는 FNR-Tree와 MON-Tree가 이동객체에 대한 궤적 모두를 유지하고 있지 못하기 때문에 성능비교를 할 수 없으므로 성능평가를 FNR-Tree와 MON-Tree가 지원할 수 있는 영역질의로 제한하였다. 영역질의는 각 차원별로 1%, 5%, 10%, 20%의 영역을 선택하여 질의 하였다. 영역 질의 성능평가에 사용한 데이타는 D4 데이타이다. 그림 14는 영역질의 성능을 비교한 그래프이다. 영역질의에 대한 성능은 MON-Tree, FNR-Tree, TMN-Tree순으로 더 짧은 질의 응답시간을 보인다. 1% 영역질의에서 FNR-Tree가 약 0.2초, MON-Tree가 0.6초, TMN-Tree가 약 0.4초이다. 20% 영역질의에서는 FNR-Tree가 약 120초, MON-Tree가 약 152초, TMN-Tree가 약 64초로 TMN-Tree가 MON-Tree와 FNR-tree에 비해 약 2배정도 향상된 성능을 가짐을 알 수 있다. 이는 TMN-Tree가 이동객체의 궤적을 B+-Tree로 색인하기 때문에, B+-Tree의 특성상 B+-Tree를 통해 궤적을 검색할 때 리프노드의 링크를 따라 순차 검색한다. 하지만 FNR-Tree나 MON-Tree의 경우는 1DR-Tree이기 때문에 주어진 시간영역을 지나는 이동객체의 궤적을 가진 리프노드를 찾은 다음 다시 상위 노드로 올라가 다음 리프노드를 검색하는 재귀적인 방법을 사용한다. 따라서 영역이 작을 때는 많은 차이가 나지 않지만 영역이 커질수록 TMN-Tree의 질의 응답시간이 점점 더 적어진다. 그리고 FNR-Tree에 비해 MON-Tree의 영역질의 성능이 더 저하되는 이유는 다음과 같다. FNR-Tree와 MON-Tree는 두 트리 모두 에지 단위로 저장하기 때문에 생성되는 2DR-Tree와 1DR-Tree의 개수가 같다. 하지만 MON-Tree는 에지를 따로 저장하는 데이타 레코드의 구조가 있기 때문에 레코드를 한 번 더 접근해야하는 오버헤드가 있는 반면, FNR-Tree는 바로 1DR-Tree와 연결되기 MON-Tree에서 발생하는

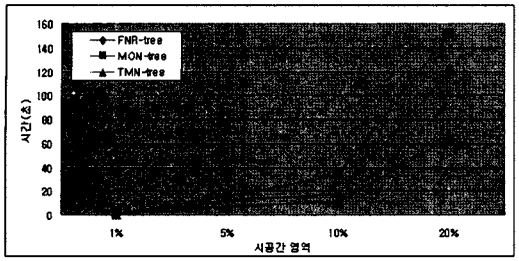


그림 14 시공간 영역 내 궤적질의 성능평가

오버헤드가 없다. 따라서 FNR-Tree가 MON-Tree보다 영역질의에서 더 향상된 성능을 보인다.

### 5.3 시간 영역 내 유사궤적질의 성능평가

본 절에서는 FNR-Tree, MON-Tree, TMN-Tree에 대한 시간영역 내 유사궤적 질의 성능평가를 수행한다. 시간 영역 내 유사궤적질의는 유사궤적의 에지 세그먼트의 개수는 2, 3, 4, 5, 6, 8, 10, 12, 14, 16, 18, 20으로 변화시키면서 성능평가를 수행하였고, 시간 영역은 전체 시간영역의 0.5%로 고정시켜 성능평가를 수행하였다. 그림 15는 시간 영역 내 유사궤적질의 성능평가를 한 그래프이다. 시간 영역 내 유사궤적질의의 성능은 FNR-Tree, MON-Tree, TMN-Tree순으로 더 짧은 질의 응답시간을 보인다. 유사궤적의 세그먼트 개수가 2개 일 때는 FNR-Tree가 약 0.02초, MON-Tree가 0.05초, TMN-Tree가 0.004초이다. 그리고 유사궤적 세그먼트 개수가 20개 일 때는 FNR-Tree가 약 0.24초, MON-Tree가 0.05초, TMN-Tree가 0.02초와 같이 증가하였다. 이는 FNR-Tree의 경우 MON-Tree나 TMN-Tree와 같이 에지 아이디를 바로 접근할 수 있는 테이블이 없기 때문에 에지의 MBR로 2DR-Tree를 검색해야 하므로 MON-Tree나 TMN-Tree에 비해서 성능이 더 저하된다. MON-Tree보다 TMN-Tree가 더 향상된 성능을 가지는 이유는, 이동객체의 궤적을 색인하는 트리가 MON-Tree는 1DR-Tree인 반면 TMN-Tree는

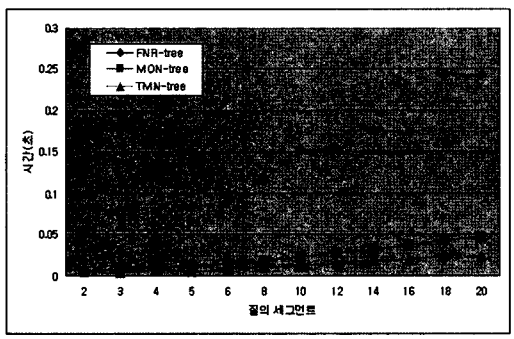


그림 15 시간 영역 내 유사궤적질의 성능평가

B+-Tree이므로 시간 영역을 탐색하는 시간이 줄어들기 때문이다. 또한 TMN-Tree는 이동객체의 궤적을 유지하기 때문에 이동객체의 유사궤적을 한 번의 접근으로 찾을 수 있지만 MON-Tree는 TMN-Tree보다 더 많은 접근을 해야 하므로 TMN-Tree보다 더 성능이 더 저하된다.

### 5.4 k-최근접 질의 성능평가

본 절에서는 보로노이(Voronoi) 다이어그램 방법[12]과 본 논문에서 제안하는 TMN- Tree에 대해 k-최근접 질의 성능평가를 수행한다. 성능평가를 위해 사용한 k의 개수는 1, 5, 10, 20, 40, 60, 80, 100개이다. 그림 16은 k-최근접 질의 성능평가를 한 그래프이다.

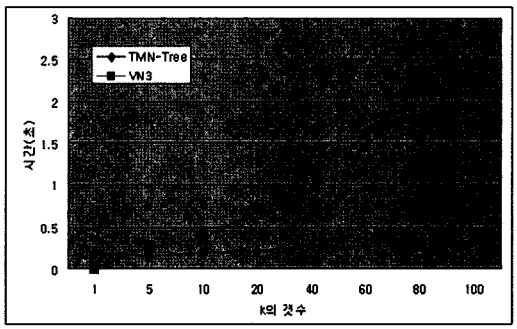


그림 16 k-최근접 질의 성능평가

k-최근접 질의 성능은 TMN-Tree에서 k-최근접 질의 성능이 보로노이 다이어그램 방법 보다 더 좋다. k의 개수가 1개일 경우에는 보로노이 방법이 약0.001초인 반면 TMN-Tree의 방법은 약 0.18초이고, k의 개수가 100개일 경우에는 보로노이 방법이 2.7초인 반면 TMN-Tree는 0.32초이다. k가 1일 경우에는 보로노이 방법이 TMN-Tree에 비해 검색 시간이 더 빠르다. 하지만 k가 증가할수록 TMN-Tree가 보로노이 방법에 비해 성능이 더 향상됨을 그림 16을 통해 알 수 있다. 예를 들어 k가 100개일 때는 TMN-Tree가 보로노이 방법에 비해 약 9배정도 더 빠른 검색 성능을 가진다. 이는 k가 1일 때는 보로노이 방법이 R-Tree 검색으로 바로 찾을 수 있지만, k가 증가할수록 네트워크 거리를 통해 보로노이 도형들을 확장해 나가야 하는데 이 과정에서 보로노이 방법은 미리 계산된 네트워크 거리를 계산할 때 디스크를 접근을 하는 오베헤드가 있다. 하지만 TMN-Tree는 공간 네트워크를 메모리에 상주시켰기 때문에 거리를 계산하기 위한 디스크 접근이 없다. 이로 인하여 검색 시간이 더 줄어들기 때문에 TMN-tree가 보르노이 방법보다 더 빠른 검색 성능을 가진다.

## 6. 결론 및 향후 연구

본 논문에서는 공간 네트워크상의 이동객체를 위한 궤적기반 저장/색인 구조인 TMN-Tree를 제안하였다. 공간 네트워크상의 이동객체를 저장 및 색인하기 위하여 공간 데이타와 이동객체의 궤적 데이타 즉 시간 데이타를 분류하여 색인하였다. 이를 위하여 공간 데이타는 효율적으로 저장 및 색인하기 위하여, R*-Tree를 사용하였다. 시간 데이타를 색인하기 위해서 B+-Tree를 사용하였다. 그리고 현재 응용 서비스에서 이동객체에 대한 질의를 시공간영역 내 궤적 질의, 시간영역 내 유사궤적질의, 연속적 k-최근접 질의와 같이 분류하고, 이를 지원하기 위해 이동객체의 궤적을 유지하기 위한 구조와 에지의 연결리스트 테이블을 구성하였다. 또한 성능평가를 통하여 본 논문에서 제시한 TMN-Tree가 기존의 공간 네트워크상의 이동객체를 저장/색인하는 구조인 FNR-Tree, MON-Tree에 비해 삽입, 영역 질의, 시간영역 내 유사궤적질의 성능이 더 좋음을 보였다. 그리고 HKUST의 연구의 INE기법에 비해 본 논문에서 제시한 저장/색인 구조가 k-최근접 질의 성능이 더 좋음을 보였다. 향후 연구로는 k-최근접 질의를 확장한 연속적 k-최근접 질의, 궤적질의를 확장한 In-Route 질의와 같은 질의 처리 알고리즘을 구현하여 본 논문에서 제안한 TMN-Tree를 확장해나가는 것이며, 본 논문에서 제안한 저장/색인 구조를 현재의 응용 서비스에 접목시켜 실세계에서 효율적임을 보이는 것이다.

## 참 고 문 헌


[1] Vazirgiannis, M., Theodoridis, Y., and Sellis, T. "Spatio-temporal Indexing for Large Multimedia Applications," In Proc. of the IEEE Conference on Multimedia Computing and Systems6(4), pp. 284-298, 1998.

[2] D. Pfoser, C.S. Jensen, and Y. Theodoridis, "Novel Approach to the Indexing of Moving Object Trajectories," In Proc. of VLDB, pp. 395-406, 2000.

[3] Tao, Y., and Papadias, D. "Mv3R-tree: a spatio-temporal access method for timestamp and interval queries," In Proc. of VLDB, pp. 431-440, 2001.

[4] A. Guttman "R-Trees: A Dynamic Index Structure for Spatial Searching," In Proc. of SIGMOD, pp. 47-57 1984.

[5] T. Brinkhoff, "A Framework for Generating Network-Based Moving Objects," In Proc. of GeoInformatica 6(2), pp. 153-180, 2002.

[6] V. Chakka, A. Everspaugh, J. Patel, Indexing "Large Trajectory Data SetsWith SETI," In Proc. of the Conf. on Innovative Data Systems Research, CIDR, Asilomar, CA, Jan. 2003.

[7] E. Frentzos, "Indexing Objects moving on fixed networks," In Proc. of the 8th In Proc. of Intl. Symp. on Spatial and Temporal Database(SSTD), pp. 289-305, 2003.

[8] D. Pfoser and C.S. Jensen, "Indexing of Network Constrained Moving Objects," In Proc. of ACM GIS, pp. 25-32, 2003.

[9] Victor Teixeira de Almeida, Ralf Hartmut Güting. "Indexing the Trajectories z'of Moving Objects in Networks," In Proc. of GeoInformatica 9(1), pp. 33-60, 2005.

[10] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao, "Query Processing in Spatial Network Databases," In Proc. of VLDB, pp. 802-813, 2003.

[11] Mohammad Kolahdouzan and Cyrus Shahabi, "Voronoi-Based K Neareast Neighbor Search for Spatial Network Databases," In Proc. of VLDB, pp. 840-851, 2004.

[12] N. Beckmann, H.-P. Kriegel, R. Schneider, B. Seeger: The "R*-Tree: An Efficient and Robust Access Method for Points and Rectangles," In Proc. of SIGMOD, pp. 322-331, 1990.



엄 정 호

2004년 전북대학교 컴퓨터 공학과(공학사). 2004년~2006년 전북대학교 컴퓨터공학과(공학석사). 2006년~현재 전북대학교 컴퓨터 공학과 박사과정. 관심분야는 공간 데이타베이스, 공간 색인 구조, GIS



장 재 우

1984년 서울대학교 전자계산기공학과(공학사). 1986년 한국과학기술원 전산학과(공학석사). 1991년 한국과학기술원 전산학과(공학박사). 1996년~1997년 Univ. of Minnesota, Visiting Scholar. 2003년~2004년 Penn State Univ., Visiting Scholar. 1991년~현재 전북대학교 컴퓨터공학과 교수. 관심분야는 공간 네트워크 데이타베이스, 상황인식, 하부저장구조