공간 네트워크 상의 이동객체를 위한 시그니처 기반의 궤적 색인구조† ††

Trajectory Index Structure based on Signatures for Moving Objects on a Spatial Network

김영진* / Young-Jin Kim, 김영창** / Young-Chang Kim,
장재우*** / Jae-Woo Chang, 심춘보**** / Chun-Bo Sim

요 약

공간 네트워크 상을 움직이는 많은 이동객체들의 궤적 분석을 통해서 많은 정보를 얻을 수 있다. 이를 위해서, 궤적을 효과적으로 검색할 수 있는 궤적 기반 색인 구조가 필요하다. 하지만 도로와 같은 공간 네트워크상의 궤적 기반 색인 구조에 대한 연구는 FNR-트리나 MON-트리를 제외하고는 연구가 많이 부족한 실정이다. 또한, FNR-트리나 MON-트리는 에지를 지난 이동객체의 이동정보인 세그먼트만을 저장할 뿐 전체 궤적을 유지하지 못하며, 궤적 질의에 대해 비효율적이다. 따라서 본 논문에서는 공간 네트워크상의 이동객체를 위한 시그니처 기반의 궤적 색인 구조인 SigMO-트리를 제안한다. 이를 위해, 이동객체를 공간과 시간 특성으로 분류하고, 전체 궤적을 유지함으로써 영역질의와 궤적질의를 동시에 처리할 수 있는 색인 구조를 설계한다. 아울러, 사용자 질의를 시공간영역 내 궤적 질의, 시간영역 내 유사궤적 질의로 분류하고, 이들을 처리하기 위한 질의 처리 알고리즘을 제안한다. 각 질의처리 알고리즘은 효율적인 검색을 위하여 시그니처 파일 기법을 이용하여 궤적을 검색한다. 마지막으로 성능평가를 통해 본 논문에서 제안한 궤적 기반 색인 구조가 기존의 색인구조인 FNR-트리, MON-트리보다 성능이 우수함을 보인다.

Abstract

Because we can usually get many information through analyzing trajectories of moving objects 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. Also, because FNR-tree and MON-tree store the segment unit of moving objects, they can't support the trajectory of whole moving objects.

† 본 연구는 교육과학기술부와 한국산업기술재단의 지역혁신 인력 양성사업으로 수행된 연구 결과임
†† 본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT연구센터 지원사업의 연구결과로 수행되었음 (IITA-2008-(C1090-0801-0047))
■ 접수일 : 2008.01.09 ■ 수정일 : 1차 2008.02.26 / 2차 2008.04.15 ■ 심사완료일 : 2008.06.05
* 전북대학교 전자정보공학부 컴퓨터공학과 석사과정(yzkim)@chonbuk.ac.kr
** 전북대학교 전자정보공학부 컴퓨터공학과 박사과정(zerowin1)@chonbuk.ac.kr
*** 교신저자 전북대학교 전자정보공학부과 컴퓨터공학 교수(jwchang)@chonbuk.ac.kr
**** 순천대학교 정보통신공학부 조교수(cbsim@sunchon.ac.kr)

In this paper, we propose an efficient trajectory index structures based on signatures on a spatial network, named SigMO-Tree. For this, we divide moving object data into spatial and temporal attributes, and design an index structure which supports not only range query but trajectory query by preserving the whole trajectory of moving objects. In addition, we divide user queries into trajectory query based on spatio-temporal area and similar-trajectory query, and propose query processing algorithms to support them. The algorithm uses a signature file in order to retrieve candidate trajectories efficiently Finally, we show from our performance analysis that our trajectory index structure outperforms the existing index structures like FNR-Tree and MON-Tree.

주요어 : 공간 네트워크 데이터베이스, 궤적, 색인구조, 시그니처
Keyword : Spatial Network Database, trajectory, index structure, signature

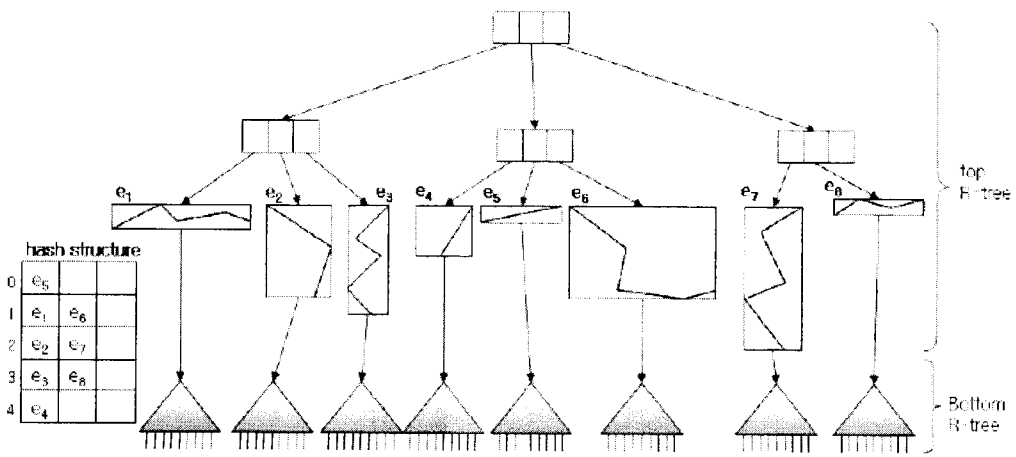
1. 서 론

기존의 공간 데이터베이스 상에서 이동객체를 위한 색인구조 연구가 많이 진행되어 왔다[1, 2, 3]. 대표적인 연구로는 3DR-트리[1], TB-트리[2]가 있다. 3DR-트리는 2DR-트리[4]에서 시간축을 확장하여 3차원의 공간상에서 움직이는 이동객체를 표현 하였다. TB-트리는 3차원의 공간상에서 움직이는 이동객체를 MBR 대신 에지를 지나는 하나의 이동객체의 이동정보인 세그먼트 형식으로 표현하여 저장하였다. 여기서 3DR-트리와 TB-트리의 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 최근 연구되었다.[5, 6, 7, 8]. 이 중에서 대표적인 연구는 아테네 국립대학에서 연구한 FNR-트리[7]와 독일 하겐 대학에서 연구한 MON-트리[9]가 있다. 유클리디언 공간상을 움직이는 이동객체는 객체가 시간에 따라 이동을 할 때 공간의 제약이 없다. 반면에 공간 네트워크상에서 움직이는 이동객체는 공간의 제약을 가진다. 이러한 차이점으로 인하여, 유클리디언 공간을 움직이는 이동객체를 색인하는 구조는 공간 네트워크상을 움직이는 이동객체를 색인하는 경우 공간상의 중복이 많이 발생하는 문제점을 지닌다. 따라서 공간 네트워크상에 움직이는 이동객체는 공간의 제약을 가지는 특성을 고려하여 공간데이터와 시간데이터를 분리하여 색인을 해야 한다. 공간 데이터는 데이터를 한 번 삽입하면 거의 데이터의 삽입이 이루어지지 않는 특성을 가지고 있다. 이와 달리 시간데이터의 경우 시간의 변화에 따른 데이터 양이 증가되는 특성이 있기 때문에 공간 데이터에 비해 삽입이 빈번히 일어난다. 따라서 본 논문에서는 공간 네트워크상에서 움직이는 이동객체의 특성을 반영하여 이동객체의 궤적을 위한 색인구조를 제안한다. 이를 위해, 이동객체를 공간과 시간 특성으로 분류하고, 전체 궤적을 유지함으로써 영역질의와 궤적질의를 동시에 처리할 수 있는 색인 구조를 설계한다. 아울러, 사용자 질의를 시공간영역 내 궤적 질의, 시간영역 내 유사궤적 질의로 분류하고, 이들을 처리하기 위한 질의 처리 알고리즘을 제안한다. 각 질의처리 알고리즘은 효율적인 검색을 위하여 시그니처 파일 기법을 이용하여 궤적을 검색한다. 본 논문의 구성은 다음과 같다. 제 2장에서는 공간 네트워크상을 움직이는 이동객체를 위한 색인구조에 대한 관련연구들을 소개하고, 제 3장에서는 궤적기반 색인구조를 제안한다. 제 4장에서는 SigMO-트리를 통한 데이터 삽입 알고리즘을 제안한다. 제 5장에서는

시그니처 기법을 이용한 질의 처리 알고리즘을 제안하며, 제 6장에서 실험을 통한 성능 분석 결과를 제시한다. 마지막으로 제 7장에서는 결론 및 향후 연구를 제시한다.

2. 관련 연구

공간 네트워크 데이터베이스(SNDB: Spatial Network DataBase)에 대한 연구는 크게 3가지 주제로 요약된다. 즉, 공간 네트워크를 위한 데이터 모델, 질의 처리 알고리즘, 마지막으로 공간 네트워크 데이터베이스를 위한 저장 및 색인 구조이다. 본 논문에서는 공간 네트워크 데이터베이스를 위한 세 번째 주제인 공간 네트워크 데이터베이스를 위한 효율적인 저장 및 색인 구조를 설계한다. 본 절에서는, 공간 네트워크 데이터베이스를 위한 효율적인 저장 및 색인 구조의 관련연구와 불필요한 데이터 검색을 효과적으로 줄일 수 있는 시그니처 파일 기법을 살펴본다. 먼저 기존의 공간 네트워크 데이터베이스를 위한 효율적인 저장 및 색인구조에 대한 연구는 공간 네트워크상의 이동객체를 색인하기 위한 연구[7, 9]로 진행되어 왔다. 먼저, FNR-트리[7]는 2003년 E. Frentzos 에 의해 연구되었다. FNR-트리는 고정된 공간 네트워크를 2DR-트리로 색인한다. 따라서 2DR-트리의 리프노드에는 네트워크를 이루는 단위인 에지가 삽입된다. 이동객체에 대해서는 이동객체가 지나간 에지별로 1DR-트리를 생성한다. 이는 이동객체의 궤적에 대해 공간 데이터와 시간 데이터를 분리했기 때문이다. FNR-트리는 3DR-트리보다 영역질의 측면에서 더 향상된 성능을 보이고 있다. 하지만 FNR-트리는 삽입이나 질의 처리 시 2DR-트리를 검색하고, 다시 이동객체의 궤적 검색을 위해 1DR-트리를 탐색해야 하는 오버헤드가 있다. 둘째, MON-트리[9]는 2005년 Ralf Hartmut Güting 등에 의해 연구되었다. MON-트리는 FNR-트리에서 삽입이나 질의 처리 시 2DR-트리와 1DR-트리 두 개의 트리를 탐색해야 하는 오버헤드를 보완하고, 아울러 공간 네트워크에 대한 새로운 모델을 적용하여 공간 네트워크상에서 움직이는 이동객체를 색인하고자 하였다. MON-트리의 전체 구조는 <그림 1>과 같다. MON-트리의 기본 구조는 FNR-트리와 유사하다. 먼저 top R-트리는 FNR-트리의 2DR-트리처럼 공간 네트워크를 저장한다. FNR-트리와 다른 점은 리프노드가 에지가 될 수도 있지만 MON-트리에서 제안하는 공간 네트워크 모델인 경로(Route)가 될 수 있다는 점이다. bottom R-트리는 에지나 경로를 지



<그림 1 MON-트리의 전체구조>

나는 이동객체를 색인하는 R-트리이다. FNR-트리와 같이 1DR-트리로 색인한다. 또한 MON-트리는 해시 테이블 구조를 유지하여, 이동객체 궤적의 삽입 시 해시 테이블을 통해 삽입한다. MON-트리는 해시 테이블을 이용함으로써 FNR-트리보다 트리 접근 횟수를 줄여, 삽입성능을 향상시켰다.

마지막으로 불필요한 데이터 검색을 효과적으로 줄일 수 있는 시그니처 파일 기법을 소개한다 [11,12]. 시그니처 파일 기법은 레코드를 구성하는 속성 값들의 비트 패턴을 계산하고 피트 패턴을 비트 OR 하여 레코드의 시그니처를 구성한 후 이를 시그니처 파일에 저장한다. 데이터를 검색할 시 먼저 질의 레코드에 대한 시그니처를 구성한 후 시그니처 파일에 저장된 각 레코드의 시그니처와 비트 AND 하여 질의 레코드 시그니처와 일치하는 후보 집합을 구한다. 따라서 질의 시그니처와 일치하지 않는 데이터들을 효과적으로 필터링할 수 있다. 마지막으로 후보 집합의 데이터를 검색하여 질의 데이터와 비교 후 질의 결과 집합을 구성하여 반환한다.

3. 궤적기반 색인구조

3.1 연구동기

FNR-트리나 MON-트리와 같은 공간 네트워크상의 이동객체를 위한 색인 구조는 크게 두 가지의 문제점을 갖는다. 첫째, 이동객체를 공간 네트워크의 에지를 지난 이동객체의 이동정보인 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 유사궤적 질의와 같은 대표적인 질의를 처리하지 못한다. 따라서 이를 해결하기 위한 공간 네트워크상의 이동객체에 대한 질의를 정의하고, 이러한 질의들을 효과적으로 지원할 수 있는 색인 구조를 설계하는 것이 필요하다. <표 1>은 본 논문에서 정의한 이동객체 질의를 분류한 표이다.

<표 1> 질의 타입의 분류

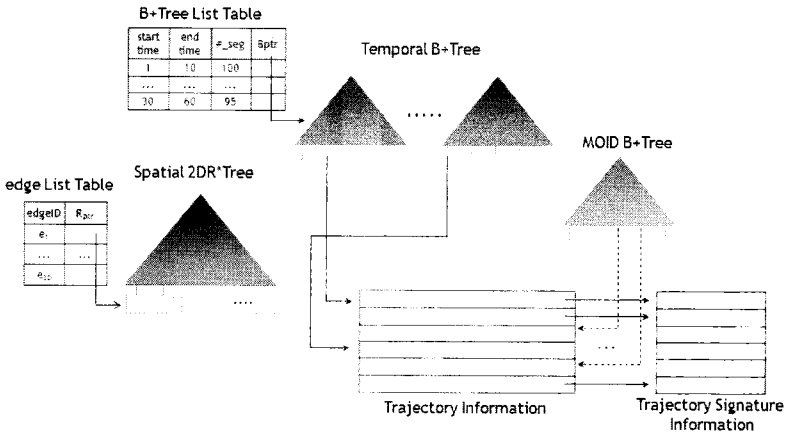
질의 타입	설 명
시공간영역 내 궤적질의	어떤 이동객체가 주어진 시공간 영역을 지나면서 어떤 궤적을 남겼는지를 찾는 질의
시간영역 내 유사궤적질의	어떤 이동객체가 주어진 시간 영역에서 주어진 궤적과 유사한 궤적을 지났는지를 찾는 질의
연속적 k-최근접 질의	이동객체가 움직이면서 자신과 가까운 k개의 POI를 찾는 질의

<표1>과 같이 분류한 이유는 다음과 같다. 기존의 공간 네트워크 데이터베이스 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 질의는 향후 연구로 남긴다. 먼저, 시공간 영역 내 궤적질의는 주어진 시공간영역을 지난 이동객체가 어떤 궤적인지를 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 시공간 영역 내 궤적질의가 많이 쓰일 수 있다. 또한 해당 영역 범위에 극심한 교통체증이 발생할 시에 해당 영역을 통과하는 차량의 이동궤적을 파악하여 주요 이동방향에 대하여 사전에 우회할 수 있는 도로를 건설을 위해 교통체증의 해결방안을 모색하기 위한 통계자료를 얻고자 할 때 많이 쓰일 수 있다. 시간영역 내 유사궤적질의는 어떤 이동객체의 궤적 또는 공간 네트워크의 에지 집합이 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. 위와 같은 시공간 영역 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. 따라서 본 논문에서는 이동객체의 궤적과 네트워크 정보를 유지함으로써 시공간영역 내 궤적질의, 시간영역 내 유사궤적질의를 효율적으로 처리할 수 있는 색인 구조인 시그니처를 이용한 궤적기반 색인 구조인 SigMO-트리를 제안한다.

3.2 SigMO-트리의 전체구조

본 논문에서 제시하는 SigMO-트리의 전체 구조는 공간 네트워크를 저장하기 위한 2DR*-트리[10], 공간 네트워크를 움직이는 이동객체를 색인하기 위한 B+-트리, 동적으로 생성되는 B+-트리의 정보를 유지하는 B+-트리 리스트 테이블, 공간 네트워크 정보를 유지하는 에지 리스트 테이블, 이동객체 아이디를 색인하기 위한 MOID B+-트리, 궤적을 유지하는 궤적 레코드와 궤적의 시그니처를 유지하는 궤적 시그니처 레코드로 구성되어 있다. <그림 2>는 SigMO-트리의 전체구조를 나타낸다. SigMO-트리의 전체 구조에서, 생성된 여러 B+-트리들 중 조건을 만족하는 B+-트리의 빠른 선택 및 접근을 위하여, B+-트리 리스트 테이블을 메모리에 유지한다. 아울러, 에지의 식별자를 통해 에지의 실제 좌표정보에 바로 접근하기 위해 에지 리스트 테이블을 메모리에 유지한다. 또한 공간 네트워크를 저장하기 위한 2DR*-트리, 공간 네트워크를 움직이는 이동객체를 색인하기 위한 B+-트리, 이동객체 아이디를 색인하기 위한 MOID B+-트리, 이동객체의 궤적 정보 레코드 그리고, 궤적 시그니처 레코드는 디스크에 유지한다. 따라서, 메모리에 상주하는 정보는 B+-트리 리스트 테이블과 에지 리스트 테이블이다. 첫째, B+-트리 리스트 테이블의 레코드는 <start time, end time, #_seg, btree_ptr>로 구성되며, 하나의 레코드는 16byte의 메모리 영역을 차지한다. 하나의 B+-트리 당 100개의 세그먼트를 유지하도록 할 경우, 성능평가 시 사용하였던 가장 큰 데이터셋인 D4는 1,000,000개의 세그먼트인 경우 총 10,000개의 레코드가 생성 된다. 따라서 B+-트리 리스트 테이블은 약 160K의 메모리 영역을 차지한다. 둘째, 에지 리스트 테이블의 레코드는



<그림 2> SigMO-트리의 전체구조

<Edge_id, Rptr>로 이루어져 있으며, 하나의 레코드 당 8byte의 메모리 영역을 차지한다. 본 논문에서 사용하는 공간 네트워크 데이터는 220,000개의 에지로 이루어져 있으므로 총 220,000개의 레코드가 생성된다. 따라서 에지 리스트 테이블은 약 1.7MB의 메모리 영역을 차지한다. 결과적으로 메모리에 유지하는 구조는 약 2MB 정도의 의 메모리 영역을 차지며 이는 시스템이 감당할 수 있는 타당한 크기이다. 이동객체의 질의를 수행할 경우, 조건을 만족하는 이동객체들의 궤적정보 및 궤적 시그니쳐 레코드는 빠른 궤적 비교를 위해서 메모리로 가져온다. SigMO-트리는 다음과 같은 특징을 지닌다. 첫째, SigMO-트리는 B+-트리 리스트 테이블, 에지 리스트 테이블을 메모리에 유지하여 궤적질의를 보다 빠르게 처리한다. 이는 궤적질의 수행 시 빈번하게 접근해야 하는 정보들을 메모리에 상주시킴으로서 디스크 접근 없이 빠르게 처리할 수 있기 때문이다.

셋째, 이동객체의 궤적을 통해 다양한 질의처리를 위해서는 궤적간의 연산이 필요하며, SigMO-트리는 시그니처 파일 기법을 사용하여 궤적간의 비교 시간을 효과적으로 줄일 수 있다[11,12]. 이는 이동객체들의 궤적정보를 비트단위 OR 연산(superimposed coding)을 통해 유지하며, 비트검사를 통해 효율적으로 궤적을 비교할 수 있다. 넷째, SigMO-트리는 MON-트리와 FNR-트리에 비해 이동객체의 궤적에 대한 시공간 영역 질의를 보다 효율적으로 처리한다. 이를 위해 이동객체의 궤적을 타임스탬프 단위로 저장하고 Temporal B+-트리로 색인한다. 이는 MON-트리나 FNR-트리처럼 1DR-트리로 시간을 색인하는 것보다 효율적이기 때문이다. 1DR-트리에 비해 B+-트리로 색인 시의 장점은 다음과 같다. 첫째, 1DR-트리가 시간 영역 질의 처리를 위해 재귀 호출하는 반면 B+-트리는 순차적으로 검색하므로 질의 처리 비용이 감소된다. 둘째, 1DR-트리의 키는 시간의 범위인 반면 B+-트리의 키는 하나의 점과 같기 때문에 색인구조의 크기가 작아진다.

3.2.1 Spatial 2DR*-트리

이동객체가 이동하는 공간 네트워크를 색인하기 위하여 노드와 에지로 이루어진 공간 네트워크 데이터를 2DR*-트리로 사용한다. 2DR*-트리에 공간 네트워크 데이터의 삽입은 이동객체가 삽입되기 전에 미리 수행되는 것으로 가정한다. 2DR*-트리의 비단말 노드의 엔트리는 <MBR, childptr>로 구성된다. MBR은 현재 노드의 자식 노드가 가지고 있는 영역을 의미하며, childptr은 현재 노드의 자식 노드를 연결한다. 2DR*-트리의 단말노드는 <MBR, Edge_info>로 구성된다. MBR은 공간 네트워크상의 에지를 포함하는 바운더리 영역을 의미하고 Edge_info는 에지 정보를 담고 있는 레코드이며 Edge_info는 <{point1, point2, ...}>과 같이 에지를 이루는 좌표들의 리스트로 구성된다.

3.2.2 에지 리스트 테이블

공간 네트워크 정보가 저장되어 있는 2DR*-트리로의 에지들의 고유 식별자를 통하여 트리의 탐색 없이 에지의 실제 좌표정보에 바로 접근하기 위하여 에지 리스트 테이블을 구성한다. 에지 리스트 테이블은 <Edge_id, Rptr>로 구성된다. Edge_id는 2DR*-트리에 저장되어 있는 에지들의 고유 식별자를 의미하며, Rptr은 고유 식별 값을 가지는 에지의 2DR*-트리의 단말 노드를 연결한다.

3.2.3 Temporal B+-트리

이동객체가 이동한 시간을 통해서 이동객체의 이동정보인 세그먼트를 색인하기 위하여 이동한 시간을 키로 하는 Temporal B+-트리를 구성한다. 이는 질의처리에 있어서 시간범위를 통해서 이동객체의 궤적으로의 효율적인 접근을 제공한다. Temporal B+-트리의 리프노드 구조는 <time, offset, trajectory_ptr>로 구성된다. time은 이동객체가 움직인 시간을 나타내며 Temporal B+-트리에서 키로 쓰인다. time을 키로 사용할 경우 키가 중복될 수

있기 때문에 Temporal B+-트리에서는 키의 중복을 허용한다. offset은 궤적 레코드 상에서 몇 번째 엔트리에 삽입이 되어 있는지를 나타내는 값이며, trajectory_ptr은 삽입된 이동객체의 전체 궤적이 저장되어 있는 궤적 레코드의 포인터이다.

3.2.4 B+-트리 리스트 테이블

Temporal B+-트리는 이동객체의 이동정보인 세그먼트를 시간을 키로 하여 색인한다. 그러나 모든 이동객체를 하나의 B+-트리에 색인하는 것은 많은 리프노드의 수로 인해서 매우 비효율적이게 된다. 따라서 하나의 B+-트리가 유지할 이동객체의 최대 세그먼트 수를 설정하고, 이동객체의 세그먼트 최대 수가 초과할 때 마다 새로운 B+-트리를 생성한다. 이를 통해 질의처리 시 해당되는 B+-트리에 접근하여 적은 수의 탐색 대상을 통하여 효율적으로 처리할 수 있다. B+-트리 리스트 테이블은 <start time, end time, #_seg, btree_ptr>로 구성된다. start time은 해당 Temporal B+-트리가 유지하고 있는 이동객체들의 세그먼트에서 제일 처음의 시간을 나타낸다. end time은 해당 Temporal B+-트리가 유지하고 있는 이동객체들의 세그먼트에서 제일 마지막 시간을 나타낸다. #_seg는 하나의 B+-트리가 현재 유지하고 있는 이동객체의 세그먼트 수를 나타내며, btree_ptr은 해당되는 B+-트리를 연결하는 포인터이다.

3.2.5 MOID B+트리

Temporal B+-트리를 통해 이동객체의 이동정보인 세그먼트를 궤적 레코드에 저장할 때, 새로운 이동객체인 경우는 새로운 궤적 레코드를 생성하여 세그먼트 정보 저장한다. 그러나 이미 이동객체의 궤적 레코드가 존재하는 경우는 해당 궤적 레코드로 접근하여 다음 entry에 세그먼트 정보를 저장하게 된다. 따라서 이동객체가 이미 저장되어 있는지 여부를 판단하고, 세그먼트가 저장될 위치를 알기 위해서 MOID B+-트리를 구성한다. MOID B+-트리의 리프노드 구조는 <MOID, trajectory_ptr>로 구성된다. MOID는 이동객체의 아이디이며 trajectory_ptr은 이동객체의 세그먼트를 저장할 궤적 레코드를 연결하는 포인터이다.

3.2.6 이동객체의 궤적 레코드

시간의 흐름에 따른 이동객체의 움직임 정보를 저장하기 위하여 이동객체의 궤적 레코드를 구성한다. Temporal B+-트리는 이 궤적 레코드와 연결되어 있으며 이동객체에 대한 질의 처리는 궤적 레코드와 연결된 Temporal B+-트리 접근을 통해서 수행된다. 이동객체의 궤적 레코드는 삽입되는 이동객체의 이동정보인 세그먼트들을 일정한 수(N)만큼 가지고 있으며, 그 수를 초과하면 새로운 궤적 레코드를 만들고 이동객체의 궤적을 저장한다. 이동객체의 궤적 레코드는 <MOID,traj_num, next_ptr, prev_ptr, signature_ptr, {segment1, segment2, ...}>와 같이 구성되어 있다. 여기서 MOID는 이동객체의 아이디를 나타낸다. traj_num은 현재 현재 궤적 레코드가 유지하고 있는 이동객체 궤적의 세그먼트의 개수를 의미한다. next_ptr과 prev_ptr은 이동객체의 현재 레코드의 궤적보다 미래의 시간을 prev_ptr은 과거의 시간을 갖는 궤적이 저장된다. 이러한 포인터를 사용하는 이유는 같은 이동객체의 아이디를 갖는 궤적을 전체적으로 유지하기 위해서이다. signature_ptr은 궤적에 대한 시그니처 정보를 유지하고 있는 궤적 시그니처 레코드를 연결한다. {segment1, segment2, ...}는 이동객체 궤적의 세그먼트 집합을 나타낸다. 이동객체의 이동정보인 세그먼트는 <t1, e1, x, y, direction>으로 구성된다. 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, 이동 객체의 에지 아이디나 좌표 중 어떤 것을 요구할지 모르기 때문이다. 아울러, 이동객체의 x, y 좌표는 일정 시간마다 샘플링하여 얻게 되며, 따라서 x, y 좌표는 샘플링 시점의 지도상에서의 절대좌표가 된다. 이동객체의 x, y좌표

를 통해서 이동객체가 어느 에지 상에 있는지 확인한 후, 이동정보인 세그먼트에 반영된다. direction은 이동객체가 움직이고 있는 방향을 나타낸다.

3.2.7 이동객체의 궤적 시그니처 레코드

이동객체 궤적의 시그니처 레코드를 통해서 이동객체의 궤적간의 효율적인 비교를 위해서 이동객체의 궤적 시그니처를 유지하는 궤적 시그니처 레코드를 구성한다. 이동객체의 궤적 시그니처 레코드의 구성방법은 다음과 같다. 첫째, 궤적을 이루는 각 에지 아이디를 얻는다. 둘째, 에지 아이디를 이용하여 해시함수를 통해 각 에지의 시그니처를 생성한다. 마지막으로, 생성된 모든 에지의 시그니처를 OR 연산하여 궤적에 대한 시그니처를 얻는다. <표 2>는 궤적 T1의 에지 구성을 나타낸다. 즉, 궤적 T1은 {e1, e2, e3, e4}의 에지들로 이루어져 있다. <표 3>을 통해 각 에지의 에지 아이디인 12, 15, 17, 22를 얻는다. 각 에지아이디를 해시함수를 통해서 각 에지의 시그니처를 생성한다. 각 에지의 시그니처에 대해 OR 연산을 통해서 <표 4>의 궤적 T1의 시그니처를 구성한다. 이는 질의처리 시 각 에지별 검사가 아닌 궤적의 시그니처 간 비트검사를 통해 효율적으로 궤적을 비교할 수 있다.

<표 2> 이동객체 궤적의 에지 구성

궤적	에지셋
T1	{e1, e2, e3, e4}

<표 3> 각 에지에 대한 시그니처

에지	에지아이디	시그니처
e1	12	01000001
e2	15	01001000
e3	17	00101000
e4	22	10000010

<표 4> 궤적 T1의 시그니처

궤적	시그니처
T1	1101011

4. 삽입 알고리즘

본 장에서는 3장에서 소개한 SigMO-트리의 전체 구조를 통해 데이터를 삽입하는 알고리즘을 설계한다. SigMO-트리의 삽입은 크게 두 단계로 이루어진다. 첫째, 공간 네트워크 색인을 위해 공간 네트워크를 구성하고 있는 에지들을 삽입하고, 둘

```
InsertNetwork()
1. Create_NodeAdjacencyListTable()
2. Create_EdgeAdjacencyListTable()
3. NodeSet = Read_NodeInfo(NodeFile) // read node information from file
4. For each node in NodeSet {
5.      Insert_NodeAdjListTable(node) // insert to Node Adjacency List Table
6. }
7. EdgeSet = Read_EdgeInfo(EdgeFile) // read edge information from file
8. for each edge in EdgeSet {
9.      Node1 = FindNode1(edge's node1 ID) from NodeAdjacencyList Table
10.     Node2 = FindNode2(edge's node2 ID) from NodeAdjacencyList Table
11.     Insert_EdgeAdjListTable(Node1, Node2)
12.     Calculate_MBR(Node1 coordinate, Node2 coordinate)
13.     RTreePtr = Insert_MBR(MBR) // insert MBR to 2DR*Tree
14. }
```

<그림 3> 공간 네트워크 삽입 알고리즘

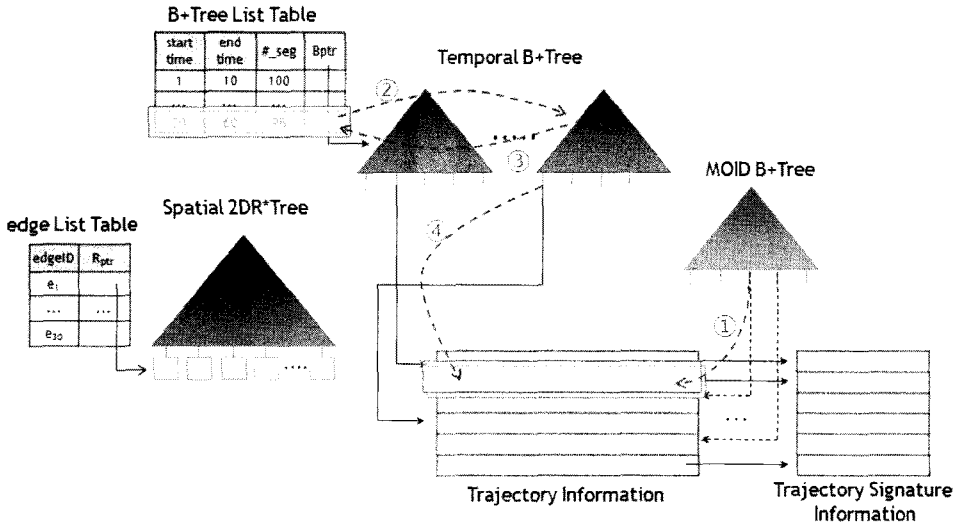
째 공간 네트워크상을 움직이는 이동객체를 삽입하는 것이다. 1절에서는 공간 네트워크의 삽입 알고리즘을 서술하며, 2절에서는 이동객체의 궤적 삽입 알고리즘을 서술한다.

4.1 공간 네트워크 삽입 알고리즘

공간 네트워크 삽입은 공간 네트워크의 구성요소인 에지를 Spatial 2DR*-트리와 에지 리스트 테이블에 삽입하는 것으로 이루어진다. 공간 네트워크의 삽입은 이동객체의 삽입 이전에 이루어지므로 본 논문에서는 공간 네트워크의 삽입이 전처리 작업에 의해 미리 저장 구조에 삽입이 되는 것을 가정한다. <그림 3>는 공간 네트워크 삽입 알고리즘을 나타낸다. 공간 네트워크의 삽입을 위해 먼저, 에지의 개수와 노드의 개수만큼 노드 테이블과 에지 테이블을 생성한다. 노드 정보 파일(NodeFile)에는 해당 노드의 좌표와 노드 아이디가 구성되어 있다. 노드 정보를 파일로부터 읽어서 노드 테이블에 노드를 삽입한다. 아울러 에지 정보를 파일(EdgeFile)로부터 읽어 에지를 구성하는 노드의 아이디를 이용하여, 이미 구성된 노드 테이블에서 아이디가 일치하는 노드를 찾는다. 노드를 찾게 되면 에지 연결리스트에 현재 에지를 저장한다. 그 후 에지를 이루는 두 노드의 좌표정보를 읽어 에지의 MBR을 구성하고 이를 Spatial 2DR*-트리에 삽입한다.

4.2 이동객체의 궤적 삽입 알고리즘

먼저, 이동객체의 궤적을 삽입할 때 삽입이 될 이동객체의 이동 정보는 지나간 에지 아이디를 가지고 있다고 가정한다. 이동객체의 삽입은 크게 두 단계로 이루어진다. 먼저 MOID B+-트리를 통해 이동객체가 삽입될 궤적 레코드를 얻어온 후 B+-트리 리스트 테이블을 통해 이동객체가 삽입될 Temporal B+-트리를 검색하여 Temporal B+-트리와 궤적 레코드와 연결한다. <그림 4>는 이동객체의 궤적 삽입 알고리즘의 과정을 보여준다. 이동객체의 삽입의 자세한 과정은 다음과 같다. 첫째, <그림 4>의 ①과 같이 이동객체의 식별자로 MOID B+-트리를 탐색한다. 둘째, 만일 이동객체의 식별자를 통해 이전의 이동객체의 궤적 세그먼트 정보가 있다면 이동객체의 이동정보를 저장하고 이동정보가 저장된 궤적 레코드의 위치를 얻어온다. 이동객체의 궤적이 이미 저장되어 있으므로, 기존 이동객체의 이동 시간 정보의



<그림 4> SigMO-트리의 이동객체의 삽입 과정

삽입을 위해 <그림 4>의 ②와 같이 B+트리 리스트 테이블에서 이동정보의 이동시간(time)을 통하여 이동객체를 색인하게 될 적합한 Temporal B+-트리(btree_ptr)를 찾는다. B+-트리 리스트 테이블에서 사전에 설정된 하나의 B+-트리당 유지할 궤적의 세그먼트 수(#_seg)를 초과하지 않은 Temporal B+-트리를 찾는다. 만약 초과하지 않은 Temporal B+-트리가 없다면 이동객체를 색인할 새로운 Temporal B+-트리를 생성하고 그 위치를 B+-트리 리스트 테이블의 btree_ptr에 저장한다.

셋째, 이동객체 삽입을 위한 B+-트리를 찾으면 <그림 4>의 ③과 같이 Temporal B+-트리의 단말노드(<time, offset, trajectory_ptr>)에 이동객체 궤적의 세그먼트 타임스탬프(time) 정보를 삽입한다. 궤적의 세그먼트가 삽입된 후 B+-트리 리스트 테이블에서 해당 B+-트리의 궤적 세그먼트 수(#_seg)를 증가시킨다. 넷째, Temporal B+-트리에 저장되는 이동객체의 offset, trajectory_ptr정보

```
InsertMovingObject(int moid, TrajSegment)
Input : 이동객체의 ID, 이동객체의 이동정보
 // ret는 Trjaectory record를 가리키는 pointer
 1. ret = Find_MOID(moid) from MOID B+-Tree
 2. if(ret≠NULL){ // 이전에 삽입된 moid가 있는 경우
 3.     Traj_Ptr = Get_TrajectoryRecord(ret) // moid에 연결된 궤적정보 요청
 4. } else { // 처음 이동을 시작한 moid의 경우
 5.     Record = Create_TrajectoryRecord() // 이동정보를 저장할 Record를 생성
 6.     // 생성된 Record에 moid의 이동정보인 TrajSegment를 세팅
 7.     Set_Record(Recod, TrajSegment)
 8.     // 궤적 레코드에 새로운 Record를 삽입 후, 해당 위치를 리턴
 9.     Traj_Ptr = Insert_TrajectoryRecord(Record)
10. }
11.  // B+Tree List Table로부터 삽입될 Temporal B+-Tree를 검색
12. BTree = Find_BTree(TrajSegment.time)
13. if(BTree.count < MaxItem) { // BTree.count 는 검색된 B+-트리의 저장 entry 개수
14.     BTree_Ptr = Get_TemporalBTree(BTree) // Temporal B+-트리의 pointer를 검색
15. }
16 else { // MaxItem을 초과한 경우
17.     BTree_Ptr = Create_TemporalBTree(TrajSegment.time);
18. }
19. Insert_BTree(BTreePtr, TrajSegment.time)
20. // MOID B+-트리로부터 얻은 이동객체의 궤적 레코드의 위치와
21.   Temporal B+-트리 테이블의 trajectory_ptr 정보에 연결
22. Set_BTree_TrajectroyPtr(Traj_Ptr)
23. Sig_Ptr = Update_TrajSignature()  // 시그니처 정보 업데이트
24. Set_SignaturePtr(Sig_Ptr) // Trajectory Information Table의 시그니처 pointer 설정
```

<그림 5> SigMO-트리의 이동객체의 궤적 삽입 알고리즘

는 MOID B+-트리로부터 얻어온 실제 궤적 레코드의 위치인 실제 궤적 레코드와 <그림 4>의 ④에 의해서 실제 궤적 레코드와 연결된다. 마지막으로, <그림 4>의 Trajectory Signature Information는 ④에 의해서 저장된 궤적 레코드들을 통해 해당 이동객체의 궤적을 구성하는 에지의 아이디를 이용하여 시그니처를 생성하고 저장한다. 그 후 저장된 시그니처 정보의 위치가 궤적 레코드의 signature_ptr로 연결된다. <그림 5>은 이동객체의 궤적 삽입 알고리즘을 나타낸다.

5. 질의 처리 알고리즘

3장에서 공간 네트워크상의 이동객체의 특성을 고려하여 질의를 3가지 즉, 시공간영역 내 궤적질의, 시간영역 내 유사궤적 질의, 연속적 k-최근접 질의로 분류하였다. 본 장에서는 분류한 질의들 중, 시공간영역 내 궤적질의, 시간영역 내 유사궤적 질의를 위한 질의 처리 알고리즘을 제시한다. 제안하는 시공간 유사 궤적 검색 알고리즘은 효과적인 검색을 위하여 시그니처 기반 액세스 기법을 사용한다. 이를 통해 불필요한 데이터 궤적의 검색을 효율적으로 줄일 수 있다. 5.1절에는 시그니처 기반 궤적 탐색 알고리즘을 제시하고 5.2 절에서는 시공간영역 내 궤적질의 처리 알고리즘을 제시하고, 5.3절에서는 시간영역 내 유사궤적 질의 처리 알고리즘을 제시한다.

5.1 시그니처 기반 궤적 비교 알고리즘

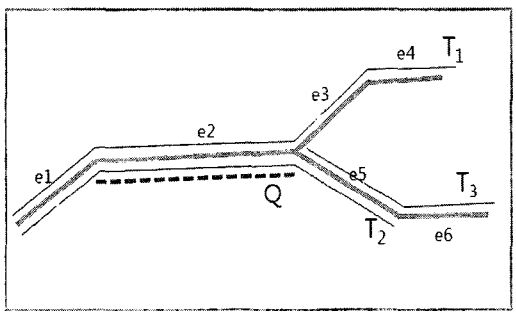
시공간영역 내 궤적질의와 시공간 유사 궤적 질의는 질의 궤적과 과거 궤적간의 비교를 통해서 질의를 만족하는 궤적들을 검색한다. 그러나 많은 수의 에지로 이루어져 있는 궤적들을 일일이 비교한다는 것은 매우 큰 비용이 든다. 따라서 이를 효과적으로 비교하기 위해 시그니처 기반 궤적 비교 알고리즘을 나타낸다. 첫째, 주어진 궤적 Q의 각 에지에 대한 시그니처를 생성한다. 둘째, 시간영역을 만족하는 과거 이동객체들의 궤적의 시그니처를 생성한다. 생성된 과거 이동객체들의 궤적 시그니처와 비트 AND연산을 통하여 질의 에지를 지나는 이동객체의 궤적을 구한다. 시그니처 기반 궤적 탐색 알고리즘은 <그림 6>과 같다.

예를 들어, <그림 7>과 같은 질의 궤적 Q와 과거 궤적들이 주어졌을 때를 가정하자. 모든 궤적들은 <표 5>에 나타난바와 같은 에지의 순서로 구성된다.

```
Trajectory_Comparison_Algorithm(Q, DT){
Input : 질의 궤적 Q={S1, S2,…,Sn}, 과거 궤적 집합 DT={T1, T2,…, Tm)
Output : 질의 궤적 Q를 지나는 데이터 궤적 집합
 1. ResultSet = Φ;
 2. Qsignature = generate_signature(Q)  // 질의궤적의 시그니처를 생성
 3. For each Ti in DT
 4.     // 질의궤적과 과거궤적의 시그니처 &연산의 결과가
 5.       질의궤적 시그니처와 동일한 경우 결과셋으로 설정
 6.     Ti_signature = generate_signature(Ti)  // 과거 궤적의 시그니처를 생성
 7.     if(Qsignature & Ti_signature == Qsignature)
 8.           ResultSet = ResultSet U Ti
 9.   }
10. }
```

<그림 6> 시그니처 기반 궤적 탐색 알고리즘

예를 들어 질의 궤적 Q={e2, e3}의 2개의 에지로 구성된다. 궤적 비교를 위해 첫째, 질의 궤적의 각 에지에 대한 시그니처를 생성한다. 질의 궤적에 대한 시그니처는 <표 6>에 나타난바와 같이 Qsignature는 {01001000, 00100100}의 비트 OR 하여 최종 시그니처, {01101100}로 구성된다. 마찬가지로 과거궤적 T1, T2, T3의 시그니처는 각각 {11101011}, {01101101}, {10100100}로 구성된다. 둘째, 과거 궤적 T1, T2, T3에 대하여 Qsignature의 시그니처와 비트 AND연산 후 그 결과가 질의궤적의 시그니처인 Qsignature와 동일여부를 통해 질의 에지를 지나는 이동객체의 궤적을 구한다. 따라서 궤적 탐색을 통해 생성되는 결과 집합 ResultSet={T2}이 된다.



<그림 7> 질의 궤적 q와 과거궤적의 예

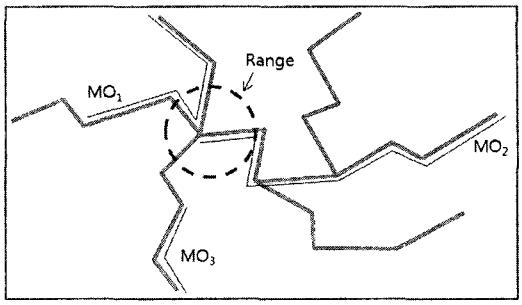
<표 5> 질의 궤적과 데이터 궤적의 구성

궤적	에지셋
Q	{e2, e5}
T1	{e1, e2, e3, e4}
T2	{e1, e2, e5}
T3	{e5, e6}

<표 6> 각 에지에 대한 시그니처

에지	시그니처
e1	01000001
e2	01001000
e3	00101000
e4	10000010
e5	00100100
e6	10000100

5.2 시공간영역 내 궤적질의 처리 알고리즘



<그림 8> 시공간영역 내 궤적질의의 예제

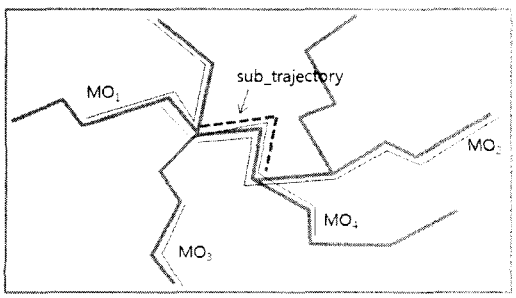
시공간영역 내 궤적질의는 먼저 시공간영역 안에 존재하는 이동객체들이 현재까지 또는 특정한 시간범위 안에 움직인 궤적을 찾는 질의이다. 시공간영역 내 궤적질의를 위해서는 공간영역과 궤적의 시간범위가 주어져야 한다. 궤적의 시간범위가 주어지지 않을 경우에는 이동객체의 모든 궤적을 찾는 것으로 가정한다. 시공간영역 내 궤적질의의 예제는 <그림 8>과 같다. MO 1은 오전 6시에 출발하여 오전 10시까지의 이동궤적을 나타낸다. MO 2는 오전 10시에 출발하여 현재시간인 오후2시까지의 이동궤적을 나타낸다. 마지막으로 MO 3은 오전 11시에 출발하여 오후 1시까지의 이동궤적을 나타낸다. 공간 범위 영역과 이동객체의 궤적을 비교하여 공간 범위를 지나는 이동객체의 궤적을 얻는다. 그 결과 MO1, MO2 또는 그 궤적이 반환된다. 그 후, 오전 11시부터 오후1시의 시간범위가 주어졌을 때, 시간범위를 만족하는 이동객체는 MO2가 된다. 따라서 시공간영역 내 궤적질의의 결과는 MO2가 된다. SigMO-트리에서 시공간영역 내 궤적 질의를 처리하기 위한 알고리즘은 다음과 같다. 첫째, 2DR*-트리를 통해서 주어진 시공간영역의 공간영역을 만족하는 공간 네트워크상의 에지들을 검색한다. 둘째, 검색된 각 에지에 대하여 시그니처를 생성한다. 셋째, B+-트리 리스트 테이블에서 주어진 시간범위로 탐색할 Temporal B+-트리를 찾는다. 넷째, 찾아진 Temporal B+-트리를 통해 주어진 시간영역으로 검색한다. 주어진 시간범위에 단말노드들이 존재하면, 각 단말노드들

```
SpatialTemporalRangeAndTrajectoryQueryAlgorithm(SR, TR){
Input : 공간영역 SR, 시간범위 TR
Output : 시공간영역을 만족하는 이동객체
 1. ResultSet = Φ;
 2. EdgeSet = RangeSearch_RTree(SR) // Spatial 2DR*트리로 부터의 영역질의
 3. For each edge in EdgeSet {
 4.     Qsignature[i] = generate_signature(edge) // 에지의 시그니처 생성
 5.     BTreePtr = Find_BTree(TR) from B+-트리 List Table
 6.     TemporalResultSet = Search_BTree(TB) from BTreePtr's Temporal B+트리
 7.     For each temporal_result in TemporalResultSet {
 8.             Traj_Records = Find_Trajectory(Trajectory_Ptr of temporal_result)
 9.             Traj_Signature[j] = Get_Signature(traj_record) from Signature Information
10.    }
11.    ResultSet = Trajectory_Comparison_Algorithm(Qsignature[i], Traj_Signature)
12. }
```

<그림 9> 시공간영역 내 궤적 질의 처리 알고리즘

의 레코드 포인터(trajectory_ptr)를 이용하여 이동객체의 궤적 레코드에 접근한다. 넷째, 각 이동객체의 궤적정보를 표현하고 있는 시그니처 정보를 공간영역 질의의 결과인 에지의 시그니처와 비교해서 시공간영역을 만족하는 이동객체의 궤적 레코드를 얻어온다. 마지막으로, 얻어온 궤적 레코드를 통해 에지 리스트 테이블을 통해 실제 에지의 좌표 정보를 반환한다. 시공간영역 내 궤적 질의 처리 알고리즘은 <그림 9>과 같다.

5.3 시간영역 내 유사궤적질의 처리 알고리즘



<그림 10> 시간영역 내 유사 궤적질의 예제

시간영역 내 유사궤적 질의는 특정한 에지들을 주어진 시간영역 동안에 유사궤적을 지난 이동객체 또는 그들의 궤적을 찾는 질의이다. <그림 10>은 주어진 두 개의 에지를 지나는 이동객체들의 궤적을 찾는 시간영역 내 유사궤적 질의의 예제를 나타낸다. MO 1은 오전 6시에 출발하여 오전 10시까지의 이동궤적을 나타낸다. MO 2는 오전 10시에 출발하여 현재시간인 오후2시까지의 이동궤적을 나타낸다. 마지막으로 MO 3은 오전 11시에 출발하여 오후 1시까지의 이동궤적을 나타낸다. 4개의 궤적과 점선으로 표시된 입력 궤적을 비교하여 입력된 궤적과 유사한 이동객체의 궤적을 찾을 수 있다. 그 결과 MO2, MO4의 궤적이 반환된다. 그 후, 오전 11시부터 오후1시의 시간범위가 주어졌을 때, 시간범위를 만족하는 이동객체는 MO2가 된다. 따라서 시공간영역 내 궤적질의의 결과는 MO2가 된다. SigMO-트리에서 시간영역 내 유사궤적 질의 알고리즘은 다음과 같다. 첫째, 입력받은 에지리스트의 시그니처를 생성한다. 둘째, B+-트리 리스트 테이블에서 주어진 시간영역으로 탐색할 Temporal B+-트리를 찾는다. 셋째, 찾아진 Temporal B+-트리를 통해 주어진 시간영역으로

```
SimilarTrajectoryAndTemporalRangeQuery(Edge_list, TR)
Input : 유사궤적 비교를 위한 Edge_list, 시간범위 TR
Output : 시간범위 TR에 Edge_list를 지나는 이동객체
 1. ResultSet = Φ;
 2. Qsignature = generate_signature(Edge_list) // Edge_list의 시그니처 생성
 3. BTreePtr = Find_BTree(TR) from B+-트리 List Table
 4. TemporalResultSet = Search_BTree(TB) from BTreePtr's Temporal B+트리
 5.     For each temporal_result in TemporalResultSet {
 6.       Traj_Records = Find_Trajectory(Trajectory_Ptr of temporal_result)
 7.       Traj_Signature[j] = Get_Signature(traj_record) from Signature Information
 8              ResultSet = ResultSet U Trajectory_Comparison_Algorithm(Qsignature,
Traj_Signature[j])
 9.     }
 10. }
```

<그림 11> 시공간영역 내 궤적 질의 처리 알고리즘

검색한다. 주어진 시간영역에 단말노드들이 존재하면, 단말노드들의 레코드 식별자(trajectory_ptr)를 이용하여 이동객체의 궤적 레코드에 접근한다. 넷째, 각 이동객체의 시그니처 정보를 입력받은 에지 리스트의 시그니처와 비교해서 시공간영역을 만족하는 이동객체의 궤적 레코드를 얻어온다. 마지막으로, 얻어온 궤적 레코드를 통해 에지 리스트 테이블을 통해 실제 에지의 좌표 정보를 반환한다. 시공간영역 내 궤적 질의 처리 알고리즘은 <그림 11>과 같다.

6. 성능평가

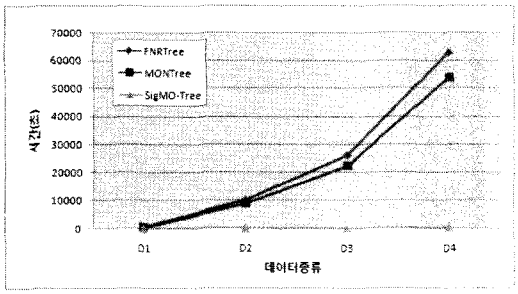
본 장에서는 본 논문에서 설계한 SigMO-트리와 기존의 연구인 FNR-트리, MON-트리와 성능평가를 수행한다. SigMO-트리를 구현한 환경은 Visual C로 작성하였으며, Intel Xeon 3.0GHz CPU와 메인 메모리 2GB, 윈도우 2003 상에서 구현하였다. 성능평가에 사용한 데이터는 Brinkhoff가 제안한 알고리즘[5]을 이용하여 이동객체를 생성하였으며, 공간 네트워크 데이터는 Brinkhoff가 제안한 알고리즘에서 제공하는 공간 네트워크 데이터중의 하나인 샌프란시스코 만 데이터를 사용하였다. 샌프란시스코 만 공간 네트워크의 에지의 수는 약 220,000개이며, 노드의 수는 약 170,000개를 가지고 있다. 이동객체는 <표 7>과 같이 구성하여 생성하였다. 이동객체의 궤적 세그먼트의 이동객체의 수는 5,000, 10,000, 20,000, 50,000개로 주었으며, 시간은 1000으로 설정하여 100,000, 200,000, 500,000, 1,000,000개의 이동객체의 궤적을 생성하였다. 각각의 데이터는 D1, D2, D3, D4로 명명한다.

<표 7> 생성된 이동객체의 궤적 세그먼트

	세그먼트 개수	이동객체 개수	타임스탬프 개수
D1	100,000	5,000	1,000
D2	200,000	10,000	1,000
D3	500,000	20,000	1,000
D4	1,000,000	50,000	1,000

사용한 R*-트리와 B+-트리의 디스크 페이지 크기는 1KB 이며, 궤적 레코드의 페이지 크기 역시 1KB로 설정하였다. 성능평가 항목은 각각 트리의 삽입시간, 저장공간 오버헤드, 영역질의 응답시간, 유사궤적 질의 응답시간을 측정한다.

6.1 삽입 성능평가

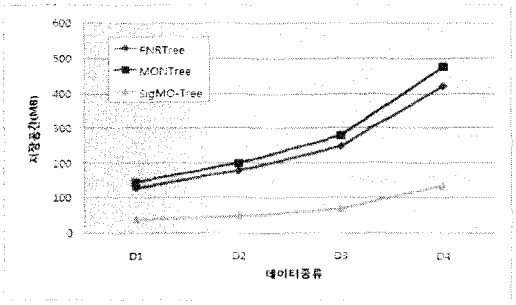


<그림 12> 삽입 성능평가

본 절에서는 FNR-트리, MON-트리, SigMO-트리의 삽입 성능평가를 수행한다. MON-트리는 경로 모델의 경우 이동객체의 유사궤적 질의처리를 위해 도로 네트워크상에 존재하는 모든 에지들을 통해 만들어 지는 모든 경로(Route)를 삽입해야 하며, 이는 삽입성능 측면에서 매우 비효율적이므로 본 연구에서는 에지 모델을 선택하여 성능을 평가한다. <그림 12>은 기존방법과 제안하는 방법의 삽입성능 비교를 나타낸다. <그림 12>에서 나타난 바와 같이 D1 데이터셋의 경우 삽입시간이 FNR-트리는 약 730초이고 MON-트리는 약 522초, SigMO-트리는 약 56초가 소요된다. D4 데이터 셋의 경우 FNR-트리가 약 62740초, MON-트리가 약 53832초 SigMO-트리가 약 533초가 소요 되어, SigMO-트리가 MON-트리, FNR-트리에 비해서 삽입시간에 있어 우수함을 알 수 있다. 이는 FNR-트리의 경우 이동객체의 궤적이 삽입되면 그 이동객체의 궤적이 어떤 에지를 지나는지 2DR-트리를 검색해야 하며,1DR-트리로 삽입을 하기 위해 다시 1DR-트리를 접근해야 하는 오버헤드가 존재하기 때문이다. 즉 트리를 두 번 접근해야 하는 오버헤드로 인해 삽입 성능이 다른 트리에 비해 저하된다. MON-트리는 FNR-트리와 달리 삽입될 위치를 저장하고 있는 해시 테이블을 유지함으로써 1DR-트리를 접근해야 하는 오버헤드를 줄여 FNR-트리보다 더 좋은 삽입 성능을 보인다.

SigMO-트리는 이동객체의 궤적이 삽입되면 MON-트리처럼, 어떤 에지를 지났는지를 찾기 위해 2DR-트리를 검색한 후 해당 에지에 연결된 트리에 궤적을 저장하지 않고, 시간에 따른 B+-트리에 바로 이동객체의 궤적을 저장한다. 따라서 에지를 찾기 위한 2DR-트리를 탐색하는 비용이 소요되지 않는다. 또한 색인하는 구조가 MON-트리는 1DR-트리인 반면 SigMO-트리는 B+트리로 색인을 한다. 따라서 B+-트리로 색인하는 SigMO-트리가 MON-트리보다 더 좋은 삽입 성능을 보인다. 또한 이러한 이유로 SigMO-트리와 MON-트리간의 성능의 차이가 MON-트리와 FNR-트리간의 성능 차이보다 더 커진다.

6.2 저장공간 오버헤드

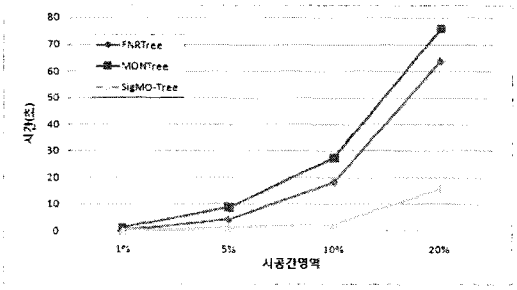


<그림 13> 저장 공간 오버헤드

본 절에서는 FNR-트리, MON-트리, SigMO-트리가 요구하는 디스크 상에서의 저장 공간 오버헤드를 통한 성능평가를 수행한다. <그림 13>은 기존방법과 제안하는 방법의 저장 공간 오버헤드의 비교를 나타낸다. <그림 13>에서 나타난 바와 같이 D1 데이터셋의 경우 저장 공간이 MON-트리는 약 146MB이고 FNR-트리는 약 130MB, SigMO-트리는 약 39MB가 소요된다. D4 데이터 셋의 경우 MON-트리가 약 478MB, FNR-트리가 약 423MB, SigMO-트리가 약 136MB가 소요 되어, SigMO-트리가 MON-트리, FNR-트리에 비해서 저장 공간에 있어 우수함을 알 수 있다. 이는 MON-트리의 경우 FNR-트리의 이동객체 삽입 시 성능개선을 위하여 별도의 해시테이블 구조가 존재하기 때문에 FNR-트리보다 많은 저장 공간을 필요로 한다. 또한, FNR-트리는 색인을 이루는 키의 값이 시간범위인

1DR-트리를 사용하는 반면, SigMO-트리는 키가 하나의 점으로 표현되는 B+-트리로 색인하기 때문에 보다 적은 저장 공간 오버헤드를 가진다.

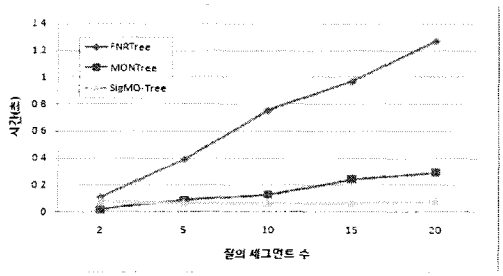
6.3 시공간 영역 내 궤적질의 성능평가



<그림 14> 시공간 영역 내 궤적질의 성능평가

본 절에서는 FNR-트리, MON-트리, SigMO-트리에 대한 영역질의 성능평가를 수행한다. 본 논문에서 제시한 시공간 영역 내 궤적질의는 FNR-트리와 MON-트리가 이동객체에 대한 전체 궤적 레코드를 유지하지 않기 때문에, 성능평가를 FNR-트리와 MON-트리가 지원할 수 있는 영역질의로 제한하였다. 영역질의 성능 비교를 위해 1%, 5%, 10%, 20%의 검색 영역에 대한 검색 시간을 비교하였다. 영역 질의 성능평가에 사용한 데이터는 D4 데이터셋이다. <그림 14>는 영역질의 성능을 비교한 그래프이다. <그림 14>에서 나타난 바와 같이 1% 영역질의에서 FNR-트리가 약 0.4초, MON-트리가 1.3초, SigMO-트리가 약 0.07초가 소요되며, 20% 영역질의에서는 FNR-트리가 약 63초, MON-트리가 약 75초, SigMO-트리가 약 16초로 SigMO-트리가 MON-트리, FNR-트리에 비해서 영역질의 응답시간에 있어서 우수함을 알 수 있다. 또한, SigMO-트리가 MON-트리와 FNR-트리에 비해 약 4배정도 성능이 향상됨을 알 수 있다. 이는 FNR-트리와 MON-트리의 경우 질의 에지를 지나는 궤적을 검색하기 위해서 후보 궤적의 모든 에지들과 질의 에지와의 일대일 비교를 수행해야 한다. 반면, SigMO-트리는 후보 궤적에 대한 시그니처를 유지하고 있기 때문에 후보 궤적들과 질의 간의 한 번의 비교만으로 질의 에지를 지나는 궤적을 검색할 수 있다. 또한 SigMO-트리가 이동객체의 궤적을 B+-트리로 색인하기 때문에, B+-트리의 특성상 궤적을 검색할 때 B+-트리 리프노드의 링크를 따라 순차 검색한다. 하지만 FNR-트리나 MON-트리의 경우는 1DR-트리를 사용하기 때문에, 주어진 시간영역을 지나는 이동객체의 궤적을 가진 리프노드를 찾은 후 다시 상위노드를 검색하는 재귀적인 방법을 사용한다. 따라서 영역이 작을 때는 비슷한 성능을 보이지만, 영역이 커질수록 SigMO-트리의 질의 응답시간이 보다 우수하다. 그리고 FNR-트리보다 MON-트리의 영역질의 성능이 더 저하되는 이유는 MON-트리는 에지를 따로 저장하는 데이터 레코드의 구조가 있기 때문에 레코드를 한 번 더 접근하는 오버헤드가 있는 반면, FNR-트리는 직접 1DR-트리와 연결되기 때문에 이러한 오버헤드가 발생하지 않는다.

6.4 시간 영역 내 유사궤적질의 성능평가



<그림 15> 시간 영역 내 유사궤적질의 성능평가

본 절에서는 FNR-트리, MON-트리, SigMO-트리에 대한 시간영역 내 유사궤적 질의 성능평가를 수행한다. 시간 영역 내 유사궤적질의는 유사궤적의 에지 개수를 2, 5, 10, 15, 20으로 변화시키고, 시간영역은 전체 시간영역의 0.5%로 고정시켜 성능평가를 수행한다. <그림 15>는 시간 영역 내 유사궤적질의 성능평가를 나타낸다. <그림 15>에서 나타난 바와 같이 유사궤적의 에지 개수가 2개 일 때는 FNR-트리가 약 0.1초, MON-트리가 0.01초, SigMO-트리가 0.07초이다. 그리고 유사궤적 에지

개수가 20개 일 때는 FNR-트리가 약 1.2초, MON-트리가 0.3초로 증가한다. 그러나 SigMO-트리는 에지 개수가 2개 일 때와 마찬가지로 0.07초의 응답시간을 보인다. 이는 FNR-트리의 경우 MON-트리나 SigMO-트리와 같이 에지 아이디를 바로 접근할 수 없기 때문에 성능이 저하된다. SigMO-트리가 MON-트리보다 더 빠른 검색 성능을 보이는 이유는 첫째, 이동객체의 궤적을 색인하는 트리로서 MON-트리는 1DR-트리를 사용하는 반면, SigMO-트리는 B+-트리를 사용하여 시간 영역 탐색시간이 감소하기 때문이다. 둘째, FNR-트리와 MON-트리의 경우 질의 궤적을 지나는 궤적을 검색하기 위해, 후보 궤적 및 질의 궤적의 모든 에지간의 일대일 비교를 수행해야 하는 반면, SigMO-트리는 후보 궤적에 대한 시그니처를 유지하기 때문에 질의 세그먼트의 수와 관계없이 후보 궤적들과 질의 궤적과의 한 번의 비교만으로 질의 궤적을 지나는 이동객체의 궤적을 검색할 수 있기 때문이다. 한편, 질의 세그먼트의 수가 2인 경우, SigMO-트리가 MON-Tree보다 성능이 저하 되는 이유는, SigMO-트리는 빠른 질의처리 수행을 위해서 질의처리 수행 전에 후보 이동객체들의 궤적을 시그니처로 변환하는 시간이 부가적으로 필요하기 때문이다. 그러나 시그니처 변환 시간은 질의 세그먼트의 수에 관계없이 거의 일정하기 때문에, 질의 세그먼트의 수가 증가하여도 일정한 질의처리 성능을 보인다.

7. 결론 및 향후연구

본 논문에서는 공간 네트워크상의 이동객체를 위한 시그니처를 이용한 궤적기반 색인 구조인 SigMO-트리를 제안하였다. 공간 네트워크상의 이동객체를 저장 및 색인하기 위하여 공간 데이터와 이동객체의 궤적 데이터 즉 시간 데이터를 분리하여 색인하였다. 이를 위하여 R*-트리와 B+-트리를 사용하여 공간 데이터와 시간데이터를 효율적으로 색인하였고, 시공간 영역 내 궤적 질의, 시간영역 내 유사궤적 질의를 지원하기 위해 시간영역을 색인하기 위한 리스트 테이블과 이동객체의 궤적을 유지하기 위한 구조를 제안하였다. 또한 제시한 질의의 효율적이고 빠른 수행을 위해서 궤적정보에 시그니처 파일방법을 적용하였다. 성능평가를 통하여 본 논문에서 제시한 SigMO-트리가 기존의 공간 네트워크상의 이동객체를 색인하는 구조인 FNR-트리, MON-트리에 비해, 삽입, 저장공간, 영역질의, 시간영역 내 유사궤적질의 측면에서 성능이 우수함을 보였다. 향후 연구로는 궤적질의를 확장한 In-Route 질의와 같은 질의 처리 알고리즘을 구현하여 본 논문에서 제안한 SigMO-트리를 확장하는 연구이다.

참고문헌

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 spatiotemporal 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 Güuting. "Indexing the Trajectories z of Moving Objects in Networks." In Proc. of GeoInformatica 9(1), pp33-60, 2005.
10. 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.
11. Faloutsos, C., and Christodoulakis, S., "Signature Files: An Access Method for Documents and Its Analytical performance Evaluation," ACM Tran. on Office Information Systems, Vol.2 No.4, 1984, pp. 267-288.
12. Zobel, J., Moffat, A., and Ramamohanarao, K., "Inverted Files versus Signature Files for Text Indexing," ACM Tran. on Database Systems, Vol.23 No.4, 1998, pp. 453-490.

김영진
2007년 전북대학교 컴퓨터 공학과(공학사)
2007년~현재 전북대학교 컴퓨터 공학과(석사과정)
관심분야 : 공간 데이터베이스, 공간 색인 구조, GIS
e-mail : yzkim@dblab.chonbuk.ac.kr

김영창
2001년 전북대학교 컴퓨터공학과(공학사)
2003년 전북대학교 대학원 컴퓨터공학과(공학석사)
2004년~현재 전북대학교 대학원 컴퓨터공학과(박사과정)
관심분야 : 데이터 마이닝, 공간 데이터베이스, 공간 색인 구조, 질의처리 알고리즘
e-mail : yckim@dblab.chonbuk.ac.kr

장재우
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
관심분야 : 공간 네트워크 데이터베이스, 상황인식, 하부저장구조
e-mail : jwchang@chonbuk.ac.kr

심춘보
1996년 전북대학교 컴퓨터공학과 졸업(공학사)
1998년 전북대학교 대학원 컴퓨터공학과 졸업(공학석사)
2003년 전북대학교 대학원 컴퓨터공학과 졸업(공학박사)
2004년~2005년 부산가톨릭대학교 컴퓨터정보공학부 전임강사
2005년~현재 순천대학교 정보통신공학부 조교수
관심분야 : 멀티미디어 DB, LBS, 유비쿼터스 컴퓨팅
e-mail : cbsim@sunchon.ac.kr