공간 네트워크 데이터베이스를 위한 저장 및 색인 구조의 설계 및 성능평가

엄 정 호[†] · 장 재 우[††]

요 약

최근 LBS(location-based service)를 지원하기 위해, 공간 네트워크를 고려한 연구가 활발히 수행 중에 있다. 본 논문에서는 공간 네트워크 데이터베이스에서의 우수한 질의처리 성능을 위해, 공간 네트워크상에 존재하는 공간 네트워크 자체의 데이터, POI(point of interest), 이동객체 데이터를 위한 효율적인 저장 및 색인 구조를 설계한다. 이를 위해 첫째, 노드와 에지로 구성된 공간 네트워크 자체의 데이터를 관리하기 위해 공간 네트워크 파일 구조를 설계한다. 둘째, 식당, 호텔, 주유소와 같은 POI 에 대한 효율적인 접근을 위해 POI 저장 및 색인 구조를 설계한다. 셋째 이동객체의 과거, 현재, 미래 궤적 정보를 효과적으로 관리하기 위해 시그니처 기반 저장 및 색인 구조를 설계한다. 마지막으로, 본 논문에서 설계한 저장 및 색인 구조가 기존의 공간 네트워크를 위한 저장구조 및 이동객체를 위한 궤적 색인구조 보다 성능이 우수함을 보인다.

키워드 : 공간 네트워크데이터베이스, 저장 및 색인구조

Design & Performance Evaluation of Storage and Index Structures for Spatial Network Databases

Jung-Ho Um[†] · Jae-Woo Chang[††]

ABSTRACT

For supporting LBS service, recent studies on spatial network databases (SNDB) have been done actively. In order to gain good performance on query processing in SNDB, we, in this paper, design efficient storage and index structures for spatial network data, point of interests (POIs), and moving objects on spatial networks. First, we design a spatial network file organization for maintaining the spatial network data itself consisting of both node and edges. Secondly, we design a POI storage and index structure which is used for gaining fast accesses to POIs, like restaurant, hotel, and gas station. Thirdly, we design a signature-based storage and index structure for efficiently maintaining past, current, and expected future trajectory information of moving objects. Finally, we show that the storage and index structures designed in this paper outperform the existing storage structures for spatial networks as well as the conventional trajectory index structures for moving objects.

Key Words : Spatial Network Database, Storage and Index Structure

1. 서 론

최근 PDA 및 PCS 등의 이동장비의 확산과, GPS등의 통신 기기의 발달로 인하여 이를 이용한 서비스들이 많이 활성화 되고 있다. 그 중에서도 사용자의 위치에 기반한 LBS (location-based service)가 활성화 되고 있으며, 대표적인 응용 서비스로는 텔레메틱스 및 관광 안내 시스템 등이 있다. 최근 LBS를 효과적으로 지원하기 위해, 이상적인 공간 대신, 실제 도로나 철도와 같은 공간 네트워크(network)를 고려한 연구가 활발하게 수행 중에 있다[1-7]. 이러한 공간 네트워크 데이터베이스 연구는 일반 공간 데이터베이스의 연구와 마찬가지로 크게 3가지 주제로 요약된다. 즉, 공간 네트워크를 위한 데이터 모델, 질의 처리 알고리즘, 마지막으로 공간 네트워크 데이터베이스를 위한 저장 및 색인 구조이다. 첫째, 공간 네트워크를 위한 데이터 모델은 그래프로 쉽게 표현할 수 있기 때문에 주로 그래프의 형태로 표현한다[5-6]. 둘째, 공간 네트워크상에서의 질의 처리 알고리즘은 기존의 유클리디안(Euclidean) 공간상에서의 질의 처리 알고리즘과 달리 네트워크 거리를 고려한 범위 질의와 k-nearest neighbor 질의, e-distance join 질의 처리 알고리즘 등이 있다[5]. 마지막으로 공간 네트워크 데이터베이스를 위한 저장 및 색인 구조는 공간 네트워크상에 존재하는 데이터를 저장 및 색인하여 효율적인 질의 처리를 가능하게

※ 본 과제(결과물)는 교육인적자원부, 산업자원부, 노동부의 출연금으로 수행한 최우수실험실지원사업의 연구결과입니다.
† 준 회 원 : 전북대학교 컴퓨터공학과 박사과정
†† 종신회원 : 전북대학교 컴퓨터공학과 교수
논문접수 : 2005년 12월 31일, 심사완료 : 2006년 3월 30일

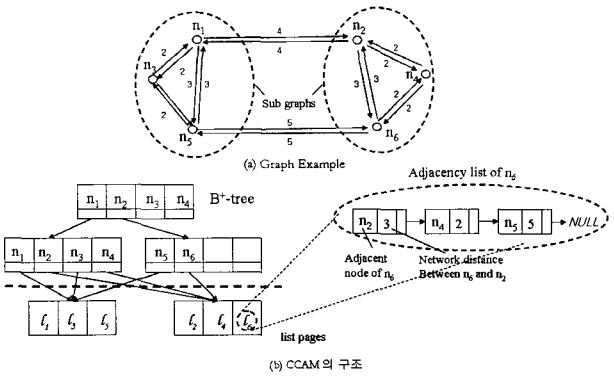
한다.[4] 이는 공간 네트워크 데이터베이스의 성능을 좌우하는 중요한 연구이다.

따라서 본 논문에서는 공간네트워크 데이터베이스를 위한 세 가지 주제 중에 세 번째 주제인 공간 네트워크 데이터베이스를 위한 효율적인 저장 및 색인 구조를 설계하고, 이를 구현하여 기존의 연구와 성능을 평가한다. 이를 위해 저장할 데이터를 크게 3가지, 즉 공간 네트워크 자체의 데이터, 공간 네트워크상에 존재하는 POI(point of interest) 데이터, 이동 객체의 데이터로 분류한다. 이러한 분류에 근거하여, 첫째 공간 네트워크 자체의 데이터를 효율적으로 저장하기 위해 노드 및 에지 정보를 지니는 공간 네트워크 파일 구조를 설계한다. 둘째, POI 데이터에 대한 효율적인 접근을 위해 POI 저장 및 색인 구조를 설계한다. 마지막으로, 이동 객체 데이터에 대한 저장 및 효율적인 궤적 검색을 위해 시그니쳐 기반 저장 및 색인 구조를 설계한다.

논문의 구성은 다음과 같다. 2장에서는 관련연구로써 기존의 공간 네트워크 데이터베이스를 위한 저장 및 색인 구조에 대해 알아본다. 3장에서는 본 논문에서 설계한 공간 네트워크 데이터베이스를 위한 새로운 저장 및 색인 구조를 제시한다. 4장에서는 공간 네트워크상의 이동객체를 위한 저장 및 색인 구조를 제시한다. 아울러 5장에서는 기존의 연구와 성능을 비교 및 평가한다. 마지막으로 6장에서는 결론을 제시한다.

2. 관련 연구

기존의 공간 데이터베이스는 제한조건이 없는 이상적인 공간을 가정한 것과는 달리, 공간 네트워크 데이터베이스에서의 공간 네트워크는 그 차제가 네트워크 구조를 지니고 있으므로, 이를 위한 저장이 필수적이다. 일반적으로 이는 그래프 형태로 변환이 용이하므로, 그래프 형태로 변환한 후 저장하는 연구들이 수행되었다. 공간 네트워크 데이터베이스를 위한 저장 및 색인 구조에 대한 관련 연구는 다음과 같다. 첫째, Univ. of Minnesota의 연구에서는, 공간 네트워크를 G=(N, E)의 그래프 형태로 변환하였다. 여기서 N은 노드의 집합을, E는 에지의 집합을 나타낸다. 아울러 변환된 그래프를 Z-order로 표현하고 연결에 따른 클러스터를 형성하여, CCAM의 접근기법에 따라 저장한다[6]. 아울러 이들 노드들에 대해 B^+-트리를 구성하여, 노드 아이디를 통한 접근을 빠르게 한다. CCAM의 구조는 (그림 1)과 같다. (그림 1)의 (a)는 CCAM의 구조에 저장된 그래프 예제이다. 그래프는 6개의 노드로 되어 있고 각 노드는 Z-Order 방식으로 노드 아이디를 설정하였다. (a)의 그래프 예제에서 두 개의 그래프로 나뉘는데, 분할 기준은 거리에 따른 가중치이다. CCAM 구조의 상단부분은 B^+-트리를 구성하며 노드 아이디를 키로 하여 색인한다. CCAM의 하단 부분은 각 노드의 인접 노드들의 리스트(Adjacency list)를, 분할된 그래프 단위로 같은 디스크 페이지에 저장한다. CCAM은 노드간의 정보를 통하여 네트워크를 정의하므로, 에지에 대한

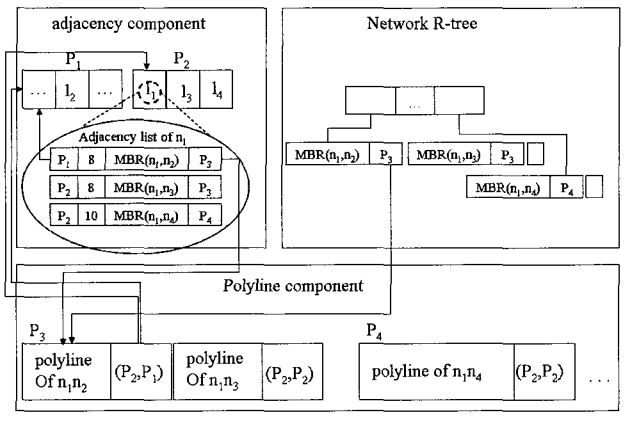


(그림 1) 그래프 예제와 CCAM의 구조

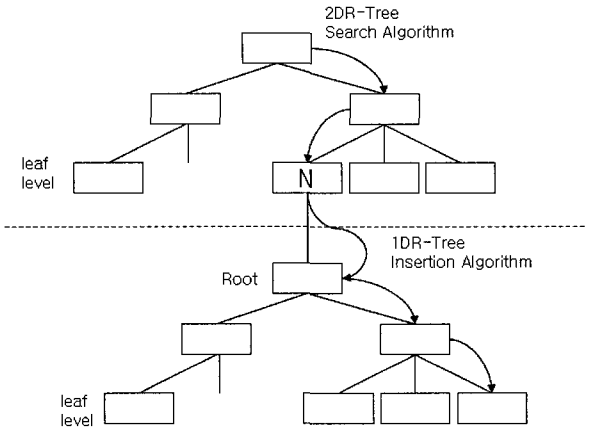
공간 질의를 수행하기에 비효율적인 구조를 가지는 단점이 있다.

둘째, HKUST의 연구에서는 도로나 철도와 같은 공간 네트워크를 연결에 따른 클러스터를 형성하여 노드들의 인접 리스트로 저장한다[5]. 단, 지역적 클러스터링 효과를 높이기 위해 Hilbert-order를 사용한다. 크게 3개의 컴포넌트들로 구성되어 전체 구조는 (그림 2)와 같다. 첫째, Adjacency 컴포넌트는 공간 네트워크 상에서 근접한 노드들의 리스트 정보를 가진 인접리스트를 포함하고 있다. 둘째, Poly-line 컴포넌트는 각 에지들의 poly-line정보를 상세히 표현하고, end point의 인접리스트를 포함하는 디스크 페이지들에 대한 포인터 쌍을 포함한다. 마지막으로 Network R-트리는 poly-lines의 MBR(Minimum Bounding Rectangle)들의 집합으로 네트워크상에서 영역 질의를 지원한다. 각 리프노드는 디스크 페이지에 대한 포인터를 지닌다. HKUST에서의 연구는 CCAM의 연구와 달리 에지간의 관계와 에지를 통한 공간 질의에 초점을 맞추고 있다. 따라서 노드를 중심으로 하는 질의에 대해서는 효율적이지 못한 단점이 있다.

마지막으로 아테네 국립기술대학의 연구에서는 고정 공간 네트워크상에서의 이동객체를 처리할 수 있는 색인구조(Fixed Network R-tree (FNR-tree))를 제안하였다 [7]. FNR-트리의 구조는 (그림 3)과 같다. FNR-트리는 공간 네트워크를 2DR-트리로 색인하고, 이동객체는 2DR-트리의 리



(그림 2) HKUST의 전체 구조

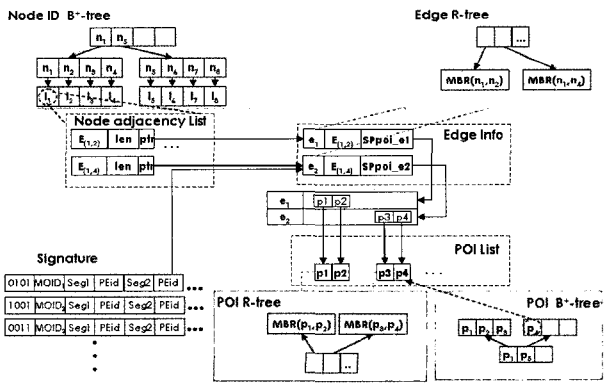


(그림 3) FNR-트리 구조

프노드에 1DR-트리를 두어 색인한다. 공간과 시간의 분리로 3DR-트리보다 시공간 영역질의에서 좋은 성능을 보이지만, 이동객체의 삽입 시 두 개의 트리를 접근해야 하는 단점과 이동객체가 지나간 에지 수만큼의 1DR-트리를 생성해야 하는 부담이 있다.

3. 공간 네트워크를 위한 저장 및 색인 구조

공간 네트워크를 위한 저장 및 색인구조의 목적은 공간 네트워크상에 존재하는 객체 데이터를 효율적으로 저장하고 색인하는 것이다. 이를 위하여 저장할 데이터를 다음과 같이 세 가지로 분류한다. 첫째, 공간 네트워크 자체의 데이터, 즉 공간 네트워크를 그래프로 변환했을 때 노드와 에지에 해당하는 데이터이다. 이들은 실세계에서 노드는 도로의 교차로, 에지는 도로의 일부분을 의미한다. 둘째, 공간 네트워크상에 존재하는 POI 데이터이다. POI 데이터는 공간 네트워크상에서 관심 있는 대상이고, 찾고자 하는 대상으로, 실세계에서 주유소, 식당 등이 이에 해당한다. 마지막으로 이동객체의 데이터이다. 이동 객체 데이터는 도로 네트워크상에서 움직이고 있는 객체 데이터를 의미하며, 실세계에서는 자동차, 사람 등이 이에 해당한다. 이들을 위한 저장 및 색인구조는 (그림 4)와 같다.



(그림 4) 공간 네트워크를 위한 저장 및 색인구조

3.1 공간 네트워크 자체의 데이터를 위한 저장 및 색인 구조

공간 네트워크는 그 자체가 네트워크 구조를 지니고 있으므로, 관련연구의 CCAM 및 HKUST의 연구와 마찬가지로, 공간 네트워크를 그래프 형태로 변환하여 이에 대한 데이터를 저장하는 것이 필수적이다. CCAM은 노드에 대한 효율적인 색인구조를 가지고 있고, HKUST의 연구에서는 에지에 대한 효율적인 색인 구조를 가지고 있다. 따라서 본 논문에서는 앞서의 두 가지 방법의 장점을 결합하여 공간 네트워크 데이터에 대한 저장 및 색인 구조를 설계한다. 노드의 저장구조로 노드의 인접 리스트와, 노드가 구성하는 에지를 가리키는 포인터 정보를 저장한다. 아울러 빠른 노드 검색을 위해 노드 아이디를 키로 사용하여 B^+-트리를 구성한다. 또한 Hilbert-order를 사용하여 노드 아이디를 부여하며, 그 이유는 Hilbert-order가 지역적으로 클러스터를 하기 때문에 질의 처리 시 디스크에 대한 I/O 횟수가 Z-order에 비해 감소하기 때문이다[8]. 에지의 저장구조로는 에지를 이루는 노드 정보와 에지 정보, 에지 상에 있는 POI 리스트를 가리키는 포인터 정보를 저장한다. 또한 공간 질의 처리를 위해 R-트리를 구성한다. 노드 인접리스트 정보와 에지 정보의 자세한 데이터 구성은 다음과 같다.

3.1.1 노드 인접리스트 정보

노드에 대한 정보 및 노드와 에지간의 관계성을 나타낸다. 노드 ni에 대한 연결 에지와의 리스트는 li로 표기되며, 리스트는 다음과 같은 집합으로 표현된다. 여기서 e(ni,nj)는 노드 ni와 nj로 만들어지는 에지를 의미한다. Ei는 ni에서 생성되는 에지의 집합을 나타낸다.

$$li = \{<ni, nj, len, Pj> \mid e(ni,nj) \in Ei, i \neq j \}$$

- li : 노드에 대한 연결 에지와의 리스트,
- e(ni, nj) : 노드 ni, nj 간의 에지,
- Ei : ni에서 생성되는 에지의 집합

3.1.2 에지 정보

노드 ni, nj 사이의 에지 e(ni,nj)에 대한 에지 정보 Re(ni,nj)는 다음과 같이 구성된다. 여기서 Eid는 에지의 아이디를 나타내고, len은 에지의 길이를 나타낸다. MBR(ni,nj)은 에지에 대한 2 차원 공간상의 좌표를 나타내며, 이는 공간 질의를 처리할 때 사용된다. 아울러 direction은 에지 e(ni,nj)의 방향을 나타내며, 이는 위치와 방향 연산자가 결합된 질의를 처리할 때 유용하게 사용된다. Num of POIs는 이 에지 상에 존재하는 POI들의 전체 개수를 나타내며, SPpoi는 POI들을 접근하고자 할 때, POI Info file 내에서의 시작점을 나타낸다.

Re(ni, nj) = <Eid, ni, nj, len, MBR(ni, nj), direction, Num of POIs, SPpoi>

- Eid : 에지 아이디, len : 에지 길이
- MBR(ni, nj) : 에지에 대한 2차원 좌표,

- direction: 에지의 방향,
- Num of POIs : 에지 상에 존재하는 POI의 전체개수
- SPpoi : POI Info file 내에서의 시작점

3.2 POI 데이터를 위한 저장 및 색인 구조

관광지 정보나 주유소, 식당과 같이 사용자의 관심 대상이 되는 장소를 POI(point of interest) 라고 한다. 이것은 특성상 다른 공간 네트워크 자체의 데이터나 자동차나 사람과 같은 이동객체와 그 특성이 매우 다르기 때문에, POI 데이터는 하나의 독립적인 파일을 구성하여 서비스를 제공하는 것이 필요하다. 즉, POI들은 대개 정해져 있으며, 가끔씩 새로운 POI가 삽입되고 삭제되기 때문에, POI 데이터는 이동객체와는 달리, 거의 정적인 데이터의 특성을 지닌다. 한편, 공간 네트워크 자체의 데이터와 다른 점은 상대적으로 데이터의 양이 매우 크다는 사실과, 공간 네트워크 자체의 데이터가 완전히 정적인 데이터인데 반해, 이것은 거의 정적인 특성을 지녔다고 볼 수 있다. 이 POI를 위한 저장 및 색인 구조의 자세한 구성은 다음과 같다.

3.2.1 POI정보

POI 데이터는 하나의 에지 e(ni,nj)에 놓여있는(종속된) POIs의 리스트 Lij로 나타낸다. 종속되었다는 의미는 Lij는 에지 e(ni,nj)에 의해 하나의 클러스터로 형성되어 한 페이지나 최소의 페이지에 저장됨을 의미한다. 여기서 E는 공간 네트워크 내의 전체 에지 집합을 나타낸다. 여기서 Pid는 시스템에 저장된 POI를 위한 아이디 이며, Pname은 사용자들이 사용하는 POI의 이름이다. 아울러, Ptype은 POI 분류 기호로써, 예를 들면, 관광지인지, 식당인지를 구별하는 것이다. rlen는 그 POI를 포함하고 있는 에지 e(ni,nj)에서, nI로부터의 길이이다. coord는 그것의 공간 좌표값 (x, y)이고, PGij는 그 POI가 추가의 정보를 지니고 있을 때 그것이 저장된 페이지를 의미한다. 아울러 에지 정보에 접근하기 위해 하나의 Lij당 해당되는 하나의 에지 e(ni,nj)를 위한 포인터(Pij)를 지닌다.

Lij = {<Pid, Pname, Ptype, rlen, coord, PGij> | e(ni,nj) $\in$ E, i $\neq$ j}

- Pid : POI를 위한 아이디
- Pname : POI의 이름
- Ptype : POI 분류기호,
- rlen : POI를 포함하고 있는 에지에서, ni로 부터의 길이
- coord : 공간 좌표값,
- PGij : POI의 추가 정보가 저장된 페이지

3.2.2 POI B^+-트리, R-트리

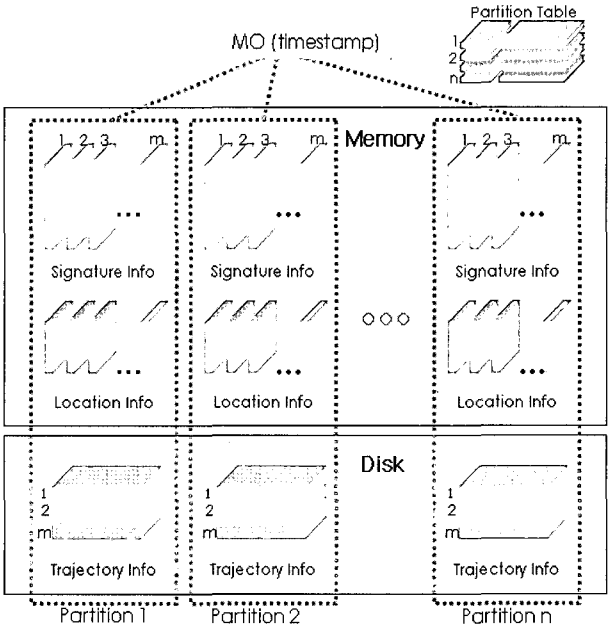
하나의 POI 정보에 대한 빠른 접근을 위하여 POI 아이디나 이름에 대하여 B^+-트리를 구성한다. 아울러 영역 질의와 같은 공간 질의의 확장을 위하여 R-트리도 함께 구성한다.

4. 이동객체 데이터를 위한 저장 및 색인 구조

4.1 이동 중인 객체 궤적의 저장 및 색인 구조

이동객체는 계속적으로 그 위치가 변하기 때문에, 일반적으로 객체 궤적 정보의 양은 매우 크다. 기존 객체 궤적 정보에 대한 빠른 접근을 위하여 TB-트리[9] 기법이 제안되었다. 그러나 이것은 다음과 같은 문제점을 지니고 있다. 첫째, TB-트리[9] 기법은 이상적인 유클리디안 공간을 가정하여 설계되었기 때문에, 공간 네트워크상에서의 궤적 색인 기법으로는 효율적이지 못하다. 즉 이동 객체는 이미 정해진 공간 네트워크상에서만 움직이므로, 이동 객체의 움직임 경로의 쏠림(skewed) 현상이 발생한다. 이로 인하여 상위 MBR 간에 중복이 매우 많이 발생하여 트리 구조를 통해 이동 객체의 궤적을 접근하는 것이 매우 비효율적이다. 둘째, TB-트리[9] 기법은 시간 축을 포함하여 3차원으로 상위 MBR을 구성하기 때문에, 공간 네트워크상에서의 궤적 색인 기법으로는 효율적이지 못하다. 즉, 이동 객체는 이미 정해진 공간 네트워크상에서 긴 경로를 움직일 때, 이상적인 유클리디안 공간에 비해 dead space가 많이 증가한다. 이는 다른 객체의 궤적과의 높은 중복을 발생시키며, 따라서 TB-트리의 분별능력(discrimination capability)이 감소하여 성능이 저하된다. 따라서 위의 문제점을 해결하는 궤적에 대한 빠른 검색을 위한 색인 구조로써 시그니쳐(signature) 기반 접근기법을 제안한다. 제안하는 시그니쳐 기반 접근기법의 구조는 (그림 5)와 같다.

먼저 제안하는 기법의 특징은 객체 궤적(trajectory) 정보와 그것의 시그니쳐 정보를 하나로 저장하지 않고, 이동 객체의 시작 시간(start time)에 근거하여 순서대로 n 개의 파티션(partition)을 만들어 메모리에서 유지한다는 점이다. 이



(그림 5) signature 파일 기반 및 색인 기법

렇게 n개의 파티션으로 저장하는 이유는 다음과 같다. 첫째, 파티션 내에 최대 객체 개수를 정의하여, 시그니쳐 파일 구성을 가능하게 하기 위함이다. 이는 시그니쳐 구성을 통해 효율적인 검색 성능뿐 아니라 저장 공간 사용 측면에서도 효율성을 제공한다. 둘째, 병렬 환경으로 확장하여, 질의를 만족하는 파티션들을 동시에 독립적으로 탐색하는 것을 가능하게 하여, 보다 효율적인 검색 성능을 달성할 수 있다. 마지막으로, 객체의 과거 궤적에 관한 질의를 처리하기 위해, 과거 궤적에 대한 정보를 효과적으로 유지해야 하며, 이러한 질의는 시간 축에 의존하므로, 시간에 따라 저장된 파티션 단위로 과거 궤적 정보를 유지하는 것이 효과적이다.

아울러 공간 네트워크상에서는 미래 위치가 현재의 네트워크 세그먼트(에지)와 전체 네트워크에 매우 의존적이므로 상대적으로 미래 위치를 예측하기가 용이하며, 적어도 가까운 미래에 대해서는 전적으로 현재의 네트워크 에지에 의존하므로 상대적으로 정확한 미래 예측이 가능하다. 따라서 제안하는 기법의 장점은 다음과 같다. 첫째, 시그니쳐 파일에 기반한 기법이므로 데이터의 쏠림현상에 전혀 영향을 받지 않는다. 둘째 트리 구조가 아니므로 dead space 문제가 발생하지 않는다. 셋째, 불연속적인 부분궤적 조건을 포함한 복잡한 질의에도 매우 효율적이다. 마지막으로, 병렬 환경으로 쉽게 구조를 구현할 수 있기 때문에, 성능이 우수한 접근 기법이 될 수 있다.

한편 (그림 5)의 시그니쳐 기반 저장 및 색인 구조는 파티션 테이블(partition table) 및 3개의 파티션 영역으로 구성되어 있으며, 이는 객체 궤적 정보 (object trajectory information) 영역, 궤적 시그니쳐 정보 (trajectory signature information) 영역, 위치 정보 (location information) 영역이다. 시그니쳐 정보 영역 및 위치 정보 영역은 그 크기가 작기 때문에 메모리에 유지한다.

4.1.1 파티션 테이블

파티션 테이블은 현재 이동 중인 객체의 궤적 세그먼트가 저장된 파티션들의 정보를 메모리에 유지하고 있다. 즉, 각 파티션내의 전체 세그먼트들을 위한 시작 및 끝 시간을 유지하고 있다. 이는 질의 처리 시 질의를 위해 탐색해야 할 파티션을 찾는 데 사용되며, 그 크기가 작기 때문에 주기억장치에 유지시킨다. 파티션 i를 위한 엔트리 Ei는 다음과 같다.

Ei = <p_start_time, p_end_time, p_expected_time, final_entry_no>
- p_start_time : 파티션내의 세그먼트들의 최소 시작시간
- p_current_time : 파티션내의 세그먼트들의 최대 현재시간
- p_end_time : 파티션내의 세그먼트들의 최대 예상 종료시간
- final_entry_no : 파티션에 저장된 마지막 엔트리 번호

4.1.2 객체 궤적 정보 영역

예측된 경로는 과거의 궤적과 마찬가지로 부분적으로는 하나의 에지에 해당되므로, 미래 궤적에 대한 처리도 과거 궤적과 동일하게 처리할 수 있다. 객체의 궤적 데이터는 객체가 이동한 세그먼트의(또는 에지) 집합으로 객체 MOi 에 대한 궤적 Ti로 나타낸다.

Ti = <MO_iid, #_actual_seg, #_future_seg, #_mismatch, $SegList_i$>
- MO_iid : MOi의 아이디
- #_actual_seg : 객체 이동에 따른 과거 궤적 세그먼트의 수
- #_future_seg : 예상되는 미래 궤적 세그먼트의 수
- #_mismatch : 예측 미래 궤적 과 일치하지 않은 세그먼트의 수
- $SegList_i$: 객체 MOi의 과거 및 미래 궤적 세그먼트 집합

아울러 n개의 과거 궤적 세그먼트와 m개의 미래 궤적 세그먼트를 가진 경우, $SegList_i$는 다음과 같이 표현된다.

$SegList_i = \{<sij, eid, start, end, ts, (te \text{ or } v)> \mid 0 \le j < n+m \}$
- sij : MOi의 j번째 세그먼트
- eid : sij를 포함하는 에지의 아이디
- start : eid로 나타내는 에지에서의 sij의 시작 위치
  만약 값이0이면 에지의 처음부터 시작함을 의미한다.
- end : eid로 나타내는 에지에서의 sij의 마지막 위치
- ts : eid로 나타내는 에지에서의 sij의 시작 시간
- te : eid로 나타내는 에지에서의 sij($0 \le j < n$)의 마지막 시간
- v : eid로 나타내는 에지에서의 sij ($n \le j < n+m$)의 평균속도

4.1.3 궤적 시그니쳐 정보 영역

객체 궤적 데이터에 대한 색인 기법으로서 시그니쳐를 효과적으로 구축하기 위해서는, 주어진 하나의 객체 궤적에 대해 그에 해당되는 시그니쳐 레코드를 효율적으로 구성하는 것이 필요하다. 시그니쳐 레코드를 구성하는 방법으로는 비트 단위 OR 연산(superimposed coding) 방법을 사용한다. 그 이유는 비트 단위 OR 연산 방법이 SNDB 응용과 같이 객체 당 궤적의 수가 매우 가변적일 경우에도 하나의 시그니쳐 레코드로 표현 가능하여 효과적으로 사용될 수 있기 때문이다[10]. 일반적으로 전체 객체의 수가 N 이고, 객체 당 평균 객체 세그먼트의 수가 r일 때, 가장 효율적인 시그니쳐 레코드 크기 S와 한 세그먼트 당 세팅되는 비트의 수 k는 다음 식에 의하여 계산될 수 있다. 여기서 Fd는 false drop을 나타내는 값으로, 이것을 특정 값으로 고정시키면 (일반적으로 1/N로 가정), S와 k를 구할 수 있다 [11].

$$\ln Fd = -(\ln 2)^2 * S / r$$
$$k = S * \ln 2 / r$$

마지막으로, 객체의 이동에 따른 새로운 세그먼트를 저장하고자 하며 이미 존재하는 예측된 미래 궤적 세그먼트와 일치하는 경우, 시그니쳐에 저장은 필요하지 않다. 한편, 비트 단위 OR 연산 방식으로 시그니쳐를 생성하였기 때문에, 불일치된 미래 궤적 세그먼트가 포함되어도 이를 삭제하지 않는다. 그 이유는 극한의 임계점(threshold)을 넘지 않으면 여전히 좋은 검색 성능을 제공하기 때문이다.

4.1.4 위치 정보 영역

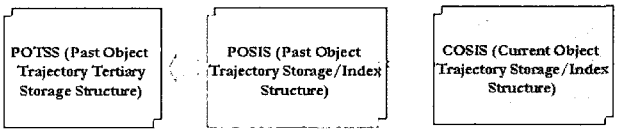
위치 정보 영역은 궤적 시그니쳐 정보에 해당하는 객체 궤적 정보 영역 내에서의 객체 궤적의 저장 위치를 나타낸다. 이는 시그니쳐에서 potential match로 판명된 객체 궤적을 실제로 탐색하여 실제 질의를 만족하는 것인지를 검색할 수 있도록 해준다. 아울러 위치 정보 영역은 start time, current time, end time 등을 포함하고 있어, 궤적 질의의 시간 조건에 대해 필터링 역할을 수행할 수 있다. 하나의 객체 MOi의 궤적 데이터에 대한, 위치 정보 Ii는 다음과 같이 나타낸다.

Ii = <MOiid, Li, strat_time, current_time, end_time >
- MOiid : MOi의 아이디
- Li : MOi의 궤적이 trajectory information component 내에 저장된 위치
- start_time : MOi의 궤적이 처음으로 입력된 시작 시간
- current_time : MOi의 궤적 가운데 최종으로 이동된 세그먼트가 저장된 시간
- end_time : MOi의 미래 궤적의 예상되는 마지막 시간. 이것이 current_time_
  stamp와 같으면, MOi는 더 이상의 궤적 이동이 없이 이것이 궤적 이동의 마지막 시간임을 의미한다.

4.2 이동 객체의 과거 궤적의 저장 및 색인 구조

현재 이동 중인 객체의 궤적뿐만 아니라, 이동이 완료된 객체의 과거 시간의 궤적을 효과적으로 처리하기 위해, 객체의 과거 궤적을 위한 색인 및 저장구조가 필요하다. 이를 위해 과거 궤적을 두 가지 경우로 구분한다. 즉, 질의의 대상이 되는 과거 시간의 궤적과 질의의 대상이 될 확률이 거의 없는 오랜 과거 시간의 궤적이다. 따라서 이를 반영한 이동 객체의 전체 구조는 (그림 6)과 같다.

(그림 6)에서 COSIS는 이동 중인 이동객체 궤적을 저장 및 색인하는 구조이고, POSIS와 POTSS는 과거의 이동객체 궤적을 저장 및 색인하는 구조이다. 여기서 COSIS는 앞 절에서 설명이 이루어졌으므로, 본 절에서는 COSIS에서 POSIS로의 이동 저장 및 POSIS 구조에 대해 다룬다.



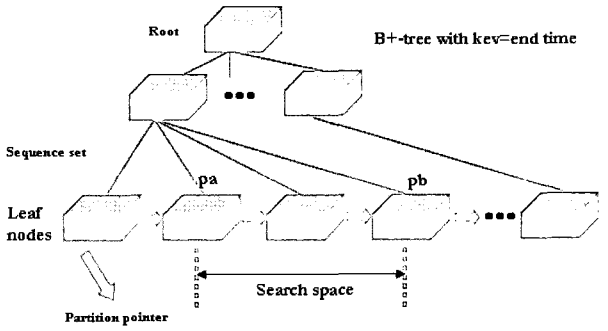
(그림 6) 이동객체 궤적을 위한 전체적인 저장/색인 구조

COSIS에 저장된 객체의 이동이 완료되면, 이것은 POSIS에서 파티션 단위로 저장이 이루어지고, 아울러 POSIS에 저장된 과거 객체 중 시간이 오래되어 더 이상 질의 검색의 대상이 될 가능성이 없는 경우, 이 과거 궤적은 POTSS로 저장이 이루어진다. 한편 COSIS는 주기억 장치와 디스크에, POSIS는 디스크에 저장이 이루어지며, POTSS는 제3의 기억장치(tertiary Storage)에 보관된다. 한편 COSIS에서 POSIS로 이동하여 저장하기 위해 고려할 사항은 다음과 같다. 첫째, 객체의 과거 궤적 저장의 비용을 최소화 하는 것이 필요하다. 둘째, 질의에 대한 효율적인 검색을 위해, 과거 궤적의 파티션을 최소로 검색하는 것이 필요하다. 셋째, 과거 궤적의 수는 매우 방대하고 시간에 따라 정렬되어 있으므로, 이를 시간에 따라 효과적으로 검색할 수 있는 색인 구조가 요구된다. 따라서 COSIS에서 POSIS로 하나의 파티션을 저장할 때, 그 파티션의 end time을 키로 하여 B^+-트리에 저장한다. B^+-트리의 구조는 (그림 7)과 같으며, 리프 노드에 저장되는 레코드 형태 Rec은 다음과 같다.

Rec: <p_start_time, p_end_time, Pid, PLoc>
- p_start_time : POSIS에 저장하고자 하는 파티션내의 세그먼트들의 최소 시작시간
- p_end_time : POSIS에 저장하고자 하는 파티션내의 세그먼트들의 최대 종료시간
- Pid : POSIS에 저장하고자 하는 파티션의 아이디
- PLoc : POSIS에 저장하고자 하는 파티션의 저장될 위치

따라서 질의가 (t1, t2) 사이의 객체 궤적을 찾고자 한다면, t1을 가지고 B^+-트리를 탐색하여 리프 노드를 얻고, 그 리프 노드로부터 B^+-트리의 순차 집합(sequence set)의 포인터를 가지고 계속 오른쪽의 리프 노드를 탐색하면서 조건 1을 만족하는 레코드를 획득한다. 한편 (t1, t2) 시간 범위의 질의를 처리하기 위한 탐색영역은, t1을 가지고 B^+-트리를 탐색하여 얻은 리프 노드(Pa)부터 시작하여, 조건1을 만족하지 않는 레코드를 포함하는 리프 노드(Pb)를 순차 집합에서 찾을 때까지이다. 이를 통하여 조건 1을 만족하는 레코드를 찾기 위해 불필요한 리프 노드의 탐색은 수행되지 않는다.

조건 1 : (p_end_time ≥ t1) AND (p_start_time ≤ t2)



(그림 7) B^+-트리 in POSIS

4.3 이동 객체 궤적의 삽입 알고리즘

이동객체 궤적의 삽입은 현재 이동 중인 객체의 궤적을 COSIS에 삽입하는 것과, 객체의 이동이 끝난 객체의 궤적을 COSIS에서 POSIS로 이동시켜 저장하는 두 가지로 구별된다. COSIS에서 POSIS로 이동하여 저장하는 알고리즘은 이미 4.2 절에서 대략의 설명이 되었으므로 생략하고, 본 절에서는 현재 이동 중인 객체의 궤적을 COSIS에 삽입하는 것을 다룬다. 한편 COSIS에 이동 객체 궤적을 삽입하는 것은, 객체의 이동에 따른 초기 저장과 객체의 궤적 세그먼트 저장으로 나뉘어 진다. 먼저, 객체의 이동에 따른 초기 저장은, 파티션 테이블에서 마지막 파티션을 찾고, 그 파티션에서 하나의 이용 가능한 entry(NE)를 얻는다. 한편 객체의 이동에 따른 초기 저장은 미래 예측 궤적을 지니지 않은 경우와 지닌 경우의 두 가지 경우로 수행이 이루어진다. 첫째, 미래 예측 궤적을 지니지 않은 객체의 초기 저장은, 비록 미래 예측이 존재하지 않는 객체이지만, 처음에 이동하고 있는 에지를 기반으로 하나의 예측 궤적 세그먼트를 생성하고, 이를 객체 궤적 정보 영역(trajectory info area)의 NE entry에 저장한다. 아울러, 이 초기 예측 세그먼트를 통해 위치 정보 영역(location info area)에 start_time(StartT), current_time(CurrentT), end_time (ExpectedET)을 저장하며, StartT와 CurrentT는 모두 객체의 처음 시작 시간으로 할당하며, ExpectedET는 NULL로 할당된다. (그림 8)은 객체 궤적 초기 저장을 위한 알고리즘 InsertFirst를 나타낸다. 둘째, 미래 예측 궤적을 지닌 객체의 이동에 따른 초기 저장은, 미래 예측 세그먼트 리스트(TrajSegList)가 주어지므로 이 리스트를 객체 궤적 정보 영역의 NE entry에 저장한다. 아울러 TrajSegList의 각 세그먼트로부터 시그니쳐를 생성하고(SSn), 이들을 비트 단위 OR 연산(bit Oring) 하여, 최종 시그니쳐를 생성한다(SigTS). 아울러 주어진 TrajSegList를 통해 위치 정보 영역에 StartT, CurrentT, ExpectedET를 저장하며, StartT와 CurrentT는 모두 객체의 처음 시작 시간으로 할당하며, ExpectedET는 미래 예측 궤적의 마지막 세그먼트의 ts, start, v를 통해 계산된다. 마지막으로 두 가지 경우 공통으로, 생성된 SigTS를 궤적 시그니쳐 정보 영역(siganture info area)의 NE entry에 저장한다. 아울러 파티션 테이블의 마지막 파티션을 위한 엔트리(LP)에 <StartT, CurrentT, ExpectedET>를 저장한다.

```
Algorithm InsertFirst(MOid, TrajSegList) /* TraSegList contains
   the information of a set of expected segments for the trajectory
   of a moving object Moid */
1. TrajSeg = the first segment of TrajSegList
2. Generate a signature SigTS from TrajSeg
3. StartT = CurrentT = ts of TrajSeg
4. Obtain final_entry_no of the entry, in the partition table, for the
   last partition, LP
5. NE = final_entry + 1 // NE = the next available entry in LP
6. Obtain the location, Loc, of the entry NE in the trajectory infor
   mation area for inserting object trajectory
7. if(end field of TrajSeg = NULL){// no expected trajectory
8.          ExpectET = NULL
9.          Store <MOid,0,1,TrajSeg> into the entry NE, pointed b
            y Loc, of the trajectory information area in LP
10. } else {// expected trajectory exists
11.         #_fseg = 1
12.         while (the next segment Sn of TrajSegList ≠ NULL) {
13.                 #_fseg = #_fseg + 1
14.                 Generate a signature SSn from Sn
15.                 SigTS = SigTS | SSn }
16.         Store <MOid,0,#_fseg,TrajSegList> into the entry NE,
            pointed by Loc, of the trajectory
            info area in LP
17.         Compute ExpectET by using ts, start, and v of the last
             segment of TrajSegList
18. } // end of else
19. Store SigTS into the entry NE of the signature information are
    a in LP
20. Store <MOid,Loc,StartT,CurrentT,ExpectET> into the entry NE
    of the location information
      area in the partition LP
21. Store <StartT,CurrentT,ExpectET,NE> into the entry for LP in
    the partition table
End InsertFirst
```

(그림 8) 객체 궤적 초기 삽입 알고리즘

한편, 객체의 궤적 세그먼트 저장은, 먼저 파티션 테이블로부터 그 객체가 이미 저장되어 있는 파티션을 찾는다. 이를 위해서는 객체 궤적의 시작 시간이(ST) 객체 궤적의 세그먼트 저장 시 사용된다. 아울러 이 ST와 객체의 아이디를 가지고, 파티션 내의 객체의 세그먼트 정보가 저장된 엔트리를 찾는다. (그림 9)는 세그먼트를 저장하는 알고리즘을 나타내고 있다. 여기서 NE는 MOid 객체의 파티션 내에서의 저장된 엔트리를 나타내며, Loc은 객체 궤적 세그먼트 영역에서의 NE 엔트리의 시작위치를 나타낸다. Loc가 필요한 이유는 객체 궤적 정보 영역에 있는 레코드는 가변길이 레코드이므로, NE 엔트리의 시작 위치를 알 수 없기 때문이다. 한편 객체의 궤적 세그먼트 저장은, 미래 예측 궤적을 지니지 않은 경우와 지닌 경우로 나누어진다. 첫째, 미래 예측 궤적을 지니지 않은 객체의 궤적 세그먼트 저장은, Loc이 가리키는 위치의 NE 엔트리에 저장하고자 하는 TrajSeg를 저장하고, TrajSeg를 통해 생성된 SigTS를 시그니쳐 정보 영역의 NE 엔트리에 저장하는 것이다. 마지막으로 위치 정보 영역의 NE 엔트리에 <MOid, Loc, StartT, CurrentT, ExpectET>를 저장한다. 둘째, 미래 예측 궤적을 지닌 객체의 궤적 세그먼트 저장은, 다음 세 가지 경우에 따라 수행된다. 이를 위해 find_seg 함수가(그림 9의 line 9)호출되며, 이는 Loc으로 가리키는 NE 엔트리의 미래궤적 세그먼트에서 TrajSeg 과 일치하는 것을 탐색하는 함수이다. 여기서 일치한다는 의미는 같은 에지 상에서 이동함을 의미한다. 일치하는 것을 발견하면, 발견된 위치를(1부터 시작) seg_pos로 리턴하며, 발견되지 않았으면 0을 리턴한다. 첫째, 발견하지 못한 경우(seg_pos=0), 이때는 미래 예측 궤적을 지니지 않은 객체의 궤적 세그먼트 저장과 동일한 작업을 수행한다. 단, Loc이 가리키는 위치의 NE 엔트리의 미래 궤적을 하나씩 뒤로 옮겨주고, (#_actual_seg)-th 세그먼트에 TrajSeg를 저장한다. 둘째, 맨 처음으로 발견한 경우(seg_pos=1), 이때는 예측한 것과 동일하므로 단지 Loc이 가리키는 NE 엔트리의 (#_actual_seg)-th 세그먼트에 TrajSeg

를 저장한다. 마지막으로, 맨 처음이 아닌 경우로 발견된 경우(seg_pos>1), 이때는 발견되기 전의 미래궤적들은 예측이 틀린 세그먼트이므로, 이들을 Loc이 가리키는 NE 엔트리의 미래궤적으로 부터 seg_pos-1만큼 제거한다. 그리고 (#_actual_seg)-th 세그먼트에 TrajSeg를 저장하고, 미래궤적을 seg_pos-2 만큼 앞으로 옮겨준다. 만약 전체 세그먼트에 대한 제거된 수(#_mismatch)의 비율이 임계점(τ)을 넘지 않으면, TrajSeg로 생성된 SigTS를 시그니쳐 정보 영역의 NE 엔트리에 저장한다. 비록 불일치하는 미래 궤적 세그먼트가 포함되어 비트 단위 OR 연산 방식으로 시그니쳐가 생성 되었어도, 임계점을 넘지 않았기 때문에 여전히 좋은 검색 성능을 제공할 수 있다. 한편 많은 개수의 예측 미래 궤적 세그먼트가 불일치하여 임계점(τ)을 초과하면, re-generate_sig 함수를 호출하여 시그니쳐를 재생성한다. 그리고 모든 경우에 공통으로, Loc이 가리키는 위치의 NE 엔트리의 #_actual_seg, #_future_seg, and #_mismatch 값들을 변경시킨다. 아울러위치 정보 영역의 NE 엔트리에 CurrentT를 변경하고, 파티션 테이블에서 해당 파티션 엔트리의 CurrentT 값을 변경한다.

```
Algorithm InsertSeg(MOid, TrajSeg, ST)
/* TraSeg contains the information of a segment for the trajectory
of a moving object Moid, to be stoted with a object trajectory's
start time, ST*/
1. Generate a signature SigTS from TrajSeg
2. Locate a partition P covering ST in the partition table
3. Locate an entry E covering ST for the moving object with
   MOid in the location information area and get its location, Loc,
   in the trajectory information area
4. Obtain #_actual_seg, #_future_seg, and #_mismatch of the trajectory
   info entry E (i.e., TE) for the MOid in P
5. if(#_future_seg = 0) { // no expected trajectory
6.          Insert TrajSeg into the (#_actual_seg+1)-th trajectory
            segment of TE
7.          Store SigTS into the entry E of the signature info area in P
8. } else {// expected trajectory exists
9.          seg_pos = find_seg(TrajSeg,Loc)
10.         #_ actual_seg++, #_ future_seg = #_ future_seg  seg_pos
11.         case(seg_pos = 0) { // find no segment
12.                 Insert TrajSeg into segment of TE and relocate
                    the future traj segments backward
13.                 Store SigTS into the entry E of the signature
                    info area in P }
14.         case(seg_pos = 1) //find the first segment
15.                 Insert TrajSeg into (#_actual_seg)-th trajectory
                    segment of TE for exchanging the old segment
16.         case(seg_pos > 1) {// find the (seg_pos)-th segment
17.                 #_mismatch = #_mismatch + seg_pos  1
18.                 Insert TrajSeg into (#_actual_seg)-th segment of
                    TE and relocate the future traj segments forward
19.                 if(#mismatch/(#_future_seg+#_actual_seg) > τ)
20.                         regenerate_sig(Loc, SigTS, E, P)
22.           }// end of case
23. } // end of else
24. Update #_actual_seg, #_future_seg, and #_mismatch of TE
25. CurrentT = te of TrajSeg
26. Store CurrentT into the current_time of the entry E of the
    location information area in the partition P
27. Store CurrentT into the p_current_time of the partition P entry
    in the partition table
End InsertSeg
```

(그림 9) 세그먼트 삽입 알고리즘

4.4 이동 객체 궤적의 검색 알고리즘

이동 객체 궤적의 검색 알고리즘은 주어진 질의 궤적 세그먼트에 포함하고 있는 객체 궤적의 객체 아이디를 찾는 알고리즘이다. 이 알고리즘에서 먼저 수행할 일은, 질의에서 주어지는 시간의 범위(lower, upper)를 가지고, 이에 해당하는 파티션을 탐색하는 것이다. (그림 10)은 이동 객체의 궤적 검색 알고리즘을 나타낸다. find_partition 함수는 질의에서 주어진 TimeRange를 만족하는 파티션을 구하기 위해, POSIS의 B^+-트리의 구조와 COSIS의 파티션 테이블을 탐색하여 partList를 생성한다. 이에 대한 탐색은 크게 다음의 3가지 경우로 요약될 수 있다. 아울러 POSIS에 저장된 마지막 파티션의 p_end_time (PEtime)을 유지하고 있어, 다음 조건으로 3 가지 경우를 쉽게 판별할 수 있다.

1. 질의 범위의 lower, upper가 둘 다 COSIS에 있는 경우, (lower > PEtime)
2. 질의 범위의 lower가 POSIS에, upper가 COSIS에 있는 경우, (upper > PEtime AND lower ≤ PEtime)
3. 질의범위의 lower, upper가 둘 다 POSIS에 있는 경우, (upper≤ PEtime)

첫째, 1의 경우에는 COSIS의 파티션 테이블을 순차적으로 탐색하여, TimeRange를 만족하는 파티션 리스트 partList를 구한다. 이를 구하기 위한 조건 2는 다음과 같다. 아울러, COSIS의 파티션 테이블의 크기는 상대적으로 작기 때문에 주기억장치에 저장되며, 파티션의 p_start_time 으로 정렬되어 있다. 따라서 lower의 값을 가지고 키 순차탐색을 수행하며, 조건 2를 만족하지 않은 엔트리를 발견한 뒤로는 더 이상의 탐색이 필요 없기 때문에, 파티션 테이블 탐색 비용은 크지 않다.

조건 2 : (p_end_time≥lower)AND(p_start_time≤upper)if
p_end_time≠NULL
(p_current_time ≥ lower)
AND (p_start_time ≤ upper) otherwise

둘째, 3의 경우에는 POSIS의 B^+-트리를 lower 값을 키로 하여 탐색하여 리프 노드를 얻는다. 아울러 그 리프 노드로부터 B^+-트리의 sequence set의 오른쪽의 리프 노드를 계속 탐색하면서 (p_end_time ≥ lower AND p_start_time ≤ upper) 조건을 만족하는 레코드의 리프 노드까지만 탐색하여 만족하는 partList를 구한다. 마지막으로, 2의 경우에는 POSIS의 B^+-트리를 lower 값을 키로 하여 탐색하여 리프 노드를 얻고, 그 리프 노드로부터 B^+-트리의 sequence set의 마지막 리프 노드까지를 탐색하여 TimeRange를 만족하는 파티션 리스트를 얻는다. 아울러 주기억장치에 저장된 COSIS의 파티션 테이블을 키 순차 탐색하여, p_start_time ≤ upper 조건을 만족하는 파티션 리스트를 구하고, 앞서 POSIS에서 구한 리스트와 결합하여 최종 partList를 구한다.

마지막으로, 질의 궤적 세그먼트(QsegList)로 부터 질의 시그니쳐(Qsig)를 생성한다. find_partition 함수를 통해 구한 partList의 각각에 대하여, 각 파티션의 시그니쳐 정보 영역을 검색하여 Qsig를 만족하는 엔트리를 CanList에 저장한다. 그리고 위치 정보 영역의 CanList에 포함하는 엔트리에 대해서 start_time, end_time, current_time를 구하고, 조건 3을 만족되는지 확인한다. 조건 3을 만족하면, 질의 궤적 세그먼트 리스트(QsegList)와 위 조건을 만족하는 객체 궤적 세그먼트 리스트 간에 서로 매치되는 지 체크한다. 즉, 파티션에 저장된 객체 궤적 세그먼트 리스트가 QsegList를 포함하고 있는 지 살펴보고 (line 15~22), 만약 그렇다면 그것의 객체 아이디를 결과 리스트(MoidList)에 저장한다.

조건 3 : (end_time ≥ lower) AND (start_time ≤ upper) if end_time≠NULL
(current_time ≥ lower) AND (start_time ≤ upper) otherwise

```
Algorithm Retrieve(QSegList, TimeRange, MOidList) /* MOidList
is a set of ids of moving objects containing a set of query segmen
ts, QsegList, for a given range time, TimeRange */
1. Qsig = 0, #qseg = 0, partList = ∅
2. t1 = TimeRange.lower, t2 = TimeRange.upper
3. for each segment QSj of QsegList {
4.          Generate a signature QSSi from Qsj
5.          QSig = QSig | QSSj, #qseg = #qseg + 1 }
        /*find partitions, partList, satisfying TimeRange by searc
        hing patiotion table of COTSS and B+-tree of POTSS*/
6. find_partition(TimeRange, partList)
7. for each partition Pn of partList {
8.          Obtain a set of candidate entries, CanList, examining th
            e signaturesof signature info area in Pn
9.          for each candidate entry Ek of CanList {
10.         Let s,e,c be start_time, end_time, current_time of the en
            try Ek of location information area
11.         if((s ≤ t2) AND (e ≥ t1 OR c ≥ t1)){
12.                     #matches = 0
13.                     Obtain the first segment ESi of the entry
                        Ek of the trajectory info area, TEk
14.                     Obtain the first segment QSj of QsegList
15.                     while(ESi ? NULL and QSj ? NULL) {
16.                     if(match(Esi, QSj)=FALSE)
                    Obtain the next segment ESi of TEk
17.                     else { #matches = #matches + 1
18.                     Obtain the first segment ESi of Tek }
19.                     if(#matches=#qseg)MOidList=MOidList∪
                        {TEk's MOid}
20.                     } } } //end of while //end of if //end of fo
                        r- CanList
21. } // end of for - partList
End Retrieve
```

(그림 10) 궤적 검색 알고리즘

5. 성능평가

공간 네트워크 데이터베이스를 위한 저장/색인 구조는 Intel Xeon Dual CPU 2.4GHz 및 memory 2GB를 사용하는 Windows Server 2003에서 C++로 구현하였다. 성능평가를 수행하기 위해 사용된 공간 네트워크 데이터는 17만개의 노드와 22만개의 에지로 이루어진 샌프란시스코 만의 공간 네트워크 데이터를 사용하였다[12]. 성능평가를 위한 POI 및 이동 객체 데이터를 위해서는, 공간 네트워크 이동객체 생성 알고리즘에 근거하여 10,000개의 POI와 5,000개의 이동객체가 100초 동안 이동한 궤적 세그먼트를 생성하였다[2]. 성능평가를 위해 공간 네트워크 데이터베이스 검색을 위한 질의 타입을 <표 1>과 같이 분류하였다. 한편, 공간 네트워크 질의와 공간 질의는 기존의 공간 네트워크 데이터베이스를 위한 저장구조로 잘 알려진 CCAM[6]과 비교를 하였다. 아울러, 궤적 질의는 기존의 궤적 색인 구조로 잘 알려진 TB-트리[8] 및 네트워크를 기반한 이동객체의 색인 구조인 FNR-트리[7]와 성능 비교를 수행하였다. 검색 성능은 질의 처리를 위해 검색하는데 걸리는 전체 시간 즉, CPU 및 디스크 I/O 시간을 합한 시간을 의미하며, 논문에서 구현한 저장/색인 구조는 SINOS(Storage and INdex Organization for Spatial network databases)라 명명한다.

<표 1> 공간 네트워크 데이터베이스를 위한 질의 타입

질의 타입	설 명
공간 네트워크 질의 (spatial network query)	정의) 아이디를 주고 공간 자체 데이터(노드, 에지)를 찾는 질의 예제) li: 노드 인접 리스트, Re:에지 집합 SQL문) SELECT * FROM li WHERE li.ni = query_ID SELECT * FROM Re WHERE exist(SELECT Pj FROM li WHERE li.ni = query_ni AND li.nj = query_nj)
공간 질의 (spatial query)	정의) 특정 점이나 특정 지역을 질의 데이터로 주고 에지나 POI를 찾는 질의 예제) Re:에지 집합, range(MBR):영역질의 함수 SQL문) SELECT * FROM Re WHERE range(query_MBR)
궤적 질의 (trajectory query)	정의) 이동 객체의 부분 궤적 또는 궤적을 찾는 질의 예제) Ti:이동객체의 궤적 집합, find(질의 에지 집합):궤적 질의 함수 SQL문) SELECT * FROM Ti WHERE find(query_edge_list)

5.1 삽입 성능 및 저장 공간 크기 비교

CCAM과 SINOS의 삽입 시간 결과는 <표 2>와 같다. SINOS가 706.9초이고 CCAM이 799.8초로, SINOS가 삽입 성능 측면에서 우수하다. 이는 CCAM의 경우는 클러스터링을 하기 위해 가중치로서 노드간의 거리를 계산하기 때문에 전체적인 구축시간이 더 많이 소요되는 반면, 본 논문에서 제안하는 기법은 hilbert ordering을 이용해 지역적인 측면을 고려하기 때문에 연산이 줄어들어 전체적인 구축 시간이 줄어들게 된다. CCAM과 SINOS의 저장 공간은 CCAM이 약 88MB정도의 크기를 가지며, SINOS는 약 116MB정도의 크기를 가진다. 이는 CCAM과 달리 SINOS에서는 POI 및 에지간를 위한 색인 구조가 더 필요하기 때문에 약 CCAM보다 1.3배정도 저장공간의 오버헤드를 갖는다.

〈표 2〉 삽입 시간 성능비교

(단위 : sec)

	CCAM	SINOS
삽입 시간	799.8	706.9

5.2 공간 네트워크 질의 성능 비교

공간 네트워크 질의는 노드의 경우에는 노드 아이디를 주고, 하나의 노드를 찾는 평균 시간을 측정하였고, 에지의 경우에는 노드 아이디를 주고 노드에 연결된 에지를 찾는 평균 시간을 측정하였다. 성능평가 결과는 <표3>과 같다. 공간 네트워크의 노드 탐색의 경우, SINOS가 0.064ms, 기존 CCAM이 0.07ms로써, SINOS가 검색 성능이 약간 우수하다. 그 이유는 CCAM은 연결성(connectivity)에 근거하여 클러스터링을 하는 반면, 본 논문에서 제안한 기법은 Hilbert-Order에 의하여 지역적 클러스터링을 수행하며, 이를 통하여 질의에 따른 보다 적은 디스크 탐색을 수행하기 때문이다. 에지 탐색의 경우, SINOS가 0.124ms, 기존 CCAM이 0.128ms로써, 본 논문에서 제안하는 SINOS와 CCAM의 에지 검색 성능이 거의 동일함을 알 수 있었다.

〈표 3〉 공간 네트워크 질의 성능비교

(단위 : ms)

	CCAM	SINOS
노드 탐색	0.070	0.0064
에지 탐색	0.128	0.124

5.3 공간 질의 성능비교

공간 질의는 점(point)을 중심으로 에지나 POI를 검색하는 경우 및 영역을 중심으로 검색하는 경우의 두 가지가 존재한다. 점을 중심으로 검색하는 질의는 에지 탐색의 경우, 랜덤하게 POI를 선택하여 그 POI를 포함한 에지를 검색하는 시간을 측정하였고, POI 탐색의 경우, 랜덤하게 선택한 POI를 가진 에지 상에 있는 모든 POI를 검색하는 시간을 측정하였다.

<표 4>는 점을 통한 공간 질의 성능 결과를 나타낸다.

〈표 4〉 점으로 검색한 공간 질의 성능비교

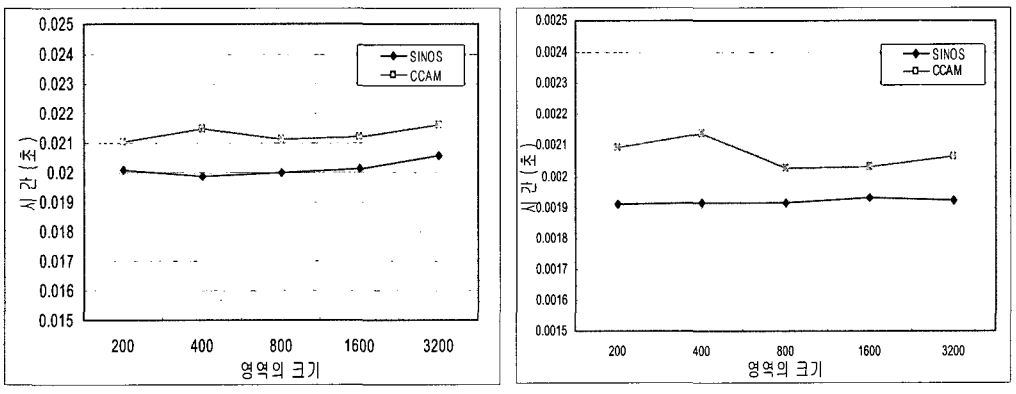
(단위 : ms)

	CCAM	SINOS
에지 탐색	25.9	26.0
POI 탐색	1.95	1.95

에지 탐색의 경우, SINOS가 26ms, 기존 CCAM이 25.9ms로써, 검색 성능이 거의 동일함을 알 수 있다. 아울러 POI 탐색에 대해서는, SINOS가 1.95ms, 기존 CCAM이 1.95ms로써, 검색 성능이 동일하다. 따라서, SINOS가 점으로 검색하는 질의 성능에서는, CCAM과 동일한 성능을 보인다고 할 수 있다. 한편, 영역을 중심으로 검색하는 경우는 영역을 200, 400, 800, 1600, 3200의 다섯 개로 나누어 성능비교를 수행하였다. 이와 같이 수행한 이유는 에지의 평균 길이를 측정한 결과, 약 1000의 길이를 가지기 때문에 이를 기준으로 구간을 나누었다. 여기서 1에 해당하는 길이는 실제 공간상에서 약 1.5 m에 해당한다. (그림 11)의 (a)는 주어진 영역을 중심으로 에지를 찾는 평균 시간을 측정한 결과이며, (그림 11)의 (b)는 POI를 주어진 영역으로 찾는 평균 시간이다. 성능 결과에서 보는 것과 같이, 에지 검색의 경우, SINOS가 약 20ms, 기존 CCAM이 21ms로써, SINOS가 에지 검색 성능에서 우수하다. 아울러 POI 검색의 경우, SINOS가 약 1.9ms, 기존 CCAM이 약 2.1ms로써, SINOS가 보다 우수하다. 그 이유는 본 논문에서 제안하는 기법이 영역질의에 효율적인 R-트리를 에지 및 POI의 색인 구조로 가지기 때문이다. 아울러 영역 구간 사이에서는 영역에 상관없이 일정한 탐색 시간을 보이고 있다. 이와 같은 이유는 성능에 영향을 줄 정도로 영역 구간이 넓지 않고, 하나의 에지 내의 범위로 영역 구간을 정했기 때문이다.

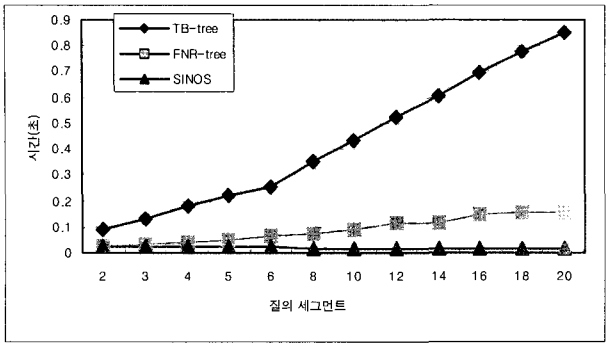
5.4 궤적 질의 성능비교

이동 객체의 궤적 질의는 질의 세그먼트의 수에 따라 해당 이동 객체의 궤적을 검색하는 시간을 측정하였다. 한편 궤적 질의의 유형을 살펴보면, 이동 객체의 모든 궤적을 찾는 것보다, 특정 시간에 이동하는 객체의 부분 궤적을 찾는



(a) 영역으로 에지 검색 (b) 영역으로 POI 검색

(그림 11) 영역 공간 질의



(그림 12) 궤적 질의

질의가 더 많다. 따라서 이에 근거하여 궤적 길이를 2에서부터 20까지 변화시키면서, 부분 궤적 질의의 성능 평가를 수행하였다. (그림 12)는 이동 객체의 궤적 질의의 검색 시간을 측정한 결과를 보여준다.

(그림 12)에서 나타난 것처럼 세그먼트가 2 인 경우, SINOS가 약 22ms, TB-트리가 94ms, FNR-트리가 25ms로써, SINOS가 TB-트리보다 약 4배의, FNR-트리보다 약 25%정도의 성능향상을 보인다. 아울러 세그먼트가 20 인 경우, SINOS가 약 19ms, TB-트리는 855ms, FNR-트리가 158ms로, SINOS가 TB-트리보다 약 45배, FNR-트리보다 약 8배의 성능 향상을 보인다. 그 이유는 본 논문에서 제안하는 기법은 질의궤적이 다수의 세그먼트를 가지고 있어도 이것을 하나의 질의 시그니쳐로 구성하여 탐색을 수행하므로, 질의궤적 세그먼트 개수에 상관없이 일정한 검색 성능을 보인다. 반면, TB-트리와 FNR-트리는 각 세그먼트마다 세그먼트의 영역 질의 후, 각 영역을 통과하는 모든 이동 객체를 탐색하기 때문에 세그먼트의 수가 증가할수록 비례적으로 검색 시간이 증가하기 때문이다. 아울러 TB-트리보다 FNR-트리가 성능이 더 좋은 것은 FNR-트리가 공간질의를 효율적으로 지원하는 R-트리로 구성되어 있기 때문이다.

5.5 공간 네트워크를 위한 색인 구조의 비교

제안한 저장 및 색인 구조인 SINOS가 기존의 연구들과 어떤 차이점을 가지고 있는지 공간 네트워크 데이터별로 비교하였다. <표 5>는 제안한 색인 구조와 기존의 연구들이 노드, 에지, POI, 이동 객체별로 어떤 색인 구조를 가지고 있는지 보여준다.

<표 5>에서 보듯이, SINOS가 공간 네트워크에서 가능한 4가지의 모든 데이터에 대해 색인구조를 제공함을 알 수 있다. 따라서 모든 데이터에 대한 빠른 접근이 가능하다. 아울러 5.1절에서 5.3절까지의 성능평가를 통해 노드, 에지, POI 각각에 대해 SINOS의 색인구조의 효율성을 알 수 있었다. 또한 5.4절의 성능평가를 통해, 이동 객체의 궤적을 색인하는 구조로 SINOS의 시그니쳐 색인구조가 기존의 FNR-트리나 TB-트리에 비해 궤적 검색 성능이 상당히 향상됨을 알 수 있었다. 그러나 5.1절에서 언급한 것처럼, SINOS가 기존 연구와 비교하여 다수의 색인 구조를 가지고 있기 때문에, 저장 공간의 오버헤드가 존재한다.

〈표 5〉 각 연구별 공간 네트워크를 위한 색인 구조의 비교

	CCAM	HKUST의 연구	FNR-트리	SINOS
노드	B^+-트리	없음	없음	B^+-트리
에지	없음	R-트리	R-트리	R-트리
POI	없음	R-트리	없음	R-트리,B^+-트리
이동 객체	없음	없음	R-트리	시그니쳐

6. 결 론

LBS(Location Based Service)를 효율적으로 지원하고, 실제 생활에 적용하기 위해서는 이상적인 공간이 아닌 실제상황과 유사한 공간 네트워크를 고려한 연구가 필요하다. 따라서 본 논문에서는 공간 네트워크를 고려한 효율적인 저장 구조와 공간 네트워크상에서 효율적인 색인기법을 설계하였다. 저장 부분에서는 노드와 에지간의 정보를 포함하고, B^+-트리를 이용해 노드에 대한 빠른 접근을 가능하게 하였다. 그리고 에지에 대한 정보뿐만 아니라, 에지 상에 존재하는 POI(point of interest)들의 리스트 정보도 지닌다. 또한 R-트리를 통한 구성으로 에지에 대한 효율적인 공간 질의도 가능하게 하였다. 아울러, 이동 객체 데이터에 대한 효율적인 궤적 검색을 지원하기 위해, 시그니쳐 기반 색인 기법을 제안하였다. 제안한 저장 및 색인 구조는 기존의 방법인 CCAM과 비교하여 보다 우수한 성능을 보임을 알 수 있었다. 특히 본 논문에서 제안한 이동 객체를 위한 시그니쳐 기반 색인구조는 기존의 TB-트리 및 FNR-트리보다 매우 우수한 성능으로 궤적 검색질의를 지원함을 알 수 있었다.

향후 연구로는 본 논문에서 제안하는 색인 및 저장 기법을 바탕으로, 다양한 과거 시간에 따른 과거 궤적 검색 질의에 대한 성능평가를 수행하는 것이다.

참 고 문 헌

[1] C. Shahabi, M.R.Kolahdouzan, M. Sharifzadeh, "A Road Network Embedding Technique for K-Nearest Neighbor Search in Moving Object Databases," GeoInformatica, Vol.7, No.3, pp.255-273, 2003.

[2] T. Brinkhoff, "A Framework for Generating Network-Based Moving Objects," GeoInformatica, Vol.6, No.2, pp.153-180, 2002.

[3] L. Speicys, C.S. Jensen, and A. Kligys, "Computational Data Modeling for Network-Constrained Moving Objects," Proc. of ACM GIS, pp.118-125, 2003.

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

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

[6] S. Shekhar and D.-R. Liu, "CCAM: A Connectivity-Clustered

Access Method for Networks and Network Computations," IEEE Tran. on Knowledge and Data Engineering, Vol.9, No.1, pp.102-119, 1997.

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

[8] B. Moon, H.V. Jagadish, C. Faloutsos, and J. H. Saltz. Analysis of the Clustering Properties of the Hilbert Space-Filling Curve. IEEE TKDE, 13(1): pp.124-141, 2001.

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

[10] J. Zobel, A. Moffat, and K. Ramamohanarao, "Inverted Files Versus Signature Files for Text Indexing," ACM Tran. on Database Systems, Vol.23, No.4, pp. 453-490, 1998.

[11] C. Faloutsos and S. Christodoulakis, "Signature Files: An Access Method for Docu-ments and Its Analytical performance Evaluation," ACM Tran. on Office Information Systems, Vol.2, No.4, pp.267-288, 1984.

[12] www.maproom.psu.edu/dcw/

엄 정 호

e-mail : jhum@dblab.chonbuk.ac.kr
2004년 전북대학교 컴퓨터공학과(공학사)
2004년~2006년 전북대학교 컴퓨터공학과 (공학석사)
2006년~현재 전북대학교 컴퓨터공학과 박사과정
관심분야 : 공간 데이터베이스, 공간 색인 구조, GIS

장 재 우

e-mail : jwchang@chonbuk.ac.kr
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