한국지리정보학회지 7권 4호 : 1~14(2004)

이동객체의 궤적처리를 위한 트리기반 색인기법의 성능분석

심춘보[1*] · 신용원[2]

Performance Analysis of Tree-based Indexing Scheme for Trajectories Processing of Moving Objects

Choon-Bo SHIM[1*] · Yong-Won SHIN[2]

요 약

본 연구에서는 GIS 응용에서 이동객체의 궤적을 인덱싱하기 위해 기존에 제안되었던 TB(Trajectory-Bundle)-트리의 성능을 개선시킬 수 있는 연결테이블(LinkTable:L-Table) 기반의 확장된 TB-트리(L-Table TB-Tree:LTB-Tree)를 제안하고 아울러 제안하는 색인기법의 성능평가를 위해 다음과 같은 사항을 고려한다. 첫째, 기존의 R*-트리, TB-트리, 그리고 제안하는 LTB-트리를 성능평가 대상으로 선정한다. 둘째, 실험 데이타 집합으로는 랜덤 데이타 집합 및 실제 데이타 집합을 이용한다. 셋째, 시스템의 가용 메모리의 제약을 고려해 메모리 버퍼 크기에 따른 성능평가를 수행한다. 넷째, 다양한 데이타 분포도를 가지고 있는 실험 데이타를 이용하여 성능평가를 수행한다. 마지막으로 삽입성능 및 검색성능(궤적질의 및 영역질의)을 평가한다. 성능평가를 분석한 결과, 제안하는 색인기법이 기존의 색인기법들에 비해 삽입과 궤적질의의 검색 측면에서 더 우수함을 보인다.

주요어: GIS, 이동객체, 색인기법, 성능 분석

ABSTRACT

In this study, we propose Linktable based on extended TB-Tree (LTB-Tree) which can improve the performance of existing TB (Trajectory-Bundle) -tree proposed for indexing the trajectory of moving objects in GIS Applications. In addition, in order to evaluate proposed indexing scheme, we ta

2004년 9월 10일 접수 Recieved on September 10, 2004 / 2004년 11월 3일 심사완료 Accepted on November 3, 2004
1 부산가톨릭대학교 컴퓨터정보공학부 School of Computer Information Engineering, Catholic University of Pusan
2 부산가톨릭대학교 병원경영학과 Dept. of Hospital Management, Catholic University of Pusan
※ 연락저자 E-mail: cbsim@cup.ac.kr

ke into account as follows. At first, we select existing R*-tree, TB-tree, and LTB-tree as the subject of performance evaluation. Secondly, we make use of random data set and real data set as experimental data. Thirdly, we evaluate the performance with respect to the variation of size of memory buffer by considering the restriction of available memory of a given system. Fourth, we test them by using the experimental data set with a variation of data distribution. Finally, we think over insertion and retrieval performance of trajectory query and range query as experimental measures. The experimental results show that the proposed indexing scheme, LTB-tree, gains better performance than traditional other schemes with respect to the insertion and retrieval of trajectory query.

KEYWORDS : GIS, Moving Objects, Indexing Scheme, Performance Analysis

서 론

최근 들어 GIS, LBS, 시공간 데이타베이스, 비디오 데이타를 다루는 멀티미디어 데이타베이스와 같은 다양한 분야에서 이동객체(moving objects)에 대한 많은 관심을 가지고 활발한 연구가 진행 중이다. 예를 들면, GPS와 같은 위치 추적 장치를 이용한 친구 찾기 서비스, 택배 및 배송 서비스, 긴급 구난 및 POI(point of interest) 서비스, 내용-기반 비디오 검색 서비스 등과 같은 다양한 응용 서비스들의 출현은 이를 한층 더 가중시키고 있다. 이와 더불어 데이타가 점점 대용량화되어가고 빠른 검색 성능을 보장해야하는 분야에서는 특히 색인기법에 대한 관심이 두드러지고 있는 추세이다.

이동객체의 색인기술에 관련된 연구로는 현재와 미래위치를 검색하기 위한 색인기술(Tayeb 등, 1998; Song 등, 2001; Tao 등, 2001), 과거위치를 검색하기 위한 색인기술(Theodoridis 등, 1996), 그리고 이동객체의 궤적을 효율적으로 검색하기 위한 궤적 색인기술(Pfoser 등, 1999; Pfoser 등 2000)로 나뉜다. 초기의 이동객체의 궤적을 색인하기 위한 기법으로는 기존의 공간 데이타를 위해 제안되었던 R-트리 기반의 색인기법(Guttman, 1984; Sellis 등, 1987; Beckmann 등, 1990; Nascimento 등, 1998)이 널리 이용되었다. 그러나 이 기법은 데이타 삽입이나 갱신으로 인해 색인이 변경될 때 단지 데이타 값만 변경해주는 것이 아니라 데이타 변경에 따른 색인의 노드들이 분할되거나 합병됨으로써 색인의 구조가 변경되는 문제점을 안고 있으며 이는 GIS 분야에서의 공간 데이타와 같이 한 번 입력으로 인해 자주 변경되지 않는 경우에는 유리하나 이동객체와 같이 삽입과 변경 연산이 많이 이루어지고 객체의 궤적을 검색하는 질의처리에는 적합하지 않다. 반면에 공간적인 특성을 가지고 있는 영역질의에 대해서 좋은 성능을 보이는 것으로 알려져 있다. 최근의 연구들 중에 이동객체의 궤적-기반 질의에 대해서 좋은 성능을 보이는 TB-트리(Pfoser 등, 2000)가 있다. 이 기법은 시간의 흐름에 따라 연속적으로 변하는 이동객체의 궤적을 보간법(interpolation)을 이용해서 라인 세그먼트의 집합으로 저장하는 방식으로 동일한 객체에 속하는 궤적 세그먼트들을 같은 단말노드에 삽입하여 유지하는 궤적 보존 전략을 채택하여 궤적-기반 질의 처리에 탁월한 성능을 보이는 것으로 알려져 있으며 아울러 저장공간의 공간활용도가 거의 100%에 가까울 정도의 성능을 보인다. 그러나 삽입의 경우, 삽입하려고 하는 라인 세그먼트의 이전 세그먼트를 포함하는 단말노드를 찾기 위해 트리 전체를 탐색해야 하며, 검색의 경우 또한 마찬가지로 원하는

궤적의 첫 번째 궤적 세그먼트를 검색하기 위해서는 트리의 루트노드부터 단말노드까지 탐색해야 하는 단점을 가지고 있다.

따라서 본 연구에서는 이동객체의 궤적-기반 질의에 대해서 검색 성능을 제공하기 위해 기존에 제안되었던 TB-트리의 성능을 향상시킬 수 있는 연결테이블(linktable) 기반의 확장된 TB-트리 즉, LTB-트리를 제안하며, 아울러 제안하는 색인기법을 다양한 측면을 고려하여 성능평가를 수행한 후 이를 분석한 결과를 보인다.

본 논문의 구성은 다음과 같다. 제2장에서는 관련 연구로서 성능평가의 대상인 기존의 R*-트리와 TB-트리에 대해서 기술하고, 제3장에서는 기존의 TB-트리를 확장한 연결테이블 기반의 제안하는 LTB-트리에 대해서 설명한다. 제4장에서는 실험환경 및 성능평가에 대해서 언급하고, 5장에서는 결론 및 향후 연구방향을 제시한다.

관련 연구

1. R*-트리

R*-트리(Beckmann 등, 1990)는 점이나 선, 면과 같은 다양한 공간 데이타를 저장하기 위해 기존에 제안된 R-트리(Guttman, 1984)의 변형된 트리이다. 기본적인 구조와 연산은 기존의 R-트리와 거의 동일하며, 삽입이나 검색 성능을 개선하여 이동객체를 위한 다양한 색인기법들이 제안되기 전에 많이 사용되었던 색인기법이다. R*-트리는 기존의 R-트리가 데이타를 삽입할 때 최소 영역 증가정책(least area enlargement policy)을 사용하여 영역 정보만을 고려했던 부분을 개선하여 중복(overlap)과 가장자리(margin) 정보를 추가적으로 고려함으로써 삽입 성능을 향상시켰을 뿐만 아니라, 재삽입 정책을 이용하여 R-트리의 공간활용도 부분도 향상시켰다. R*-트리는 기본적으로 모든 단말노드의 높이가 동일한 균형 트리이며, 다양한 공간 데이타를 처리하기 위해 최소 경계 사각형, 즉 MBR(minimum bounding rectangle)을 이용하여 데이타를 인덱싱한다. 그림 1은 R*-트리의 구조를 도식화한 것이다.

그림 1의 왼쪽의 실선으로 표현된 사각형은 공간 객체를 의미하며, 점선으로 나타낸 사각형들은 비단말노드들을 의미한다. 비단말노드에는 객체를 모두 포함하는 MBR 정보와 자식 노드를 가르키는 포인터 정보로 구성된다. 그에 반해 단말노드는 각 객체의 MBR 정보와 실제 공간 객체를 가리키는 포인터 정보로 구성되어 있다. FUR(frequent updates in R-tree)(Mong 등, 2003)는 R-트리에서 기존의 top-down 방식의 갱신 비용을 줄이기 위해 단말노드의 모든 객체들을 가리키는 정보를 해쉬 테이블을 이용하여 직접 접근할 수 있는 구조를 제안하였다. 그러나 이 구조는 본 논문에서 제안하는 방법과 유사한 테이블 구조를 이용하고 있으나, 이동객체의 첫번째 세그먼트와 마지막 세그먼트를 포함하고 있는 노드의 정보를 테이블에 담고 있는 것과는 달리 단말노드의 모든 객체들에 대한 정보를 가지고 있으며, 또한 테이블에 단말노드의 객체에 대한 포인터만을 가지고 있는 구조이다.

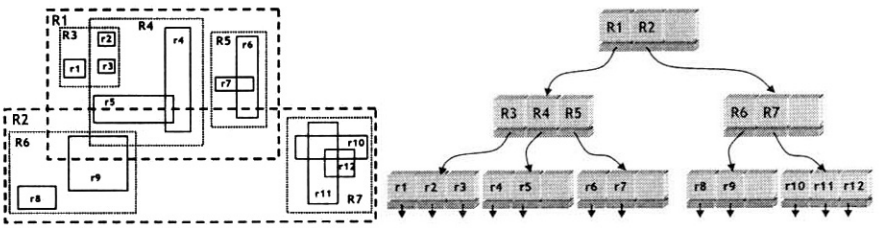


FIGURE 1. Overall structure of R*-tree

그리고 R-트리의 갱신 비용을 고려한 구조이지 궤적질의를 위한 것은 아니다.

2. TB-트리

TB-트리(Pfoser 등, 2000)는 좌표 같은 공간 정보는 고려하지 않고 극단적으로 이동객체의 궤적에만 초점을 맞추어 제안된 색인기법이다. 즉, 시간에 따라 연속적으로 변화하는 위치 정보를 표현하기 위해 이동객체의 궤적을 보간법을 이용하여 라인 세그먼트(line segment)로 저장하는 방법이다. 동일한 이동객체에 속하는 라인 세그먼트들을 같은 단말노드에 삽입하여 유지하는 궤적 보존 전략(trajectory preservation strategy)을 채택하고 있다. 따라서 공간적으로 근접하더라도, 같은 궤적의 세그먼트가 아니면 다른 단말노드에 저장된다. 그림 2는 TB-트리의 전체적인 구조를 나타낸 것이다. 비단말노드는 하위 자식 노드에 속한 객체의 궤적 세그먼트들을 모두 포함하는 MBB 정보를 저장하며, 단말노드는 객체의 궤적 세그먼트들을 저장하고 하나의 단말노드에는 반드시 같은 궤적에 속하는 라인 세그먼트만을 포함한다. 즉, 그림 2에서 객체 A에 속하는 라인 세그먼트, a1에서 a6는 단말노드 LN1과 LN4에 삽입되며, 단말노드(LN1)가 모두 가득 차있는 경우는 새로운 단말노드(LN4)를 생성해서 저장한 후 같은 궤적 세그먼트를 포함하고 있는 단말노드들 사이에 양방향 연결리스트(doubly linked list)를 이용하여 연결시키도록 설계되어 있다.

TB-트리의 삽입 알고리즘은 새로운 세그먼트가 삽입되면 삽입된 세그먼트의 이전의 선행 세그먼트가 있는 단말노드에 삽입을 하기 위해 STR(Spatio-Temporal R-tree)-트리(Pfoser 등, 2000)와 동일한 방식으로 FindNode() 함수를 호출하여 선행 단말노드의 위치를 찾는다. 그리고 검색된 선행노드에 새로운 세그먼트에 대한 엔트리를 추가한다. 만약에 선행노드가 가득 차 있을 경우에는 노드의 분할이 없이 단지 새로운 단말노드를 생성해서 엔트리를 추가하게 된다. 효율적인 궤적 추출을 위해, 동일한 이동객체의 궤적을 저장하고 있는 단말노드들간에는 양방향 연결 리스트로 링크되어 있어 임의의 단말노드로부터 해당하는 전체 궤적을 추출하는 데 매우 용이하다. TB-트리는 궤적 보전 전략을 사용하기 때문에 저장 공간의 오버헤드가 없다. 즉, 저장 공간활용도가 거의 100%에 가깝다. 아울러 이동객체의 궤적-기반 질의에 매우 탁월한 성능을 보인다. 반면에 TB-트리는 이동객체의 새로운 라인 세그먼트를 삽입할 때마다 궤적 보존 전략의 특성상 동일한 객체의 선행 라인 세그먼트가 삽입된 단말노드를 찾아야 한다. 따라서 루트노드에서부터 단말노드까지 트리를 순회한다. 이는 삽입시 불필요한 많은 노드를 접근해야 하는 문제점을 가지고 있다. 특히 시간이 지남에 따라 계속해서 증가하는 이동객체의 특성상 트리의 크기가 커질수록 삽입될 객체의 선행 노드를 검색하는데 많은 비용이 들게 된다.

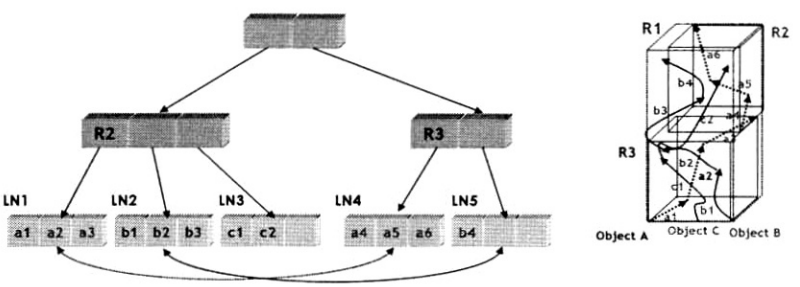


FIGURE 2. Overall structure of TB-tree

제안하는 색인기법

본 연구에서는 기존의 TB-트리의 단점을 보완하고 이동객체의 궤적을 구성하는 라인 세그먼트를 삽입할 때의 삽입 성능을 개선하며 저장공간 측면과 궤적-기반 질의처리의 효율성을 높이기 위해 기존의 TB-트리를 변형시킨 구조의 색인기법을 제안한다. 제안하는 색인기법의 설계시 고려사항 및 성능 분석시 고려사항을 정리하면 다음과 같다.

1. **설계시 고려사항**
   1) LTB-트리는 기존의 TB-트리의 구조에 객체의 첫번째 라인 세그먼트와 마지막 세그먼트를 포함하는 노드의 포인터 정보와 디스크의 페이지 번호로 구성된 연결 테이블을 결합한 구조로서 연결 테이블 정보를 이용하여 기존 TB-트리의 삽입 및 검색 성능을 향상시킨다.
   2) LTB-트리는 디스크 기반의 트리 구조로서 삽입 및 갱신으로 인해 메모리에 상주된 색인정보가 변경되는 즉시 디스크에 반영함으로써 색인의 일관성을 유지하도록 설계한다.
   3) 버퍼링 정책을 이용하여 시스템의 가용메모리의 제약으로 인해 색인 전체가 메모리에 상주되지 못하는 경우를 고려하여 설계한다.
2. **성능분석시 고려사항**
   1) 성능평가는 기존의 R*-트리와 TB-트리, 그리고 제안하는 LTB-트리와 비교한다.
   2) 실험 데이타는 랜덤 데이타와 실제 데이타를 모두 이용한다.
   3) 대용량 데이타 및 Random과 Gaussian 데이타 분포를 가지는 데이타 집합을 이용한다.
   4) 버퍼 크기에 따른 성능평가 및 삽입, 검색 성능(영역질의, 궤적질의)에 대해서 성능 분석을 수행한다.

제안하는 색인구조는 그림 3과 같이 기존의 TB-트리의 구조와 전체적으로 유사하며, 단지 선행 노드를 직접적으로 접근할 수 있도록 하기 위해 각 이동객체의 처음 세그먼트와 마지막 세그먼트가 저장된 단말노드를 가리키는 메모리 포인터 정보와 노드를 저장하기 위한 디스크 페이지 번호를 유지하는 연결테이블(L-Table)이 별도로 존재한다. L-Table 구조는 헤더 정보와 단말노드에 대한 포인터 정보와 디스크의 페이지 번호로 구성된다. L-Table 헤더는 total_rec과 id_len으로 구성되며, total_rec은 저장된 전체 이동객체의 수를 나타내며, id_len은 이동객체의 부가적인 세부 정보를 가리키는 식별자를

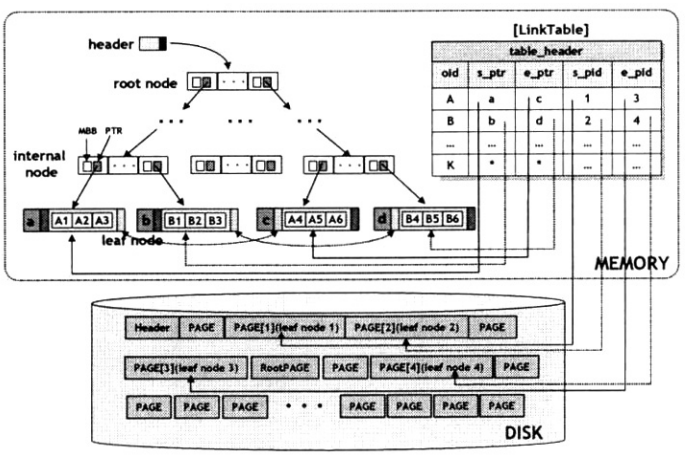


FIGURE 3. Overall structure of LTB-tree

위한 식별자 길이 정보를 의미한다. L-Table의 각 레코드에는 oid, s_ptr, e_ptr, s_pid, e_pid 필드가 있으며, 각각은 이동객체의 식별자, 이동객체의 첫 번째 및 마지막 라인 세그먼트를 저장하고 있는 단말노드의 메모리 포인터와 각각 디스크 상에서의 페이지 번호들을 나타낸다.

제안하는 색인구조는 궤적에 대한 라인 세그먼트를 삽입할 때 기존의 TB-트리처럼 트리의 루트노드에서부터 단말노드까지 순회하지 않고 L-Table에서 해당 객체의 마지막 라인 세그먼트를 포함하고 있는 단말노드를 가리키는 포인터(e_ptr)와 디스크 상에서의 페이지 번호(e_pid)를 이용하여 한 번에 직접적으로 단말노드에 접근함으로써 삽입 성능의 효율성을 향상시킬 수 있다. 또한 이동객체의 궤적질의 처리 시 해당 객체의 선행 궤적 정보를 페이지 번호를 통해 직접 디스크로부터 읽기 때문에 페이지 접근 횟수를 줄일 수 있다. 디스크 기반의 구조를 가지고 있어 새로운 이동객체의 라인 세그먼트를 삽입시 해당 객체의 궤적을 바로 디스크에 반영해 메모리 상의 트리 구조와 디스크 상의 트리 구조 사이의 일관성을 유지한다.

제안하는 색인구조는 여섯 개의 클래스 즉, cETBTree 클래스, cETBTreeBuffer 클래스, cETBTreeLinkTable 클래스, cETBDirNode 클래스, cETBLeafNode 클래스, cETBEntry 클래스로 구성되어 있다. 표 1은 각 클래스에 대한 기능 설명 및 세부 함수들을 나타낸다.

ETB-트리의 삽입 알고리즘은 기존의 TB-트리의 삽입 알고리즘과 거의 유사하다. 단지 삽입 시 L-Table 정보를 이용해서 이동객체의 선행 라인 세그먼트를 찾으며, 아울러 새로운 단말노드가 생성되게 되면 L-Table 정보를 갱신한다. 이동객체의 라인 세그먼트를 삽입하기 위해서는 그림 4의 LTB-트리 삽입 알고리즘에서와 같이 LTB_Insert()함수를 호출한다. LTB_Insert()함수는 삽입하려고 하는 객체가 기존의 트리에 존재하는 객체인지 확인하기 위해 LTB_FindLeafNode() 함수를 호출한다. 이 함수는 트리에 삽입하려고 하는 객체와 같은 객체가 이미 존재한다면 그 객체의 마지막 라인 세그먼트가 저장되어 있는 단말노드를 반환한다. 만일 삽입하려는 객체와 같은 객체가 존재하지 않는다면 즉, 처음 삽입하는 객체라면 널(Null)값을 반환을 한다. LTB_Insert() 함수는 LTB_FindLeafNode()함수에서 반환되는 값을 확인하고 새로운 단말노드를 만들 것인지 아니면 트리에 있는 단말노드에 삽입할 것인지를 결정한다. 만약에 단말노드에 여유 공간이 없다면 새로운 단말노드를 생성하고 생성된 단말노드에 삽입한 후 종료한다. 그렇지 않고 선행 라인 세그먼트의 단말노드가 존재하지 않으면 새로운 이동객체의 첫 번째 라인 세그먼트로 간주하고 새로

TABLE 1. Description for each class

Class Name	Description
cETBTree	트리의 생성 및 세그먼트 삽입과 질의처리를 위한 함수를 포함하는 클래스
cETBTreeBuffer	버퍼링에 관련된 함수를 포함하는 클래스
cETBTreeLinkTable	L-Table에 대한 자료 구조 및 레코드 관리에 대한 함수를 포함하는 클래스
cETBDirNode	트리의 비단말노드에 대한 자료 구조 및 MBR에 관련된 함수를 포함하는 클래스
cETBLeafNode	트리의 단말노드에 대한 자료 구조 및 이동객체의 세그먼트를 저장하기 위한 함수를 포함하는 클래스
cETBEntry	트리의 비단말노드를 구성하는 엔트리들을 위한 함수를 포함하는 클래스

운 단말노드를 생성해서 저장하고 종료한다. 삽입 후 단말노드에서부터 루트노드까지의 propagation 처리는 기존의 TB-트리와 매우 유사하며 보다 단순하게 처리하기 위해 부모 노드와 자식 노드 사이에 양방향 포인터를 이용하고 있다. 즉, 자식 노드에 새로운 세그먼트가 삽입이 되었거나 엔트리의 정보가 변경되었을 경우에 양방향 포인터를 통해 부모 노드를 탐색하여 해당 노드에 대한 변경된 정보를 반영하고 이는 루트노드까지 전파된다.

LTB_FindLeafNode는 그림 5와 같이 해당

Algorithm 1: LTB_Insert Algorithm

```
Procedure LTB_Insert(L, oid)
    Input
       L : 새로 삽입될 라인 세그먼트(line segment)
       oid : 라인 세그먼트 L이 속해 있는 이동객체의 식별자
    Output : 성공 혹은 실패

1:     1 삽입될 라인 세그먼트 L의 선행 세그먼트가 저장된 단말노드 N'를 찾기 위해
       LTB_FindLeafNode(oid, link_type) 함수 호출
2:   2 만약 단말노드 N'가 발견되면
3:       2.1 단말노드 N'에 여유 공간이 있다면
4:           2.1.1 라인 세그먼트 L을 단말노드 N'에 삽입후에 변경 정보를 부모 노드(루트까
                 지)에 전파
5:       2.2 여유 공간이 없다면 즉, 가득 차 있으면
6:           2.2.1 새로운 단말노드 생성후에 라인 세그먼트 삽입 및 변경 정보를 부모 노드(루
                 트까지)에 전파
7:   3 단말노드 N'가 발견되지 않으면(N'가 NULL이면)
8:     3.1 기존의 삽입된 선행 라인 세그먼트가 없기 때문에 즉, 처음 삽입된
           이동객체의 라인 세그먼트이므로 새로운 단말노드 생성후, 라인 세그먼트를 삽입과
           링크테이블에 갱신 및 변경 정보를 부모 노드(루트까지)에 전파
```

FIGURE 4. Algorithm for insertion of LTB-tree

oid와 첫 번째 라인 세그먼트의 단말노드인지 아니면 마지막 라인 세그먼트의 단말노드인지를 가르키는 link_type을 가지고 L-Table 정보를 참조해서 oid의 s_ptr 즉, 첫 번째 라인 세그먼트의 단말노드 또는 e_ptr 즉, 마지막 라인 세그먼트의 단말노드를 반환한다.

성능평가 및 분석

1. 실험환경

본 연구에서는 펜티엄-Ⅳ 2.8GHz CPU와 메인 메모리 1GB, 윈도우 2000에서 C++ 언어를 이용하여 제안하는 색인구조를 구현하였으며, 삽입 성능과 검색 성능에 대해서 성능평가를 수행하였다. 특히 검색 성능의 경우에는 각각 궤적질의와 영역질의에 대해서 성능을 평가하였다. 성능평가는 시스템의 하드웨어 환경에 영향을 받지 않는 노드 접근 횟수를 측정하였다. 아울러 기존의 이동객체의 영역질의에 대해서 우수한 성능을 나타내는 R*-트리와 궤적질의에 대해서 우수한 성능을 보이는 TB-트리, 그리고 제안하는 LTB-트리를 성능평가 대상으로 선택했다. R*-트리, TB-트리, 그리고 제안하는 LTB-트리 모

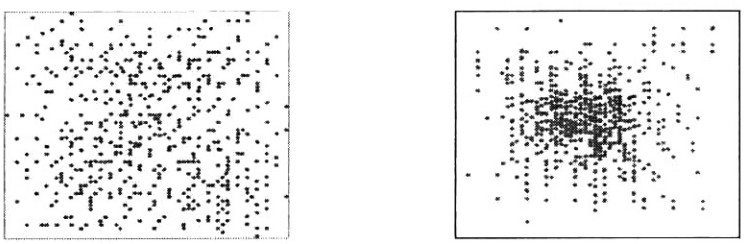
Algorithm 2: LTB_FindLeafNode Algorithm

```
Algorithm
    Procedure LTB_FindLeafNode(oid, link_type)
    Input
        oid : 라인 세그먼트 L이 속해 있는 이동객체의 식별자
          link_type : First 혹은 Last값을 가지며 해당 oid의 첫 번째 단말노드를 찾을
                      것인지 아니면 마지막 단말노드를 찾을 것인지를 나타내는 flag
    Output : 검색한 단말노드 혹은 Null 값을 반환

1:     1 oid를 이용해서 테이블 정보를 검색
2:     2 해당 oid가 테이블 존재하면
3:       2.1 link_type이 First이면
4:    2.1.1 첫 번째 라인 세그먼트를 가리키는 s_ptr를 반환
5:          2.2 link_type이 Last이면
6:              2.2.1 마지막 라인 세그먼트를 가리키는 e_ptr를 반환
7:      3 테이블에 존재하지 않으면
8:          3.1 Null 반환
```

FIGURE 5. Algorithm for FindLeafNode of LTB-tree

두 디스크 페이지의 크기는 4KB로 설정했다. 성능평가를 위해 사용한 실험 데이터는 GSTD(Theodoridis 등, 1999)와 Runtime21(Census, 1994; Brinkhoff, 2002)을 이용하여 생성한 데이타로서 특히 Runtime21은 로드맵(loadmap)에 따라 이동하는 자동차들과 같이 지리 네트워크 정보(TIGER/Line)를 이용해 실제 데이타와 거의 유사한 데이타 집합을 생성할 수 있는 도구이다. GSTD는 이동객체를 위한 랜덤 데이타를 생성하기 위해 가장 많이 이용하는 데이타 생성 도구로서 다양한 형태의 데이타 분포도를 고려하여 생성할 수 있는 장점을 가지고 있다. 따라서 본 연구에서는 랜덤 데이타를 위해서는 표 2와 같이 데이타 분포도를 고려하여 GSTD 알고리즘을 이용해 각 분포도에 따라 객체의 수가 1,000(1K)에서부터 10,000(10K)까지 100초 동안 이동한 데이터를 toriod 좌표계를 사용해 생성하였다. Uniform 과 Gaussian의 초기 분포도를 그림을 나타내면 그림 6과 같다. toriod 좌표계는



(a) Uniform (b) Gaussian

FIGURE 6. Uniform and Gaussian distribution

GSTD알고리즘에서 데이타 생성시 이동객체의 궤적이 좌표평면상에서 어떤 한 축으로 벗어나는 경우, 벗어나는 축을 기준으로 대칭시켜 이동시켜 좌표를 얻어내는 좌표계를 말한다. 실제 데이타를 위해서는 Runtime21을 이용하여 표 3과 같이 생성하였다.

TABLE 2. Random dataset generated by GSTD algorithm

	Initial distribution	Moving pattern
U-R 데이타세트	Uniform	Random
U-D 데이타세트	Uniform	Directed
G-R 데이타세트	Gaussian	Random
G-D 데이타세트	Gaussian	Directd

TABLE 3. Real dataset generated by Runtime21

# of segments	# of moving objects	time
50,000	500	100 sec.
200,000	2,000	100 sec.
500,000	5,000	100 sec.

2. 성능분석

본 연구에서는 비교적 다양한 조건하에서 성능평가를 수행하기 위해 다음의 사항들을 고려하였다. 첫째, 제안하는 색인기법은 디스크 기반의 색인구조로서 시스템의 가용 메모리의 제약을 고려해 메모리의 버퍼 크기에 따른 성능평가를 수행하였다. 둘째, 성능평가를 위한 실험 데이터는 다양한 형태의 분포도를 가지고 있는 랜덤 데이터 뿐만 아니라 로드맵에 따라 도로 위를 이동하는 자동차와 같이 도로 네트워크에 제약을 받는 실제 데이터를 이용하였다. 셋째, 성능평가는 성능평가를 수행하는 시스템의 환경에 독립적인 노드 접근 횟수를 측정하는 방법을 이용하였다. 마지막으로 성능평가는 삽입 성능과 검색 성능, 특히 검색 성능은 이동객체의 궤적 특성을 고려해 궤적질의와 영역질의를 각각 측정하였다.

첫 번째 실험의 버퍼의 크기에 따른 성능평가는 표 3의 Runtime21 실제 데이타 집합을 이용했으며, 제안하는 색인기법만을 평가 대상으로 하였다. 왜냐하면 기존의 R*-트리와 TB-트리는 버퍼를 이용하고 있지 않기 때문이다. 그림 7은 버퍼 크기의 변화에 따른 삽입 성능에 대한 노드 접근 횟수를 측정한 결과로 버퍼 크기가 클 수록 삽입 성능이 향상됨을 알 수 있다. 특히 데이타 집합의 크기가 커질수록 버퍼 크기가 노드 접근 횟수에 큰 영향을 미치는 것을 확인할 수 있다. 이동객체의 라인 세그먼트의 수가 500,000일 때 버퍼의 크기가 10%일 때에 비해 50%일 때 약 2배 정도의 노드 접근 횟수의 감소가 일어남을 알 수 있다. 따라서 비교적 시스템의 가용 메모리가 클수록 색인의 많은 부분이 메모리에 상주할 수 있기에 성능을 향상시킬 수 있다.

영역질의에 대한 성능평가를 위해 질의영역의 크기는 전체 데이터 공간의 0.1%, 1%, 10%로 나누어 성능평가를 수행하였다. 그러나 지면관계상 본 논문에서는 그림 8과 같이 버퍼 크기의 변화에 따라 영역질의 1%에 대한 검색 성능에 대해 노드 접근 횟수를 측정한 결과를 보여준다. 버퍼 크기가 10%일 때는 500,000일 경우 가장 큰 노드 접근 횟수를 보이다가 버퍼 크기가 50%로 증가할 때 가장 적은 수의 노드 접근 횟수를 나타내고 있다. 이는 50,000의 경우와 같

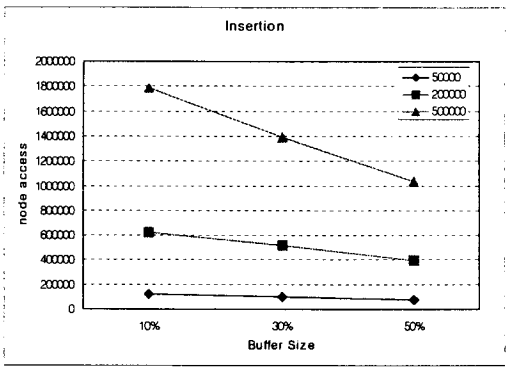


FIGURE 7. Experimental results for insertion with buffer size

이 데이타의 크기가 적은 경우에는 버퍼 크기에 큰 영향을 받지 않지만, 데이타의 크기가 큰 경우 즉, 500,000의 경우 버퍼 크기가 작은 경우에는 색인의 일부만이 메모리에 상주되기 때문에 노드 접근 횟수가 증가하는 반면에 버퍼 크기가 큰 경우에 색인의 많은 부분이 메모리에 상주될 수 있기 때문에 노드 접근 횟수가 적어 검색 성능이 향상됨을 알 수 있다.

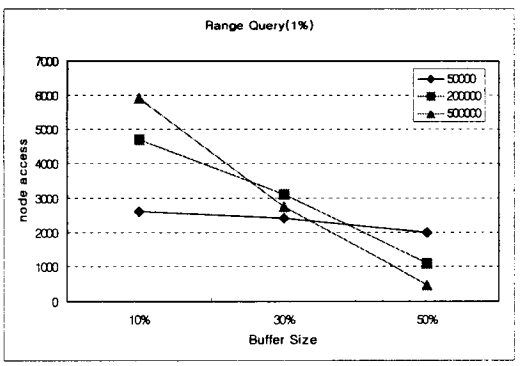


FIGURE 8. Experimental results for range query(1%) with buffer size

그림 9는 버퍼 크기의 변화에 따른 궤적질의에 대한 검색 성능에 대해 노드 접근 횟수를 측정한 결과를 보여준다. 궤적질의는 임의의 100개의 질의 이동객체 식별자(oid)를 선별해서 해당 이동객체의 전체 궤적시간 영역 중에 10% 구간을 검색하는 방식으로 질의를 처리하였다. 그림 8의 영역질의와 비슷한 형태를 보이고 있으며, 대체적으로 라인 세그먼트의 수 즉, 데이터의 크기가 적은 경우는 버퍼의 크기에 비교적 큰 영향을 받지 않는 반면에 데이터의 크기가 큰 경우에는 버퍼의 크기를 얼마만큼 설정하느냐에 따라 검색 성능에 지대한 영향을 미칠 수 있다는 걸 알 수 있다.

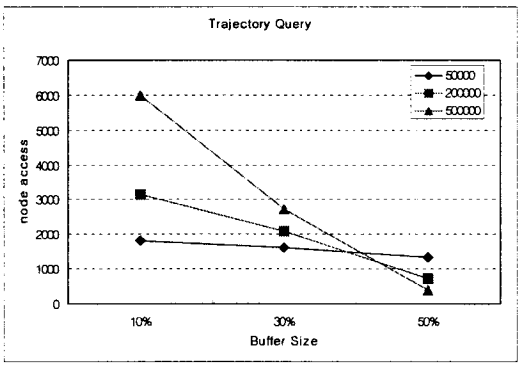


FIGURE 9. Experimental results for trajectory query with buffer size

다음으로 데이타 분포도에 따른 성능평가는 표 2의 GSTD 데이터세트를 이용하였으며, 성능평가는 기존의 R*-트리, TB-트리 그리고 제안하는 LTB-트리에 대해서 성능을 비교하였다. 그림 10은 데이터 분포도에 따른 삽입 성능에 대한 노드 접근 횟수를 측정한 결과이다. 데이터 집합이 가지는 분포도에 따라 다소 차이는 보이지만 대체적으로 궤적질의를 위해 기존에 제안되었던 TB-트리가 가장 성능이 좋지 않다. 이는 객체의 라인 세그먼트가 삽입될 때마다 이전 세그먼트를 탐색하기 위해 루트노드에서부터 단말노드까지 트리 전체를 탐색해야 하는 단점을 가지고 있기 때문이다. 그에 반해 R*-트리는 공간적으로 가까운 위치에 존재하는 세그먼트는 같은 MBR에 삽입하는 정책을 적용하기 때문에 트리 전체를 탐색하는 오버헤드는 없다. 따라서 TB-트리보다는 삽입 성능에서 더 좋은 결과를 보인다. 대체적으로 제안하는 LTB-트리

가 가장 좋은 성능을 나타내며 객체의 수가 증가할수록 기존의 TB-트리와 R*-트리에 비해 훨씬 더 완만한 그래프를 나타내고 있다. 아울러 데이터 집합이 가지는 분포도에도 그리 큰 영향을 받지 않는 편이다.

그림 11은 데이터 분포도에 따른 검색 성능 중에서 영역질의에 대한 노드 접근 횟수를 측정한 결과이다. 버퍼 크기에 따른 실험과 마찬가지로 영역질의는 전체 데이터 공간을 100%로 간주하고 그 중에서 0.1%, 1%, 10% 질의 영역에 대해서 영역질의를 실험하였으나 그 중에서 1% 질의영역에 대한 영역 질의 결과이다. 일반적으로 알려진 바와 같이 영역질의에 대해서는 기존의 R*-트리가 가장 좋은 성능을 보인다. 특히 R*-트리는 객체의 수의 변화에 큰 영향을 받지 않으며 일정한 성능을 보인다. 단지 G-R 데이타세트에 경우에만 다소 변화를 보일 뿐이다. 그에 반해 기존의 TB-트리와 제안하는 LTB-트리는 거의 비슷한 성능을 나타내며 객체의 수가 증가할수록 성능이 가파르게 떨어짐을 알 수 있다. 이는 TB-트리와 LTB-트리 모두 영역질의를 위한 특별한 정책이 없으며 그에 따라 질의영역에 포함되는 모든 객체의 궤적 세그먼트를 검색하기 위해서는 트리 전체를 탐색해야 하는 문제점을 안고 있기 때문에 성능이 좋지 않다. 따라서 이러한 영역질의를 향상시킬 수 있는 특별한 기법이 더 추가되어져야 할 것

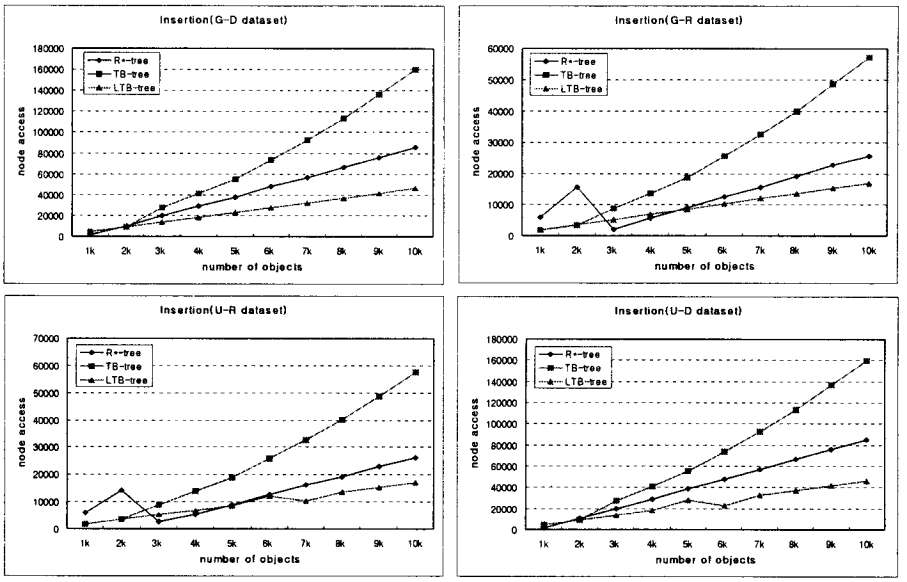


FIGURE 10. Experimental results for insertion with data distribution

이다.

마지막으로 그림 12는 데이터 분포도에 따른 검색 성능 중에서 궤적질의에 대한 노드 접근 횟수를 측정한 결과이다. 이전의 영역질의에 대한 성능평가 결과와는 달리 궤적질의에 대해서는 모든 데이터 분포도에 대해서 기존의 R*-트리가 가장 낮은 성능을 보이고 있다. 이는 R*-트리가 궤적질의 즉, 임의의 이동객체의 궤적을 구성하는 모든 라인세그먼트를 탐색하기 위해서는 트리 전체를 탐색해야 하기 때문이다. 그리고 TB-트리와 LTB-트리는 데이터의 분포도에 관계없이 거의 일정한 성능을 나타내며, 특히 LTB-트리는 궤적 질의처리시 기존의 TB-트리가 원하는 궤적의 첫 번째 세그먼트를 찾기 위해 트리의 루트노드에서부터 단말노드까지 트리 탐색 과정이 없이 메모리 상에 상주되어 있는

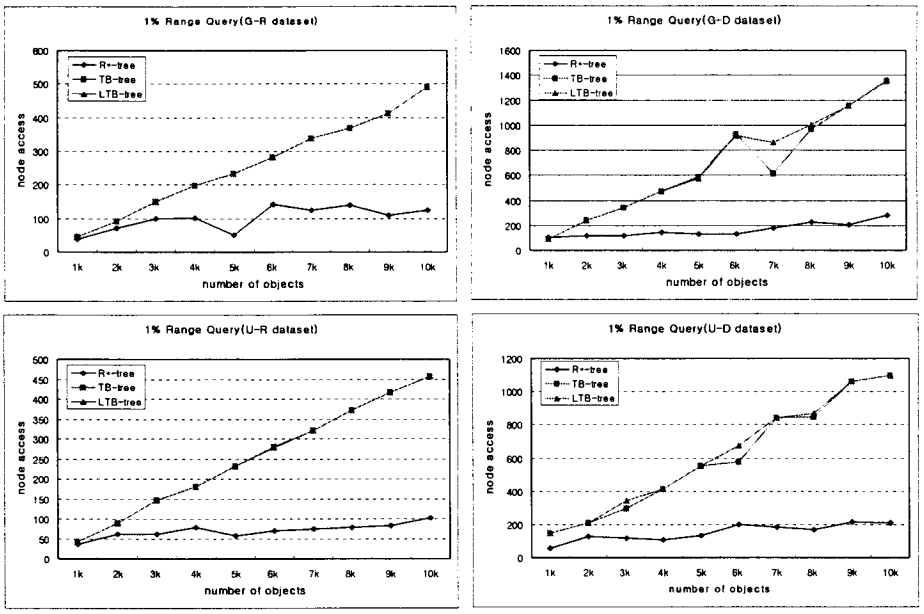


FIGURE 11. Experimental results for range query(1%) with data distribution

L-Table을 통해서 첫 번째 궤적 세그먼트를 포함하고 있는 노드 정보를 가져와서 처리하기 때문에 단지 양방향 연결 리스트로 되어 있는 단말노드만을 탐색하는 데 드는 비용만 요구한다. 따라서 제안하는 LTB-트리는 궤적질의 처리에는 기존의 TB-트리보다 약 10배 정도의 성능 향상을 가져올 수 있다.

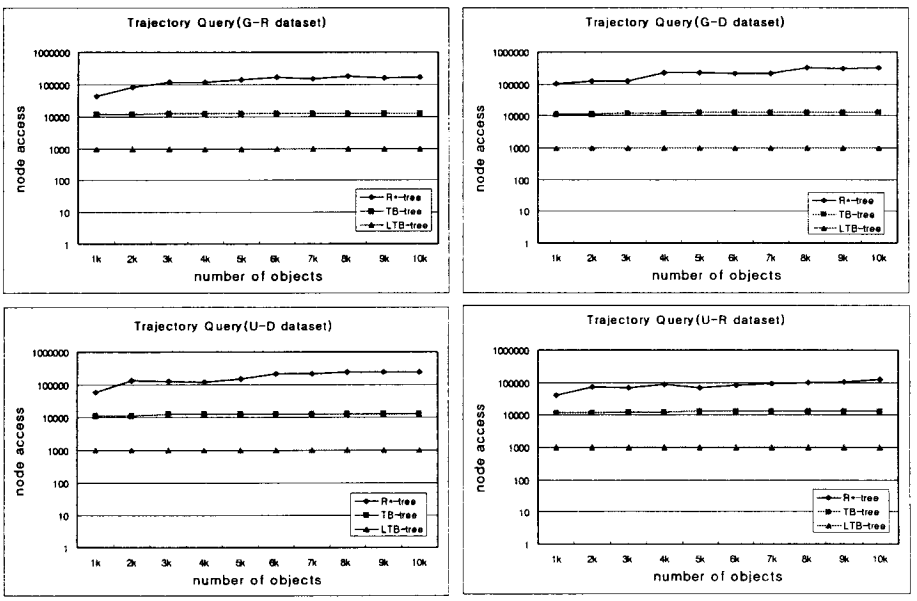


FIGURE 12. Experimental results for trajectory query with data distribution

결론 및 향후 연구

GPS와 같은 위치 추적장치의 빠른 보급과 기술의 발달로 인해 일상생활에서도 쉽게 관심이 있는 이동객체에 대한 위치파악 및 추적기능을 이용한 다양한 응용서비스 즉, 차량 네비게이션 시스템, 친구찾기, 전자상거래, 물류 및 택배 서비스 등을 쉽게 이용할 수 있게 되었다. 본 논문에서는 이러한 이동객체를 이용한 다양한 응용서비스를 위해 이동객체의 궤적 정보를 효율적으로 색인하기 위해 기존에 제안되었던 TB-트리의 성능을 개선시킬 수 있는 연결테이블 기반의 확장된 TB-트리를 제안하였고 아울러 제안하는 색인기법을 기존의 R*-트리와 TB-트리와 비교하여 다양한 실험 데이타를 이용하여 삽입 및 검색성능에 대해서 성능 분석을 수행하였다. 성능분석 결과, 영역질의에 대한 검색 성능을 제외하고는 제안하는 LTB-트리가 기존의 R*-트리와 TB-트리에 비해 삽입성능 및 검색성능 특히 궤적 기반 질의에 대해서 더 우수함을 증명할 수 있었다. 앞으로는 향후 연구로는 제안하는 색인구조를 실제 LBS 응용이나 GIS 시스템에서 동작할 수 있는 응용서비스에 접목시켜 실제 시스템에 적용시키는 연구와 가상적인 공간에서 이동하는 이동객체가 아닌 실제 도로나 철도와 같은 로드맵(loadmap)을 고려한 공간 네트워크 환경(Pfoser 등, 2003; Shahabi 등, 2003; Speicys 등, 2003)에서도 효율적으로 동작할 수 있는 색인기법에 관한 연구를 수행할 것이다. KAGIS

참고문헌

Beckmann, N., H. Kriegel, R. Schneider and B. Seeger. 1990. The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. In Proc. of the 1990 ACM SIGMOD Int'l. Conf. on Management of Data, pp.322-331.

Brinkhoff, T. 2002. A Framework for Generating Network-Based Moving Objects. GeoInformatica 6(2):153-180.

Tiger/Lines Precensus Files. 1994 Technical Documentation. U.S. Bereau of Census, Technical Report.

Guttman, A. 1984. R-Trees: A Dynamic Index Structure for Spatial Searching. In Proc. of the 1984 ACM SIGMOD Conference on Management of Data. pp.47-57.

Mong, L. L., H. Wynne, S. J. Christian, C. Bin. and L. T. Keng. 2003. Supporting Frequent Updates in R-trees:A Bottom-Up Approach. In Proc. of the 29th VLDB Conf. pp.608-619.

Nascimento, M. A. and J. R. O. Silva. 1998. Towards historical R-trees. ACM symposium on Applied Computing, pp.235-240.

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

Pfoser, D., C. S. Jensen. and Y. Theodoridis. 2000. Novel Approaches in Query Processing for Moving Objects. International Conference on VLDB, pp.395-406.

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

Sellis, T. K., N. Roussopoulos. and C. Faloutsos. 1987. The R+-tree: A Dynamic Index for Multidimensional Objects. In Proc. of 13th Conference on VLDB. pp.507-518.

Shahabi, C., M. R. Kolahdouzan. and M. Sharifzadeh. 2003. A Road Network Embedding Technique for K-Nearest Neighbor Search in Moving Object Databases. GeoInformatica 7(3):255-273.

Song, Z. and N. Roussopoulos. 2001. Hashing

Moving Object. International Conference on Mobile Data Management, pp.161-172.

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

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

Tayeb, J., O. Ulusoy. and O. Wolfson. 1998. A quadrate based dynamic attribute indexing method. The Computer Journal 41(3):185-200.

Theodoridis, Y., M. Vazirgiannis. and T. K. Sellis. 1996. Spatio-Temporal Indexing for Large Multimedia Applications. IEEE International Conference on Multimedia Computing and Systems, pp.441-448.

Theodoridis, Y., J. R. O. Silva. and M. A. Nascimento. 1999. On the generation of spatiotemporal datasets. International Symposium on Spatial Datasets, pp.147-164.