공간 네트워크 데이터베이스에서 실체화 기법을 이용한 범위 및 k-최근접 질의처리 알고리즘†

Range and k-Nearest Neighbor Query Processing Algorithms using Materialization Techniques in Spatial Network Databases

김용기* / Yong-Ki Kim, 니하드 카림 초우더리** / Nihad Karim Chowdhury
이현조*** / Hyun-Jo Lee, 장재우**** / Jae-Woo Chang

요 약

최근 LBS(location-based service) 및 텔레매틱스(telematics) 응용의 효율적인 지원을 위해, 유클리디언(Euclidean) 공간을 대신하여 실제 도로나 철도와 같은 공간 네트워크(network)를 고려한 연구가 활발하게 수행중이다. 그러나 기존 연구에서의 범위 질의 및 k-최근접 질의 처리 알고리즘은 범위나 k 값의 증가에 따라 검색에 필요한 노드 검색 및 거리 계산의 비용 증가로 인하여 선형적인 성능 감소를 보인다. 따라서, 본 논문에서는 공간 네트워크를 위한 기존 질의처리 알고리즘의 성능을 향상시키기 위해, 실체화 기법을 이용한 효율적인 범위 및 k-최근접 질의처리 알고리즘을 제안한다. 아울러, 기존 알고리즘과의 성능 비교를 통하여 제안하는 알고리즘이 우수함을 보인다.

Abstract

Recently, to support LBS(location-based services) and telematics applications efficiently, there have been many researches which consider the spatial network instead of Euclidean space. However, existing range query and k-nearest neighbor query algorithms show a linear decrease in performance as the value of radius and k is increased. In this paper, to increase the performance of query processing algorithm, we propose materialization-based range and k-nearest neighbor algorithms. In addition, we make the performance comparison to show the proposed algorithm achieves better retrieval performance than the existing algorithm.

주요어 : 질의 처리 알고리즘, 범위 질의, k-최근접 질의, 공간 네트워크 데이터베이스

Keyword : Query Processing Algorithm, Range Query, k-Nearest Neighbor Query, Spatial network databases

† 이 논문은 교육인적자원부, 산업자원부, 노동부의 출연금으로 수행한 최우수실험실지원사업의 연구결과이며, 아울러 이 연구에 참여한 연구자는 2단계 BK21 사업의 지원비를 받았음

■ 논문접수 : 2007.6.7 ■ 심사완료 : 2007.8.16

* 전북대학교 컴퓨터공학과 박사과정(ykkim@dblab.chonbuk.ac.kr)
** 전북대학교 컴퓨터공학과 석사과정(nihad@dblab.chonbuk.ac.kr)
*** 전북대학교 컴퓨터공학과 석사과정(hjlee@dblab.chonbuk.ac.kr)
**** 교신저자 전북대학교 컴퓨터공학과 교수(jwchang@chonbuk.ac.kr)

1. 서론

최근 LBS(location-based service) 및 텔레매틱스(telematics) 응용의 효과적인 지원을 위해, 유클리디언(Euclidean) 공간을 대신하여 실제 도로나 철도와 같은 공간 네트워크(network)를 고려한 연구가 활발하게 수행 중에 있다[1,2,3,4,5,6,8]. 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 시스템이다.

첫째, 공간 네트워크 데이터베이스에서 데이터 모델에 관한 연구는 대표적으로 덴마크의 Aalborg 대학의 연구가 있다[2,7]. 우선 Aalborg 대학의 연구에서는 네트워크 내의 이동 객체를 위한 데이터 모델링을 제시하였다. 여기서는 도로 네트워크(road network) 및 고정/이동 객체의 2차원 표현 및 그래프 표현 방법을 제시하고, 2차원 표현을 그래프 표현으로 변환하는 방법을 제시하고 있다[2]. 아울러, 또 다른 연구에서는 교통 인프라구조(infrastructure)에서의 데이터 모델링은 제안하였고, 이를 위해 kilometer post, link-node, 지리좌표 표현, segment 표현 의 4가지 표현 방법을 제시하고 있다[7].

둘째, 공간 네트워크 데이터베이스에서 질의처리 알고리즘에 관한 연구는 대표적으로 범위 질의처리 알고리즘 및 k-최근접 질의처리 알고리즘에 관한 연구가 존재한다[5,6]. 우선 HKUST(HongKong University of Science and Technology)에서는 유클리디언 공간을 확장한 방법과 네트워크를 확장하는 방법으로 범위 및 k-최근접 질의처리 알고리즘을 제안하였다[5]. 한편, Southern California 대학에서는 Voronoi 다이어그램을 기반으로 하여 k-최근접 질의처리 알고리즘을 제안하였다. 이는 각각의 POI(Point of Interest)를 기준으로 하여 Voronoi 영역별로 공간 네트워크를 분할하여 저장함으로써 효율적인 k-최근접 질의처리 알고리즘을 수행하였다[6].

마지막으로, 공간 네트워크 데이터의 효율적인 색인 및 저장 시스템의 연구로써 공간 네트워크를 그래프 형태로 변환한 후 저장하는 연구들이 수행되었다[5,9]. 미네소타 대학의 연구에서는, 공간 네트워크를 G=(N, E)의 그래프 형태로 변환한다[9]. 여기서 N은 노드의 집합을, E는 에지의 집합을 나타낸다. 이렇게 변환된 그래프를 Z-order로 표현하여 연결에 따른 클러스터를 형성한 노드들에 대해 B+-트리를 구성함으로써 노드 식별자를 통한 접근을 빠르게 하는 연구가 수행되었다. 아울러 HKUST에서는 미네소타 대학의 연구와 마찬가지로 연결에 따른 클러스터를 지역적 클러스터링 효과를 높이기 위하여 Hilbert order를 사용하여 노드들의 인접 리스트로 저장하는 연구가 수행되었다[5].

본 논문에서는 위의 3가지 주제 가운데, 공간 네트워크 데이터베이스를 위한 효율적인 질의처리 알고리즘을 설계 및 구현하고자 한다. 최근 공간 네트워크상에서의 범위 및 k-최근접 질의처리 알고리즘으로, 유클리디언 공간을 검색하여 확장하는 기법과 네트워크를 확장하는 기법이 제안되었으나, 이 기법들은 응용 환경에 따라서 다수의 디스크 I/O 나 불필요한 연산 등이 존재한다[5]. 아울러, Voronoi 기반의 네트워크 k-최근접 질의처리 알고리즘이 수행되었으나, 이 기법은 k가 커짐에 따라 이웃하는 지역들을 합침으로써 지역이 갖고 있는 POI와 인접 POI까지의 중간지점인 Border-Point들을 재계산 및 네트워크 구성하여야 하는 추가적인 계산 비용이 존재한다[6]. 따라서 본 논문에서는 네트워크 확장 방법에 추가적으로 실체화 기법을 도입하여 보다 효율적으로 범위 질의 및 k-최근접 질의를 처리하는 알고리즘을 제안한다. 제안하는 알고리즘의 효율성을 보이기 위해 기존 범위 및 k-최근접 질의처리 알고리즘과 성능비교를 수행한다.

논문의 구성은 다음과 같다. 2장에서는 관련 연구를 소개하고, 3장에서는 본 논문에서 제안하는 범위 및 k-최근접 질의처리 알고리즘을 설계한다. 4장에서는 제안하는 알고리즘과 기존 알고리즘과의

성능비교를 수행하고, 마지막으로 5장에서는 결론 및 향후연구를 제시한다.

2. 관련 연구

공간 네트워크 데이터베이스를 위한 범위 및 k-최근접 질의처리 대표적인 알고리즘으로써 HKUST에서의 연구와 Southern California 대학에서의 연구가 있다.

우선 HKUST 연구에서는 유클리디언 공간을 확장하는 기법과 공간 네트워크를 확장하는 기법을 제안하였다[5]. 이 기법을 기반으로 하여 범위 질의처리 알고리즘으로써 RER (Range Euclidean Restriction) 및 RNE (Range Network Expansion), k-최근접 질의처리 알고리즘으로써 IER (Incremental Euclidean Restriction) 및 INE (Incremental Network Expansion)를 제안하였다. 첫째, RER 기법은 유클리디언 거리로써 R-tree를 탐색하여 범위 질의를 수행한 후, 주어진 범위 내의 에지상에 존재하는 POI들을 검색하여 실제 네트워크 거리를 계산하여 범위 내에 존재하는 POI들을 탐색하는 방법이다. 둘째, RNE 기법은 네트워크의 특성을 이용한 네트워크 확장 방법으로 질의가 속한 에지의 MBR(Minimum Bounding Rectangle)을 탐색하여, 그 MBR의 집합을 통하여 POI R-tree와의 조인 기법을 이용하여 탐색하는 방법이다. 셋째, IER 기법은 유클리디언 거리로써 R-tree를 탐색하여 k 개의 최근접점을 검색한다. 검색된 결과를 네트워크 거리의 계산을 통해 정렬한 후에 후보 집합으로 저장한다. 그리고 유클리디언 거리로 그 다음 가까운 POI를 검색하여 네트워크 거리를 계산한 후 후보 집합의 k 번째의 POI보다 네트워크 거리가 작으면 탐색결과에 포함시키고 그렇지 않으면 다음을 검색한다. 검색해야할 POI의 유클리디언 거리가 후보집합의 k 번째 POI의 네트워크 거리보다 크면 알고리즘은 종료한다. 마지막으로, INE 기법은 질의가 속한 에지(edge)를 탐색하여, 이 에지를 시작으로 에지상에 존재하는 k개의 최근접점을 찾을 때까지 에지를 별도의 저장 공간에 추가하면서 네트워크를 확장해가는 방법이다. 이를 통해 얻어지는 k-최근접 POI들은 그들의 거리가 공간 네트워크상의 거리를 의미한다. 새로이 추가되는 에지까지의 거리가 k번째 최근접 네트워크 거리보다 커지면 알고리즘은 종료된다. 그러나 INE 알고리즘은 기본적으로 Dijkstra 알고리즘을 사용하여 최근접점을 찾기 때문에, 지도 데이터에 비해 POI 밀도가 낮은 경우 전체적인 검색 시간이 증가하게 된다.

또 다른 연구로써 Southern California 대학에서는 유클리디언 환경에서 연구되어진 Voronoi 다이어그램을 네트워크 데이터베이스에 적용하여 효율적인 k-최근접 질의처리를 제안하였다. Voronoi 기반 k-최근접 질의처리 알고리즘은 첫째, 질의점이 포함되어 있는 Voronoi 영역(Network Voronoi Polygon)을 검색함으로써, 가장 가까운 POI을 찾는다. 둘째, 최근접점 POI가 포함되어 있는 영역(질의영역)의 이웃하는 영역들 중에서 그 다음의 POI를 찾을 수 있으므로, 이웃하는 영역을 검색한다. 이 때, 이웃영역에 속해있는 POI까지의 실제거리를 미리 계산 되어진 BtoP(Border-point to POI), BtoB(Border-point to Border-point), 그리고 질의점과 Border-point까지의 거리를 이용하여 다음 최근접점을 구하게 된다. 아울러 질의점이 속해있는 영역과 다음 최근접점이 포함된 영역을 합쳐 하나의 질의영역으로 만들고 해당영역의 Border-point까지의 거리를 재계산 한다. 이와 동시에 해당 영역들만을 합치는 것이 아니라 다음의 최근접 POI를 용이하게 찾기 위하여 새로이 생성된 질의영역이 가진 질의점과 Border-point들간의 네트워크를 구성한다. 위와 같은 방식으로 계속 이웃하는 영역에서 다음의 최근접점을 찾아가는 확장방법을 사용한다.

그러나 HKUST 연구의 범위 질의처리 알고리즘 및 k-최근접 질의처리 알고리즘들은 다음과 같은 단점들을 가지고 있다. 먼저 유클리디언 공간을 이용하여 범위 및 k-최근접 질의를 처리하는 알고리즘은 유클리디언 공간에서 범위 질의를 수행하여 범위 내의 POI를 탐색하고, 아울러 결과 POI들의

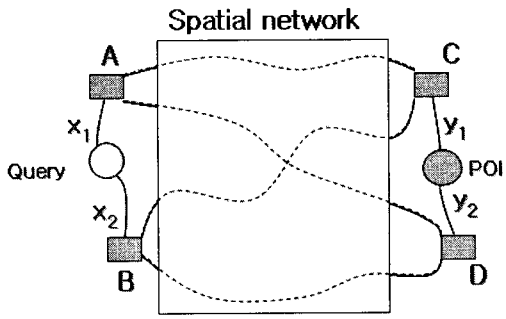
실제 네트워크 거리를 계산해야한다. 한편, 네트워크를 확장하여 POI를 검색하는 범위 및 k-최근접 질의처리 알고리즘은 각 노드의 정보를 디스크로부터 읽고, 질의점으로부터 각 노드까지의 거리를 계산해야 하므로, 디스크 I/O 증가 및 CPU 시간의 증가로 인하여 범위 및 k의 수가 증가할수록 질의 응답시간이 기하학적으로 증가하는 단점이 존재한다. 한편 Southern California 대학의 연구에서는 질의 영역을 확장하기 위하여 이웃하는 영역의 POI까지의 거리를 모두 계산하여야 하며, 동시에 새로운 네트워크를 구성해야 하는 오버헤드가 존재하게 된다. 아울러 구성된 네트워크를 통하여 질의점으로부터 border들까지를 새로이 계산하여야 하는 단점이 존재한다.

3. 공간 네트워크 질의처리 알고리즘

본 논문에서는 기존의 범위질의와 및 k-최근접 질의처리 알고리즘의 성능을 향상시키기 위하여, 한 노드로부터 다른 모든 노드까지의 거리를 미리 계산하여 저장하는 실체화 기법을 이용한 새로운 알고리즘을 제안한다.

3.1 실체화 개념 및 실체화 파일 구조

사용자가 질의를 주었을 때 질의를 수행하는 방법은 크게 2가지로 나눌 수 있다. 첫째, on-line computation 기법을 사용하여 질의처리를 수행하는 방법이다. 이는 질의를 수행하면서 동적으로 메모리 할당 및 네트워크를 탐색하여 질의를 수행한다. 이는 갱신 비용이 최소화 되며, 부가적인 저장공간이 필요치 않다. 다만, 네트워크를 탐색하여 질의를 수행하므로 질의응답 속도가 느리다. 대표적인 기법으로는 INE 기법이 있다[5]. 둘째, pre-computation 기법이 있다. Pre-Computation 기법은 질의처리를 수행하기 전에 네트워크 정보를 미리 계산하여 저장함으로써 질의응답시간을 최소화 한다. 대표적인 기법으로는 Voronoi 기법이 있다[6]. 따라서 본 논문에서는 pre-computation 기법으로 실체화 기법을 사용한다. 실체화 기법은 노드에서 POI까지의 거리를 계산하는 기법과 노드에서 노드까지의 거리를 계산하는 기법이 있다. 노드에서 POI까지의 거리를 계산하는 기법은 질의점이 주어지면 질의점을 포함하는 두 노드를 검색하여 각각이 갖고 있는 POI까지의 거리들을 이용하여 질의를 수행하는 방법이다. 이 방법은 POI가 고정되어 있어야 하는 가정과 POI의 밀도에 따라 미리 계산해야 하는 양이 크게 달라진다. 일반적으로 노드는 고정되어 있는 반면에 POI는 자주 업데이트가 일어나므로 노드와 POI간의 거리를 저장하는 방법은 실제 응용에 적용하기가 용이하지 않다. 반면에 노드에서 노드까지의 거리를 계산하는 기법은 질의점이 주어지면 질의점을 포함하는 두 노드가 갖고 있는 노드들까지의 거리들을 이용하여 질의를 수행하는 방법이다. 이는 네트워크상의 노드가 고정되어 있으므로 업데이트 비용이 거의 필요치 않으며 POI의 밀도나 자주 업데이트 되는 상황에도 영향을 받지 않는다. 따라서 본 논문에서는 노드-노드 거리를 미리 계산하여 저장하는 실체화 기법을 사용한다. 노드-노드 실체화 파일을 이용하여 다음 그림 1과 같이 질의점으로부터 POI까지의 거리를 계산할 수 있다.



<그림 1> 질의점과 POI간의 거리계산의 예

정의 1. 네트워크 거리 (Network Distance) 계산식

$$ND(Q, POI) = MIN\ (ND(A,C)+x1+y1,\ ND(A,D)+x1+y2,\ ND(B,C)+x2+y1,\ ND(B,D)+x2+y2)$$

위 정의 1은 질의점(Query Point)에서 POI까지의 거리 계산을 정의한다. 질의점이 주어지면 질의점을 포함하고 있는 두 노드를 읽어오고 POI에 해당하는 각각의 두 노드간의 거리중에 최소의 거리를 구할 수 있다. 그러나 데이터의 노드의 수가 증가할수록 실체화 파일의 크기가 증가하기 때문에 노드 정보를 읽는 시간이 오래 걸린다는 단점이 존재한다. 이는 실체화 파일에 포함된 순서정보를 이용하여 상위 m번째의 노드까지만 저장하는 축약된 실체화 파일을 사용함으로써 해결할 수 있다. 즉, 질의를 수행함에 있어 기본적으로 질의점과 가까운 범위만을 검색하게 되기 때문에 대부분의 질의가 축약된 실체화 파일로써 수행되어질 수 있다. 따라서 축약된 실체화 파일을 이용하여 질의를 수행하면, 전체 노드가 저장된 파일을 이용하지 않고 상위 노드만을 저장함으로써 전체 용량 및 메모리 로딩 시간을 줄일 수 있다. 축약된 실체화 파일은 질의를 수행하기 전에 한번만 메모리에 상주시킴으로써, 질의를 여러번 수행할 수 있음으로써 질의수행의 효율성을 높인다. 아울러 이를 위해서는 모든 노드간의 거리를 보조기억장치에 저장하는 것이 필요하다. 실체화 파일에 저장되는 하나의 레코드 크기는 약10바이트로, 약 20만개의 노드로 구성된 데이터를 저장할 때, 400GB (=200K*200K*10바이트)의 부가적인 저장 공간이 필요하다. 하지만, 하드디스크는 매해 용량이 증가하고 있다. 저장장치의 발달로 인한 저장장치의 용량증가 및 비용감소가 이를 가능하게 한다. 현재 200-500GB 용량의 하드디스크는 Maxter, Western Digital, Segate사 등에서 보편화되어 있으며, 최근 히타치 글로벌 스토리지 테크놀로지스(Hitachi GST)는 세계 최초로 테라 바이트급 하드디스크를 '2007 CES'에서 선보였다. 아울러 Segate도 조만간 테라바이트 하드디스크 양산에 들어갈 계획이다[10]. 따라서 네크워크 정보를 미리 계산(Pre-compute)하여 저장하는 실체화 접근 방법을 사용하는 것이 가능하다[11].

Start Node ID	Destination Node ID (int : 4 byte)	Distance (double : 8byte)
1	2	3424.23
	4	3862.41
	...	...
	331 (mth)	25421.82
2	3	2731.87
	5	3212.48
	...	...
	285 (mth)	20845.56
...	...	...
n	325	1821.13
	298	2768.56
	...	...
	5124 (mth)	32465.75

<그림 2> 실체화 파일의 구조

본 논문에서 제시하는 실체화 파일의 레코드 구성은 그림 2와 같다. 먼저 Start Node ID는 기준점의 시작지점 노드식별자를, Destination Node ID는 도착지점 노드식별자를, Distance는 Start Node와 Destination Node 사이의 거리를 나타낸다.

실체화 파일을 사용함으로써 다음과 같은 장점이 있다. 첫째, 노드식별자만을 가지고 노드와 노드 사이의 최소 거리를 경로탐색 알고리즘을 사용하지 않고 바로 검색할 수 있다. 둘째, 순서정보를 이용하여 임의의 노드에서 가장 가까이에 있는 노드 또는 i번째에 있는 노드를 바로 알 수 있다. 이로 인하여 범위 및 k-최근접 질의처리에 필요한 노드정보 검색을 위한 디스크 I/O 수를 줄일 수 있다. 아울러 노드간의 거리계산을 줄이므로 질의 처리를 위한 CPU 시간을 줄일 수 있다. 따라서 질의처리를 위한 전체검색시간이 감소한다. 그림 3은 실체화 파일을 생성하는 알고리즘이다.

일반적으로 POI는 자주 변경이 발생하는 반면에 노드는 삽입 또는 삭제가 거의 일어나지 않기 때문에 실체화 파일을 실제 응용에 적용하기가 용이하다. 그러나 빈번하지는 않지만 노드에 대한 삽입(새로운 도로 개설에 따른 교차로 삽입) 및 삭제가 일

```
01. create_materialization_file()
02. foreach(all node) {
03.     node_info = single_all_destination(a node, 0);
04.     sort_by_distance(node_info);          //calculate order
05.     fwrite(node_info);
06.     //node info <destination node id, order, distance>;
07. }
08.
09. single_all_destination(node, distance)
10. while(a node has adjacency node) {
11.     node_list = expand_adjacency_node( node );
12.     foreach(all node in node_list) {
13.         destination_distance = distance + d(node, adjacency node);
14.           //store node info <adjacency node id, 0, destination_distance>;
15.           single_all_destination(adjacency node, destination_distance);
16.     }
17. }
```

<그림 3> 실체화 파일 생성 알고리즘

어날 경우는 도달 가능한 모든 노드들에 대한 최소 거리 계산이 다시 수행되어야 한다. 이를 해결하기 위하여 노드의 삽입 및 삭제 발생지역에서 일정 범위내의 노드까지의 거리만을 계산함으로써 거리 계산 수행의 오버헤드를 줄일 수 있다.

3.2 범위 질의처리 알고리즘

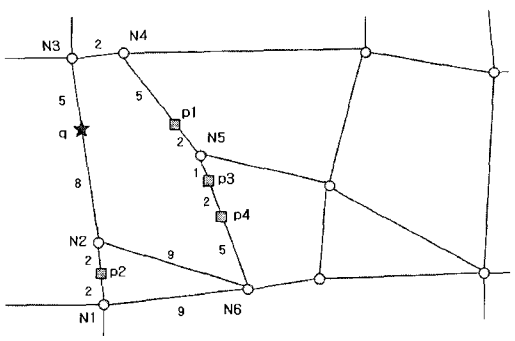
실체화 기법을 이용한 범위 질의 처리 알고리즘은, R-Tree를 탐색하여 결과 POI를 구하고, 이들의 실제 거리 계산을 위해 실체화 기법을 사용하는 방법이다. 이는 실제 거리를 계산하는데 있어, 질의점을 포함하고 있는 에지와 POI를 포함하고 있는 에지를 바로 알 수 있기 때문에, 네트워크 거리 계산 알고리즘에 근거하여 실제거리를 쉽게 계산할 수 있는 장점이 있다. 그러나 R-Tree 전체를 탐색해야 하는 단점이 존재하므로, 트리를 검색하지 않고 네트워크를 확장하면서 범위 내의 POI를 검색하여 이를 개선할 수 있다. 하지만, 노드를 확장할 때 인접한 다른 모든 노드까지의 에지들을 확장하기 때문에 이미 검색된 에지인가를 비교해야 한다. 만약, 비교를 하지 않는다면 이미 검색된 에지를 확장할 수 있기 때문에 중복된 검색이 발생할 수 있다. 이는 검색된 노드를 별도의 저장 영역에 저장하여 최소한의 에지 탐색만을 수행함으로써 개선할 수 있다. 아울러 노드 정보(ns)에 대한 큐와 에지 정보 (ES(Edge Set) 셋, EES(Extra Edge Set) 셋)는 질의점에서 POI까지의 Path를 제공한다.

제안하는 범위 질의처리 알고리즘은 그림 4와 같다. 먼저, 축약된 실체화 파일은 메모리에 상주시킨다. 따라서 질의를 수행함에 있어 첫 번째로 주어진 질의점이 존재하는 에지상의 POI들을 검색하여 결과 셋에 저장하고, 해당 에지의 양 노드를 노드 셋에 저장한다. 둘째, 메모리에 상주된 실체화 파일 정보에서 양 노드에 대한 정보를 얻어온다. 셋째, 확장해야 할 노드의 인접노드가 노드 셋에 포함된 경우에는 확장하고. 그렇지 않은 경우는 확장 에지 집합에 저장한 후, 검색된 노드를 노드집합에 저장한다. 이때, 메모리에 상주된 축약된 실체화 파일 정보에서 노드에 대한 정보가 없다면 디스크에서 해당 실체화 파일을 읽어온다. 넷째, 에지집합에 포함된 에지상에 존재하는 POI를 검색하여 거리를 계산한 후 범위 이내에 존재하면 결과 집합에 저장한다. 다섯째, 그 다음으로 가까운 노드를 검색하여 질의점으로부터의 거리가 범위보다 클 때까지 반복한다. 마지막으로 확장 에지 집합에 포함된 에지상

```
MAR(q, r)       // MAterialization based Range Algorithm
// q : qeury point, r : range
result = find_poi(q.ni, q.nj);
ns = {ni, nj};
node_info = load_ns_from_mem(ns);
nn = find_nearest_node(node_info);
while(d(q, nn)<r) {
        if (nn do not exist from node_info) ns = load_ns_from_disk(ni, nj);
        ES = EES = ∅;
        foreach(adjacency node of nn) {
                if(adjacency node∈ns)
                        ES = ES ∪ expand(adjacency node, nn);
                else
                        EES = EES ∪ expand(adjacency node, nn);
        }
        ns = ns ∪ nn;
        foreach(edge∈ES) {
                poi=find_poi(edge);
                if(d(q, poi)<r)
                        result = result ∪ poi;
        }
        nn = find_nearest_node(node_info);
}
foreach(edge∈EES) {
        poi=find_poi(edge);
        if(d(q, poi)<r)
                result = result ∪ poi;
}
```

<그림 4> 범위 질의처리 알고리즘

에 존재하는 POI중 범위내의 POI들을 결과 집합에 포함시킨다.



<그림 5> 범위 질의처리 알고리즘 예

그림 5는 제안하는 범위 질의처리 알고리즘을 이용하여 "q에서 범위 13이내의 POI를 검색"하는 예제이다. 첫째, 질의 q가 속한 에지 e(n2,n3)을 탐색하고 n2, n3 노드의 실체화 파일을 통해 큐 <(n3,5),(n4,7),(n2,8),(n1,12),(n5,13),(n6,17)>를 얻어오며 , e(n2,n3)는 탐색을 했기 때문에 초기 ns의 구성은 <n2,n3>가 된다. 둘째, 실체화 파일에서 질의점으로부터 가장 가까이에 있는 노드가 n3와 n2이지만, ns에 이미 포함되어 있으므로 ES 셋에 삽입하고 e(n2,n3)를 탐색한다. 이와 동시에 n3와 n2의 인접 노드인 N1, N4, N6는 ns셋에 속해 있지 않으므로 ns에 삽입하고 해당 e(n2,n1), e(n2,n6), e(n3,n4)을 EES셋에 삽입한다. 실체화

파일에서 그 다음 노드인 n4와 인접 노드를 읽는다. n4는 ns에 포함되어 있으므로 e(n2,n3)를 EES 셋에서 삭제하고 ES 셋에 삽입과 동시에 POI를 탐색한다. 아울러 인접 노드 n5는 ns에 포함되어 있지 않으므로 EES에 포함시키고 ns에 삽입한다. 실체화 파일에서 다음 노드를 읽고(n1), n1의 인접 노드는 <n2,n6>이다. 인접 노드 중에서 ns에 포함되지 않은 것은 n6이므로, ns에 n6를 삽입하고 e(n1,n6)를 EES에 넣음과 동시에 n2는 ns에 포함되어 있으므로 ES에 e(n1,n2)를 삽입하고 탐색함으로써 POI p2를 찾는다. 마지막으로, 다음 노드는 범위 13을 넘으므로 현재 EES에서 에지 e(n4,n5), e(n2,n6)를 탐색함으로써 POI p1을 찾고 프로그램을 종료한다. 따라서 최종 결과 POI 집합은 {(p2,10), (p1,12)}가 된다.

3.3 k-최근접 질의처리 알고리즘

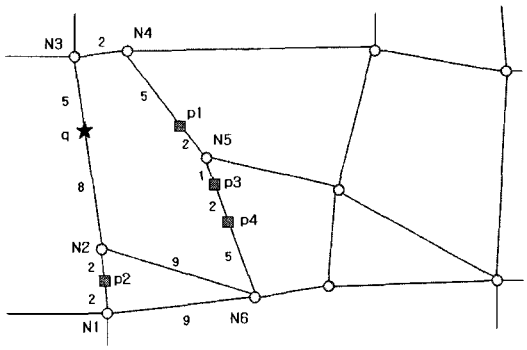
k-최근접 질의처리 알고리즘 역시 범위 질의처리 알고리즘과 마찬가지로 노드간의 거리를 미리 계산하여 저장하는 실체화 기법을 사용하여 디스크 I/O 및 검색에 필요한 노드간의 거리계산을 줄임으로써 성능을 향상시킬 수 있다. 본 논문에서 제안하는 k-최근접 질의처리 알고리즘은 그림 6과 같다.

```
MAK (q, k)       // MAterialization based k-NN Algorithm
// q is the query point
// EES is the Extra Edge Set
// result is the POI(Point Of Interesting) Set
// ns is the Node length Store, Materialization File
// hn is the History Node
result = ∅;
hn = ∅;
dmax = ∞ ;
result= result ∪ find_poi(q.ni,q.nj);
ns = load_ns_from_mem(ni, nj);
hn = hn ∪ ni ∪ nj;
while(nsx-1.length < dmax)        {
       if(count(result) >= k)     {
                result = sort_kcut(result, k);
                dmax = max(result);        }
       if(hn⊉nsx)         {
                hn = hn ∪ nsx;
                foreach(nsx)       {
                         if(hn∋nsx.nj)     {
                         result = result ∪ find_poi(nsx.ni, nsx.nj);
                                  EES = EES - nsx.edge_idj; }
                         else              {
                                  EES = EES ∪ nsx.edge_idj;}}}
       nsx = nsx+1;
   if(nsx do not exist from node_info) ns = load_ns_from_disk(ni, nj);
 }
foreach(EES){
       result = result ∪ find_poi(EES); }
result = sort_kcut(result, k);
return result;
```

<그림 6> k-최근접 질의처리 알고리즘

첫째, 질의점을 포함하는 에지를 탐색하여 얻어진 POI 집합을 후보 결과 집합에 넣는다. 탐색된 에지의 양 끝 노드는 HN(History Node)에 포함시킨다. 즉, HN에는 탐색된 노드들의 정보를 포함한다. 둘째, 질의점이 포함되어 있는 에지의 양 끝 노드로부터 모든 노드까지의 거리와 순서정보를 실체화 파일 정보에서 가져온다. 셋째, 실체화 파일 정보를 이용하여 질의점에서 가장 가까운 노드부터 검사한 후, 해당 노드가 HN에 포함되어 있으면, 실체화 파일 정보에서 질의점으로부터 해당 노드 다음으로 가까운 노드를 읽고, 그렇지 않으면 해당 노드를 HN에 포함시키고, 노드의 정보를 읽어서 인접노드 리스트를 생성한다. 이때, 이미 메모리상에 존재하는 실체화 파일의 노드정보를 모두 읽었다면 다음 노드정보를 검색하기 위해서 디스크로부터 해당 실체화 파일을 읽는다. 인접노드 중 HN에 포함되어 있는 노드와 실체화 파일 정보에서 바로 전에 읽었던 노드로 이루어진 에지를 탐색하여 POI 집합을 구하고, 이를 후보 결과 집합에 포함시킨다. 인접노드 리스트 중에서 HN에 포함되지 않는 노드와 실체화 파일에서 바로 전에 읽었던 노드로 이루어진 에지는 나중에 탐색하기 위해서 EES에 넣는다. 넷째, 후보 결과 집합의 개수가 k보다 크거나 같으면 질의점으로 부터 거리가 작은 순으로 정렬 한 후, 상위 k개만을 선택하여 k번째 POI와 질의점까지의 거리를 dmax로 정의한다. 다섯째, 실체화 파일에서 바로 전에 읽었던 노드와 질의점까지의 거리가 dmax 보다 크거나 같게 되면, EES에 들어 있는 에지상의 모든 POI를 검색하여 후보 결과 집합에 넣는다. 마지막으로 후보 결과 집합을 정렬한 후, 상위 k개를 최종 결과 집합으로 전달한다.

그림 7은 제안하는 k-최근접 질의처리 알고리즘을 이용하여 "q에 가까이 있는 최근접 POI를 검색"하는 예제이다. 첫째, 질의 q가 속한 에지 e(n2,n3)을 탐색하고 n2, n3 노드의 실체화 파일을 통해 큐를 생성한다. 즉, <(n3,5),(n4,7),(n2,8),(n1,12),(n5,13), (n6,17)> 이다, e(n2,n3)는 탐색을 했기 때문에 초기 HN의 구성은 <n2,n3>가 된다. 둘째, 실체화 파일에서 질의점으로 부터 가장



<그림 7> k-최근접 질의처리 알고리즘 예

가까이에 있는 노드가 n3이지만, HN에 이미 포함되어 있으므로, 그 다음 노드인 n4를 읽는다. n4는 HN에 포함되어 있지 않으므로 HN에 포함시킨다. n4의 인접 노드는 <n3, n5>이다. 인접노드중 HN에 포함되어 있는 것은 n3 이므로, e(n4,n3)을 탐색하고 e(n4,n5)는 EES에 넣는다. 셋째, 아직 POI가 검색되지 않았기 때문에, 실체화 파일에서 다음 노드를 읽고(n2), 이것이 HN에 포함되어 있기 때문에, 그 다음 노드인 n1을 읽는다. n1은 HN에 포함되어 있지 않으므로 HN에 포함시킨다. 넷째, n1의 인접 노드는 <n2,n6>이다. 인접 노드중 HN에 포함되지 않은 것은 n6이므로, e(n1,n6)를 EES에 넣고 e(n1,n2)를 탐색하여서 POI p2를 찾는다. 이때 dmax 를 q에서 p2 까지의 거리인 10으로 설정한다. 마지막으로, 현재 EES에서 에지 e(n4,n5), e(n1,n2)을 탐색하면 POI p1를 찾는다. 질의점에서 n1까지의 거리가 12 이므로 이는 dmax 보다 크다. 따라서 최종 결과 POI 집합은 {(p2,10)}이 된다.

4. 성능평가

성능평가를 위해 본 논문에서 제안하는 알고리즘을 Memory 3GB, CPU 3.0GHz를 지닌 Windows Server 2003 Enterprise 환경에서 Visual C++ 7.1을 사용하여 구현하였다. 지도 데이터는 실제 샌프란시스코 만 지도를 사용하였고 [12], 전체 노드 수는 175,343개, 전체 에지 수는

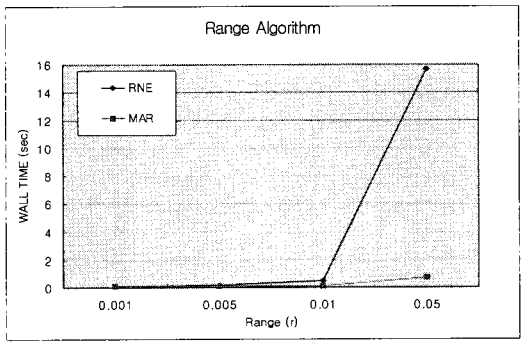
<표 1> 범위 질의 성능비교

비교대상	범 위	Disk I/O(횟수)	CPU Time(초)	Wall Time(초)
RNE	0.001	168	0.026069	0.094676
	0.005	3766	0.076568	0.176035
	0.01	15181	0.335413	0.511619
	0.05	95795	13.56704	15.654860
MAR	0.001	36	0.011310	0.094016
	0.005	566	0.014743	0.104170
	0.01	2053	0.021063	0.113396
	0.05	32490	0.151916	0.748551

223,199개로 구성되어 있다. 아울러, POI는 RunTime21[13] 알고리즘을 사용하여 임의로 생성한 10846개를 사용하였고, 질의점은 임의로 1000개를 주어서 사용하였다. 마지막으로 축약된 실체화 파일의 크기는 약 2GB이며, 각 노드에 저장되는 Destination 노드의 개수는 1000개씩이다. 이는 디스크에서 실체화 파일을 불러오는 것을 최소로 줄이기 위한 개수이다. 질의 성능평가 및 알고리즘 구현을 위해 이미 개발된 공간 네트워크를 위한 저장/색인 구조를 사용하였다[14]. 기존 알고리즘들 중에서 RNE 알고리즘이 RER 알고리즘보다 성능이 우수하기 때문에 RNE만을 성능비교 대상으로 삼았으며, INE가 IER보다 성능이 우수하기 때문에 INE과 Voronoi 기반 k-최근접 알고리즘을 성능비교 대상으로 하였다.

표 1은 제안하는 범위 질의처리 알고리즘(MAterialization based Range : MAR)과 RNE의 성능비교 결과를 나타낸다. 표에서 DISK는 DISK I/O 개수를 나타내며, CPU는 디스크 I/O에 소요되는 시간을 제외한 CPU Time(sec)을 나타내며, WALL은 전체 검색 시간(sec)을 나타낸다. 범위는 전체 지도 데이터 크기의 비율을 의미한다.

표 1에 나타난바와 같이 범위가 0.001일 때, RNE과 MAR은 비슷한 성능을 보이나, 0.005인 경우 RNE는 약 0.17초, MAR은 약 0.1초가 소요된다. 제안하는 알고리즘이 기존 RNE 기법에 비해 좋은 성능을 나타냄을 알 수 있으며, 범위가 증가할수록 RNE는 급격한 성능감소를 보이는 반면, MAR은 훨씬 완만한 증가세를 보임을 알 수 있다. 이는 실체화 파일에 저장된 정보를 사용함으로써 기존 알고리즘에서 필요한 노드 접근을 위한 디스크 I/O 및 거리 계산에 필요한 시간을 줄일 수 있을 뿐만 아니라, 네트워크를 확장하여 범위를 설정하기 때문이다. 그림 8은 RER과 MAR의 전체 검색시간을 범위에 따라서 그래프로 표시한 것이다.



<그림 8> 범위 질의 성능비교

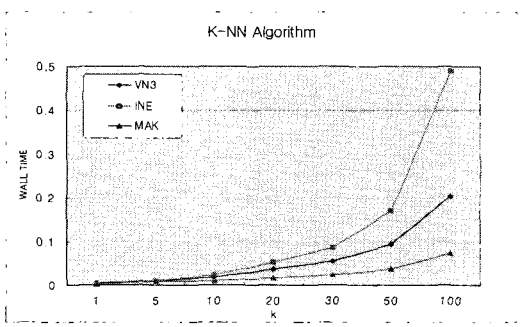
표 2는 제안하는 k-최근접 질의처리 알고리즘(MAterialization based K-NN : MAK)과 기존 INE, VN3와의 성능비교 결과를 나타낸다. 성능측정은 k의 값을 1부터 100까지 변화시키면서 전체 검색 시간(Wall Time)을 측정한 것이다.

<표 2> k-최근접 질의 전체 검색 시간

단위 : 초(sec)

k 값	1	5	10	20	30	50	100
VN3	0.005639	0.009942	0.019003	0.036254	0.053598	0.094188	0.203535
INE	0.001903	0.010873	0.023369	0.052161	0.086692	0.169527	0.489689
MAK	0.003921	0.008360	0.010939	0.017621	0.023062	0.036770	0.072848

표 2에 나타난바와 같이 k=1일 때 전체 검색시간은 INE의 경우 0.0019초, VN3의 경우 0.0056초이며 MAK는 0.0039초로 비슷한 성능을 나타낸다. 하지만 k=10일 때 INE의 전체 검색시간은 0.0536초, Voronoi는 0.0190초, 그리고 MAK는 0.0233초가 되며, k값이 5일 때부터 MAK가 INE나 VN3에 비해서 성능이 우수함을 알 수 있다. 예를 들면, k값이 증가하여 100일 경우에는, INE는 0.4896초, VN3은 0.2035초, 그리고 MAK는 0.0728초로써 제안하는 알고리즘인 MAK가 기존 연구인 INE와 VN3에 비해서 훨씬 효율적임을 알 수 있다. 그림 9는 INE, VN3와 MAK의 전체 검색시간을 k값에 따라서 그래프로 표시한 것이다. INE, VN3, 그리고 MAK 모두 k값이 증가 할수록 전체검색시간이 증가함을 알 수 있다. 하지만 본 논문에서 제안하는 MAK는 INE 보다 훨씬 완만하게 증가함을 볼 수 있다.



<그림 9> k-최근접 질의 성능비교

표 3은 제안하는 실체화 파일의 저장 공간 크기와 HKUST 연구 및 VN3와의 저장 공간 크기의 비교를 나타낸다. 축약된 실체화 파일의 크기는 기존의 VN3 연구의 저장 공간에 비해 약 7배, HKUST 연구(RNE, INE)의 저장 공간에 비해 약 17배의 오버헤드가 존재한다. 그러나 앞서 언급한 바와 같이 저장장치의 발달로 인한 저장장치 비용 감소 및 실체화 파일을 이용한 제안하는 알고리즘 검색 성능의 우수함을 통해 이러한 저장 공간의 오버헤드를 극복(또는 완화)할 수 있다.

<표 3> 저장 공간 비교

비교 대상	저장 공간 크기
VN3	280MB
RNE / INE	112MB
MAR / MAK	1960MB

5. 결론

본 논문에서는 공간 네트워크 데이터베이스를 위한 기존 알고리즘의 문제점인 디스크 I/O 횟수와 거리 계산으로 인한 성능감소를 극복하기위해 실체화 기법을 이용하여 보다 효율적인 범위 및 k-최근접 질의처리 알고리즘을 제안하였다. 제안한 범위 질의처리 알고리즘은 기존 RNE 알고리즘보다 범위가 커짐에 따라 크게는 약 20배가량 성능이 좋음을 알 수 있다. 또한, 제안한 k-최근접 질의처리 알고리즘은 k값이 커짐에 따라 기존 INE보다는 약 2배에서 7배까지 Voronoi 알고리즘보다는 약 3배까지 성능이 우수함을 입증하였다. 향후 연구로는 여러 타입의 POI를 고려한 Join 기반(Closest Pairs, e-distance Join) 질의 처리 알고리즘들을 실체화 기법을 사용하여 설계하는 것이다.

참고문헌

1. S. Shekhar et al., "Spatial Databases Accomplishments and Research Needs," IEEE Tran. on Knowledge and Data Engineering, Vol. 11, No. 1, 1999, pp. 45-55.
2. L. Speicys, C.S. Jensen, and A. Kligys, "Computational Data Modeling for Network-Constrained Moving Objects," Proc. of ACM GIS, 2003, pp. 118-125.
3. C.S. Jensen, J. Kolar, T.B. Pedersen, and I. Timko, "Nearest Neighbor Queries in Road Networks," Proc. of ACM GIS, 2003, pp. 1-8.
4. 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, 2003, pp. 255-273.
5. D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao, "Query Processing in Spatial Network Databases" Proc. of VLDB, 2003, pp, 802-813.
6. M. Kolahdouzan and C. Shahabi, "Voronoi-Based K Nearest Neighbor Search for Spatial Network Databases", Proc. of VLDB, 2004, pp. 840-851.
7. C.S. Jensen, T.B Pedersen, L. Speicys, and I. Timko, "Data Modeling for Mobile Services in the Real World", Proc. of SSTD, 2003, pp. 1-9.
8. Z. Song, and N. Roussopoulos, "K-Nearest neighbor Search for Moving Query Point", Proc. of SSTD, 2001, pp. 79-96.
9. S. Shekhar, and D.R. Liu, "CCAM: A Connectivity-Clustered Access Method for Networks and Network Computations", Proc. of IEEE Tran. on Knowledge and Data Engineering, Vol. 9, No. 1, 1997, pp. 102-119.
10. http://www.danawa.com.
11. R.Agrawal, S,Dar, and H.V.jagadish, "Direct Transitive Closure Algorithms : Design and Performance Evaluation", ACM Tran. on Database Systems, Vol. 15, No. 3, 1990, pp. 427-458.
12. http://www.fh-oow.de/institute/iapg/personen/brinkhoff/generator/
13. T. Brinkhoff, "A Framework for Generation Network - Based Moving Objects" , GeoInformat.
14. 강홍민, 장재우 "공간 네트워크 데이터베이스를 위한 저장 및 색인 구조의 설계" , 한국정보과학회 가을 학술발표논문집, 제31권, 제2호, 2004, pp.133-136.

김용기
2002년 전북대학교 컴퓨터공학과(공학사)
2005년 전북대학교 대학원 컴퓨터공학과(공학석사)
2006년~현재 전북대학교 대학원 컴퓨터공학과 박사과정
관심분야 : 공간 데이터베이스, 질의처리 알고리즘, 공간 색인 구조

니하드 카림 초우더리
2004년 Chittagong University of Engineering & Technology (공학사)
2006년~현재 전북대학교 대학원 컴퓨터공학과 석사과정
관심분야 : 데이터 마이닝, 공간 데이터베이스, 질의처리 알고리즘

이현조
2006년 전북대학교 컴퓨터공학과(공학사)
2006년~현재 전북대학교 대학원 컴퓨터공학과 석사과정
관심분야 : 데이터 마이닝, 공간 데이터베이스, 공간 색인 구조

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