공간 데이타베이스에서 최근접 K쌍을 찾는 효율적 기법

(An Efficient Method for Finding K Nearest Pairs in Spatial Databases)

신 효 섭 † 이 석 호 ††
(Hyoseop Shin) (Sukho Lee)

요 약 R 트리와 같은 다차원 인덱스로 구성된 2개의 공간 데이타 집합들에 대하여 거리가 가까운 순서대로 점진적으로 객체 쌍을 찾는 거리조인(distance join) 알고리즘이 이전에 제안된 바 있다. 본 논문에서는 찾고자 하는 객체 쌍의 개수 K를 미리 정할 때 거리 우선순위 큐를 이용한 효율적인 K-거리조인 기법을 제안한다. 특히 양쪽 노드 확장 방식과 스위핑 축 및 방향의 선택 기법을 이용한 최적화된 평면 스위핑 가지치기 기법을 통한 거리조인 알고리즘을 개발한다. 실제 지리정보 데이타 집합을 가지고 실험을 수행하여 본 논문에서 제안한 알고리즘이 기존의 알고리즘들보다 좋은 성능을 나타냄을 확인한다.

Abstract *The distance join has been introduced previously, which finds nearest pairs in the order of distance incrementally among two spatial data sets built with multidimensional indexes like R-trees. We propose efficient K-distance joins when the number(K) of pairs to find is preset. Especially, we develop a distance join algorithm with bi-directional expansion and optimized plane sweeping using selection method of sweep axis and direction. The experiments on real spatial data sets show that the proposed algorithm is much better than the former algorithms.*

1. 서 론

공간 데이타베이스 관리 기술은 지리정보 시스템(geographic information systems), 이미지 처리 시스템(image processing systems), VLSI 및 CAD/CAM 등에서 핵심적인 기술로 자리잡고 있다. 공간 데이타베이스에서 질의는 대상 데이타 집합의 개수에 따라서 크게 두가지 유형으로 나눌 수 있다. 하나의 데이타 집합에 대해서는 포함 질의(containment query), 교차 질의(intersection query) 등의 질의와 더불어 최근 들어 많은 연구 결과를 산출하고 있는 최근접점 질의(nearest neighbor query)[1,2,3,4,5,6]가 있다. 다른 한가지는 포함이나 교차 조건을 바탕으로 2개 이상의 데이타 집합들에 대한 공간 조인(spatial join)[7,8,9] 질의가 있다. 예를 들면 "A 거리 상에 위치해 있는 빌딩들을 나열하라." 가 그것이다. 이 예에서는 거리 데이타 집합과 빌딩 데이타 집합에 대한 교차 공간 조인을 수행함으로써 결과값을 구할 수 있다.

한편 최근에 2 개 이상의 다차원 데이타 집합에 대하여 공간 상의 거리를 기준으로 하여 서로 간에 가장 근접해 있는 객체쌍들을 거리 순서대로 찾는 문제가 제안된 바 있다[10]. 이러한 질의는 지리정보 시스템 뿐만 아니라 이미지 데이타베이스 시스템(image database systems) 등에서 많은 필요성을 찾을 수 있다. 예를 들어 분리된 2개의 인물 데이타 집합들로부터 생김새가 비슷한 사람들을 찾는다든지, 서로 다른 종류의 지형 정보를 담고 있는 두 개의 데이타 집합들로부터 지리적으로 근접해 있는 객체 쌍을 찾는 문제가 그러한 예이다.

공간적인 거리에 기반하여 두 개 이상의 데이타 집합들로부터 가장 가까운 쌍을 찾는 SQL 질의 예는 다음과 같다.

질의 1 : 지리적으로 서로 가까운 순으로 <레스토랑,

† 학생회원 : 서울대학교 컴퓨터공학부
hsshin@db.snu.ac.kr
†† 종신회원 : 서울대학교 컴퓨터공학부 교수
shlee@comp.snu.ac.kr
논문접수 : 1999년 5월 13일
심사완료 : 2000년 3월 24일

호텔> 100 쌍을 검색하라.

```
SELECT R, S
FROM Restaurant R, Hotel S
ORDER BY distance(R.location, S.location)
STOP AFTER 100 ;
```

질의 1에서처럼 2차원 이상의 다차원 공간에서 두 개의 데이타 집합에 대하여 공간적인 거리에 기반하여 서로 거리가 가까운 순으로 정해진 개수의 객체쌍을 찾는 연산을 **거리 조인(Distance Join)**이라고 정의한다. 개념적으로 거리조인을 수행하는 가장 단순한 방법은 R에 속하는 모든 객체와 S에 속하는 모든 객체로 이루어진 순서쌍 R*S의 모든 원소를 상호간의 거리 순으로 정렬하여 상위 K개를 출력하는 방식이다. 물론 이 방법은 O(|R|*|S|)으로서 좋지 않은 성능을 나타낸다.

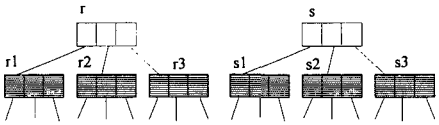
[10] 논문에서는 R-Tree 계열의 공간 인덱스[11, 12,13,14]를 이용하여 거리조인을 수행하는 알고리즘을 제시한 바 있다. 해당 논문은 특히 거리 조인의 결과값의 개수를 미리 정해놓지 않고 상위 모듈의 요구에 따라 결과값들을 점진적으로 반환하는 기법을 제시하였다.

본 논문에서는 찾고자 하는 객체 쌍의 개수 K가 미리 정해진 상황에서 점진적으로 찾는 기존 기법보다 효율적인 알고리즘들을 제시한다. 객체 쌍의 개수 K를 미리 정해 놓으면 현재까지 생성된 K개의 최소 거리들보다 거리가 더 큰 노드쌍들은 고려하지 않아도 된다는 장점을 가진다. 특히 본 논문에서는 기존의 한쪽 노드 확장 기법 이외에 최적화된 평면 스위핑(plane sweeping) 가지치기 기법을 활용한 양쪽 노드 확장 기법을 새롭게 제시한다. 이전 논문[10]에서 오버헤드가 많다고 지적된 양쪽 노드 확장 기법은 최적화된 평면 스위핑의 높은 가지치기 효율 때문에 오히려 훨씬 더 효율적인 알고리즘이 될 수 있다.

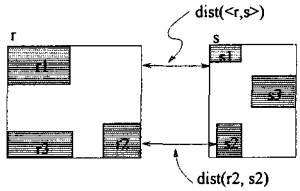
본 논문의 구성은 다음과 같다. 2절에서는 공간 인덱스를 이용한 거리조인 기법의 배경 지식 과 기존의 점진적 거리조인 알고리즘 및 K-거리조인에 대하여 설명한다. 3절에서는 양쪽 노드 확장 방식의 K-거리조인 알고리즘에 대하여 기술한다. 4절에서는 기존의 알고리즘들과 제시된 알고리즘의 성능을 비교 평가한다. 5절에서는 결론 및 추후 연구 과제를 제시한다.

2. 배경 지식 및 기존 연구

공간 데이타베이스를 구축하는 데 있어서 공간 데이타 집합을 효율적으로 운용하고 접근하기 위하여 다차원 트리 구조의 공간인덱스[11,12,13,14,15,16]를 이용하는 것은 보편화되어 있다. (그림 1)과 같은 트리 구조의 공간 인덱스들은 트리의 계층에 따라 공간 역시 계층적으로 분포한다는 특성을 가진다. 즉, 트리에서 상위 노드는 자기의 모든 하위 노드를 공간적으로 포함하고 있다. (그림 1)처럼 비단말 노드인 r과 s가 있다고 하자. 이때 r과 s사이의 최소거리는 r과 s의 자식노드들끼리의 최소거리보다 항상 작거나 같으며, 마찬가지로, r과 s사이의 최대거리는 r과 s의 자식노드들끼리의 최대거리보다 항상 크거나 같다. 이러한 특징에서 성질 1을 이끌어 낼 수 있다.



(a) Tree-Structured Spatial Indes



(b) Spatial Containment

그림 1 공간 인덱스의 공간 계층성

성질 1. 트리구조의 공간 인덱스 R과 S에 대해서 루트가 아닌 r∈R, s∈S 두 노드에 대하여, 다음 부등식들이 성립한다.

$$\begin{aligned} dist(r,s) &\geq dist(parent(r), parent(s)), \\ dist(r,s) &\geq dist(r, parent(s)), \\ dist(r,s) &\geq dist(parent(r), s). \end{aligned} \quad (1)$$

단, $dist(r,s)$는 r과 s의 MBR 간의 최소 거리

증명. 공간 인덱스의 공간 계층적인 특성으로부터 자명하다. □

성질1 로부터 공간인덱스를 통하여 거리조인을 수행할 때 하향식(top-down) 방식, 즉 루트노드부터 단말노드 방향으로 양쪽 트리를 확장하면 불필요한 하위 노드들 간의 거리 계산을 피할 수 있게 된다. 예를 들어 <r,s> 의 거리가 요구하는 거리보다 큰 값이면, r이나

s의 자식노드 들로 이루어진 노드쌍에 대한 확장은 더 이상 필요하지 않다. 거리조인을 수행할 때 공간인덱스를 하향식 방식으로 확장하는 기법은 2 가지 방식을 고려할 수 있다. 즉, 노드 쌍 <r,s>에 대하여 한 쪽은 자식 노드, 다른 한 쪽은 그대로 부모 노드로 짝을 짓는 한쪽 노드 확장 방식과 양 쪽 모두 자식 노드들로 짝을 짓는 양쪽 노드 확장 방식이 있다.

2.2 주 우선 순위 큐(Main Priority Queue)

노드 확장에서 만들어지는 쌍들은 주 우선 순회 큐에서 관리된다. 큐의 우선 순위는 각 쌍에서 두 노드 간의 **거리**(distance)이다. 즉, 큐에 들어있는 쌍들 중에서 두 노드 간의 거리가 가장 작은 쌍을 먼저 순회한다. 그리고 거리가 같은 쌍 중에서는 트리 상에서 단말노드에 더 가까이 있는 노드의 쌍을 먼저 우선하여 고려하도록 한다. 이는 거리조인의 결과집합을 구성하는 객체 타입 쌍 <object,object>를 가능한 빨리 검색하기 위함이다. 주 우선 순위 큐에서 나온 쌍의 종류가 객체-객체(<object,object>)이면 답으로 출력하고, 그 밖에 객체-중간노드(<object,node>), 중간노드-객체(<node, object>), 중간노드-중간노드(<node,node>) 유형이면 노드 확장을 수행한다. 이러한 방식으로 사용자가 원하는 개수 만큼의 <object,object> 노드쌍을 점진적으로 생성하는 알고리즘을 **점진적 거리조인**(IDJ, Incremental Distance Join)으로 정의한다. (그림 2)는 점진적 거리 조인의 기본 구조를 나타낸 것이다.

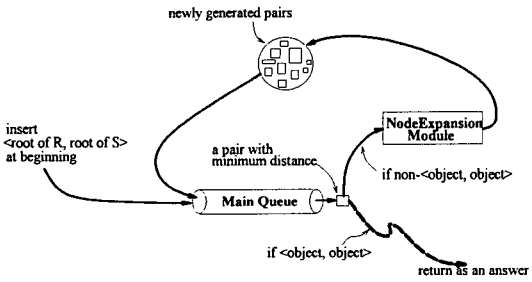


그림 2 점진적 거리조인의 기본 구조[10]

2.3 K 거리조인(KDJ, K Distance Join)

거리조인을 수행하기 이전에 검색할 객체 노드쌍의 개수 K를 미리 정해 놓으면 알고리즘의 성능 향상을 기대할 수 있다. 즉, K를 미리 알고 있으면 점진적 알고리즘보다도 불필요하게 발생하는 노드 쌍들을 더 많이 가지치기(pruning) 할 수 있다.

노드 확장(node expansion)을 통해서 만들어진 객체 노드쌍 <object,object> 중에서 현재까지 거리 중 가장 작은 K개를 유지하고, 그 중 가장 큰 거리값을 **qDmax** 라고 정하고, 생성되는 노드쌍 중 이 값보다 더 큰 값을 가지는 불필요한 노드쌍들을 주 우선 순위 큐에 삽입하지 않고 가지치기하는데 사용한다. 이를 위하여 K개의 <object,object> 쌍들에 대한 거리를 유지하기 위하여 주 우선순위 큐와는 별도의 **거리 우선순위 큐**(distance priority queue)를 둔다. qDmax는 현재까지 발생한 거리값들을 기반으로 지정되기 때문에 처음에는 무한대값으로 지정되며, 알고리즘이 수행됨에 따라 점점 더 작아지며, 최종적인 실제 Dmax 값보다는 항상 크거나 같다. 이러한 거리 우선 순위 큐를 이용한 거리조인을 **K 거리조인(KDJ, K Distance Join)**이라고 정의한다. K 거리조인은 노드 확장에 의한 노드쌍이 모두 주 우선순위 큐에 들어가지 않고 qDmax에 의해서 많이 가지치기 되므로 점진적 거리조인보다 성능 향상이 기대된다. (그림 3)은 K 거리조인의 기본 구조를 표현한 것이다.

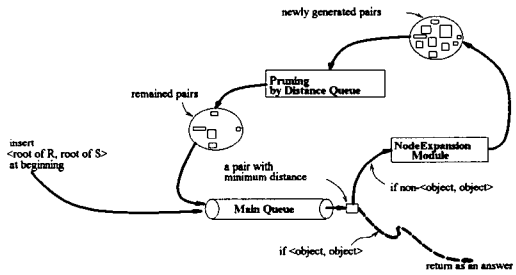


그림 3 K 거리조인의 기본 구조

2.4 기존 연구

[10] 논문에서는 한쪽 노드만을 처리하여 확장하는 방식을 이용한 점진적 거리조인과 K 거리조인을 제안하였다. 한쪽 노드만을 처리하는 방식은 양쪽 노드만을 처리하는 방식보다 한번에 생성하는 노드쌍의 개수가 줄어든다는 장점이 있다. 하지만 이 방식은 공간인덱스의 각 노드를 필요 이상으로 접근하는 오버헤드가 발생하므로 양쪽 노드 순회보다 R-tree등의 공간 인덱스 디스크 노드 접근횟수가 증가한다. 예를 들어 (그림 1)에서 <r1, s1> 노드쌍을 접근하기 위해서는 <r,s> 와 <r,s1> 의 두 번의 노드쌍 접근이 선행적으로 필요하다. 반면 양쪽 노드를 동시에 처리하는 노드 확장 방식에서는 <r,s>에서 바로 <r1,s1>을 만들어낼 수 있다. 한쪽 노드를 순회하는 방식의 또 다른 문제점은 노드 확장시 모든 노드간의 비교가 불가피하다는 점이다. 예를 들어 <r,s>에서 s 노드를 확장한다면 r에 대해서

s1, s2, s3 등 모든 상대 노드에 대해서 짝을 짓게 되어 많은 거리 계산 비용이 요구된다.

3. 양쪽 노드 확장 방식의 K 거리조인 (BKDJ, K Distance Join with Bi-directional node expansion)

양쪽 노드 확장 방식은 어느 한 노드쌍을 처리할 때 양쪽 노드들의 모든 자식 노드들에 대한 순서쌍을 생성하여 우선 순위 큐에 삽입한다. 따라서 이 방식은 트리 순회시 트리의 노드 방문 횟수를 줄여주긴 하지만 한 번 처리시 한쪽 노드 확장 방식보다 훨씬 많은 양의 노드쌍을 생성하는 노드쌍 폭발 현상을 일으켜 성능 상의 문제가 된다.

3.1 평면 스위핑(plane sweeping) 가지치기

양쪽 노드 순회 방식에서는 많은 노드쌍들에 대한 거리 계산 및 우선순위 큐 삽입하는 연산에 드는 비용이 매우 크다. 이러한 문제점은 기존의 공간조인에서 사용한 평면 스위핑 기법[7,8,9]을 변형하여 거리조인에 적용함으로써 극복될 수 있다. 평면 스위핑은 다차원의 공간 데이타를 한 축(x축, y축 등등)을 기준으로 정렬한 다음, 정렬한 순서대로 축 방향으로 스위핑하면서 데이타를 처리하는 방식이다.

K 거리조인에서 평면 스위핑 가지치기의 기본 원리는 다음과 같다. 즉, 평면 스위핑시 발생하는 각 노드쌍에 대해서 노드쌍의 스위핑 축 간의 거리가 거리 우선순위 큐에 의해서 지정되는 qDmax보다 큰 것은 노드쌍의 실제거리가 qDmax보다 크다는 것이 자명하기 때문에 따로 계산을 필요하지 않게 된다 ($axis\_distance(r,s) \le real\_distance(r,s)$). 물론 평면 스위핑을 수행하기 위해서는 각 노드가 스위핑 축 방향으로 정렬되어야 하는 추가 비용이 든다. 이 비용은 O(|r| * log |r|)에 불과한 반면, 평면 스위핑을 이용하지 않을 시 드는 비용은 O(|r| *|s|)에 달하며, 평면 스위핑은 이를 O(|r|+|s|)으로 줄여주는 효과를 보여주기 때문에 정렬 비용에도 불구하고 많은 성능 개선을 기대할 수 있게 된다.

알고리즘 1은 평면 스위핑을 이용한 양방향 확장 거리조인 알고리즘이다. 프로시져 PlaneSweep이 <object,object> 가 아닌 노드쌍에 대한 양방향 노드 확장을 수행한다. 24번째 줄에서 축방향 거리와 qDmax와의 비교를 통한 스위핑 축에 대한 가지치기가 있고, 그 조건을 통과한 노드쌍만이 실제 거리에 대한 가지치기를 수행한다(25번째 줄). 특히 첫 번째 축간 거리(axis_distance) 비교 조건은 루프(while문)를 빠져나가는 조건으로서 많은 불필요한 노드쌍간의 계산을 없애줄 수 있다. 두 단계 조건들을 모두 통과한 노드쌍은 주 우선순위 큐와 거리 우선 큐에 삽입한다(26, 27번째 줄). 거리 우선 순위 큐에는 <object,object> 노드쌍에 대한 거리값만 삽입되며, 삽입되면 qDmax 값이 더 작은 값으로 갱신된다.

알고리즘 1 BKDJ : 평면 스위핑을 이용한 양쪽 노드 확장 거리조인

```
1   set AnswerSet ← an empty set;
2   set QM, QD ← empty main and distance queues;
3   insert a pair <R.root, S.root> into the main queue QM;
4   while |AnswerSet| < k and QM ≠ 0 do
5       set c ← dequeue(QM);
6       if c is an <object, object> then AnswerSet ← {c} ∪
        AnswerSet;
7       set PlaneSweep(c);
8   end
9
10  procedure PlaneSweep(<l,r>)
11  set L ← sort_axis({child nodes of l});   // Sort the child nodes of l
                                                by axis values.
12  set R ← sort_axis({child nodes of r});   // Sort the child nodes of r
                                                by axis values.
13  while L ≠ 0 and R ≠ 0 do
14      n ← a node with the min axis value ∈ L ∪ R;
15      if n ∈ L then
16              L ← L - {n}; SweepPruning(n, R);
17      else
18              R ← R - {n}; SweepPruning(n, L);
19      end
20  end
21
22  procedure SweepPruning(n, List)
23  for each node m ∈ List in an increasing order of axis value do
24      if axis_distance(n,m) > qDmax then return;   // No more
                                                        candidates
25      if real_distance(n,m) ≤ qDmax then
26              insert <n,m> into QM;
27              if <n,m> is an <object, object> then insert
                                        real_distance(n,m) into QD;
28                          //qDmax modified
29      end
302 end
```

예를 들어, (그림 4)에서 r1에 대해서 s1, s2, s3, s4는 r1과의 스위핑 축인 x 축간의 거리가 qDmax보다 작기 때문에 실제 거리를 계산하지만 <r1, s5>, <r1, s6>, <r1, r7>은 실제 거리를 계산하지 않고 바로 가지치기가 된다.

3.2 평면 스위핑의 최적화

앞 절의 BKDJ에서는 평면 스위핑 가지치기 시 노드쌍을 구성하는 양쪽노드의 공간 상에서의 위치관계를 고려하지 않고 단순히 한 축에 대해서 정렬하는 방식을 택하고 있다. 이와는 달리 각각의 노드쌍에 대하여 두 노드간의 위치 관계를 고려하여 데이타를 스위핑하는 기준축을 달리하거나 스위핑의 방향을 순방향 혹은 역방향으로 할 때 알고리즘의 성능은 더욱 개선될 수 있다.

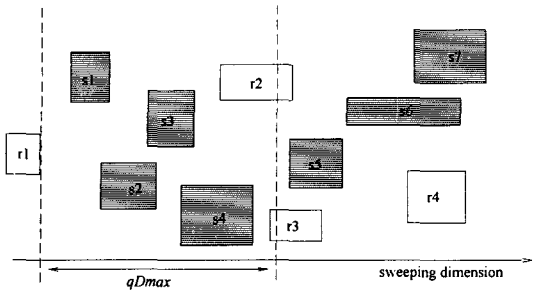


그림 4 BKDJ에서의 평면 스위핑 가지치기

교차(intersection) 조건 공간조인[7,8,9]에서 평면 스위핑은 스위핑하는 축을 선택하거나 스위핑의 방향을 달리해도 두 노드 간의 스위핑 축 간의 교차 검사 횟수나 혹은 실제 노드들 사이의 교차 검사 횟수가 달라지진 않는다. 하지만 K 거리 조인에서는 유동적으로 변하는 qDmax값에 따라서 두 노드 간의 스위핑 축 거리와 실제 거리에 의해서 2단계로 가지치기를 수행하므로 스위핑 축의 선택이나 스위핑 방향에 의해 스위핑 축 거리 및 실제 거리 계산 비용을 절감할 수 있다.

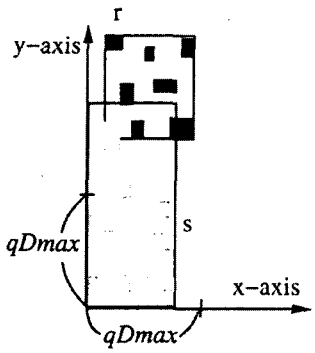


그림 5 스위핑 축의 선택

(1) 스위핑 축의 선택

주 우선순위 큐에서 삭제되어 처리될 어느 노드 쌍의 두 노드가 공간적으로 (그림 5)와 같이 위치해 있다고 가정하자. 이 두 노드의 자식노드 쌍 들에 대하여 평면 스위핑 가지치기를 수행한다고 할 때, 만약 x축을 스위핑 축으로 고정한다면 두 노드의 자식 노드들 사이의 x 축 간 거리가 qDmax보다 항상 작기 때문에 결국 모든 자식 노드들간의 실제 거리를 계산하여야 한다. 하지만 y축 방향으로 스위핑하게 되면 많은 자식 노드들간의 y 축 간 거리가 qDmax보다 크기 때문에 그 노드들 사이끼리는 실제 거리를 계산할 필요가 없게 되어 평면 스위핑의 효과가 더욱 두드러진다. 이와 같이 스위핑 축의 선택에 따라서 가지치기 효율이 달라질 수 있다.

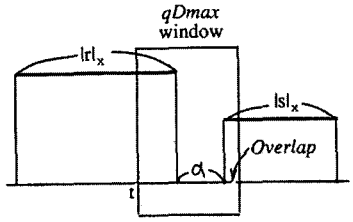
직관적으로는 축방향 길이가 가장 긴 축을 선택하는 것이 더 이로워 보이나, 두 MBR의 겹침 정도나 qDmax 값에 따라서 꼭 그렇지는 않다. 본 논문에서는 스위핑 축의 정확한 선택을 위한 척도로서 **스위핑 인덱스**(sweeping index)를 새롭게 정의한다. 즉, 주 우선순위 큐로부터 삭제된 각 노드쌍 <r,s>와 주어진 qDmax 에 대해서 각 축에 대한 스위핑 인덱스 값을 계산하고 그 값들을 비교함으로써 그 노드쌍에 대한 스위핑 축을 정한다. 개념적으로 스위핑 인덱스 값은 해당 축에 대한 실제 거리(real distance)를 계산하는 횟수의 상대적인 값을 나타낸다. 축 x에 대한 스위핑 인덱스는 다음 공식으로 표현한다.

$$Sweeping\ Index_x = \int_0^{|r|_x} \frac{Overlap(qDmax, r, t)}{|s|_x}\,dt + \int_0^{|s|_x} \frac{Overlap(qDmax, s, t)}{|r|_x}\,dt \quad (2)$$

위의 공식에서 $|r|_x$와 $|s|_x$는 각각 r과 s의 x축 방향 너비를 나타낸다. 공식의 첫 번째 적분식에서 Overlap(qDmax, r, t)는 x축 상에서 윈도우 [t, t + qDmax]와 겹치는 s의 x 축 방향 길이를 나타낸다. 따라서 Overlap(qDmax, r, t)는 r 의 자식 노드 중 x 축 상에서 t 지점에 있는 노드와 실제 거리를 계산해야 하는 s의 자식 노드들이 포함되어 있는 구간의 x 축 길이를 나타낸다. 이 값을 $|s|_x$으로 나누면 s의 전체 자식노드들 중 t 지점에 있는 r의 자식노드와 실제 거리를 계산해야 하는 s 자식 노드의 개수 비율을 구할 수 있다. t가 0에서 $|r|_x$ 까지 변할 때 이 값들을 적분하면 x축에 대한 실제 거리 계산 비율값이 나온다. 마찬가지로, 두 번째 적분식은 s에 대한 r 자식노드들의 실제 거리 계산 비율을 나타낸다.

예를 들어, r 과 s 노드의 x축에 대한 위치 관계가 (그림 6)의 위쪽 그림과 같다고 하자.

이 예에서는 r이 s의 왼쪽에 위치하기 때문에 공식 (2) 두 번째 적분식은 항상 0이 된다. 한편 r에 대해서는 t가 [0, $|r|_x$]사이에서 변화할 때 주어진 qDmax 윈도우 크기에 대하여 Overlap 값을 계산한 그래프가 그림 6의 아래쪽 그림과 같다. 그래프에서 t가 [0, $|r|_x + a - qDmax$] 사이에 있을 때는 qDmax 윈도우와 s가 겹치는 구간이 없기 때문에 Overlap 값이 0이 된다는 것을 알수 있다(a는 r과 s의 최소거리). 빗금친 부분의 면적을 $|s|_x$ 로 나눈 값이 공식 (2)의 왼쪽 적분 값이며, 이 값이 곧 x축의 스위핑 인덱스 값이 된다.



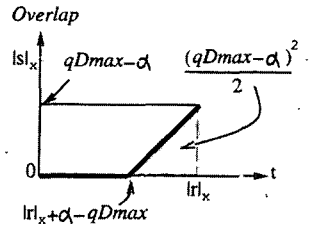


그림 6 스위핑 인덱스

각 축에 대한 스위핑 인덱스 값이 모두 계산되면, 그 중 가장 작은 값을 가지는 축을 해당 노드쌍에 스위핑 축으로 선택한다. 스위핑 인덱스를 계산하여 축을 선택하는 과정은 (알고리즘 1)에서 10번째 줄과 11번째 줄의 사이에 들어간다. 여기서 한가지 주목할 점은 스위핑 인덱스가 적분으로 표현되었다고 하지만, 이 적분식은 결국 간단한 산술식으로 표현될 뿐 아니라 각 부모 노드쌍에 대하여 단 한 번 계산되므로, 수 천개의 자식 노드쌍을 생성하는 비용과 비교했을 때, 알고리즘의 많은 성능 향상을 기대할 수 있다.

(2) 스위핑 방향의 선택

스위핑 축이 일단 결정되면, 그 다음 과정으로 스위핑 방향을 선택한다. 스위핑 방향은 순방향 스위핑과 역방향 스위핑으로 결정지어 진다. 순방향 스위핑은 r과 s의 자식 노드들을 선택된 축에 대해서 오름차순으로 스위핑하는 방법이고, 역방향 스위핑은 내림차순으로 스위핑하는 방법이다.

노드 r과 s가 스위핑 축에 사상되었을 때, 그 사상된 이미지는 세가지의 연속된 구간으로 나누어 진다. 예를 들어 그림 7(a) 처럼 두 노드가 겹쳐 있을 때는 r만이 사상된 가장 왼쪽 구간, r과 s가 동시에 사상된 가운데 구간, s만이 사상된 가장 오른쪽 구간으로 나눌 수 있다. 그림 7(b) 처럼 두 노드가 분리되어 있을 때는 중간 구간은 두 노드에 의해서 사상되지 않는 구간이 된다. 또한 그림 7(c)처럼 한 노드가 다른 노드를 완전히 포함하는 경우는 겹쳐지는 구간을 중간 구간으로 볼 수 있다.

(a) intersected (b) separated (c) contained

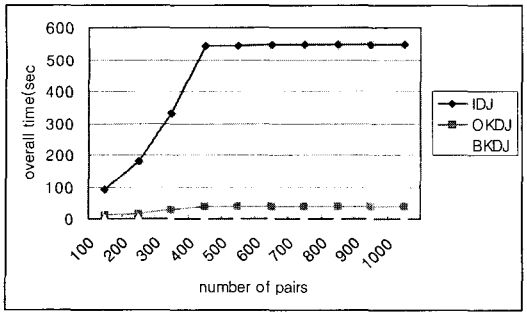
그림 7 스위핑 축에 사상된 r과 s에 의한 세가지 구간

이 때, 스위핑 방향은 세 구간 중 가장 왼쪽 구간의 길이와 가장 오른쪽 구간의 길이를 비교하여 결정한다. 즉, **가장 왼쪽 구간의 길이가 가장 오른쪽 구간의 길이보다 작다면 순방향 스위핑을, 그렇지 않으면 역방향 스위핑을 한다.** 스위핑 방향을 이와 같은 전략으로 선택함으로써 거리가 작은 노드쌍부터 먼저 생성하여 주 우선순위 큐와 거리 우선 순위 큐에 삽입하게 되고 (<object, object>쌍인 경우), 이에 따라 qDmax 값을 좀 더 빨리 감소시키는 효과를 보여준다.

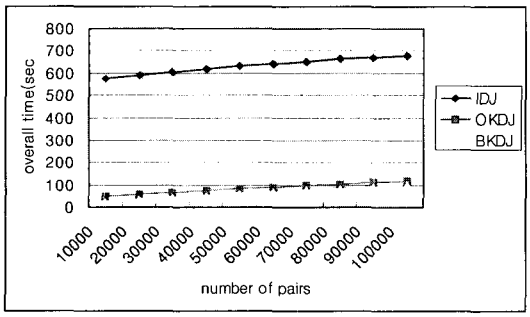
4. 성능 평가

점진적 거리조인 **IDJ**, 한쪽노드를 확장하는 K-거리조인 **OKDJ**, 양쪽노드를 동시에 확장하는 K-거리조인 **BKDJ**의 성능을 비교 실험하였다. 실험에 사용한 컴퓨터는 Solaris 2.5.1가 탑재된 Sun4m-sparc, 메인메모리 108M, 디스크 1G 규격이다. 실험에 사용한 데이타는 [6]에서 사용했던 것으로써 미국 Bureau of Census에서 제공하는 Tiger/Line 지리정보 데이타 중 미국 캘리포니아 주의 강과 도로를 나타내는 각각 128,971개와 131,461개의 실제 직선 데이타들이며, 이를 R*-트리[3]로 구성하여 사용하였다. 성능의 기준은 전체 수행 시간과 주 우선순위 큐에 삽입되는 노드쌍의 개수 등 2가지로 나누어 평가하였다. 또한 K가 100에서 1,000까지 비교적 적은 개수의 노드쌍을 검색할 때와 10,000에서 100,000까지 비교적 많은 개수의 노드쌍을 검색할 때 2가지로 나누어 실험결과를 알아보았다.

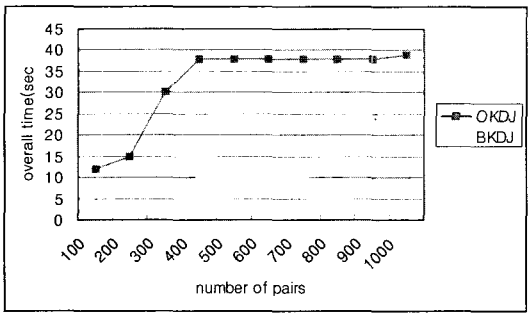
먼저 전체 수행 시간에서는 K가 작은 경우와 큰 경우 모두 점진적 거리조인보다 K-거리조인들이 훨씬 좋은 성능을 나타내었다(그림 8(a),(b)). BKDJ는 IDJ보다 최대 수십배의 시간을 단축시켜 주었다. 이는 거리 우선순위 큐를 이용한 가지치기의 효율이 높다는 것을 입증한 것이다. 한편, OKDJ와 BKDJ의 수행 시간을 보면 BKDJ가 OKDJ보다 K가 작은 범위(그림 8(c))에 있을 때는 많은 차이로 BKDJ가 수행 시간을 단축하고 있다는 것을 알 수 있다. K가 10000이상인 경우(그림 8(d))에도 수행 시간의 우세를 보이고 있으나 비율로 따져보



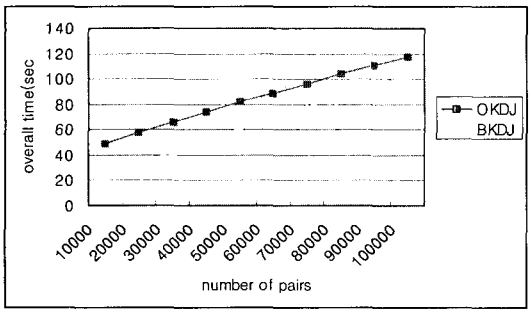
(a) K<=1000, IDJ와 KDJ의 수행 시간



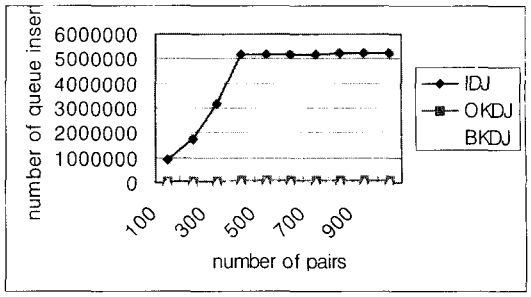
(b) K>=10000, IDJ와 KDJ의 수행 시간



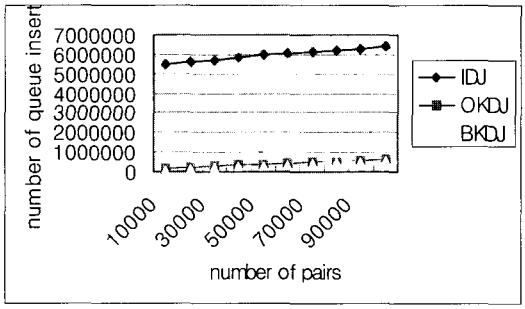
(c) K<=1000, OKDJ와 BKDJ의 수행 시간



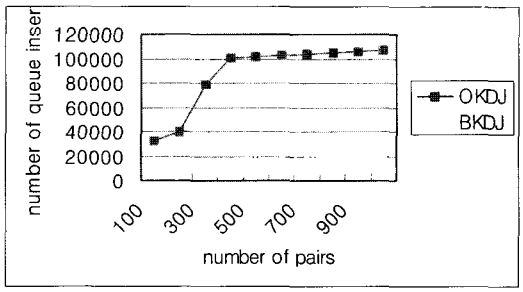
(d) K>=10000, OKDJ와 BKDJ의 수행 시간



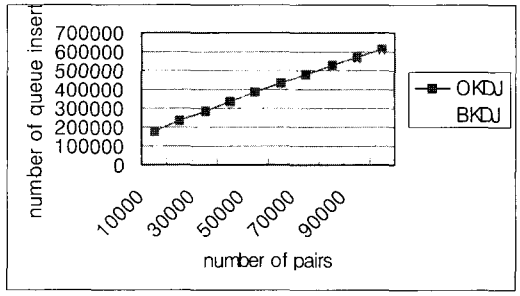
(e) K<=1000, IDJ와 KDJ의 큐 삽입수



(f) K>=10000, IDJ와 KDJ의 큐 삽입수



(g) K<=1000, OKDJ와 BKDJ의 큐 삽입수

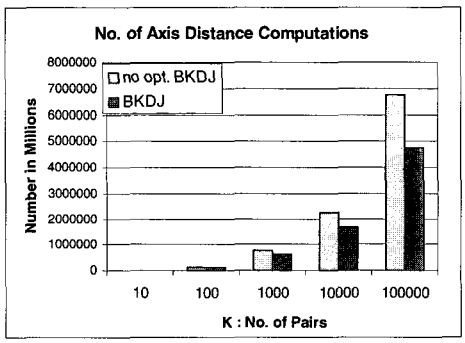


(h) K>=10000, OKDJ와 BKDJ의 큐 삽입수

그림 8 IDJ, OKDJ, BKDJ의 성능 비교

면 K가 작은 범위에 있을 때보다는 그 차이가 줄어 든다. 일반적으로 KDJ에서는 K가 커질수록 거리 우선 순위 큐의 크기가 커지고, Dmax 값의 감소 속도가 둔화되며 이에 따라 가지치기의 효율도 떨어지기 마련인데, OKDJ보다는 BKDJ가 더욱 더 민감하게 K에 영향을 받는다는 것을 보여준다.

주 우선 순위 큐에 삽입하는 노드 쌍의 수 역시 전체 수행 시간과 아주 유사한 결과를 보였다(그림 8(e),(f),(g),(h)). 즉, KDJ가 IDJ보다 훨씬 나은 결과를 보였으며, BKDJ가 OKDJ보다 좋은 성능을 보였다. 전체 수행 시간은 노드쌍을 큐에 삽입하는 횟수에 비례한다고 볼 수 있다.



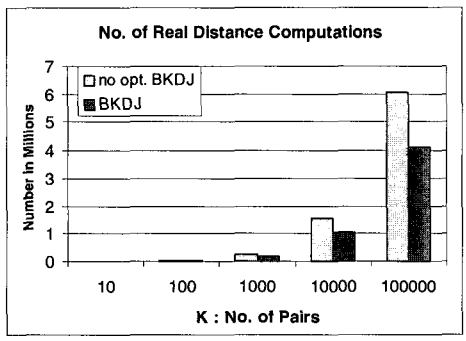


그림 9 최적화된 평면 스위핑에 의한 성능 향상

4.1 스위핑 축 선택에 따른 성능 향상

공식 (2)를 적용하여 스위핑 축을 각 노드쌍 마다 적용하는 최적화된 평면 스위핑을 사용하는BKDJ와 이를 적용하지 않은 BKDJ(no opt. BKDJ) 사이의 성능을 비교하였다. 최적화되지 않은 BKDJ는 스위핑 축을 x 축으로 고정하고 실험을 하였다. (그림 9)가 보여주듯이 최적화된 평면 스위핑 기법은 축 간 거리와 실제 거리 계산 횟수를 30에서 42 퍼센트까지 줄여주었다.

4.2 거리 우선 순위 큐의 구현

거리 우선 순위 큐의 원소를 <object,object>쌍으로만 구성할 수 있고, 기타 <node,object>, <object, node>, <node,node>등의 모든 유형을 고려하여 구현할 수 있다. <node,object>, <object,node>, <node, node> 등 중간 노드를 포함하는 쌍은 하나 이상의 <object,object> 쌍을 내포할 수 있다. 임의의 쌍 <node1,node2>로부터 미래에 생성 가능한 <objeect, object> 쌍의 개수는 <node1,node2>의 각 노드 node1,node2의 트리에서의 깊이와 트리의 최소 팬아웃(fanout) 값을 통하여 계산할 수 있다. 따라서 이론적으로 거리 우선 순위 큐의 길이가 K보다 작은 값으로 유지될 수 있다. 하지만 중간 노드쌍에 대하여 거리 우선 순위 큐에서의 우선 순위가 되는 거리는 두 노드 간의 최소 거리(dmin)가 아니고 최대 거리(dmax)이며, 반면 <object,object>쌍에 대해서는 최소 거리가 우선 순위이기 때문에 중간 노드를 포함하는 노드쌍 <node, object>, <object,node>, <node,node>들은 <object, object>에 비해서 거리 우선 순위 큐 안에서 우선 순위가 상당히 떨어질 수 있다. 실제로 실험을 통해서 알아본 결과 알고리즘이 진행됨에 따라 거리 우선 순위 큐에는 거의 <object,object> 유형만 남게 되어 중간 노드쌍을 고려하는 효과가 거의 없게 된다. 더군다나 중간 노드쌍들까지 고려하는 경우에는 거리 우선 순위 큐를 유지하는 비용이 증가하였다. (그림 10)은 BKDJ에 대하여 <object,obejct>쌍 만을 고려한 경우와 모든 종류의 노드 쌍을 고려했을 경우 성능을 비교한 것이다. 성능에 별다른 차이를 볼 수 없으며 <object,obejct>쌍이 약간 나은 반응 시간을 나타내는 것을 알 수 있다. 이는 거리 우선 순위를 관리하는 비용의 차이라고 볼 수 있다.

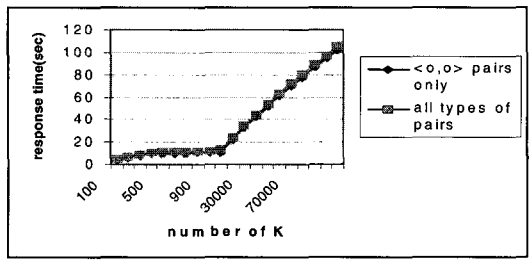


그림 10 거리 우선 순위 큐의 구현 기법 비교

5. 결론 및 추후 연구 과제

본 논문에서는 최근에 제안된 공간 조인 기법인 거리

조인을 3가지 알고리즘으로 세분화하여 제시하였다. [11]에서 제안한 기법을 점진적 거리조인(IDJ)으로 분류하였고, 또한 이와 구분하여 거리 우선 순위 큐를 이용한 K개의 노드쌍을 찾는 K 거리조인(KDJ) 기법을 정의하였다.

특히, 본 논문에서는 양쪽 노드 확장 방식과 스위핑 축 및 방향의 선택 기법을 이용한 최적화된 평면 스위핑 가지치기 기법을 통한 거리조인 알고리즘(BKDJ)을 개발하였으며, 기존에 제안된 한쪽 노드만을 확장하는 거리 조인 알고리즘들에 비하여 매우 우수한 성능을 나타냄을 실험을 통하여 확인하였다.

K 거리조인의 문제점은 K값이 커지면 거리 우선순위 큐에 의한 가지치기 효율이 떨어짐에 따라 알고리즘의 성능이 좋지 않게 된다는 점이다. 이는 추후 연구할 과제이다.

참 고 문 헌

[1] Arya S., Mount D. M., Netanyahu N. S., Silverman R., "An Optimal Algorithm for Approximate Nearest Neighbor Searching in Fixed Dimensions," Journal of ACM, to appear, 1999

[2] Berchtold S., Bohm C., Keim D. A., Kriegel H. P., "A Cost Model for Nearest Neighbor Search in High-Dimensional Data Space," ACM PODS, 1997

[3] Donjerkovic D. and Ramakrishnan R., "Probabilistic Optimization of Top N Queries," CS-TR-99-1395, UW-Madison, 1999

[4] Hjaltason G. R. and Samet H., " Ranking in Spatial Databases," Proc. of 4th Symp. on Spatial Datanases, 1995

[5] RoussopoulosN. , Kelley S. , Vincent F. , "Nearest Neighbor Queries," Proc. of ACM SIGMOD, 1995

[6] Seidl T. and Kriegel H.P. , "Optimal Multi-Step k-Nearest Neighbor Search," Proc. of ACM SIGMOD, 1998

[7] Arge L., Procepiue O., Ramaswamy S., Suel T., Vitter J. S., "Scalable Sweeping-Based Spatial Join" Proc. of VLDG conf., 1998

[8] Brinkhoff T., Kriegel H. P., Seeger B., "Efficient Processing of Spatial Joins Using R-Trees," Proc. of ACM SIGMOD, 1993

[9] Patel J. M. and DeWitt D. J., "Partition Based Spatial-Merge Join", Proc. of ACM SIGMOD, 1996

[10] Hjaltason G. R. and Samet H. , "Incremental Distance Join Algorithms for Spatial Databases," Proc. of ACM SIGMOD, 1998

[11] Beckmann N., Kriegel H. P., Schneider R., Seeger B., "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles," Proc. of ACM SIGMOD, 1990

[12] Berchtold S., Keim D., Kriegel H. P., " The X-Tree: An Index Structure for High-Dimensional Data," Proc. of VLDB conf, 1996

[13] Guttman A., "R-Trees: A Dynamic Index Structure for Spatial Searching," Proc. of ACM SIGMOD, 1984

[14] Sellis T. , Roussopoulos N. , Faloutsos C. , "The R+-Tree: A Dynamic Index for Multi-Dimensional Objects," Proc. of VLDB conf. 1987

[15] Henrich A., "The LSD^h-tree: An Access Structure for Feature Vectors", Proc. of ICDE, 1998

[16] Robinson J. T., "The K-D-B-tree: A Search Structure for Large Multidimensional Dynamic Indexes," Proc. of ACM SIGMOD, 1981

신 효 섭

1994년 서울대학교 컴퓨터공학과 졸업. 1996년 서울대학교 컴퓨터공학과에서 석사학위 취득. 1996년 ~ 현재 서울대학교 컴퓨터공학부 박사과정. 1994년 4월 ~ 2000년 3월 미국 아리조나주립대학교 방문 연구활동. 관심분야는 데이타베이스 질의 처리, 다차원 데이타 처리, 멀티미디어 서버, 공간데이타베이스등.

이 석 호

1964년 연세대학교 정치외교학과 졸업. 1975년, 1979년 미국 텍사스대학교 전산학 석사와 박사학위 취득. 1979년 ~ 1982년 한국과학원 전산학과 조교수. 1982년 ~ 1986년 한국정보과학회 논문편집위원장. 1986년 ~ 1988년 한국정보과학회 부회장. 1988년 ~ 1989년 미국 IBM T.J. Watson 연구소 객원교수. 1988년 ~ 1990년 데이타베이스연구회 운영위원장. 1989년 ~ 1991년 서울대학교 중앙교육연구전산원 원장. 1994년 한국정보과학회 회장. 1997년 ~ 현재 한국학술진흥재단 부설 첨단학술정보센터 소장. 1982년 ~ 현재 서울대학교 컴퓨터공학부 교수. 관심분야는 데이타베이스, 멀티미디어 데이타베이스