DOT 색인을 이용한 효율적인 공간 조인 기법
(An Efficient Spatial Join Method Using DOT Index)

백 현 [†] 윤 지 희 [††] 원 정 임 [†††] 박 상 현 [††††]
(Hyun Back) (Jee-Hee Yoon) (Jung-Im Won) (Sang-Hyun Park)

요 약 지리정보시스템에서 빈번히 사용되는 공간 조인 연산자의 성능을 보장하기 위해서는 효율적인 색인 기법의 선택이 중요하며, 대표적인 색인 기법으로는 R^*-tree를 이용한 방법이 알려져 있다. 본 논문에서는 DOT(DOuble Transformation) 공간 색인을 이용한 효율적인 공간 조인 처리 기법을 제시하고 이를 R^*-tree를 이용한 공간 조인 처리 기법과 비교한다. DOT 공간 색인 기법은 공간 객체의 MBR 정보를 공간 순서화 곡선을 사용하여 하나의 1차원 값으로 변환한 후 그 값을 검색 키로 갖는 B^+-tree 색인 구조를 구성하는 방법으로서, 이를 이용하면 전통적인 데이타베이스의 주 색인을 적용할 수 있다는 중요한 특징을 가진다. 본 논문에서는 공간 객체의 MBR 정보를 하나의 1차원 값으로 변환하기 위해 사용되는 공간 순서화 곡선의 규칙성을 분석함으로써 공간 변환 연산의 횟수를 대폭 감소시킨 효율적인 DOT 색인 기반의 공간 조인 알고리즘을 제안한다. 제안된 알고리즘에서는 반복적으로 수행되는 공간 변환 연산의 횟수를 줄이기 위하여 질의 영역을 공간 순서화 곡선이 연속 운행하는 가능한 최대 크기의 면적으로 분할하는 쿼터 분할 기법을 사용한다. 다양한 분포와 크기를 갖는 데이타 집합을 대상으로 R^*-tree를 이용한 공간 조인 처리 기법과의 비교 실험을 수행한 결과 최대 약 3배의 성능 이익을 얻을 수 있음을 확인할 수 있었다.

키워드 : 공간 조인, 공간 색인 기법, 공간 순서화 곡선, DOT 색인

Abstract The choice of an effective indexing method is crucial to guarantee the performance of the spatial join operator which is heavily used in geographical information systems. The R^*-tree based method is renowned as one of the most representative indexing methods. In this paper, we propose an efficient spatial join technique based on the DOT(DOuble Transformation) index, and compare it with the spatial join technique based on the R^*-tree index. The DOT index transforms the MBR of an spatial object into a single numeric value using a space filling curve, and builds the B^+-tree from a set of numeric values transformed as such. The DOT index is possible to be employed as a primary index for spatial objects. The proposed spatial join technique exploits the regularities in the moving patterns of space filling curves to divide a query region into a set of maximal sub-regions within which space filling curves traverse without interruption. Such division reduces the number of spatial transformations required to perform the spatial join and thus improves the performance of join processing. The experiments with the data sets of various distributions and sizes revealed that the proposed join technique is up to three times faster than the spatial join method based on the R^*-tree index.

Key words : spatial join, spatial indexing method, space filling curve, DOT index

· 이 논문은 2006년도 한림대학교 교비연구비(HRF-2006-030)에 의하여 연구되었음. 또한 이 연구에서는 한국과학기술원의 Database and Multimedia Lab에서 개발된 버퍼 관리 프로그램을 사용하였음
† 정 회 원 : 한림대학교 컴퓨터공학과
backhyun@sysgate.co.kr
†† 종신회원 : 한림대학교 정보통신공학부 교수
jhyoon@hallym.ac.kr
††† 정 회 원 : 한양대학교 정보통신학부 교수
jiwon@hanyang.ac.kr
†††† 종신회원 : 연세대학교 컴퓨터과학과 교수
sanghyun@cs.yonsei.ac.kr
논문접수 : 2006년 3월 3일
심사완료 : 2007년 5월 15일

1. 서 론

지리정보시스템(Geographic Information System: GIS)은 많은 응용 분야에서 사용되고 있으며 그 중요성이 점차 증가하고 있다. 지리정보시스템은 공간 및 비공간 데이타를 동일 시스템 내에서 효율적으로 관리하는 일종의 데이타베이스 시스템으로 볼 수 있다. 따라서 전통적인 데이타베이스 시스템의 기능을 확장하여 지리정보시스템을 구현하는 경우 데이타베이스 시스템이 가지고

있는 대용량의 저장, 검색, 질의 최적화, 동시성 제어 및 고장으로부터의 회복 등 다양한 기능을 계승받을 수 있게 된다. 그러나 이를 위해서는 복합 구조를 가지는 공간 객체를 정형적으로 표현하는 방법과 다차원의 공간 객체를 효율적으로 접근할 수 있는 색인 및 공간 연산자를 필수적으로 제공해야 한다[1,2].

지리정보시스템에서 빈번히 사용되는 중요한 공간 연산자로는 점 질의(point query), 영역 질의(range query), 공간 조인(spatial join) 등이 있다. 이 중에서 공간 조인은 두 개의 공간 객체 집합에 대하여 교차(intersection), 포함(containment) 등의 공간 관련성을 만족하는 객체 쌍들을 추출하는 연산으로서 가장 많은 계산량과 디스크 액세스를 필요로 한다. 공간 조인의 성능을 보장하기 위해서는 효율적인 색인 기법의 선택이 중요하며, 색인 기법으로는 R-tree[3,4], Grid File[5] 등 다차원 색인을 이용한 방법이 알려져 있다[6,7].

본 논문에서는 DOT(DOuble Transformation) 공간 색인[8,9]을 이용한 효율적인 공간 조인 처리 기법을 제시하고 이를 R^*-tree를 이용한 공간 조인 처리 기법과 비교한다. DOT 공간 색인 기법은 원 공간(initial space)에 존재하는 공간 객체의 최소 경계 사각형(Minimum Bounding Rectangle: MBR) 정보를 공간 순서화 곡선(space filling curve)을 사용하여 하나의 1차원 값으로 변환한 후 그 값을 검색 키(search key)로 갖는 B^+-tree 색인 구조를 구성하는 방법으로서, 이를 이용하면 전통적인 데이타베이스의 주 색인을 적용할 수 있다는 중요한 특징을 가진다. 또한 DOT 공간 색인을 주 색인으로 갖는 공간 객체의 집합은 공간상의 상호 인접성이 유지되도록 디스크 상에 클러스터링 되어 저장된다. 따라서 DOT 공간 색인을 기반으로 하는 공간 조인 알고리즘은 LRU 버퍼 방식을 사용하면 기존의 R^*-tree를 이용한 공간 조인 처리 기법에 비하여 디스크 액세스 면에서 유리한 장점이 있다[10].

그러나 DOT 색인을 이용한 공간 조인 알고리즘은 다차원 질의 영역을 1차원 값의 집합으로 변환하는 많은 수의 공간 변환 연산을 필요로 한다. 본 논문에서는 이 후부터 이 연산을 질의 영역 공간 변환 연산이라 부른다. 따라서 DOT 색인을 이용한 공간 조인 알고리즘은 R^*-tree를 이용한 공간 조인 처리 기법에 비하여 전체적인 성능이 만족스럽지 않다. 본 논문에서는 공간 객체의 MBR 정보를 하나의 1차원 값으로 변환하기 위해 사용되는 공간 순서화 곡선의 규칙성을 분석함으로써 공간 변환 연산의 횟수를 대폭 감소시킨 효율적인 DOT 색인 기반의 공간 조인 알고리즘을 제안한다. 제안된 알고리즘에서는 반복적으로 수행되는 공간 변환 연산의 횟수를 줄이기 위하여 질의 영역을 공간 순서화 곡선이 연속 운행하는 가능한 최대 크기의 면적으로 분할하는 쿼터 분할 기법을 사용한다. 우리들이 알기로는, DOT 색인 기법을 이용한 공간 조인 알고리즘은 아직 제안된 바 없다. 다양한 분포와 크기를 갖는 데이타 집합을 대상으로 R^*-tree를 이용한 공간 조인 처리 기법과의 비교 실험을 수행한 결과 최대 약 3배의 성능 이익을 얻을 수 있음을 확인할 수 있었다.

본 논문의 구성은 다음과 같다. 제2장에서는 공간 조인 연산과 관련된 기존의 연구들을 간략히 소개한다. 제3장에서는 DOT 색인을 이용한 영역 질의 방식에 대하여 논하며, 쿼터 분할 기법에 의한 질의 영역 공간 변환 연산의 최적화 방식을 제안한다. 제4장에서는 DOT 색인을 이용한 효율적인 공간 조인 처리 기법을 제안하고, 공간 조인 알고리즘을 보인다. 제5장에서는 제안된 알고리즘의 성능을 검증하기 위한 실험 결과를 기술하고, 마지막으로 제6장에서는 본 연구의 공헌을 요약하고 앞으로의 연구 방향을 제시한다.

2. 관련 연구

본 장에서는 공간 조인 연산에 관한 관련 연구 및 배경 기술에 대하여 요약한다. 먼저, 제 2.1절에서는 공간 조인 연산에 대한 기존 연구 결과를 요약, 설명하고, 제 2.2절에서는 DOT 색인 기법의 정보 추출 및 저장 방식에 대하여 간략히 설명한다.

2.1 공간 조인 연산

지리정보시스템에서 빈번히 사용되는 중요한 공간 연산자로는 점 질의, 영역 질의, 공간 조인 등이 있다[1]. 이러한 연산자는 이동 통신 기술과 결합하여 위치 기반 서비스(Location-Based Service: LBS)에 활용될 수 있다. 예를 들면 영역 질의를 수행하여 사용자 근처의 맛있는 음식점을 찾을 수 있으며, 공간 조인을 실행하여 가까운 위치에 있는 친구들의 쌍을 검색할 수 있다.

공간 연산자 중 점 질의나 영역 질의는 단일 검색 질의(single scan query)의 일종이며, 이를 위한 다수의 색인 기법[2]이 제안되어 있다. 공간 조인은 점 질의나 영역 질의와는 달리 다중 검색 질의(multiple scan query)의 일종이며, 질의 처리를 위해 모든 공간 객체를 반복적으로 액세스해야 하므로 가장 많은 계산량과 디스크 액세스를 필요로 한다.

공간 조인 연산은 여과와 정제의 두 단계로 나누어진다[6]. 여과 단계(filtering step)에서는 공간 객체의 MBR을 대상으로 조인을 수행하여 최종 결과로 남을 객체 쌍의 후보를 걸러내며, 정제 단계(refinement step)에서는 여과 단계에서 얻어서 후보 객체 쌍에 대하여 정밀한 기하 연산을 적용하여 최종 조인 결과를 산출한다. 정제 단계는 기하 연산의 복잡성으로 인하여 많은

양의 계산을 필요로 하지만 공간 조인 연산에 있어 고정 비용과 같은 성격을 가지고 있다. 따라서 전체 비용을 줄이기 위해서는 여과 단계의 비용을 줄여야 하며, 이를 위해서는 효율적인 색인 기법의 선택이 중요하다.

공간 객체의 MBR을 이용한 공간 색인 기법은 크게 세 가지로 분류할 수 있다. 첫 번째 방법은 k차원의 공간 객체를 2k차원의 점으로 변환하여 저장하는 방식으로 Grid-file[5], KDB-tree[11], LSD-tree[12] 등이 이에 속한다. 두 번째 방법은 공간 순서화 곡선을 사용하여 k 차원의 객체를 일련의 1차원 점으로 변환하는 방법으로써 z 순서 색인 기법[13,14], DOT 색인 기법[8,9] 등이 여기에 속한다. 세 번째 방법은 k차원의 공간을 분할하는 방식으로 R-tree[3], R^+-tree[15], R^*-tree[4], Cell-tree[16] 등을 예로 들 수 있다.

위의 세 가지 기법 중 다차원 공간을 분할하는 방식(즉, 세 번째 방법)이 현재까지는 가장 많이 사용되고 있으며, 이를 위치 기반 서비스에 적용하는 연구도 최근 활발히 진행되고 있다. 즉, 3DR-tree[17], HR-tree[17], STR-tree[18], TB-tree[19], MV3R-tree[20]와 같은 공간 색인 기법은 이동 객체의 위치 정보를 2차원의 공간 좌표로 표현하고, 이를 3차원 이상의 R-tree 색인에 저장한다. 또한, 실세계의 도로 네트워크를 이용하려는 최근의 연구 [21,22]는 객체의 이동 가능한 범위를 도로 위로 제한하여 이동 객체의 위치를 (가장 가까운 도로의 식별자, 가장 가까운 도로로부터의 거리)의 쌍으로 표현한 후, 이러한 도로 좌표에 적합한 색인을 제안하였다.

이러한 색인 기법을 이용한 공간 조인 알고리즘으로는 z 순서 색인 기법을 이용한 알고리즘[13,14], R^*-tree를 이용한 알고리즘[6], 구석점 변환 기법을 사용한 알고리즘[7] 등이 알려져 있다. 관련 연구 [13,14]에서는 k차원의 MBR 공간 객체를 겹치지 않는 일정한 크기의 조각으로 나누어 각각에 유일한 1차원 코드 값(z 순서 값)을 부여함으로써 일련의 1차원 점의 집합을 얻은 후, 이들 값으로부터 구축된 B-tree 색인을 이용하여 조인을 수행하는 직관적인 방식을 제안하였다. R^*-tree를 이용한 공간 조인 방식에서는 조인 대상의 두 공간 객체에 대하여 R^*-tree 색인을 구성한 후, 두 색인의 루트 노드로부터 비교를 시작하여, 비교 대상의 두 노드에 속한 엔트리의 MBR이 겹치면 그 자식 노드에 대하여 비교를 계속하는 방식으로서, 결과로 얻어진 리프 노드들의 MBR 쌍이 후보 객체 쌍이 된다. 관련 연구 [6]에서는 적절한 버퍼 크기가 유지되는 경우 각 데이타 페이지를 한번만 액세스하는 최적화된 공간 조인 알고리즘을 제시하고 있다. 또한 관련 연구 [7]의 구석점 변환 기법에서는 한 파일의 인접한 두 영역에 대한 상대방 파일의 두 조인 대상 영역이 서로 많은 공통 부분을 가진다는 특성을 활용하여 최적화된 공간 조인 기법을 제시하고 있다.

비교 연구 결과에 의하면, 이 들 중 R^*-tree를 이용한 공간 조인 알고리즘이 비교적 우수한 성능을 가지는 것으로 알려져 있다[6]. 한편, 구석점 변환 기법을 사용한 공간 조인 알고리즘[7]이 LRU 페이지 대치 알고리즘을 이용하는 경우 디스크 접근 면에서 R^*-tree 보다 우수한 성능을 보인다는 실험 결과가 보고되어, 변환 기법이 일반적으로 원 공간(initial space)에서의 객체 간 근접성을 유지시키지 못한다는 기존의 생각[23]에 대한 반설로 제시되었다. 그러나, R^*-tree를 이용한 방식과 구석점 변환 기법을 이용한 방식 모두 다차원 공간 색인 기법을 사용한 방식으로서 기존 데이타베이스 시스템의 주 색인 기법을 그대로 적용할 수는 없다는 한계를 가지고 있다.

2.2 DOT 공간 색인 기법

DOT 공간 색인 기법[8,9]은 공간 객체의 MBR 정보를 공간 순서화 곡선을 사용하여 하나의 1차원 값으로 변환한 후 그 값을 검색 키로 갖는 B^+-tree 색인 구조를 구성하는 방법이다. 단계적인 색인 생성 방식은 다음과 같다.

첫 번째 단계로서, k차원에 존재하는 공간 객체의 MBR 정보를 다음과 같이 두 번의 변환 과정에 의하여 1 차원 값으로 변환한다.

(1) k차원의 원공간(initial space)에 존재하는 공간 객체를 표현하는 MBR 정보를 2k차원의 중간 공간(intermediate space)상에 단편화 없이 하나의 점으로 사상한다. 이것을 1차 변환이라고 한다.

(2) 2k차원의 중간 공간에 하나의 점으로 사상된 공간 객체를 공간 순서화 곡선을 이용하여 1차원 상의 한 점을 나타내는 값으로 변환하고, 이 값을 x값(x-value)이라고 부른다. 이것을 2차 변환(공간 변환)이라고 한다.

여기에서 2차 변환에 적용되는 공간 순서화 곡선으로서 공간 객체 간 근접성을 최대한 보존하는 방식을 채택하는 것이 유리하며, 이는 원공간 상의 공간 객체를 상호 인접성이 유지되도록 디스크 상에 클러스터링 하여 저장하는 효과를 가진다[24-26]. 참고 논문 [8,9]에서는 2차원 중간 공간을 위한 공간 순서화 곡선으로서 tri-Hilbert 곡선을 사용하고 있다.

두 번째 단계로서 각 공간 객체의 x값을 이용하여 이 값을 검색 키로 갖는 B^+-tree 구조의 색인을 구성하며, 이와 같이 얻어진 B^+-tree 색인 구조를 DOT 공간 색인이라 부른다. DOT 공간 색인은 이와 같이 기본적으로 B^+-tree 색인 구조를 이용한다. 따라서 공간 객체의 동적 삽입이나 삭제 과정은 두 번의 공간 변환 함수(1

차 변환, 2차 변환)를 선행으로 호출하는 과정을 가지는 것이 차이가 있을 뿐, 본질적으로 B^+-tree 색인 구조에서의 동적 삽입이나 삭제 과정과 동일하다. 따라서 공간 객체들의 키 값이 동일한 영역에 편중되어 삽입/삭제된다면, B^+-tree 색인 구조의 인덱스 노드 분할/합병에 따르는 오버 헤드를 예상할 수 있다.

3. DOT 색인을 이용한 영역 질의 기법

본 장에서는 공간 조인 처리를 위하여 반복적으로 수행되는 영역 질의 알고리즘의 성능 을 보장하기 위하여 공간 변환 연산 비용을 줄이는 쿼터 분할 기법을 제안하고, 이 쿼터 분할 기법을 적용한 영역 질의 알고리즘을 제시한다. 제3.1절에서는 DOT 색인을 이용한 기본적인 영역 질의 수행 방식 및 알고리즘을 보이고 그 문제점을 지적한다. 제3.2절에서는 질의 영역을 1차원 값의 집합으로 변환하는 질의 영역 공간 변환 연산의 연산 비용을 줄이기 위한 쿼터 분할 기법을 제안한다. 또한 쿼터 분할 기법을 적용한 영역 질의 알고리즘을 보이고, 영역 질의 알고리즘의 성능 개선 효과를 정량적으로 분석한다. 마지막으로 제3.3절에서는 삼각형 쿼터(Tri-Quarter) 분할 기법을 이용한 추가적인 성능 개선 방식을 제안한다.

3.1 영역 질의 알고리즘

영역 질의는 주어진 질의 영역과 교차하는 모든 공간 객체를 검색하는 연산이다. DOT 공간 색인을 이용한 영역 질의 과정을 그림 1의 예를 들어 간단히 설명하면 다음과 같다. 그림 1(a)는 1차원의 원 공간에서 공간 객체 A와 B 그리고 질의 영역을 나타내는 q를 표현하고 있다. 다음의 그림 1(b)는 1차 변환 결과에 의한 공간 객체와 질의 영역을 나타낸다. 1차원 원 공간 상에 존재하는 각 공간 객체의 MBR 정보는 객체의 시작점이 X_s 축의 좌표로, 끝점이 X_e축의 좌표로 이용되어 중간 공간인 2차원 공간상의 하나의 점으로 표현되며, 공간 객체 A와 B는 각각 (1,3), (5,6)의 점으로 사상된다. 이 때 1차원 객체의 끝점은 항상 시작점보다 크거나 같은 값을 가지므로 모든 공간 객체는 그림 1(b)와 같이 중간 공간의 대각선을 포함한 위쪽 직각 삼각형에 사상되는 것을 알 수 있다.

또한 DOT 공간 색인 기법에서 질의 영역은 원 공간이 1차원인 경우 중간 공간상에 면적으로 표현된다. 시작점을 q_s로, 끝점을 q_e로 갖는 질의 영역 q는 $X_s \le q_e$와 $X_e \ge q_s$의 조건을 만족하는 영역으로 표현되며 그림 1(b)의 음영 처리된 부분이 그 영역을 나타내고 있다. 이 때 공간 객체가 존재하지 않는 대각선 아래 부분

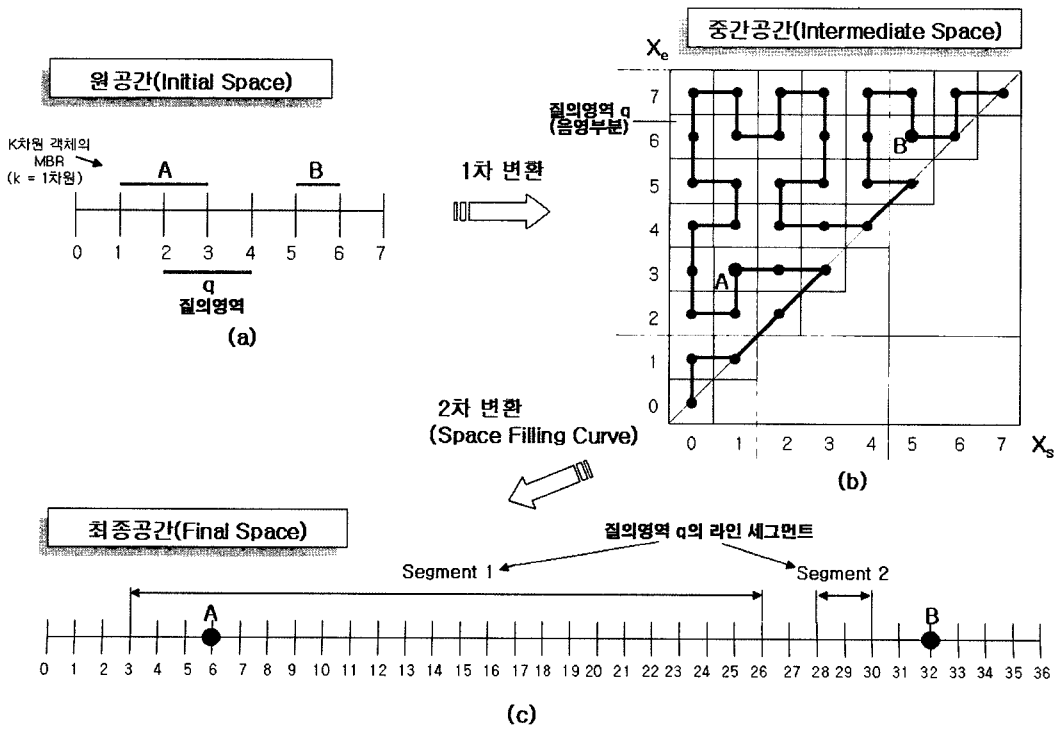


그림 1 DOT 색인 기법의 영역 질의 과정

은 사용되지 않음을 알 수 있다.

다음의 그림 1(c)는 2차 변환(공간 변환)에 의하여 최종 공간인 1차원 공간에 사상된 각 공간 객체와 질의 영역을 나타낸다. 그림 1(b)의 2차원 중간 공간에 대하여 tri-Hilbert 순서화 곡선[8]을 적용하여 공간 객체 및 질의 영역에 대한 1차원의 x값을 추출하면, 공간 객체는 각각 1차원 공간상의 하나의 값(A는 6, B는 32의 x값을 가짐)으로 변환되며, 질의 영역 q는 그림 1(c)에서 segment1과 segment2로 표현된 1차원 공간상의 일련의 점의 집합인 라인 세그먼트의 집합([3~26]과 [28~30]사이의 x값을 가짐)으로 변환된다.

DOT 색인 기법에서는 이와 같이 얻어진 각 공간 객체의 x값을 이용하여, 그 값을 검색 키로 갖는 B^+-tree 구조의 색인을 구성한다. 따라서 DOT 공간 색인을 이용한 영역 질의는 질의 영역 q의 라인 세그먼트의 집합이 나타내는 범위의 검색키를 가지는 공간 객체를 B^+-tree 구조의 DOT 색인 상에서 찾아내는 과정으로 설명될 수 있다. 즉 그림 1의 영역 질의는 DOT 색인 상에서 탐색 키의 값이 3에서 26사이 또는 28에서 30사이의 객체를 찾는 것에 해당하며, 결국 탐색 키의 값이 6인 객체 A가 검색된다.

DOT 색인을 이용한 영역 질의 처리는 알고리즘 1과 같다. 알고리즘 Range-Query는 원 공간의 질의 영역인 (q_s, q_e)와 중간 공간의 한 변의 크기(Grid Size) n을 입력으로 받아 다음과 같이 영역 질의를 수행한다. 우선, 함수 QueryRangeToLineSegments()를 호출하여 질의 영역을 1차원 상의 라인 세그먼트들로 변환하고, 이들 값을 LS에 저장한다(line 1). 여기서 LS는 시작점과 끝점으로 표현되는 라인 세그먼트들의 집합을 저장하는 구조체 변수이다. 다음, 함수 GetObjectsUsingBTree-Index()를 호출하여 LS에 해당하는 x값을 가지는 공간 객체를 B^+-tree 구조의 DOT 색인을 이용하여 검색한 후, 그 결과를 QR에 저장하여 반환한다(lines 2~3).

함수 QueryRangeToLineSegments()는 질의 영역에 해당하는 중간 공간상의 점들을 1차원 x값으로 변환하여 라인 세그먼트의 집합 LS를 구성한다. 여기에서 사용된 함수 TriHilbert()는 tri-Hilbert 공간 순서화 곡선의 운행 경로에 따라 중간 공간의 점을 최종 공간의 값으로 변환하는 공간 변환 연산 함수이다. 또한 ConcatenateAdjacentXvalue(SORT(LS))는 라인 세그먼트를 구성하는 함수로서 질의 영역의 모든 점들에 대하여 x값으로 변환된 결과를 인접한 점들이 연속되는 구간별로 구분하고 시점과 종점을 판단하여 라인 세그먼트를 구성하는 처리를 수행한다. GetObjectsUsing-BTreeIndex()는 기존의 B^+-Tree 색인을 이용한 영역 질의 과정을 나타내므로 자세한 기술을 생략한다.

알고리즘 1: DOT 영역 질의 알고리즘 Range-Query

Input : Query Start q_s, Query End q_e, Grid Size n
Output : Set of Results QR

```
1. LS = QueryRangeToLineSegments(qs, qe, n);
2. QR = GetObjectsUsingBTreeIndex(LS);
3. Return QR;
```

알고리즘 2: 공간변환 알고리즘 QueryRangeToLineSegments

Input : Query Start q_s, Query End q_e, Grid Size n
Output : Set of Line Segments LS

```
1. LS = Φ;
2. for(Xs = 0; Xs <= n-1; Xs++)
3.     for(Xe = Xs; Xe <= n-1; Xe++)
4.         if(Xs <= qe and Xe >= qs)
5.             LS = LS ∪ TriHilbert(Xs, Xe);
6. LS = ConcatenateAdjacentXvalue(SORT(LS));
7. Return LS;
```

DOT 영역 질의 알고리즘이 좋은 성능을 갖기 위해서는 질의 영역을 최종 공간상의 라인 세그먼트의 집합으로 변환하는 함수 QueryRangeToLineSegments()의 연산 비용을 최소화 시켜야 한다[알고리즘 2]. 그 이유는 질의 영역의 크기에 비례하여 공간 변환 연산을 수행하는 함수 TriHilbert()의 실행 횟수가 증가하기 때문이다. 일반적으로 질의 영역은 중간 공간상에 면적으로 표현되는데, 공간 순서화 곡선의 운행은 이 면적 내에서 반드시 연속적이라 단정할 수는 없다. 따라서 이 면적에 해당하는 모든 점들에 대해 공간 변환 연산을 적용해야만 최종 공간에서 연속적인 라인 세그먼트의 집합을 구성할 수 있다.

질의 영역 (q_s, q_e)의 q_s를 X_s축의 좌표로, q_e를 X_e축의 좌표로 이용하여 중간 공간의 한 점으로 사상한 점을 q라 하였을 때, 이 점 q의 위치에 따라 질의 영역의 크기가 달라진다. 예를 들어 원 공간 전체가 질의 영역인 경우 점 q는 중간 공간의 좌상점(upper left)에 위치하게 되며 중간 공간 전체가 질의 영역이 된다. 이때 중간 공간의 한 변의 길이가 n이라면 $(n^2+n)/2$번의 공간 변환 연산이 필요하게 되며, 예를 들어 n=1024의 경우에는 524,800번의 공간 변환 연산이 필요하다. 질의 영역에 해당하는 면적내의 모든 점들에 대해 공간 변환 연산을 적용하는 경우 점 q의 위치에 따른 변환 연산의 횟수는 다음과 정리된다.

(1) 점 q가 중간 공간의 좌상점에 위치할 때 질의 영역의 면적이 가장 크며, 그 면적은 중간 공간 면적의 절반이 된다(대각선 위쪽 삼각형의 면적). 이 경우 질의 영역에 속하는 점들의 개수는 $(n^2+n)/2$개이며, 공간 변환 연산 횟수도 $(n^2+n)/2$번이 된다.

(2) 점 q가 중간 공간의 우상점(upper right)이나 좌하점(lower left)에 위치하는 경우는 원 공간의 시점과 종점 중 한 점을 질의하는 경우로서 질의 영역 면적이 가장 작으며, 그 면적은 중간 공간의 한 변의 길이와 같다. 이 경우 질의 영역에 속하는 점들의 개수는 n이며, n번의 공간 변환 연산이 필요하다.

따라서 DOT 영역 질의 수행을 위한 공간 변환 연산의 횟수를 Nt라고 정의하면 Nt는 $[n \leq Nt \leq (n^2+n)/2]$의 크기를 가지며, 이는 최악의 경우 $O(n^2)$에 해당하므로 DOT 영역 질의 알고리즘의 성능을 저하시키는 중요한 요인으로 작용하게 된다.

3.2 쿼터 분할 기법을 이용한 질의 영역 공간 변환 연산

본 절에서는 공간 변환 연산 횟수를 최소화하여 DOT 영역 질의를 효율적으로 처리할 수 있는 쿼터 분할 기법을 제안한다. 질의 영역의 모든 점들에 대하여 공간 변환 연산을 적용하는 대신, 질의 영역을 공간 순서화 곡선이 연속 운행되는 부분 영역의 집합으로 분할할 수 있다면 각 분할된 영역의 크기를 이용하여 공간 변환 연산의 비용을 절감할 수 있다. 쿼터 분할 기법은 한 변의 길이가 n인 중간 공간을 한 변의 길이가 n/2인 정방사각형의 쿼터로 재귀적으로 분할하며, 공간 순서화 곡선의 운행 경로를 분석하여 곡선의 운행이 연속되는 쿼터가 질의 영역에 포함되는지를 판별하여, 질의 영역에 포함되는 경우에는 그 시작점에 대하여 쿼터 당 1회의 공간 변환 연산을 적용하여 최종 공간에서의 라인 세그먼트를 찾아내는 방법이다.

공간 순서화 곡선으로 적용된 Hilbert 곡선은 중간 공간을 재귀적인 규칙을 가지고 운행한다[25,26]. 그림 2에 각 차수(Order)에 따른 Hilbert 곡선의 운행 방식을 보인다. 그림과 같이 Hilbert 곡선은 크기가 n인 중간 공간을 한 변의 길이가 n/2인 쿼터로 재귀적으로 분할하였을 때 각 쿼터 내에서 연속적인 값을 갖도록 운행하는 성질을 가지고 있다. 따라서 이러한 성질을 이용하면 분할되는 쿼터로부터 최종 공간에서의 연속적인 x값의 범위를 예측할 수 있다. 이와 함께 고려되어야 할 것은 Hilbert 곡선이 분할된 쿼터의 어느 점에서 시작되는지를 판단하는 것이다. 이는 각 쿼터 내에서의 곡선의 시작점의 x값을 알게 되면 라인 세그먼트의 범위는 시작점의 x값에 쿼터의 한 변의 길이의 제곱을 합하고 1을 빼는 것으로 쉽게 계산될 수 있기 때문이다.

Hilbert 곡선의 운행 규칙은 다음과 같이 요약될 수 있다. 여기에서는 쿼터의 각 사분면을 그림 3(a)와 같이 오른쪽 위의 사분면부터 시계 방향으로 각각 1 사분면, 2 사분면, 3 사분면, 4 사분면으로 부른다.

(1) Hilbert 곡선은 차수(Order)가 1인 경우, 중간 공간을 3-4-1-2 사분면의 순서로 운행한다. 이것을 Pattern3412의 패턴이라고 부른다. 즉, 중간 공간은 처음에 Order=1인 쿼터로 간주되며 운행 순서는 ⌐와 같이 표현된다. 차수(Order)가 2인 경우, 분할된 각 쿼터에서의 Hilbert 곡선의 운행 패턴은 그림 3(b)와 같다.

(2) 쿼터의 재귀적인 분할에 대한 Hilbert 곡선의 운행 패턴은 표 1과 같이 일정한 규칙을 가진다. Hilbert 곡선은 Pattern3412, Pattern3214, Pattern1432, Pattern1234의 4가지 패턴으로 이루어져 있으며, 쿼터가 분할됨에 따라 각 사분면의 쿼터는 각각 표 1과 같이 정해진 운행 패턴을 가지게 된다. 표 1을 이용

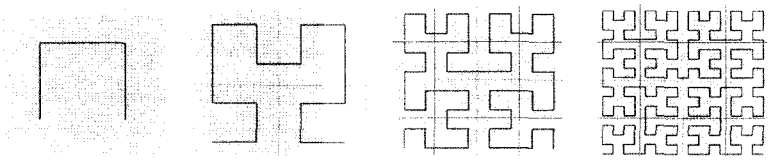


그림 2 차수(Order)의 증가에 따른 쿼터 분할과 Hilbert 곡선의 운행 방식

표 1 Hilbert 곡선의 운행 규칙

Order	Order + 1			
	1 사분면 쿼터	2 사분면 쿼터	3 사분면 쿼터	4 사분면 쿼터
Pattern3412	Pattern3412	Pattern1432	Pattern3214	Pattern3412
Pattern3214	Pattern3214	Pattern3214	Pattern3412	Pattern1234
Pattern1432	Pattern1234	Pattern3412	Pattern1432	Pattern1432
Pattern1234	Pattern1432	Pattern1234	Pattern1234	Pattern3214

하여 상태천이도(그림 4)를 구성하면 쿼터의 다음 분할에 따른 각 사분면 쿼터에서의 곡선의 운행 패턴과 시작점을 쉽게 찾을 수 있다. Pattern3412, Pattern3214의 운행 패턴을 갖는 쿼터는 Hilbert 곡선의 시작점이 쿼터의 좌하점(lower left)이 되며, Pattern1432, Pattern1234의 운행 패턴을 갖는 쿼터는 Hilbert 곡선의 시작점이 쿼터의 우상점(upper right)이 된다.

다음의 알고리즘 3은 쿼터 분할 기법을 적용한 질의 영역의 공간 변환 알고리즘을 나타낸다. 이 알고리즘에서 사용되는 변수 Q와 Q_{sub}는 쿼터를 표현하는 구조체 변수이다. 쿼터 변수는 쿼터의 크기를 나타내는 size 필드, 중간 공간상에서 쿼터의 위치를 판별하기 위하여 쿼터의 좌상점에 해당하는 점의 위치를 나타내는 Upper-Left 필드, 그리고 Hilbert곡선의 운행 패턴을 판별하기 위한 Pattern 필드로 구성되어 있다. 우선, InitializeQuarter()는 질의 영역을 쿼터로 분할하기 위하여 중간 공간의 크기를 이용하여 처음으로 비교하게 될 쿼터 Q의 size 및 좌상점의 위치 정보를 초기화 하는 작업을 수행하며, Hilbert 곡선의 초기 운행 방향인 Pattern3412를 초기 운행 패턴으로 지정한다(line 2). 다음으로, 함수 SplitQueryRangeByQuarter()는 전달된 쿼터 Q가 질의 영역에 속하는 경우에는 쿼터의 운행 패턴을 기준으로 공간 순서화 곡선의 시작 위치를 찾아(line 2의 Start-Point함수 이용) 단 한 번의 공간 변환 연산으로 쿼터 크기만큼의 연속된 라인 세그먼트를 반환한다. 이때 쿼터의 좌상점에 해당하는 좌표 값 upperleft는 함수 StartPoint()에서 시작점의 위치를 계산하는데 이용된다. 구조체 변수 LS_{new}는 이렇게 계산된 라인세그먼트를 저장하는 변수이다. 한편, 전달된 쿼터가 질의 영역에 속하지 않는 경우에는 이것을 다시 4개의 서브 쿼터로 나누어 각 서브 쿼터에 대하여 공간 순서화 곡선의 운행 경로와 서브 쿼터의 위치를 설정하여 재귀적으로 SplitQueryRangeByQuarter()를 호출한다(line 10). 함수 IsInRegion(Q, q_s, q_e)는 분할된 쿼터 Q가 (q_s, q_e)로 설정된 질의 영역에 속하는 지를 판별하는 함수로서, 분할된 쿼터가 중간 공간의 대각선 위쪽에 위치하면서 설정된 질의 영역에 완전히 포함되는 경우 TRUE 값을 반환하며 자세한 기술은 생략하였다.

알고리즘 3: 쿼터 분할을 이용한 공간 변환 알고리즘 QueryRangeToLineSegments

Input : Query Start q_s, Query End q_e, Grid Size n
Output : Set of Line Segments LS

1. LS = Φ;
2. Q = InitializeQuarter(q_s, q_e, n);
3. SplitQueryRangeByQuarter(Q, q_s, q_e, LS);
4. Return LS;

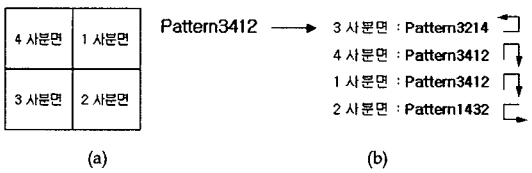


그림 3 쿼터의 사분면 정의와 Hilbert 곡선의 초기 운행 방식

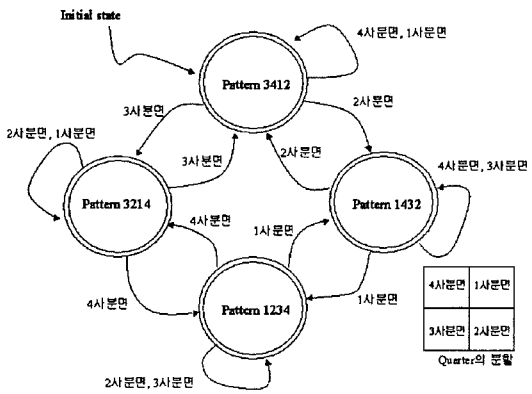


그림 4 Hilbert 곡선의 운행 규칙을 나타낸 상태천이도

알고리즘 3의 SplitQueryRangeByQuarter는 분할된 쿼터마다 한 번의 공간 변환 연산을 수행한다. 즉, 쿼터 분할 기법을 적용한 공간 변환 연산의 횟수는 분할되는 쿼터의 개수와 같으며, 따라서 분할되는 쿼터의 수가 알고리즘의 성능을 좌우하는 중요한 요소가 된다. 중간 공간의 한 변의 크기를 n이라고 하였을 때, 분할되는 쿼터의 개수는 질의 영역을 중간 공간에 사상한 점 q에 의한 질의 영역의 범위에 따라 다르나 다음과 같이 추정할 수 있다.

질의 영역이 가장 큰 경우인 중간 공간의 위쪽 삼각형 전체를 고려하였을 때, 쿼터의 분할은 중간 공간의 대각선 상의 점들을 이진 탐색(binary search)의 순서로 검색하면서 각 검색된 점에서 X축(삼각형의 윗변)과 Y축(삼각형의 좌측변)에 수선을 내려 얻어진 면적으로 분할하는 것과 일치한다(그림 5). 이렇게 얻어진 쿼터의 개수는 대각선에 걸치지 않고 질의 영역에 완전히 포함되는 쿼터의 개수로 n-1개가 된다. 또한 대각선에 걸친 n개의 쿼터의 개수를 고려하면, 결국 대각선을 이진 탐색하면서 얻어진 쿼터의 수 n-1개와 대각선 상의 크기가 1인 쿼터들의 수 n개를 합한 2n-1이 분할되는 쿼터의 수가 된다. 따라서 알고리즘 3-1의 공간 변환 연산

알고리즘 3-1: 쿼터를 이용한 질의 영역 분할 알고리즘 SplitQueryRangeByQuarter

Input : Quarter Q, Query Start q_s, Query End q_e, Set of Line Segments LS
Output : Set of Line Segments LS

```
1. if(IsInRegion(Q, qs, qe))
2.     LSnew.Start = TriHilbert(StartPoint(Q));
3.     LSnew.End = LSnew.Start + Q.size^2 - 1;
4.     LS = LS ∪ LSnew;
5. else if(Q.size > 1)
6.     Qsub.size = Q.size / 2;
7.     for(SubQuarter = 1; SubQuarter <= 4; SubQuarter++)
8.         Qsub.pattern = nextpattern(Q, SubQuarter);
9.         Qsub.upperleft = nextupperleft(Q, SubQuarter);
10.        SplitQueryRangeByQuarter(Qsub, qs, qe, LS);
11. Return LS;
```

의 횟수 Nt는 $n \leq Nt \leq 2n-1$로 알고리즘 1에 비하여 향상된 성능을 보인다. 즉 최악의 경우, 공간 변환 연산의 복잡도가 $O(n^2)$에서 $O(n)$으로 개선될 수 있음을 알 수 있다.

3.3 삼각형 쿼터(Tri-Quarter) 분할을 이용한 추가 성능 개선

제3.2절에서 보인 쿼터 분할 기법에서는 질의 영역에 대하여 정방사각형의 쿼터 분할을 재귀적으로 수행함으로써 공간 변환 연산의 비용을 절감시킨다. DOT 색인 기법에서는 모든 공간 객체와 질의 영역이 대각선 위쪽 삼각형 부분에만 적용되므로 공간 순서화 곡선도 그림 1의 경우처럼 대각선 위쪽 삼각형만을 운행하는 tri-Hilbert 곡선을 사용할 수 있다. 따라서 쿼터의 분할 과정에 있어 대각선에 걸쳐지는 쿼터는 반드시 사각형이 아니라 대각선을 빗변으로 하는 직각삼각형의 경우도 연속되는 쿼터라고 가정할 수 있다. 이 경우, 단 1회의 공간 변환 연산에 의하여 삼각형의 면적으로 라인 세그먼트를 계산한다면 그림 5의 경우처럼 질의 영역의 대각선 경계 부분에서 보다 작은 크기의 쿼터(최소크기 = 1)로 분할되는 경우를 최적화 할 수 있어 추가로 연산 비용을 절감할 수 있게 된다.

그림 6은 제 3.2절에 보인 쿼터 분할 기법을 사용한 질의 영역의 쿼터 분할 예를 보인다. 다음의 그림 7은 같은 질의 영역에 대하여 삼각형 쿼터 분할 방식을 추가적으로 적용한 예로서, 삼각형 쿼터로 분할하였을 때 굵은 선으로 표시된 영역들이 각각 하나의 삼각형 쿼터로 간주되어 모두 14회의 공간 변환 연산 횟수를 절감할 수 있음을 나타내고 있다. 삼각형 쿼터 분할 기법을 그림 5의 경우에 적용하면 전체 질의 영역은 하나의 삼

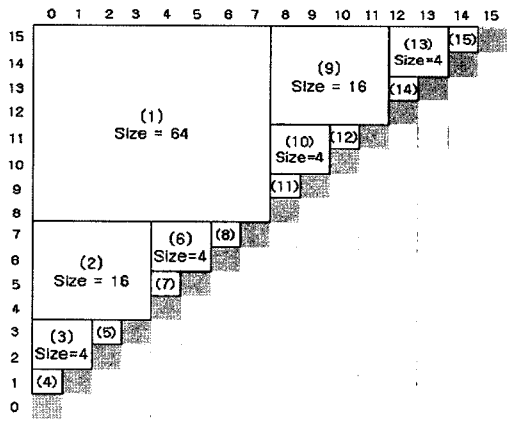


그림 5 대각선의 이진 탐색으로 분할되는 쿼터 (n=16의 경우)

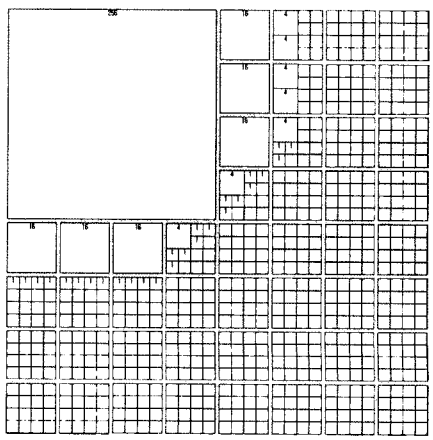


그림 6 임의의 질의 영역을 쿼터 분할한 예제 (n=32의 경우)

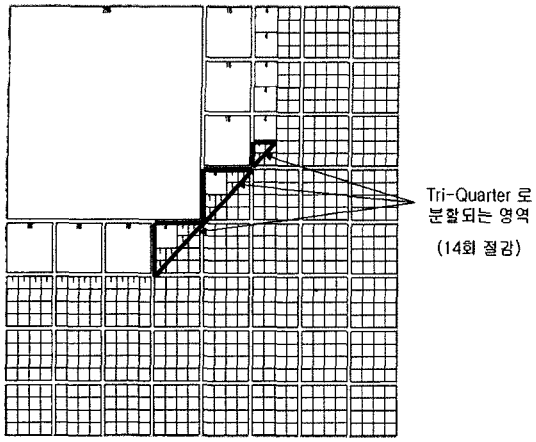


그림 7 삼각형 쿼터로 분할하였을 때 추가적인 성능 개선 효과

각형 쿼터로 인식되어 단 1회의 공간 변환 연산에 의하여 라인 세그먼트를 구할 수 있다.

다음의 알고리즘 3-2는 삼각형 쿼터 분할을 적용하여 알고리즘 3-1을 추가로 개선한 알고리즘이다. 이때 쿼터를 표현하는 변수의 구조에 삼각형 쿼터인지 여부를 기억하는 새로운 필드 TriQuarter를 추가하고, 함수 IsInRegion(Q, q_s, q_e)은 삼각형 쿼터인 경우 TriQuarter 필드에 TRUE를 반환하도록 수정되었다. 또한 삼각형 쿼터의 라인 세그먼트를 계산하는 방법은 ($Q.size^2$ + Q.size)/2 - 1과 같다(line 4).

4. DOT 색인을 이용한 공간 조인 기법

본 장에서는 DOT 색인을 이용한 효율적인 공간 조인 알고리즘을 제안한다. 제 4.1절에서는 직관적 이해를 돕기 위하여 예를 이용한 공간 조인 방식을 설명하고, 그 성능 개선 방식을 제안한다. 제 4.2절에서는 DOT 색인을 이용한 공간 조인 알고리즘을 보인다.

4.1 공간 조인 방법

공간 조인 연산은 두 개의 공간 객체 집합에 대한 것으로서 영역 질의나 점 질의의 경우와는 달리 질의 영역이 고정되지 않는다. 두 개의 파일에 대하여 공간 조인 연산을 행하는 경우, 색인이 존재하지 않는다면 하나의 파일에 있는 각각의 공간 객체에 대하여 다른 파일에 있는 모든 공간 객체를 비교해야 하므로 그 처리 비용이 매우 높아지게 된다.

DOT 색인을 가지고 있는 두 개의 파일 R과 S가 있다고 가정하면 R과 S의 공간 조인 연산은 제 3장에서 설명한 영역 질의 과정을 확장 적용함으로써 얻어질 수 있다. 즉, 파일 R을 구성하는 모든 공간 객체들에 대해서, 각 공간 객체의 MBR을 질의 영역으로 하여 파일 S에 존재하는 모든 공간 객체들을 검색하는 영역 질의를 반복하는 처리라고 생각할 수 있다.

이와 같은 공간 조인 과정을 그림 8의 예를 들어 설명하면 다음과 같다. 그림 8(a)는 1차원 원 공간에서 조인의 대상이 되는 R과 S를 나타낸 것이며, R과 S는 동일 파일로서 a, b, c, d의 4개의 공간 객체를 포함한다.

알고리즘 3-2: 삼각형 쿼터를 이용한 질의 영역 분할 알고리즘 SplitQueryRangeByQuarter

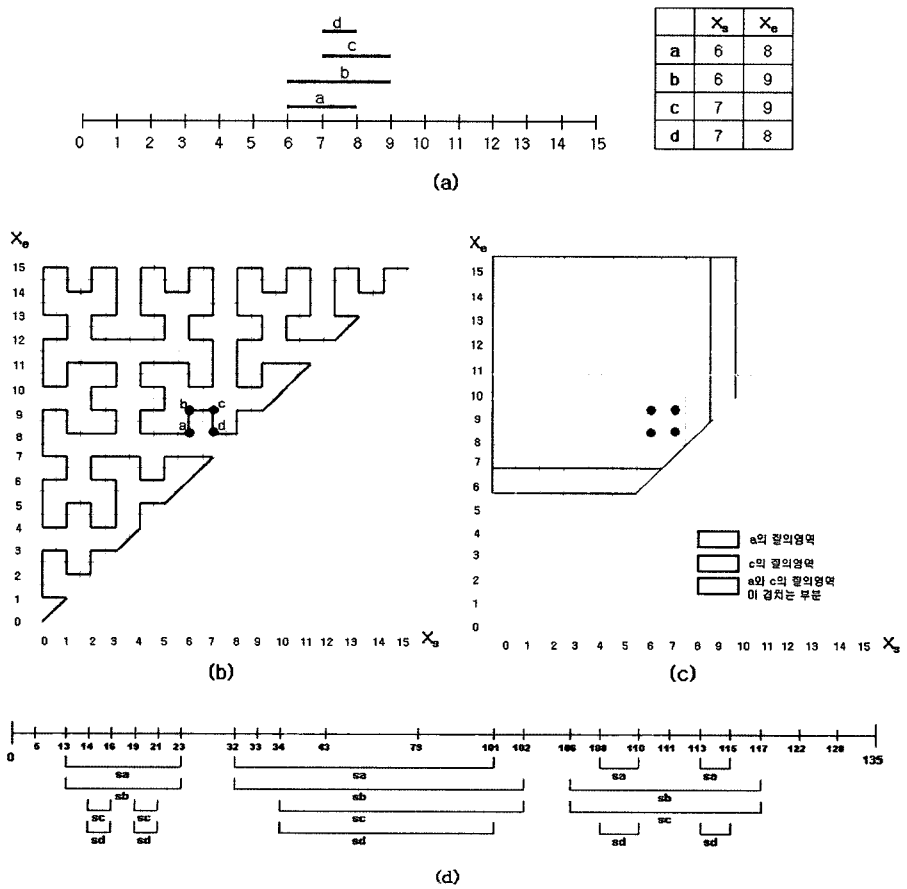
Input : Quarter Q, Query Start q_s, Query End q_e, Set of Line Segments LS
Output : Set of Line Segments LS

```
1.  if(IsInRegion(Q, qs, qe))
2.  |  LSnew.Start = TriHilbert(StartPoint(Q));
3.  |  if(Q.TriQuarter = TRUE)
4.  |  |  LSnew.End = LSnew.Start + (Q.size^2 +Q.size) /2 - 1;
5.  |  else
6.  |  |  LSnew.End = LSnew.Start + Q.size^2 - 1;
7.  |  LS = LS ∪ LSnew;
8.  else if(Q.size > 1)
9.  |  Qsub.size = Q.size / 2;
10. |  for(SubQuarter = 1; SubQuarter <= 4; SubQuarter++)
11. |  |  Qsub.pattern = nextpattern(Q, SubQuarter);
12. |  |  Qsub.upperleft = nextupperleft(Q, SubQuarter);
13. |  |  SplitQueryRangeByQuarter(Qsub, qs, qe, LS);
14. Return LS;
```

이들 객체는 1차 변환에 의하여 그림 8(b)와 같이 2차원 중간 공간상에 점으로 표현된다. 이 경우 파일 R과 S의 조인 연산은 파일 R의 각 a, b, c, d 객체에 대하여 해당 MBR을 질의 영역으로 하는 영역 질의를 파일 S에 반복 적용하는 것과 같으며, 최종 조인 결과는 각 영역 질의의 결과를 모두 합한 것과 같다. 예를 들어 R의 중간 공간상에서 (6,8)의 좌표를 갖는 점으로 표현되는 객체 a는 S의 중간 공간상에서 $X_s \leq 8$ and $X_e \geq 6$를 만족하는 2차원 공간상의 대각선 윗부분의 음영으로 표시된 부분에 해당하는 질의 영역을 갖는다(그림 8(c)). 객체 b, c, d도 유사한 질의 영역을 가지며 이러한 중간 공간상의 질의 영역을 2차 변환에 의하여 최종 1차원 공간상에 표현하면 그림 8(d)와 같은 결과를 얻게 된다. 여기에서 sa, sb, sc, sd는 각각 객체 a, b, c, d의 질의 영역을 의미한다. 다음, 각 질의 영역이 갖는 x값을 이용하여 파일 S상의 객체를 검색하게 되면 파일 R상의 객체 a, b, c, d는 모두 파일 S상의 객체 a, b, c, d와 겹쳐져 16개의 객체 쌍의 집합을 얻게 된다.

이때, 예제로 사용된 파일 R상의 객체들은 원 공간상에서 서로 인접하게 위치하고 있으므로, 1차 변환 후 중간 공간상에서 인접한 점들로 변환된다. 이 점들은 2차 변환 후 디스크상에 인접한 위치에 저장되게 된다. 그 이유는 2차 변환에 사용되는 공간 순서화 곡선이 인접성을 최대한 유지하는 곡선이기 때문에 중간 공간에서 인접한 공간 객체는 서로 인접한 x값을 가지게 되며, 결과적으로 B^+-tree의 특징상 동일 클러스터에 저장될 가능성이 높아진다. 특히 파일 R상의 객체 b와 같이 시작점이 가장 작고, 끝점이 가장 큰 객체의 질의 영역은 나머지 객체의 질의 영역을 모두 포함하고 있음을 알 수 있다. 본 논문에서는 이와 같은 객체를 대표 객체라 부른다.

이와 같은 성질을 이용하여 파일 S의 접근 횟수를 줄일 수 있는 방법을 생각할 수 있다. 즉, 파일 R상의 각 객체에 대한 영역 질의를 행하는 대신 다수 객체의 대표 객체에 대하여 질의 영역을 설정하여 영역 질의를 행하게 되면, 각각의 객체들의 질의 영역을 모두 포함한



	X_s	X_e
a	6	8
b	6	9
c	7	9
d	7	8

그림 8 질의 영역들 사이의 상호 관계

영역 질의를 수행하게 된다. 그 후에 실제적인 객체 간 겹침 관계를 검사하는 방식을 채택하면 파일 S를 최소한으로 접근하면서 조인 처리를 수행할 수 있다. 여기서 대표 객체의 질의 영역에 의해 정해지는 최종 공간상의 라인 세그먼트의 집합을 조인 후보 영역이라 부른다. 본 연구에서는 각 데이타 페이지 당 1개의 대표 객체를 설정하는 것으로 가정한다.

다음의 그림 9는 DOT 색인 기법을 이용한 공간 조인 방법을 단계적으로 표현한 것이다. 그림 9(a)는 1차원 원 공간에서의 동일한 파일 R과 S를 나타낸 것으로 각각 A에서 L까지 12개의 공간 객체를 포함한다. 이들 객체는 1차 변환에 의하여 그림 9(b)와 같이 중간 공간상의 점으로 표현된다. 그림 9(c)는 R과 S의 DOT 색인 구조를 나타내며, 그림 9(d)는 DOT 색인의 각 데이타 페이지들의 대표 객체 설정에 의한 조인 후보 영역을 나타낸다. 여기에서 데이타 페이지의 대표 객체 값은 동일 페이지에 존재하는 각 데이타 항목의 최소 X_s값과 최대 X_e값을 이용하여 설정한다.

그림 9(d)로부터 각 페이지에 대한 조인 후보 영역들이 역시 상당히 겹치는 것을 볼 수 있다. 예를 들어 그림 9(b)에서 첫 번째 페이지의 공간 조인 후보 영역은 ①, ②, ③ 영역으로 표현되며, 두 번째 페이지의 공간 조인 후보 영역은 ②, ③, ④ 영역으로 표현된다. 즉 ②, ③의 영역이 겹치는 부분이 된다.

이와 같이 연속적으로 접근되어야 하는 공간 조인 후보 영역이 반복적으로 중첩되는 성질은 전체적인 공간 조인 알고리즘의 성능 개선에 활용될 수 있다. 즉 LRU 버퍼링 방식을 사용하여 공간 조인 연산을 수행하면 파일 S의 디스크 접근 횟수를 획기적으로 줄이는 효과를 얻게 된다.

4.2 DOT 공간 조인 알고리즘

알고리즘 4는 DOT 색인을 이용하는 공간 조인 알고리즘이다. 파일 R과 S는 DOT 색인을 갖는 1차원 객체의 파일이라고 가정하며, n은 중간 공간의 한 변의 크기(Grid Size)를 나타낸다. 버퍼 관리 기법으로는 일반적으로 널리 이용되는 LRU 방식을 사용하고, 공간 조인 연산을 위한 공간 관련성은 교차(intersection) 관계만을 고려한다.

이 알고리즘의 수행과정을 간단히 설명하면 다음과 같다. 외부의 for문은 파일 R의 데이타 페이지 수만큼 반복된다(line 2). for문 내부의 처리는 다음과 같다. 파일 R로부터 해당 데이타 페이지를 읽어 들인 후(line

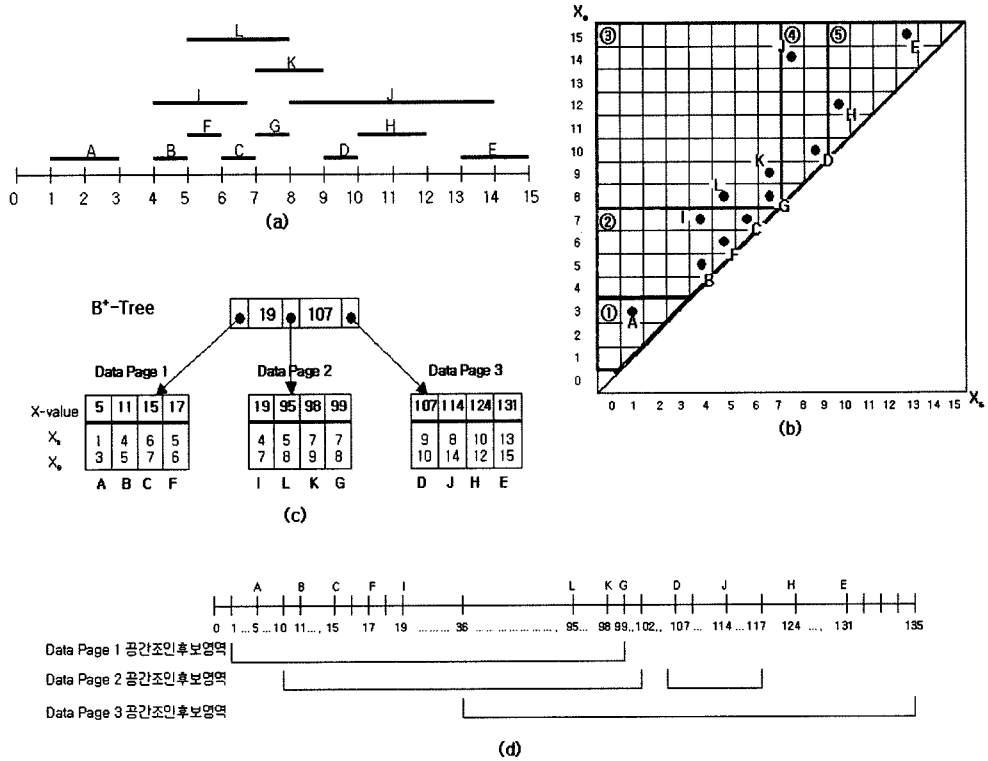


그림 9 DOT 색인을 이용한 공간 조인

3), 각각의 데이타 엔트리를 비교하여 질의 영역을 결정하는 대표 객체의 좌표를 구한 후(lines 4~5), 이 대표 객체의 값으로 조인 후보 영역을 산출한다(line 6). 여기에서 호출하는 함수 QueryRangeToLineSegments()는 제3.3절에서 보인 쿼터 분할 기법을 이용한 질의 영역 공간 변환 함수를 사용한다. 다음, 조인 후보 영역에 해당하는 파일 S의 객체들을 DOT 색인을 이용하여 접근하고(line 8) 이 객체들과 PageR상의 객체들의 실질적인 교차 여부를 검사하여 조인 결과 집합에 저장시킨다(line 9).

알고리즘 4: 조인 알고리즘 DOT-Join

```
Input : File R, File S, Grid Size n
Output : set of Results RS

1.  RS = Φ;
2.  for(i = 0; i < NumOfDataPage(R); i++)
3.  |  PageR = GetDataPage(R, DataPage_i);
4.  |  q_s = MIN(PageR.X_s);
5.  |  q_e = MAX(PageR.X_e);
6.  |  LS = QueryRangeToLineSegments(q_s, q_e, n);
7.  |  for(j = 0; j < |LS|; j++)
8.  |  |  PageS = GetDataPageUsingBTree(S, LS_j);
9.  |  |_ RS = RS ∪ Intersect(PageR, PageS);
10. Return RS;
```

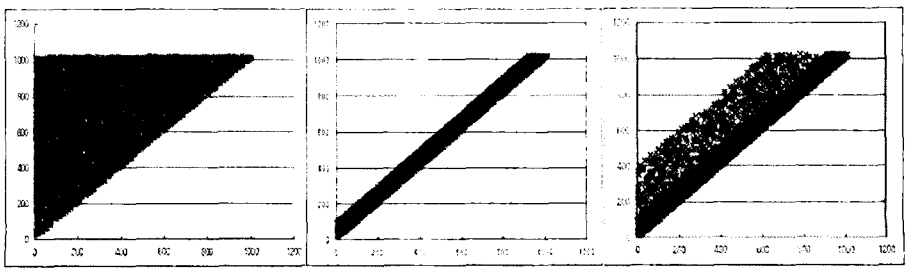
5. 성능 평가

본 장에서는 실험에 의한 성능 분석을 통하여 제안하는 기법의 우수성을 규명한다. 다양한 공간 객체 분포에 따른 실험을 수행하기 위하여 세 개의 서로 다른 분포를 가지는 공간 객체들로 구성된 실험 데이타를 사용하여 공간 조인 연산을 수행한다. 그림 10은 객체의 수가 10,000개이고, 0부터 1023 사이의 다양한 크기를 가지는 공간 객체 집합의 서로 다른 분포 유형을 나타낸다. 그림 10(a)는 다양한 크기의 공간 객체들로 구성된 균등 분포를 나타내며, 그림 10(b)는 전체 공간의 크기에 비해 크기가 비교적 작은 객체들이 좁은 띠 형태의 분포를 가지는 경우를 나타내며, 그림 10(c)는 좁은 띠 형태의 분포와 크기가 비교적 큰 객체들의 균등 분포를 혼합한 형태를 나타낸다. 실험을 위하여 이들 각각 세 개의 분포 유형에 대하여 객체 수를 10,000부터 300,000까지 증가시키며 데이타를 생성한다.

성능 비교를 위하여 DOT 색인을 이용하는 공간 조인 기법과 R*-tree를 이용한 공간 조인 기법(이후 R*-tree라고 부른다)을 사용한다. 구석점 변환 기법을 사용한 공간 조인 알고리즘[7]은 그 소스가 일반에 공개되어 있지 않으며, 구현이 어려워 금번 비교 대상에서 제외하였다.

DOT 색인을 이용하는 기법으로는 DOT-TriQuarter, DOT-RectQuarter, DOT-Range, DOT-TriQuarter-Naive의 서로 다른 4 가지 기법을 비교한다. DOT-TriQuarter는 본 논문에서 제안한 공간 조인 기법으로서, 삼각형 쿼터 분할의 영역 질의 방식을 사용한 방식을 나타낸다. DOT-RectQuarter는 사각형 쿼터 분할의 영역 질의 방식을 이용한 공간 조인 기법을 나타내며, DOT-Range는 별도의 쿼터 분할 기법을 사용하지 않는 영역 질의 방식을 이용한 공간 조인 기법을 나타낸다. 또한 DOT-TriQuarter-Naive는 데이타 페이지 당 하나의 대표 객체를 사용하여 않고, 모든 객체에 대하여 개별적으로 삼각형 쿼터 분할에 의하여 영역 질의를 수행하는 공간 조인 기법을 나타낸다. 이들 방식에서는 중간 공간의 한 변의 크기(Grid Size) n을 1024로 한다. 한편 R*-tree는 다차원 인덱스 구조를 기반으로 하는 잘 알려진 기존의 기법으로서, 본 실험에서는 Bernhard Seeger에 의해 개발된 R*-tree와 공간 조인 알고리즘을 사용한다[27]. 본 실험에서는 페이지 크기를 4K로 가정한다. 실험을 위한 하드웨어 플랫폼으로는 페도라코어 3 Linux를 운영 체제로 사용하고, 1GB의 주기억 장치, 120GB 디스크를 갖는 Pentium IV 2GHz의 PC를 사용한다.

제안된 기법의 성능을 평가하기 위하여 각 방식의 공



(a) 분포 유형 1 (b) 분포 유형 2 (c) 분포 유형 3

그림 10 실험을 위한 세 가지 공간 객체 분포 유형

간 변환 횟수, 공간 조인 연산을 수행하는데 필요한 디스크 액세스 횟수, 실행 시간 등을 기존 방식과 비교 분석한다. 본 실험에서는 동일한 두 개의 파일을 사용한 공간 조인 연산(셀프 조인)을 평가 대상으로 한다. 각 실험 데이타에 대하여 공간 조인 연산을 수행하여 얻어지는 결과 집합의 개수를 표 2에 보인다. 다양한 크기의 공간 객체들로 구성된 균등 분포의 유형 1의 경우, 유형 2 및 유형 3에 비하여 공간 조인 연산 결과 집합의 개수가 상당히 많음을 알 수 있다.

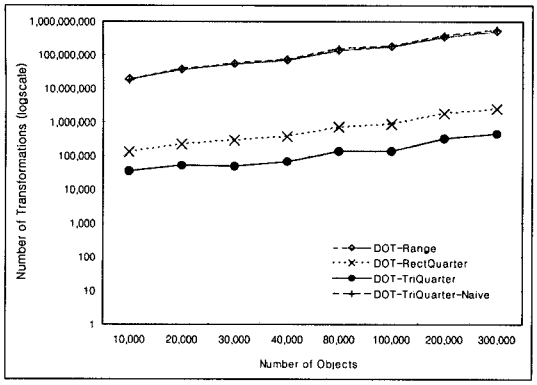
먼저, 실험 1에서는 공간 변환 횟수를 기반으로 하여 DOT-TriQuarter, DOT-RectQuarter, DOT-Range, DOT-TriQuarter-Naive의 성능을 비교한다. 그림 11은 세 개의 공간 객체 분포 유형에 대하여 객체 수의 변화에 따르는 공간 변환 횟수를 비교한 결과를 나타낸다. X축은 공간 객체의 개수를 나타내며, Y축은 공간 변환 횟수를 나타낸다. 실험 결과로부터 세 가지 공간 객체 분포에 대하여 DOT-TriQuarter와 DOT-RectQuarter는 DOT-Range와 DOT-TriQuarter-Naive에 비하여 공간 변환 횟수가 현저히 감소되어 있음을 알 수 있다. 즉, DOT-TriQuarter와 DOT-RectQuarter는 쿼터 분할 기법을 사용하여 DOT-Range에 비하여 공간 변환 연산의 횟수가 현저히 감소되었음을 알 수 있다. 또한 DOT-TriQuarter는 데이타 페이지 당 대표 객체에 대한 한 번의 공간 변환 연산만을 수행하므로 모든 객체에 대하여 공간 변환 연산을 수행하는 DOT-TriQuarter-Naive에 비하여 공간 변환 연산의 횟수가 현저히 감소되었음을 알 수 있다.

균등 분포를 나타내는 유형 1의 경우, DOT-TriQuarter는 DOT-RectQuarter에 비하여 뚜렷한 성능 개선 효과를 보이고 있다. 그러나 유형 2나 유형 3의 경우, DOT-TriQuarter는 DOT-RectQuarter에 비하여 약간 우수한 성능 개선 효과만을 보이고 있다. 그 이유는 유형 2나 유형 3의 경우와 같이 크기가 작은 공간 객체들이 대각선에 밀집되는 데이타 분포의 경우, 각 질의 영역이 매우 작은 크기의 쿼터로 분할되는 경우가 많아져 이 두 방식의 성능이 크게 차이가 나지 않는 결과를 보이기 때문이다. 그림 11(a)의 결과로부터 유형 1의 경우 DOT-TriQuarter는 DOT-TriQuarter-Naive에 비하여 약 514.5~1305.8배, DOT-Range에 비하여 약 533~1250.8배, DOT-RectQuarter에 비하여 약 3.5~6.2배의 성능 개선 효과를 보임을 알 수 있다. 또한 그림 11(b)의 결과로부터 유형 2의 경우 DOT-TriQuarter는 DOT-TriQuarter-Naive에 비하여 약 461.6~1032.8배, DOT-Range에 비하여 약 210~420.7배, DOT-RectQuarter에 비하여 약 1.4~1.6배의 성능 개선 효과를 보이며, 그림 11(c)의 결과로부터 유형 3의 경우 DOT-TriQuarter는 DOT-TriQuarter-Naive에 비하여 약 605.4~1079.3배, DOT-Range에 비하여 약 300.1~474배, DOT-RectQuarter에 비하여 약 1.5~1.9배의 성능 개선 효과를 보임을 알 수 있다.

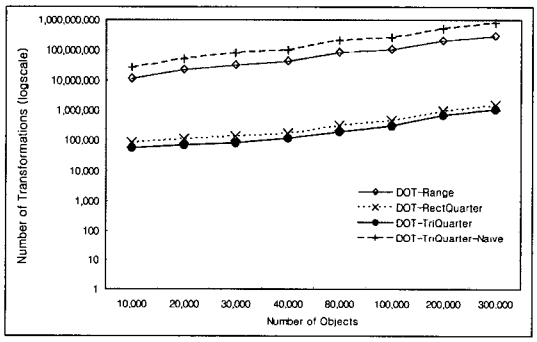
실험 2에서는 공간 조인 연산을 위한 디스크 액세스 횟수를 기반으로 하여 DOT-TriQuarter와 DOT-TriQuarter-Naive, R*-tree의 성능을 비교한다. DOT-Range와 DOT-RectQuarter는 공간 조인을 위한 디스크 액세스 횟수가 DOT-TriQuarter와 동일하므로 비교 대상에서 제외한다. 본 실험에서는 세 가지의 공간 객체 분포에 대하여 300,000개의 공간 객체를 대상으로 한다. 유형 1, 유형 2, 유형 3의 데이타 분포에 대하여 DOT-TriQuarter (혹은 DOT-TriQuarter-Naive)의 데이타 페이지 수는 각각 1510, 1471, 1490이며, R*-tree의 데이타 페이지 수는 2106, 2059, 2075이다. 세 개의 공간 객체 분포 유형에 대하여 버퍼의 크기 변화에 따르는 디스크 액세스 횟수를 비교한 결과는 그림 12와 같다. X축은 LRU 버퍼 대치 전략을 위한 사용 버퍼의 크기를 나타내며, Y축은 디스크 액세스 횟수를 나타낸다. 그림 12에서 보이는 바와 같이 버퍼 크기가 비교적 작은 부분에서 DOT-TriQuarter와 DOT-TriQuarter-Naive는 인접한 데이타 페이지 간에 중첩되는 공간 조인 후

표 2 객체 개수 증가에 따른 공간 조인 연산 결과의 개수

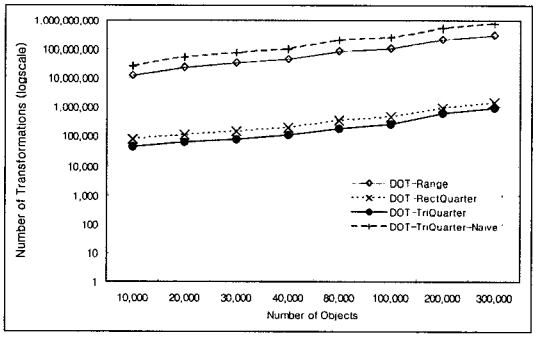
객체 개수	공간 객체 분포 유형 1	공간 객체 분포 유형 2	공간 객체 분포 유형 3
10,000	66,929,252	10,014,068	13,723,694
20,000	266,379,234	40,433,010	55,335,800
30,000	597,947,354	90,972,662	124,521,912
40,000	1,064,387,048	161,918,012	221,119,502
80,000	4,258,726,502	648,011,424	885,332,730
100,000	6,653,736,926	1,012,205,664	1,383,839,254
200,000	26,619,366,432	4,045,790,570	5,537,010,744
300,000	59,957,189,994	9,106,230,276	12,470,990,674



(a) 공간 객체 분포 유형 1의 경우

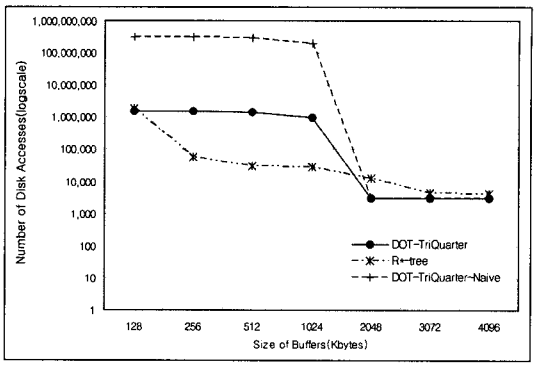


(b) 공간 객체 분포 유형 2의 경우

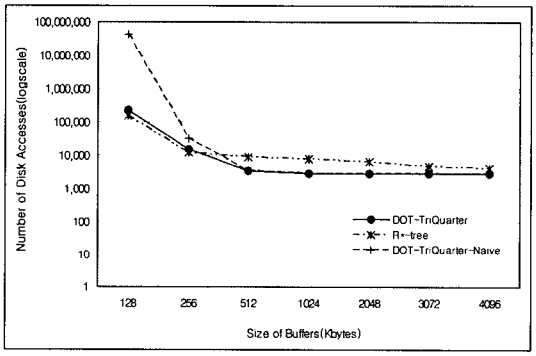


(c) 공간 객체 분포 유형 3의 경우

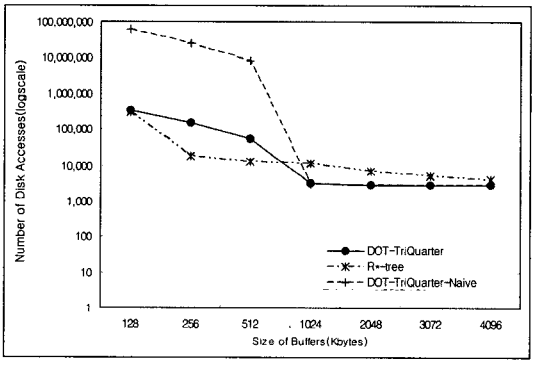
그림 11 객체 개수 증가에 따른 공간 변환 횟수의 비교



(a) 공간 객체 분포 유형 1의 경우



(b) 공간 객체 분포 유형 2의 경우



(c) 공간 객체 분포 유형 3의 경우

그림 12 분포 유형 및 버퍼 크기 증가에 따른 디스크 액세스 횟수

보 영역을 버퍼에 유지 시키지 못하므로 R*-tree 보다 많은 디스크 액세스를 필요로 한다. 특히, DOT-TriQuarter-Naive의 경우 페이지 당 대표 객체를 사용하지 않고 객체별로 질의 영역을 설정하므로 DOT-TriQuarter에 비하여 매우 많은 디스크 액세스 유발한다. 그러나 버퍼의 크기가 어느 정도 증가하면 두 방식 모두 중첩되는 공간 조인 후보 영역이 버퍼에 유지되어 R*-tree 보다 우수한 성능을 보이고 있다.

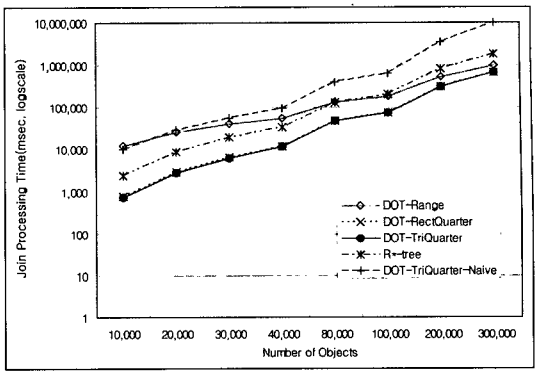
최적의 공간 조인 알고리즘은 최소한의 버퍼를 사용하여 모든 데이타 페이지를 단 한 번씩만 엑세스하는 방식으로 가정할 수 있다. 그림 12의 결과로부터 유형 1과 유형 3의 경우 DOT-TriQuarter와 DOT-TriQuarter-Naive는 버퍼 크기가 2048Kbyte(전체 데이타 페이지의 약 17%)의 경우, 최적의 디스크 액세스 횟수를 나타내며, 유형 2의 경우 DOT-TriQuarter와 DOT-TriQuarter-Naive는 버퍼 크기가 1024Kbyte(전체 데이타 페이지의

약 8.7%)의 경우 최적의 디스크 액세스 횟수를 나타냄을 알 수 있다. 한편 유형 1과 유형 2의 경우 R*-tree는 버퍼 크기가 3072Kbyte(전체 데이타 페이지의 약 18%)의 경우 최적의 디스크 액세스 횟수를 나타내며, 유형 3의 경우 R*-tree는 버퍼 크기가 4096Kbyte(전체 데이타 페이지의 약 24.6%)의 경우 최적의 디스크 액세스 횟수를 나타냄을 알 수 있다.

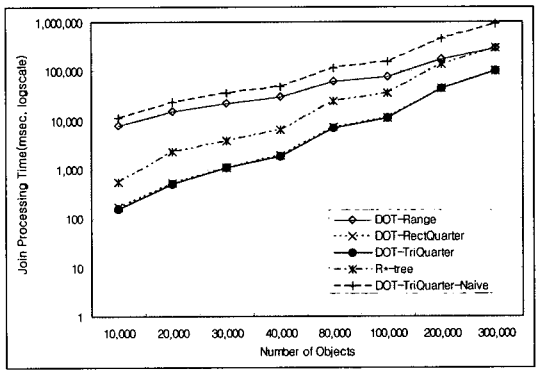
실험 3에서는 제안하는 공간 조인 기법과 R*-tree의 공간 조인 처리 시간을 비교한다. 이때 DOT 공간 색인을 이용하는 DOT-TriQuarter, DOT-RectQuarter, DOT- Range, DOT-TriQuarter-Naive의 공간 조인 처리 시간은 질의 영역에 해당하는 중간 공간상의 점들을 1차원 값으로 변환하여 얻어진 라인 세그먼트의 집합을 산출하는 시간과 이 집합이 나타내는 범위의 검색키를 가지는 공간 객체를 B+-tree 구조의 DOT 색인상에서 검색하는 시간, 그리고 후처리 시간을 합한 시간이다. 여기에서 후처리 시간은 검색된 각 공간 객체에 대하여 질의 영역과 실제로 교차되는지를 검증하는 과정을 의미한다. 세 개의 공간 객체 분포 유형에 대하여 객체 수의 변화에 따르는 공간 조인 처리 시간을 비교한 결과는 그림 13과 같다. 단, 이 실험에서는 버퍼 크기를 2048Kbyte로 가정하였다. 각 그래프에서 Y축은 로그 눈금 간격으로 표시되어 있고, 시간 측정 단위는 msec이다.

그림 13(a)는 분포 유형 1의 실험 데이타에 대한 공간 조인 연산 시간을 비교한 결과이다. DOT-TriQuarter와 DOT-RectQuarter는 공간 객체의 개수 증가에 무관하게 거의 비슷한 성능을 보이며, DOT-TriQuarter-Naive에 비하여는 약 8.1~14.7배, DOT-Range에 비하여는 약 1.5배~17.5배, R*-tree에 비하여는 약 2.7~3.3배의 성능 개선 효과를 보임을 알 수 있다. 또한 이 결과로부터 분포 유형 1에 대한 DOT-TriQuarter와 R*-tree의 공간 조인 연산은 다른 두 가지 분포 유형 2와 3의 공간 조인 연산에 비하여 비교적 많은 처리 시간을 필요로 함을 알 수 있다. 그 이유는 분포 유형 1의 실험 데이타의 경우, 다양한 객체 크기를 갖는 공간 객체가 전체 공간상에 고르게 분포하고 있으므로 표 2에 보이는 바와 같이 많은 수의 결과 집합을 얻기 위하여 많은 처리 시간을 필요로 하기 때문이다.

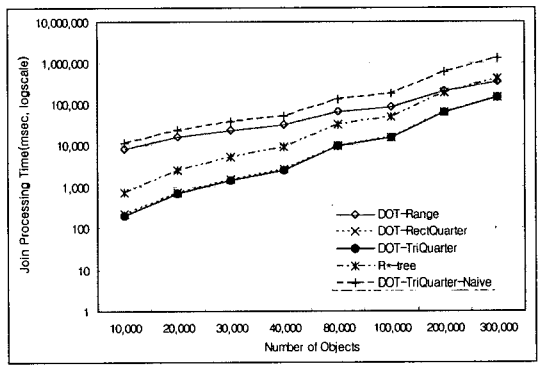
그림 13(b)와 그림 13(c)는 각각 분포 유형 2와 분포 유형 3의 실험 데이타에 대한 공간 조인 연산 시간을 비교한 결과이다. 분포 유형 2와 분포 유형 3에서 DOT-TriQuarter와 DOT-RectQuarter는 공간 객체의 개수 증가에 무관하게 거의 비슷한 성능을 보인다. 분포 유형 2에 대하여는 DOT-TriQuarter가 DOT-TriQuarter-Naive에 비하여는 약 9~70.7배, DOT-Range에 비하여는 약 3~50배, R*-tree에 비하여는 약 2.9배~4.5배의 성능 개선 효과를 보였으며, 분포 유형 3에 대하여는 DOT-TriQuarter가 DOT-TriQuarter-Naive에 비하여는 약 9~57.4배, DOT-Range에 비하여는 약 2.5~45.5배, R*-tree에 비하여는 약 2.9배~3.8배의 성능 개선 효과를 보임을 알 수 있다.



(a) 공간 객체 분포 유형 1의 경우



(b) 공간 객체 분포 유형 2의 경우



(c) 공간 객체 분포 유형 3의 경우

그림 13 객체 수 변화에 따른 공간 조인 처리 시간 (버퍼 크기 2048Kbyte)

이들 실험 결과에 의하여 제안된 DOT-TriQuarter기법은 공간 객체의 분포와 유형에 무관하게 적절한 버퍼 크기와 페이지 대치 알고리즘을 이용하는 경우 R^*-tree보다 우수한 성능을 가지며, 최대 약 3배의 성능 개선 효과를 갖는 것으로 나타났다.

앞에서 기술한 바와 같이 이 실험 결과는 버퍼 크기를 2048Kbyte로 설정한 경우를 나타낸다. 이 외에 버퍼 크기를 3072Kbyte, 4096Kbyte로 설정한 실험을 추가로 수행하였으며, 그 결과 그림 13과 거의 유사한 결과를 얻었다. 그러나 실험 2에서 검토한 바와 같이 버퍼 크기가 작아지는 경우, DOT-TriQuarter (DOT-RectQuarter, DOT-Range), 혹은 DOT-TriQuarter-Naive는 R^*-tree에 비하여 그 성능이 저하될 수 있다.

6. 결 론

DOT 공간 색인 기법은 원 공간에 존재하는 공간 객체의 최소 경계 사각형 정보를 공간 순서화 곡선을 사용하여 하나의 1차원 값으로 변환한 후 그 값을 검색키로 갖는 B^+-tree 색인 구조를 구성하는 방법으로서, 이를 이용하면 전통적인 데이타베이스의 주 색인을 적용할 수 있다는 중요한 특징을 지닌다. 본 논문에서는 1차원 공간 객체에 대하여 DOT 공간 색인 기법을 이용한 공간 조인 알고리즘을 제안하였다.

제안된 알고리즘에서는 여과 단계에서 반복적으로 수행되는 공간 변환 연산의 횟수를 줄이기 위하여 질의 영역을 공간 순서화 곡선이 연속 운행하는 가능한 최대 크기의 면적으로 분할하는 쿼터 분할 기법을 사용한다. 쿼터 분할 기법은 공간 순서화 곡선을 이용하는 모든 공간에 적용할 수 있으며, 특히 대각선 위쪽 삼각형에만 적용되는 DOT 공간 색인 기법이나 구석점 변환 기법의 경우에는 삼각형 쿼터 분할 기법을 사용함으로써 더 큰 성능 향상을 기대할 수 있다. 따라서 본 논문에서 제안한 쿼터 분할 기법은 응용 범위가 매우 넓다고 할 수 있으며, 2차원 공간에 대한 영역 질의 알고리즘이 1차원의 복잡도를 가지도록 하므로 반복적인 영역 질의를 사용하는 공간 연산에 효율적으로 이용될 수 있다.

다양한 분포와 크기를 갖는 데이타 집합을 대상으로 실험을 수행한 결과, 본 논문에서 제안한 DOT 공간 색인 기반의 공간 조인 알고리즘은 적절한 버퍼 크기와 페이지 대치 알고리즘을 이용하는 경우, R^*-tree를 이용한 공간 조인 알고리즘에 비해 최대 3배의 성능 이익을 얻을 수 있음을 확인할 수 있었다. 현재 원 공간을 2차원으로 확장하여 적용할 수 있는 DOT 공간 조인 알고리즘에 대한 연구가 진행 중에 있다.

참 고 문 헌

[1] H. Kriegel, M. Schiwietz, R. Schneider, and B. Seeger, "Performance Comparison of Point and Spatial Access Methods," SSD89, Lecture Note, pp. 89-114, 1989.

[2] C. Bohm, S. Beerchtold, and D. Keim, "Searching in High Dimensional Spaces - Index Structures for Improving the Performance of Multimedia Databases," ACM Computing Surveys, Vol. 33, No. 3, pp. 322-373, 2001.

[3] A. Guttman, "R-trees: A Dynamic Index Structures for Spatial Searching," Proc. ACM SIGMOD, pp. 47-57, 1984.

[4] N. Beckmann, H. Kriegel, and R. Schneider, "The R*-tree : An Efficient and Robust Access Method for Points and Rectangles," Proc. ACM SIGMOD, pp. 322-331, 1990.

[5] K. Hinrichs, J. Nievergelt, "The Grid File : A Data Structure Designed to Support Proximity Queries on Spatial Objects," Proc. Workshop on Graph Theoretic Concepts in Computer Science, pp. 100-113, 1983.

[6] T. Brinkhoff, H. P. Kriegel, and B. Seeger, "Efficient Processing of Spatial Joins Using R-trees," Proc. ACM SIGMOD, pp. 237-246, 1993.

[7] J. Song, K. Whang, Y. Lee, M. Lee, and S. Kim, "Spatial Join Processing Using Corner Transformation," IEEE Trans. Knowledge and Data Eng., vol. 11, no. 4, pp. 688-695, 1999.

[8] C. Faloutsos, and S. Roseman, "Fractals for Secondary Key Retrieval," Proc. PODS, pp. 247-252, 1989.

[9] C. Faloutsos, and Y. Rong, "DOT : A Spatial Access Method Using Fractals," Proc. 7th Intl. Conf. on Data Engineering, pp. 152-159, 1991.

[10] 최익수, 윤지희, 이건배, "DOT 공간색인 기법을 이용한 공간조인 알고리즘", 한국정보과학회 데이타베이스 연구회, 동계 데이타베이스 학술대회논문집, 제14권 1호, pp. 21-26, 1998.

[11] J. T. Robinson, "The K-D-B-tree: A Search Structure for Large Multidimensional Dynamic Indexes," ACM SIGMOD Conf., pp. 10-18, 1981.

[12] A. Henrich, H.-W. Six, and P. Widmayer, "The LSD Tree: Spatial Access to Multidimensional Point and Non Point Objects," Proc. 15th VLDB Conf., pp 45-53, 1989.

[13] J. Orenstein, "Spatial Query Processing in an Object-Oriented Database System," Proc. ACM SIGMOD, pp. 326-336, 1986.

[14] J. Orenstein, F. Manola, "PROBE Spatial Data Modeling and Query Processing in an Image Database Applications," IEEE Trans. on Software Engineering, Vol 14, No. 5, pp. 611-629, 1988.

[15] T. Sellis, N. Roussopoulos and C. Faloutsos, "The R^+-tree: A Dynamic Index for Multidimensional

Objects," Proc. 13th VLDB Conf., pp. 507-518, 1987.
[16] O. Gunther, The Cell Tree: An Index for Geometric Data, Memorandum No. UCB/ERL M86/89, Univ. of California, Berkeley, December 1986.
[17] Y. Theodoridis, M. Vazirgiannis, and T. K. Sellis, "Spatio-Temporal Indexing for Large Multimedia Applications," Proc. IEEE Intl. Conf. on Multimedia Computing and Systems, pp. 441-448, 1996.
[18] D. Pfoser, Y. Theodoridis, and C. S. Jensen, "Indexing Trajectories in Query Processing for Moving Objects," Chorochronos Technical Report, CH-99-3, 1999.
[19] D. Pfoser, C. S. Jensen, and Y. Theodoridis, "Novel Approaches in Query Processing for Moving Objects", Proc. VLDB Conf., pp. 395-406, 2000.
[20] Y. Tao and D. Papadias, "MV3R-Tree: A Spatio-Temporal Access Method for Timestamp and Interval Queries", Proc. VLDB Conf., pp. 431-440, 2001.
[21] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao, "Query Processing in Spatial Network Databases," Proc. VLDB Conf., pp. 802-813, 2003.
[22] S. Gupta, S. Kopparty, and C. Ravishankar, "Roads, Codes, and Spatiotemporal Queries," Proc. ACM PODS, pp. 115-124, 2004.
[23] H. W. Six and P. Widmayer, "Spatial Searching in geometric Databases," Proc. 4th Intl. Conf. on Data Engineering, pp. 496-503, 1988.
[24] J. Lawder and P. King, "Querying Multi-dimensional Data Indexed Using the Hilbert Space-Filling Curve," ACM SIGMOD Record, Vol. 30, No. 1, pp. 19-24, 2001.
[25] B. Moon, H. V. Jagadish, C. Faloutsos, and J. Saltz, "Analysis of the Clustering Properties of Hilbert Space-Filling Curve," IEEE Trans. on Knowledge and Data Eng., Vol. 13, No. 1, pp. 124-141, 2001.
[26] H. Dai and H. Su, Approximation and Analytical Studies of Inter-cluster Performances of Space-Filling Curves, Discrete Mathematics and Theoretical Computer Science, pp. 53-68, 2003.
[27] http://www.rtreeportal.org (accessed 10 May 2007).

백 현
1987년 숭실대학교 전자계산학과(학사) 1997년 한림대학교 경영대학원(석사). 1999년 한림대학교 컴퓨터공학과(박사과정) 2001년~현재 (주)시스게이트 기술연구소 관심분야는 공간데이타베이스/GIS, IT서비스관리, IT Governance

윤 지 희
정보과학회논문지 : 데이타베이스 제 34 권 제 3 호 참조

원 정 임
정보과학회논문지 : 데이타베이스 제 34 권 제 2 호 참조

박 상 현
정보과학회논문지 : 데이타베이스 제 34 권 제 1 호 참조