변환기반 공간 파티션 조인
(Transformation-based Spatial Partition Join)

이 민 재 † 한 욱 신 †† 이 재 길 † 황 규 영 †††
(Min-Jae Lee) (Wook-Shin Han) (Jae-Gil Lee) (Kyu-Young Whang)

요 약 공간 조인이란 주어진 공간 관계를 만족하는 공간 객체의 쌍들을 찾는 질의이다. 본 논문에서는 원공간상의 데이타를 이용하여 색인을 사용하지 않고 변환 공간(transform space) 상에서 공간 조인을 수행하는 새로운 알고리즘인 변환기반 공간 파티션 조인(*transformation-based spatial partition join*)을 제안한다. 기존 알고리즘들은 원공간(original space) 상에서 크기를 가지는 공간 객체를 다루기 때문에 공간 객체들의 복제를 필요로 하거나 상대적으로 공간 파티션이 복잡하여 성능이 저하되는 문제점을 가지고 있다. 이에 반해 제안하는 알고리즘은 원공간 상의 크기를 가지는 공간 객체를 변환공간 상의 크기를 가지지 않는 점 객체로 별도의 추가비용 없이 변환 해석한 후에 공간 조인을 수행하기 때문에 공간 객체들의 복제가 필요 없고, 공간 파티션이 단순하여 성능이 향상되는 장점을 가진다. 다양한 실험을 수행한 결과, 제안하는 변환기반 파티션 조인은 기존 조인 알고리즘들과 비교하여 수행 시간 측면에서 20.5~38.0% 더 우수한 성능을 보인다.

키워드 : 공간 조인, 변환 기법, 지리 정보 시스템

Abstract Spatial joins find all pairs of spatial objects that satisfy a given spatial relationship. In this paper, we propose the *transformation-based spatial partition join algorithm* (*TSPJ*), a new spatial join algorithm that performs join in the transform space without using indexes. Since the existing algorithms deal with extents of spatial objects in the original space, they either need to replicate the spatial objects or have a relatively complex partition structure—resulting in degrading performance. In contrast, TSPJ transforms objects in the original space into points in the transform space and deals only with points having no extents. The transformation does not incur any additional overhead. Thus, our algorithm has advantages over existing ones in that it obviates the need for replicating spatial objects, and its partition structure is simple. As a result, it always has better performance compared with existing algorithms. Extensive experiments show that TSPJ improves performance by 20.5 ~ 38.0% over the existing algorithms compared.

Key words : Spatial Join, Transformation Technique, GIS

1. 서 론

공간 조인이란 주어진 공간 조건을 만족하는 모든 공간 객체의 쌍들을 찾는 공간 질의의 한 종류로서, 공간 데이타베이스 시스템의 기본 연산 중의 하나이다[1]. 공간 조인의 예로는 서로 교차하는 도로와 강의 쌍을 찾는 연산을 들 수 있다. 공간 조인은 많은 디스크 I/O와 처리시간을 필요로 하므로, 이를 효과적으로 처리하는 알고리즘들에 대한 많은 연구들이 진행되어 왔다[1-6].

공간 색인을 사용하는 공간 조인 알고리즘으로는 조인되는 두 파일에 모두 색인이 있는 경우를 다루는 **깊이 우선 탐색 R 트리 조인**(*Depth-First Traversal R-Tree Join*) [1], **넓이 우선 탐색 R 트리 조인**(*Breadth-First Traversal R-Tree Join*) [2], 그리고 **변환기반 공간 조인**(*Transform-Based Spatial Join*) [6] 등과 한쪽 파일에만 색인이 있는 경우를 다루는 **시드 조인**(*Seed Join*) [3]과 **슬롯 색인 공간 조인**(*Slot Index Spatial Join*) [7] 등이 있다. 이들 방법들은 색인이 구축되어 있지 않은 파일들에는 적용될 수 없는 문제점을 가진다. 예를 들어, 복잡한 공간 질의 처리 중

· 본 연구는 첨단정보기술연구센터를 통하여 한국과학재단으로부터 지원을 받았음
† 비 회 원 : 한국과학기술원 전자전산학과
mjlee@mozart.kaist.ac.kr
jglee@mozart.kaist.ac.kr
†† 종신회원 : 경북대학교 컴퓨터공학과 교수
wshan@knu.ac.kr
††† 종신회원 : 한국과학기술원 전자전산학과 교수
kywhang@mozart.kaist.ac.kr
논문접수 : 2003년 6월 9일
심사완료 : 2004년 3월 17일

에 발생하는 중간 결과 파일들이나, 병렬 공간 질의 처리 시 교환되는 파일들에는 이들 알고리즘들을 직접 적용할 수 없다[5]. 이 경우에는 색인을 사용하지 않는 공간 조인 알고리즘을 사용하는 것이 필요한데, 본 논문에서는 이러한 조인 알고리즘을 논의 대상으로 한다.

색인을 사용하지 않는 공간 조인 알고리즘들로는 **파티션 기반 공간 병합 조인**(*Partition Based Spatial Merge Join*) [5], **공간 해쉬 조인**(*Spatial Hash Join*) [4], **크기 분할 공간 조인**(*Size Separation Spatial Join*) [8], 그리고 **스케일러블 스위핑 기반 공간 조인**(*Scalable Sweeping-Based Spatial Join*) [9] 등이 있다. 이들 알고리즘들은 조인되는 두 파일 R과 S에 대응되는 공간을 파티션들로 분할한 뒤, 공간 객체들을 대응되는 파티션들에 분배한다. 다음으로, 서로 조인 관계에 있는 파티션 쌍들을 조인한다. 파티션들을 정의하는 방법은 각 알고리즘마다 다른데, 이 방법에 따라 알고리즘의 특성이 결정된다.

파티션 기반 공간 병합 조인은 데이타 분포가 균일하다고 가정하고 공간을 그리드 형태의 파티션들로 분할한다. 공간 객체들은 크기를 가지기 때문에, 파티션의 가장자리에 위치한 공간 객체들은 복수의 파티션들과 서로 겹칠 수 있다. 파티션 기반 공간 병합 조인은 이러한 객체들을 겹치는 모든 파티션에 복제한다. 이러한 복제로 인해 데이타 분포에 따라 성능이 크게 떨어지며, 중복된 조인 결과가 나타남으로 이들을 제거하는 부담을 가진다.

공간 해쉬 조인은 데이타 분포가 균일하지 않을 수 있다고 가정하고 샘플링(sampling)을 통해 데이타 분포에 맞는 파티션들을 정의한다. 파티션 기반 공간 병합 조인과 마찬가지로 공간 객체들이 복수의 파티션과 서로 겹칠 수 있는데, 한쪽 파일에 대해서만 복제함으로써, 중복된 조인 결과가 나타나지 않도록 한다. 중복 제거를 하지 않기 때문에 파티션 기반 공간 병합 조인보다 좋은 성능을 보이지만, 여전히 한쪽 파일에서는 공간 객체들이 복제되기 때문에 데이타 분포에 따라 성능이 크게 떨어진다. 스케일러블 스위핑 기반 공간 조인도 역시 객체들을 복제하기 때문에 이와 유사한 문제점을 가진다.

크기 분할 공간 조인은 공간 객체의 크기에 따라 분배할 수 있는 파티션들을 정의한다. 공간 객체들은 단 하나의 파티션에 포함되기 때문에 복제가 필요 없다. 그렇기 때문에 복제를 필요로 하는 다른 알고리즘들 보다 약간 좋은 성능을 보인다. 그러나, 파티션들의 구조가 복잡하고 파티션들을 정의한 다음에 별도의 소트를 하여야만 이들을 조인할 수 있는 부담을 가진다.

기존에 제안된 알고리즘들에서 공간 객체들이 복제되거나 복잡한 파티션을 필요로 하는 이유는 원공간(original space) 상의 공간 객체는 크기를 가지므로 항상 공간을 분할하는 복수의 파티션들과 겹칠 수 있기 때문이다. 만약, 원공간 상의 공간 객체들을 변환공간(transform space) 상의 점 객체들로 변환하여 다룬다면, 점 객체들은 크기를 가지지 않으므로 데이타 분포에 상관없이 오직 하나의 파티션에만 포함된다. 따라서, 공간 객체들의 복제가 필요없고 공간 파티션이 단순해져 좀 더 좋은 성능을 가진 알고리즘을 제안할 수 있다.

본 논문에서는 원공간상의 데이타를 가지고 색인을 사용하지 않고 변환공간 상에서 공간 조인을 수행하는 새로운 알고리즘인 **변환기반 공간 파티션 조인**(*Transformation-Based Spatial Partition Join*)을 제안한다. 제안하는 알고리즘은 조인되는 두 파일 R과 S에 포함된 원공간 상의 객체들을 변환공간 상의 점들로 별도의 추가비용 없이 변환 해석하면서 이 점들을 소트하고, 소트 결과를 파일에 쓰면서 파티션들을 정의한다. 다음으로, 서로 조인 관계에 있는 R과 S의 파티션 쌍들을 찾아 이들에 포함된 공간 객체들을 조인한다. 파티션들을 조인하는 과정은 본 논문의 저자들이 제안한 변환공간 조인 알고리즘 [6]을 활용한다. 제안하는 알고리즘은 1) 공간 객체의 복제를 필요로 하지 않기 때문에 기존 알고리즘들 [4, 5, 9]처럼 데이타 분포에 따라 성능이 떨어지는 문제가 없으며, 2) 단순한 파티션 구조를 사용하여 알고리즘의 구성이 간단하기 때문에 기존 알고리즘들에 비해 좋은 성능을 보인다. 실험 결과, 비교된 기존 알고리즘들 [5, 8]에 비해 20.5 ~ 38.0% 성능이 향상됨을 보인다.

본 논문의 구성은 다음과 같다. 제2장에서는 관련 연구로서 기존의 공간 조인 알고리즘들에 대해 설명한다. 제3장에서는 변환기반 파티션 조인 알고리즘을 제안하고, 제4장에서는 제안하는 공간 조인 알고리즘의 성능 평가를 수행하고 그 결과를 제시한다. 마지막으로 제5장에서는 결론을 내린다.

2. 관련 연구

본 장에서는 색인을 사용하지 않는 대표적인 공간 조인 알고리즘인 **파티션 기반 공간 병합 조인**(*Partition Based Spatial Merge Join: PBSM*) [5]과 **크기 분할 공간 조인**(*Size Separation Spatial Join: SSSJ*) [8]에 대해 설명한다. 파티션 기반 공간 병합 조인은 객체를 복제하는 대표적인 알고리즘이며, 객체를 복제하는 다른 알고리즘들로는 공간 해쉬 조인 (Spatial Hash Join) [4]과 스케일러블 스위핑 기반 공간 조인 (Scalable Sweeping-Based Spatial Join) [9] 등이 있다. 크기 분할 공간 조인은 객체를 복제하지 않는 유일한 알고리즘

이다.

2.1 파티션 기반 공간 병합 조인

파티션 기반 공간 병합 조인(PBSM)은 Patel과 DeWitt이 제안한 알고리즘 [5]으로서 크게 1) 파티션, 2) 조인, 3) 중복제거의 세 과정으로 구성된다.

파티션 과정에서는 데이타 분포가 균일하다는 가정하에 조인되는 두 파일 R과 S를 각각 그리드 형태의 파티션들로 분할한다. 다음으로, 파일 R과 S에 포함된 공간 객체들을 각각의 공간 객체와 겹치는 파티션들에 분배한다. 이 때, 한 공간 객체는 복수의 파티션들과 겹칠 수 있는데, 파티션 기반 공간 병합 조인에서는 겹치는 모든 파티션에 공간 객체를 복제한다.

데이타 분포가 균일하지 않은 경우에는 몇몇 파티션들에 많은 수의 공간 객체들이 편중되는 경우가 발생할 수 있다. 이 경우, 하나의 파티션이 메인 메모리 크기보다 커서 조인 과정을 수행하지 못하는 경우가 발생한다. 파티션 기반 공간 병합 조인은 이 문제를 해결하는 방법으로 두 가지 방법을 사용한다. 첫 번째 방법은 공간을 재구성하여 재구성된 공간에서의 데이타 분포가 가능한 균일 분포를 가지도록 한 다음에 이 공간에서 파티션들을 정의하는 방법이다. 파티션 기반 공간 병합 조인은 이를 위해 잘게 나누어진 영역들을 해쉬 함수등을 사용하여 재배치하는 방법을 사용한다. 이와 같은 방법은 멀티프로세서 환경에서 각 프로세서에 주어지는 작업의 양을 균등하게 만드는 데도 자주 사용된다[5]. 두 번째 방법은 크기가 큰 파티션들을 다시 분할하는 방법이다. 첫 번째 방법을 사용하더라도 경우에 따라서는 파티션의 크기가 메인 메모리 크기보다 클 수가 있다. 이 경우, 이들을 다시 분할하여 메인 메모리에서 읽을 수 있는 크기로 만든다.

조인 과정에서는 서로 조인 관계에 있는 모든 두 파티션들을 차례대로 메모리에 읽어 들여 공간 조인을 수행하고 그 결과를 파일에 쓴다. 이 때, 각 파티션들에는 중복된 공간 객체들이 들어 있으므로, 결과 파일에는 중복된 결과가 들어갈 수 있다. 따라서, 중복된 결과들을 제거하는 것이 필요하다. 중복제거 과정에서는 결과 파일을 소트한 다음에 중복된 결과를 찾아내어 제거한다. 이와같이, 파티션 기반 공간 병합 조인은 공간 객체들을 여러 파티션들에 중복시키기 때문에, 주어진 공간 객체들의 수보다 더 많은 수의 공간 객체들을 조인해야 하며, 조인 결과에서 중복된 결과를 제거해야 하는 부담을 가진다.

예 1 (파티션 기반 공간 병합 조인): 그림 1은 R과 S가 4개의 파티션들로 분할되었을 때, 조인하는 과정을 설명한다. 우선 R의 파티션 1과 S의 파티션 1이 서로 조인 관계를 가지므로, 이들에 포함된 객체 a, b, c와 객체 g, h를 읽어 조인한다. 다음으로 R의 파티션 2와 S의 파티션 2가 서로 조인 관계를 가지므로, 객체 b, c와 객체 h, i를 읽어 조인한다. 이때, 객체 b, c와 객체 h는 파티션 1과 2가 조인될 때, 중복되어 읽히므로 중복된 조인 결과를 발생시킬 수 있다. 나머지 파티션 3과 4에 대해서도 파티션에 포함된 객체들을 읽어 조인하는 과정을 반복한다. □

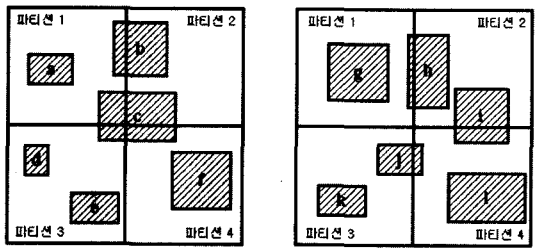


그림 1 그리드 형태의 파티션들 간의 조인 예

2.2 크기 분할 공간 조인

크기 분할 공간 조인(SSSJ)은 Koudas와 Sevcik이 제안한 알고리즘 [8]으로서 크게 1) 파티션, 2) 소트, 3) 조인의 세 과정으로 구성된다.

파티션 과정에서는 공간을 크기에 따라 계층적으로 분할하는 level file이라는 것을 사용한다. R_j는 파일 R의 j번째 level file로 공간을 2^j개의 그리드 영역들로 분할한다. j의 값이 커질수록 공간을 분할하는 영역의 크기는 작아진다. 크기 분할 공간 조인은 파일 R에 들어 있는 모든 공간 객체들을 차례대로 읽어 들이면서 각 공간 객체를 하나의 level file R_j에 저장한다. 여기서, R_j는 j의 값이 가장 크면서 R_j의 그리드 영역들과 주어진 공간객체가 겹치지 않고 포함되는 것을 선택한다. 공간 객체는 크기가 클수록 작은 j 값을 가지는 R_j에 저장되며, 크기가 작을수록 큰 j 값을 가지는 R_j에 저장된다. 파일 S의 공간 객체들도 같은 방법으로 S_j들에 저장된다. 크기 분할 공간 조인은 이렇게 다양한 크기로 공간을 분할하는 level file들을 정의함으로써, 크기를 가지는 공간 객체들을 복제없이 파티션들에 분배한다.

다음으로, 소트 과정에서는 R과 S의 모든 level file들에 저장된 공간 객체들을 공간 객체의 중심점에 대한 힐버트 값으로 소트한다. 소트시 힐버트 값을 매번 계산하는 부담을 줄이기 위해, 파티션 과정에서 공간 객체들을 level file에 넣을 때 힐버트 값을 미리 계산하여 추가한다.

조인 과정에서는 R과 S의 모든 level file들을 다음과 같이 읽으면서 조인을 수행한다. R과 S가 각각 L개의 level file들로 구성된다고 할 때, level file R_{L-1}과 S_{L-1}의 영역들을 힐버트 순서로 각각 선택하여 r_{L-1}과 s_{L-1}로

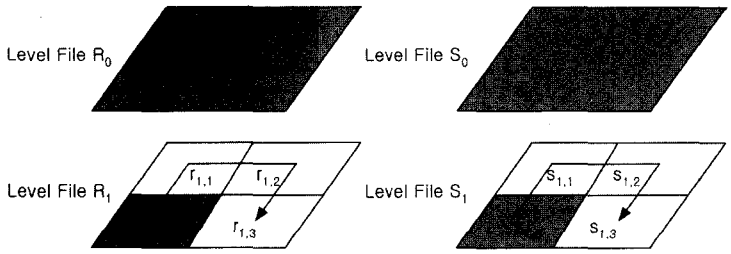


그림 2 Level file들 간의 조인 예

가리킨다. 다음으로, level file R_{L-2}, R_{L-3}, ..., R_0에서 r_{L-1}을 포함하는 영역 r_{L-2}, r_{L-3}, ..., r_0를 선택하고, level file S_{L-2}, S_{L-3}, ..., S_0에서 s_{L-1}을 포함하는 영역 s_{L-2}, s_{L-3}, ..., s_0을 선택한다. 다음으로, r_{L-2}, r_{L-3}, ..., r_0과 s_{L-2}, s_{L-3}, ..., s_0에서 아직 서로 조인된 적이 없는 모든 쌍의 영역들을 선택하여 조인한다. 이 과정을 모든 영역들이 선택되어 조인될 때까지 반복한다.

예 2 (크기 분할 공간 조인): 그림 2는 R과 S가 각각 두 개의 level file들로 구성된 경우의 조인 과정을 설명한 그림이다. 우선, R_1의 영역들을 힐버트 순서로 선택하고 S_1의 영역들을 힐버트 순서로 선택한다. 따라서, 처음에는 $r_{1,0}$과 $s_{1,0}$을 선택하고 $r_{1,1}$과 $s_{1,1}$, $r_{1,2}$와 $s_{1,2}$, 마지막으로 $r_{1,3}$과 $s_{1,3}$을 선택한다. 그리고, R_0의 영역들 중에서 $r_{1,0}$을 포함하는 영역인 $r_{0,0}$를 선택하고 S_0의 영역들 중에서 $s_{1,0}$을 포함하는 영역인 $s_{0,0}$을 선택한다. 다음으로 R에서 선택된 영역 $r_{0,0}$, $r_{1,0}$와 S에서 선택된 영역 $s_{0,0}$, $s_{1,0}$에서 아직 서로 조인된 적이 없는 모든 쌍의 영역들을 선택하여 조인한다. 이 과정을 모든 영역들이 선택되어 조인될 때까지 반복한다. □

크기 분할 공간 조인은 공간 객체들의 중복이 필요 없기 때문에 파티션 기반 공간 병합 조인 보다 약간 빠른 장점을 가진다. 하지만, 파티션들의 구조와 조인 과정이 다소 복잡한 부담을 가진다.

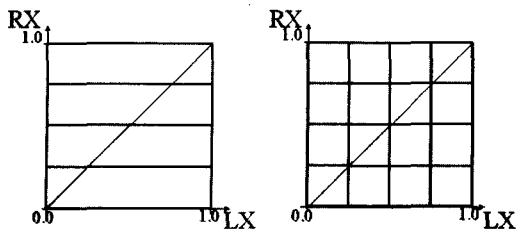
3. 변환기반 공간 파티션 조인

변환기반 공간 파티션 조인 (Transformation-Based Spatial Partition Join)은 크게 1) 소트-파티션 과정과 2) 조인 과정으로 구성된다. 소트-파티션 과정에서는 조인되는 두 파일 R과 S에 포함된 원공간 상의 객체들을 변환공간 상의 점들로 별도의 추가비용 없이 변환 해석하면서 이 점들을 소트하고, 소트 결과를 파일에 쓰면서 파티션들을 정의한다. 조인 과정에서는 서로 조인 관계에 있는 파티션 쌍들을 조인한다.

원공간 상의 객체들을 변환공간 상의 점들로 변환하는 방법으로는 구석점 변환 기법[10]을 사용한다. 구석점 변환 기법이란 n차원 원공간 상의 공간 객체를 $2n$차원 변환공간 상의 점 객체로 해석하는 기법으로, n차원 원공간의 각 차원 축에 대한 원공간 객체의 최소값과 최대값을 $2n$차원 점 객체의 좌표값으로 해석한다. 예를 들어, 1차원 원공간에서 공간 객체의 최소값과 최대값이 각각 lx와 rx인 공간 객체는 구석점 변환 기법에 의해 2차원 변환공간 상의 점 객체 $<lx, rx>$로 해석된다. 이와 같이 해석 방법에 따라 원공간 상의 객체들이 변환공간 상의 점들로 변환되므로 변환으로 인한 추가 비용은 들지 않는다.

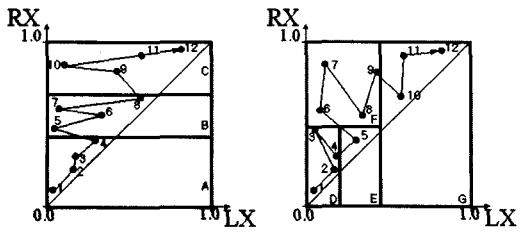
우선 1차원 원공간에 대응되는 2차원 변환공간에서의 소트-파티션 과정과 조인 과정을 방법을 설명한다음에 n차원 원공간에 대응되는 $2n$차원 변환공간으로 이들을 확장한다. 소트-파티션 과정에서는 변환공간 상의 점들로 변환된 파일 R과 S의 객체들을 Z-order로 소트하고, 소트된 점 객체들을 파일에 차례대로 쓰면서 파일 R과 S에 대응되는 변환공간 TS(R)과 TS(S)를 분할하는 파티션들을 정의한다. 이 때, TS(R)의 파티션들은 그림 3(a)에서와 같이 가로띠(horizontal strip) 형태를 하도록 정의하고, TS(S)의 파티션들은 그림 3(b)에서와 같이 그리드(grid) 형태를 하도록 정의한다. 여기서, TS(R)을 그리드 형태의 영역들이 아닌 가로띠 형태의 영역들로 분할하는 이유는 조인 과정에서 가로로 인접한 영역들을 가로띠 형태로 묶는 과정을 생략하기 위함이다. 조인 과정에서 가로로 인접한 영역들을 가로띠 형태로 묶는 이유는 조인시 발생하는 디스크 액세스 수를 줄이기 위해서이다. 이에 대한 자세한 설명은 조인 과정에서 설명한다.

그림 3과 같이 변환공간을 파티션들로 나누기 위해 Z-order의 클러스트링(clustering) 특성을 이용한다. Z-order 상에서 가까이 있는 점들은 공간 상에서도 가까이 있으므로, Z-order로 소트된 점들의 그룹핑(grouping)을 통해 공간 상에서 가까이 있는 점들을 묶는 영역들을 정의할 수 있다. 이 때, 정의된 영역들이 서로 겹치지 않도록 함으로써, 각 영역들을 공간을 분할하는 파티션으로서 사용한다. 한 파티션으로 묶여지는 점들의 수는 점들의 크기의 총 합이 한 페이지 크기 이



(a) TS(R)의 파티션들 (b) TS(S)의 파티션들

그림 3 TS(R)과 TS(S)의 파티션들



(a) 가로띠 형태의 파티션 (b) 그리드 형태의 파티션

그림 4 파티션의 정의 예

하가 되도록 제한한다.

정의되는 파티션들의 형태는 점 객체들을 Z-order로 소트할 때, 점 객체의 Z-order 값을 정의하는 방법에 따라 조절된다. 점 객체 <*lx*, *rx*>의 Z-order 값을 *lx*, *rx*의 2차원 Z-order 값, 즉 *lx*와 *rx*의 각 bit들을 interleaving한 bit string으로 정의하면, 그리드 형태의 파티션들이 정의되고, *rx*의 1차원 Z-order 값, 즉 *rx*의 bit string으로 정의하면 가로띠 형태의 파티션들이 정의된다.

예 3 (파티션 정의 과정): 그림 4는 변환공간 상에 12개의 점이 있을 때, 한 파티션에 4개 이하의 점이 들어가도록 파티션들을 정의한 예이다. 그림 4(a)는 가로띠 형태의 파티션 A, B, C를 정의한 것이고, 그림 4(b)는 그리드 형태의 파티션 D, E, F, G를 정의한 것이다. 그림 4(a)에서는 가로띠 형태의 파티션들을 정의하기 위해 각 점 <*lx*, *rx*>을 rx의 1차원 Z-order 값으로 소트한 다음에 소트된 순서에 따라 점들을 4개 이하씩 그룹핑하여 이들을 포함하는 영역을 파티션으로 정의한다. 그림 4(a)에서 점들에 붙여진 숫자는 소트된 순서에 의해 부여된 숫자이다. 그림 4(b)에서는 그리드 형태의 파티션들을 정의하기 위해 각 점 <*lx*, *rx*>을 *lx*, *rx*의 2차원 Z-order 값으로 소트한 다음에 소트된 순서에 따라 점들을 4개 이하씩 그룹핑하여 이들을 포함하는 영역을 파티션으로 정의한다. 이 때, 정의된 파티션들이 서로 겹치지 않도록 파티션들에 들어가는 점들의 수를 조정한다. 예를 들어, 파티션 E가 6번 점을 포함하는 경우, 파티션의 크기가 커져 다른 파티션들과 겹치게 된다. 정의상 파티션들은 서로 겹칠 수 없으므로, 6번 점은 파티션 E에 포함시키지 않는다. □

조인 과정에서는 TS(R)의 각 파티션과 서로 조인 관계에 있는 TS(S)의 파티션들을 찾아 이들에 포함된 객체들을 서로 조인한다. 서로 조인 관계에 있는 파티션들은 공간 조인 윈도우 [6]를 사용하여 찾는다. **공간 조인 윈도우**(*spatial join window*) SJW(P)란 TS(R)에 있는 사각형 영역 P에 대해 사각형 영역 P에 포함될 수 있는 모든 객체들과 교차관계를 가지는 객체들이 존재하는 TS(S)내의 최소 영역이다. TS(R)의 한 파티션과 대응되는 영역을 P라고 할 때, TS(R)의 파티션은 SJW(P)와 겹치는 TS(S)의 모든 파티션들과 서로 조인 관계를 가진다.

사각형 영역 P에 대한 SJW(P)의 영역은 다음의 과정으로 구해진다. P의 임의의 객체와 교차 관계를 가지기 위해서는, P의 좌상점과 대응되는 객체 $\hat{q}$=<*lx*, *rx*>와 반드시 교차관계를 가져야 한다. 이는 $\hat{q}$이 P의 모든 객체를 포함하는 가장 큰 객체이기 때문이다. $\hat{q}$과 교차하는 모든 객체들이 존재하는 TS(S)내의 최소 영역, 즉 SJW(P)는 보조정리 1에 따라 [0, *rx*] × [*lx*, 1]이 된다.

보조정리 1 (SJW(P)의 영역): [6] TS(R)내의 원공간 객체 $\hat{q}$=<*lx*, *rx*>와 원공간에서 교차관계를 가질 수 있는 모든 객체들이 존재하는 TS(S)내의 최소 영역은 [0, *rx*] × [*lx*, 1]이다. □

증명: 원공간 객체 $\hat{q}$과 원공간 객체 $\hat{u}$=<*lx'*, *rx'*>이 서로 교차관계에 있을 때, 이 두 객체들은 서로 떨어져 있지 않은 관계를 가지므로 not (*lx* > *rx'* or *lx'* > *rx*)의 관계를 만족한다. 여기서 *lx* > *rx'*과 *lx'* > *rx*는 두 객체 $\hat{q}$과 $\hat{u}$이 서로 떨어져 있음을 뜻한다. 이 관계를 풀어 쓰면 (*lx* ≤ *rx'* and *lx'* ≤ *rx*)이 된다. 이 관계에 의해 *lx'*은 구간 [0, *rx*]에 존재하고, *rx'*은 구간 [*lx*, 1]에 존재하므로, $\hat{q}$과 교차하는 모든 객체들이 존재하는 TS(S)내의 최소 영역은 [0, *rx*] × [*lx*, 1]이 된다. □

예 4 (사각형 영역 P에 대한 SJW(P)의 영역): 그림 5는 TS(R)내의 사각형 영역 P에 대한 TS(S)내의 영역 SJW(P)를 나타낸다. 그림 5에서 영역 P는 짙은 사각형으로, SJW(P)는 옅은 사각형으로 표시된다. □

서로 인접한 TS(R)의 두 영역들의 두 공간 조인 윈도우들 사이에는 다음의 두 가지 특성에 의해 매우 큰 겹침이 존재한다. 이 특성들을 이용하면 파티션들의 조인 시, 디스크 액세스 횟수를 크게 줄일 수 있다.

첫 번째 특성은 가로로 인접한 두 영역들의 공간 조인 윈도우들은 서로 포함관계를 가지는 것이다. 그림 6(a)에서와 같이 가로로 인접한 두 영역들 중 대각선에서 먼쪽을 P_1, 가까운 쪽을 P_2라고 할 때, 이 두 영역의

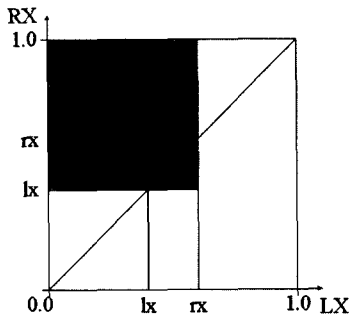


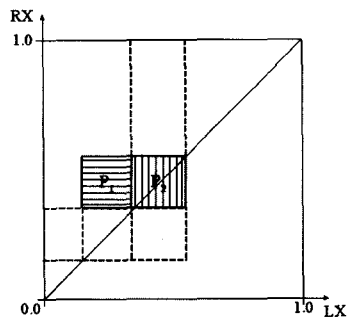
그림 5 공간 조인 윈도우

SJW들 사이에는 그림 6(a)에서와 같이 SJW(P_1) ⊃ SJW(P_2)인 관계가 성립한다. 만약 P_1과 SJW(P_1)을 조인한 후에 P_2와 SJW(P_2)를 조인하면, SJW(P_2)는 이미 버퍼에 있으므로 별도의 디스크 액세스 없이 조인을 처리할 수 있다. 그러므로, 가로로 인접한 영역들을 가지는 파티션들을 가로띠로 묶어 가로띠에 속한 파티션들을 함께 조인한다. 그런데, 변환기반 공간 파티션 조인에서는 소트-파티션 과정에서 파티션들을 이미 가로띠 형태로 정의하였으므로 파티션들을 가로띠로 묶을 필요 없이 하나의 파티션을 하나의 가로띠로 정의하여 사용한다.

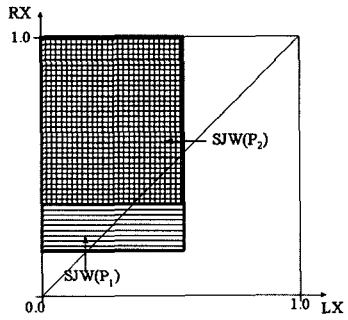
두 번째 특성은 그림 7에서와 같이 세로로 인접한 두 가로띠 $HSTR_1$과 $HSTR_2$의 공간 조인 윈도우 SJW($HSTR_1$)과 SJW($HSTR_2$) 사이에는 큰 겹침이 존재하는 것이다. 만약, $HSTR_1$과 SJW($HSTR_1$)을 조인한 후에 $HSTR_2$와 SJW($HSTR_2$)를 조인하면, SJW($HSTR_2$)의 대부분은 이미 버퍼에 있으므로 적은 수의 디스크 액세스로 조인을 처리할 수 있다. 그러므로, 가로띠들을 세로로 인접하게 선택하면서 조인한다.

위 두 가지 특성을 이용하여 조인 과정에서는 다음과 같이 파티션들을 조인한다. 우선, TS(R)의 파티션들을 순차적으로 읽는다. 읽혀진 하나의 파티션은 그림 3(a)에서와 같이 하나의 가로띠 HSTR과 대응된다. 그림 8은 이상에서 설명한 변환기반 공간 파티션 조인을 정리한 알고리즘이다.

지금까지 1차원 원공간에 대응되는 2차원 변환공간에서 조인을 수행하는 변환기반 공간 파티션 조인에 대해 설명하였다. 이를 n차원 원공간에 대응되는 $2n$차원 변

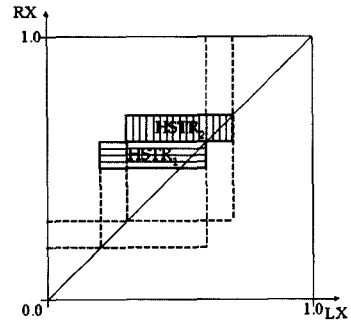


(a) 가로로 인접한 두 영역 P_1과 P_2

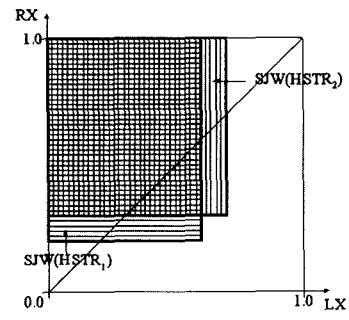


(b) SJW(P_1)과 SJW(P_2)

그림 6 가로로 인접한 두 영역들의 공간 조인 윈도우들



(a) 세로로 인접한 두 가로띠 $HSTR_1$과 $HSTR_2$



(b) SJW($HSTR_1$)과 SJW($HSTR_2$)

그림 7 세로로 인접한 두 가로띠들의 공간 조인 윈도우들

Algorithm TSPJ(File R, File S)
1. File R의 변환공간 TS(R)을 가로띠 형태의 파티션들로 분할한다.
   1.1 $2n$차원 TS(R)의 각 점 $<l_1, r_1, \cdots, l_i, r_i, \ldots, l_n, r_n>$의 Z-order 값을 r_i들의 n차원 Z-order 값으로 정의하여 소트한다.
   1.2 소트결과를 쓰면서 소트된 점들을 페이지 크기 이하가 되게 그룹핑하여 파티션들을 정의한다.

2 File S의 변환공간 TS(S)를 그리드 형태의 파티션들로 분할한다.
   2.1 $2n$차원 TS(S)의 각 점 $<l_1, r_1, \cdots, l_i, r_i, \ldots, l_n, r_n>$의 Z-order 값을 l_i, r_i들의 $2n$차원 Z-order 값으로 정의하여 소트한다.
   2.2 소트결과를 쓰면서 소트된 점들을 페이지 크기 이하가 되게 그룹핑하여 파티션들을 정의한다.

3 TS(R)의 파티션들과 TS(S)의 파티션들을 서로 조인한다.
   3.1 TS(R)의 각 파티션 HSTR을 순차적으로 선택한다.
   3.2 SJW(HSTR)과 겹치는 TS(S)의 파티션들을 선택한다.
   3.3 선택된 TS(R)의 파티션과 TS(S)의 파티션들에 포함된 객체들을 서로 조인한다.

그림 8 변환기반 공간 파티션 조인 알고리즘

환공간으로 확장하는 방법은 다음과 같다. 소트-파티션 과정은 $2n$차원 변환공간을 분할하는 파티션들을 정의하도록 확장된다. 즉, TS(R)을 가로띠 형태의 파티션들로 분할하기 위해 TS(R)의 $2n$차원 변환공간상의 각 점$<l_1, r_1, \ldots, l_i, r_i, \ldots, l_n, r_n>$의 Z-order 값을 r_i들의 n차원 Z-order 값으로 정의하여 소트하면서 파티션들을 정의한다. 그리고, TS(S)를 그리드 형태의 파티션들로 분할하기 위해 TS(S)의 각 점 $<l_1, r_1, \ldots, l_i, r_i, \ldots, l_n, r_n>$의 Z-order 값을 l_i, r_i들의 $2n$차원 Z-order 값으로 정의하여 소트하면서 파티션들을 정의한다. 정의된 파티션들을 조인하는 조인 과정은 2차원 변환공간에서와 동일하다.

변환기반 공간 파티션 조인은 다음과 같은 특징을 가진다. 첫째, 변환기반 공간 파티션 조인은 변환공간 상의 크기가 없는 점들을 다루기 때문에 객체들의 복제가 없다. 따라서, 복제를 필요로 하는 기존의 연구들보다 좋은 성능을 보인다. 둘째, 파티션 구조가 간단하기 때문에 다른 알고리즘들보다 간단한 과정들로 구성된다. 따라서, 변환기반 공간 파티션 조인은 파티션 기반 공간 병합 조인과 크기 분할 공간 조인 보다 좋은 성능을 보인다. 다음 장에서는 실험을 통해 변환기반 공간 파티션 조인과 기존 알고리즘들과의 성능을 비교한다.

4. 실 험

본 장에서는 변환기반 공간 파티션 조인과 기존에 제안된 공간 조인 알고리즘과의 성능을 비교한다. 기존 알고리즘들로는 색인을 사용하지 않는 대표적인 공간 조인 알고리즘인 파티션 기반 공간 병합 조인 [5]과 크기 분할 공간 조인 [8]을 선택하였다. 성능 평가의 척도로서 수행 시간(elapsed time)을 사용하며, 각 알고리즘들의 성능을 수행 과정별로 구분하여 측정한다. 표 1은 비교하는 알고리즘들을 정리한 표이고, 표 2는 각 알고리즘들을 과정별로 정리한 표이다.

표 1 실험에 사용된 알고리즘들

이 름	의 미
TSPJ	변환기반 공간 파티션 조인 알고리즘
PBSM	파티션 기반 공간 병합 조인 알고리즘 [5]
SSSJ	크기 분할 공간 조인 알고리즘 [8]

표 2 실험에 사용된 알고리즘들의 과정별 요약

이 름	단 계	의 미
TSPJ	Sort -Partition	파일 R과 S를 소트를 통해 파티션들로 분할한다.
	Join	서로 조인 관계에 있는 모든 두 파티션들을 조인하고 결과를 파일에 쓴다.
PBSM	Partition	파일 R과 S를 읽어 파티션들로 분할한다.
	Join	서로 조인 관계에 있는 모든 두 파티션들을 조인하고 결과를 파일에 쓴다.
	Sort	결과 파일안의 결과들을 소트한 다음에 중복된 결과를 제거하여 파일에 쓴다.
SSSJ	Partition	파일 R과 S를 읽어 level 파일들로 분할한다.
	Sort	Level 파일들안의 공간 객체들을 소트한다.
	Join	Level 파일들을 읽으면서 조인을 하고 결과를 파일에 쓴다.

실험 데이타로는 균일(uniform), 지수(exponential), 그리고 실제(real) 분포를 가지는 데이타들을 사용한다. 균일 분포를 가지는 실험 데이타는 두 데이타 집합 U1과 U2로 구성되고, 지수 분포를 가지는 실험 데이타는 두 데이타 집합 E1과 E2로 구성된다. 각 데이타 집합은 13만개의 MBR들 (2.6 MBytes)로 구성된다. U1과 U2 내의 MBR들의 중심점들은 전체 공간에 대해 균일 분포를 가지고, E1과 E2내의 MBR들의 중심점들은 각 축에서의 평균값이 1/4인 지수 분포를 가진다. U1과 U2의 공간 조인 결과 수는 87,434개이고, E1과 E2의 공간 조인 결과 수는 87,346개이다. 실제 분포를 가지는 실험 데이타는 두 데이타 집합 tiger1과 tiger2로 구성된다 [11]. tiger1은 캘리포니아 지역의 도로 131,461개의 MBR (2.6 MBytes)로 이루어진 데이타 집합이고 tiger2는 강과 철도 128,971개의 MBR (2.5 MBytes)로 이루어진 데이타 집합이다. tiger1과 tiger2의 공간 조인 결과 수는 86,094개이다.

실험은 Pentium 3 (800 MHz) PC에서 수행하였다. 운영체제에 의한 버퍼링 효과를 피하고, 실제 디스크 I/O를 보장하기 위해 raw disk를 사용하였다. 참고 문헌 [8]과 동일하게 SSSJ와 PBSM에서 사용가능한 메모리의 크기는 조인되는 파일의 10% 크기로 제한하였고, TSPJ에도 같은 제한을 두었다. 사용된 페이지의 크기는 4 KBytes이다. 표 3은 실험 환경과 실험 데이타를 정리한 표이다.

그림 9(a), 그림 9(b), 그리고 그림 9(c)는 각각 균일 분포, 지수 분포, 실제 분포를 가지는 데이타에 대해 알고리즘들의 성능을 비교한 것이다. 실험 결과, 제안하는 TSPJ는 모든 데이타 분포에 대해 다른 알고리즘들 보다 좋은 성능을 보인다. TSPJ는 PBSM와 비교하여 모든 데이타 분포에서 수행 시간을 25.1~38.0%[1] 단축시키고, SSSJ와 비교하여 20.5~23.3% 단축시킨다.

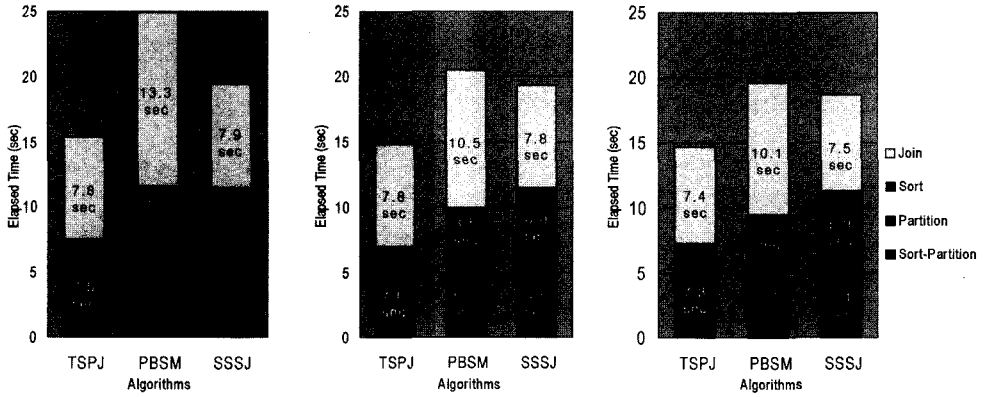
표 3 실험 환경 요약

시스템 설명	
CPU	800 Mhz
OS	Windows 2000
Memory	조인되는 파일 크기의 10% (약 500 KBytes)로 제한
Disk	Raw Disk
Page Size	4 KBytes
실험 데이타 설명	
균일 분포 데이타	균일 분포를 가지는 13만건 데이타(2.6 MBytes) U1, U2
지수 분포 데이타	지수 분포를 가지는 13만건 데이타(2.6 MBytes) E1, E2
실제 분포 데이타	실제 분포를 가지는 131,461건 데이타(2.6 MBytes) tiger1과 128,971만건 데이타(2.5 MBytes) tiger2

1) $\frac{elapsed\ time(other) - elapsed\ time(TSPJ)}{elapsed\ time(other)} \times 100$

각 과정별로 알고리즘들의 수행 시간을 분석해 보면 다음과 같다. TSPJ의 소트-파티션 과정은 소트와 파티션을 함께 수행하므로, PBSM이나 SSSJ와 비교할 때는 소트 과정과 파티션 과정을 함께 묶어서 비교한다. 소트 과정과 파티션 과정의 경우, TSPJ는 PBSM와 비교하여 모든 데이타 분포에서 수행 시간을 23.1~35.3% 단축시키고, SSSJ와 비교하여 34.7~39.1% 단축시킨다. 이는 TSPJ의 경우, 소트 결과를 파일에 쓰면서 파티션들을 정의하므로 파티션을 위한 별도의 수행 시간이 필요없는 반면, PBSM이나 SSSJ의 경우, 소트와 파티션을 따로 수행하므로 각각의 수행 시간이 필요하기 때문이다. 조인 과정의 경우, TSPJ는 PBSM와 비교하여 모든 데이타 분포에서 수행 시간을 25.7~40.6% 단축시키고, SSSJ와 비교하여 거의 비슷한 수행 시간을 가진다. 이는 TSPJ의 파티션들에는 중복된 객체들이 없어서 중복된 객체들이 존재하는 PBSM에서 보다 더 적은 수의 객체들을 조인하기 때문이다. TSPJ와 SSSJ은 같은 수의 객체들을 조인하기 때문에 거의 비슷한 수행 시간을 필요로 한다. TSPJ와 SSSJ는 모든 데이타 분포에서 비슷한 성능을 보이는 반면, PBSM은 균일 분포에서의 성능이 떨어지는데, 이는 균일 분포에서 중복된 객체가 더 많이 발생했기 때문이다.

이상을 정리하면, TSPJ의 소트-파티션 과정은 PBSM과 SSSJ의 소트와 파티션 과정보다 빠르고, TSPJ의 조인 과정도 PBSM과 SSSJ의 조인 과정보다 빠르거나 같기 때문에 전체적으로 TSPJ가 빠른 성능을 보인다. 이와 같이 TSPJ가 좋은 성능을 보이는 이유는 PBSM과 SSSJ가 원공간에서 크기가 있는 객체를 다루는 반면, TSPJ는 변환공간에서 크기가 없는 점을 다룸



(a) 균일 분포 (b) 지수 분포 (c) 실제 분포

그림 9 수행 시간 비교

으로써 객체들의 복제가 필요 없고 알고리즘이 상대적으로 간단하여 불필요한 과정들을 생략할 수 있기 때문이다.

5. 결 론

본 논문에서는 변환 공간(transform space) 상에서 색인이 없는 경우에 공간 조인을 수행하는 **변환기반 공간 파티션 조인**(*transformation-based spatial partition join*) 알고리즘을 제안하였다. 변환기반 공간 파티션 조인 알고리즘은 크게 소트-파티션 과정과 조인 과정으로 구성된다. 소트-파티션 과정에서는 조인되는 두 파일 R과 S에 포함된 원공간 상의 객체들을 변환공간 상의 점들로 별도의 추가비용 없이 변환 해석하면서 이 점들을 소트하고, 소트 결과를 파일에 쓰면서 파티션들을 정의한다. 조인 과정에서는 서로 조인 관계에 있는 파티션 쌍들을 조인한다.

다음으로 여러가지 실험을 통해 제안하는 알고리즘의 우수성을 보였다. 제안하는 알고리즘은 공간 객체의 복제를 필요로 하지 않기 때문에 기존 알고리즘들처럼 데이타 분포에 따라 성능이 떨어지는 문제가 없으며, 단순한 파티션 구조를 사용하여 알고리즘의 구성이 간단하기 때문에 기존 알고리즘들에 비해 좋은 성능을 보인다. 실험 결과, 제안하는 알고리즘은 기존 알고리즘들과 비교하여 수행 시간을 20.5~38.0% 줄였다.

참 고 문 헌

[1] T. Brinkhoff, H.-P. Kriegel, and B. Seeger, "Efficient Processing of Spatial Joins Using R-Trees," In *Proc. Int'l Conf. on Management of Data*, ACM SIGMOD, pp. 237-246, 1993.

[2] Y.-W. Huang and N. Jing, "Spatial Joins Using R-trees: Breadth-First Traversal with Global Optimizations," In *Proc. the 23rd Int'l Conf. on Very Large Data Bases*, pp. 396-405, 1997.

[3] M.-L. Lo and C.-V. Ravishankar, "Spatial Joins Using Seeded Trees," In *Proc. Int'l Conf. on Management of Data*, ACM SIGMOD, pp. 209-220, 1994.

[4] M.-L. Lo and C.-V. Ravishankar, "Spatial Hash-Joins," In *Proc. Int'l Conf. on Management of Data*, ACM SIGMOD, pp. 247-258, 1996.

[5] J.-M. Patel and D.-J. Dewitt, "Partition Based Spatial-Merge Join," In *Proc. Int'l Conf. on Management of Data*, ACM SIGMOD, pp. 259-270, 1996.

[6] J. Song, K. Whang, Y. Lee, M. Lee, and S. Kim, "Spatial Join Processing Using Corner Transformation," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 11, No. 4, pp. 688-695, 1999.

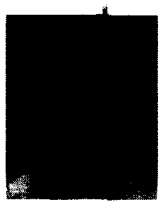
[7] N. Mamoulis and D. Papadias, "Slot Index Spatial Join," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 15, No. 1, pp. 211-231, 2003.

[8] N. Koudas and K.-E. Sevcik, "Size Separation Spatial Join," In *Proc. Int'l Conf. on Management of Data*, ACM SIGMOD, pp. 324-335, 1997.

[9] L. Arge, O. Procopiuc, S. Ramaswamy, T. Suel, and J.-S. Vitter, "Scalable Sweeping-Based Spatial Join," In *Proc. the 24th Int'l Conf. on Very Large Data Bases*, pp. 570-581, 1998.

[10] B. Seeger and H.-P. Kriegel, "Techniques for Design and Implementation of Efficient Spatial Access Methods," In *Proc. the 14th Int'l Conf. on Very Large Data Bases*, pp. 360-371, 1988.

[11] Bureau of the Census, Tiger/Line Precensus Files: 1990 Techical Documentation, Bureau of the Census, Washington, DC, 1989.



이 민 재

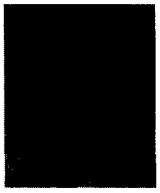
2003년 3월~현재 포스트닥, 첨단정보기술연구센터, KAIST. 2003년 2월 한국과학기술원 전자전산학과 전산학전공 박사. 1997년 2월 한국과학기술원 전자전산학과 전산학전공 석사. 1995년 2월 한국과학기술원 전자전산학과 전산학전공 학사. 관심분야는 지리 정보 시스템, 객체 관계형 데이타베이스 시스템

한 욱 신

2003년 3월~현재 전임강사, 컴퓨터공학과, 경북대학교. 2002년 9월~2003년 2월 연구교수, 첨단정보기술연구센터, KAIST. 2001년 9월~2002년 8월 포스트닥, 첨단정보기술연구센터, KAIST. 1996년 3월~2001년 8월 Ph.D, 전산학과, KAIST. 1994년 3월~1996년 2월 MS, 전산학과, KAIST. 1990년 3월~1994년 2월 BS, 컴퓨터공학과, 경북대학교. 2002년 7월 방문연구원, 미국 HP Labs. 2001년 7월~2001년 8월 Visiting Scholar, 미국 HP Labs

이 재 길

1999년 3월~현재 한국과학기술원 전자전산학과 전산학전공 박사 과정. 1997년 3월~1999년 2월 한국과학기술원 전자전산학과 전산학전공 석사. 1993년 3월~1997년 2월 한국과학기술원 전자전산학과 전산학전공 학사. 2001년 7월~2001년 8월 Visiting Scholar, 미국 HP Labs. 관심분야는 객체 관계형 데이타베이스 시스템, 정보 검색, 질의 최적화, XML, 데이타베이스, 데이타베이스 보안



황 규 영

1973년 서울대학교 전자공학과 졸업(B.S). 1975년 한국과학기술원 전기 및 전자학과 졸업(M.S.). 1982년 Stanford University(M.S.). 1983년 Stanford University(Ph.D.). 1975년~1978년 국방과학연구소(ADD), 선임연구원. 1983년~1990년 IBM T.J. Watson Research Center, Research Staff Member. 1992년~1994년 한국정보과학회 데이타베이스 연구회(SIGDB) 운영위원장. 1995년 한국정보과학회 이사 겸 논문지 편집위원장. 1999년~2000년 한국정보과학회 부회장. Editor-in-Chief: The VLDB Journal, 2003년~현재. Editor: IEEE Transactions on Knowledge and Data Engineering, 2002년~현재. Editor: The VLDB Journal, 1990년~2003년. Editor: Distributed and Parallel Databases: An International Journal, 1991년~1995년. Editor: International Journal of Geographical Information Science, 1994년~현재. Associate Editor: The IEEE Data Engineering Bulletin, 1990년~1993년. 1998년~2004년 Trustee, The VLDB Endowment. 1999년~2005년 Steering Committee Member, DASFAA. 1999년~현재 한국과학기술원 전자전산학과 전산학전공 교수. 1990년~현재 첨단정보기술연구센터(과학재단 우수연구센터) 소장. 관심분야는 데이타베이스 시스템, 멀티미디어, GIS