적응형 행 기준 순서: 변환공간 뷰 조인의 성능 최적화 방법

(Adaptive Row Major Order: a Performance Optimization Method of the Transform-space View Join)

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

요 약 변환공간 색인이란 원공간 상의 공간 객체들을 변환공간 상의 크기가 없는 점들로 변환하여 색인한 후에 이들을 다루는 구조로, 이를 활용하는 조인 알고리즘은 크기가 없는 점들을 다루기 때문에 최적화가 상대적으로 단순하다는 장점을 가진다. 하지만, R 트리와 같은 원공간 색인에는 적용될 수 없는 단점을 가진다. 이러한 단점을 해결하는 방법으로 저자들 은 변환공간 뷰라는 개념을 사용하여 두 원공간 색인들을 변환공간에서 조인하는 **변환공간 뷰 조인 알고리즘**(*transform-space view join algorithm*)을 제안한 바 있다. 여기서 **변환공간 뷰**(*transform-space view*)란 원공간 색인에 대한 가상의 변환공간 색인으로서 이미 구축된 원공간 색인을 구조적으로 변경하지 않고서도 가상의 변환공간 색인으로 해석하여 원공간 색인이 변환공간에서 조인될 수 있게 한다. 변환공간 뷰 조인 알고리즘에서 디스크 페이지 액세스 순서는 공간 채움 곡선에 의해 결정되는데, 이는 조인 성능에 큰 영향을 미친다. 본 논문에서는 변환공간 뷰 조인 알고리즘을 최적화 하는 방법으로 새로운 공간 채움 곡선인 **적응형 행 기준 순서**(*adaptive row major order: ARM order*)를 제안한다. 적응형 행 기준 순서는 주어진 버퍼 크기에 따라 디스크 페이지 액세스 순서를 적응적으로 조정하여 원패스 버퍼 크기(한 페이지 당 한번의 디스크 액세스를 보장하는 최소 버퍼 크기)와 디스크 액세스 횟수를 크게 줄인다. 정형적인 분석과 실험을 통하여 적응형 행 기준 순서를 사용하는 변환공간 뷰 조인 알고리즘의 우수성을 보인다. 실험 결과, 다른 공간 채움 곡선을 사용하는 변환공간 뷰 조인 알고리즘과 비교하여 적응형 행 기준 순서는 원패스 버퍼 크기를 최대 21.3배 줄이고, 디스크 액세스 횟수를 최대 74.6% 줄인다. 또한, R 트리를 원공간에서 조인하는 알고리즘들과 비교하여 원패스 버퍼 크기를 최대 15.7배 줄이고, 디스크 액세스 횟수를 최대 65.3% 줄인다.

키워드 : 공간 조인, 변환 기법, 지리 정보 시스템

Abstract A transform-space index indexes objects represented as points in the transform space. An advantage of a transform-space index is that optimization of join algorithms using these indexes becomes relatively simple. However, the disadvantage is that these algorithms cannot be applied to original-space indexes such as the R-tree. As a way of overcoming this disadvantages, the authors earlier proposed the *transform-space view join algorithm* that joins two original-space indexes in the transform space through the notion of the transform-space view. A *transform-space view* is a virtual transform-space index that allows us to perform join in the transform space using original-space indexes. In a transform-space view join algorithm, the order of accessing disk pages — for which various space filling curves could be used — makes a significant impact on the performance of joins. In this paper, we propose a new space filling curve called *the adaptive row major order* (*ARM order*). The ARM order adaptively controls the order of accessing pages and significantly reduces the one-pass buffer size (the minimum buffer size required for guaranteeing one disk access per page) and the number of disk accesses for a given buffer size. Through analysis and experiments, we verify the excellence of the ARM order when used with the transform-space view join. The transform-space

· 본 연구는 첨단정보기술연구센터를 통하여 한국과학재단으로부터 지원을 받았음
† 비 회 원 : 한국과학기술원 전산학과
mjlee@mozart.kaist.ac.kr
†† 종신회원 : 경북대학교 컴퓨터공학과 교수
wshan@knu.ac.kr
††† 종신회원 : 한국과학기술원 전자전산학과 교수
kywhang@mozart.kaist.ac.kr
논문접수 : 2004년 6월 7일
심사완료 : 2005년 3월 21일

view join with the ARM order always outperforms existing ones in terms of both measures used: the one-pass buffer size and the number of disk accesses for a given buffer size. Compared to other conventional space filling curves used with the transform-space view join, it reduces the one-pass buffer size by up to 21.3 times and the number of disk accesses by up to 74.6%. In addition, compared to existing spatial join algorithms that use R-trees in the original space, it reduces the one-pass buffer size by up to 15.7 times and the number of disk accesses by up to 65.3%.

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. 지금까지 제안된 공간 조인 알고리즘들은 공간 색인을 사용하는 것과 사용하지 않는 것으로 나눌 수 있다. 색인을 사용하지 않는 공간 조인 알고리즘들은 조인되는 어느 파일에도 공간 색인이 구축되어 있지 않거나, 조인되는 한 파일에만 공간 색인이 있는 경우에 유용하다. 그러나, 공간 색인이 양쪽 파일에 이미 구축되어 있는 경우, 이들을 사용하는 알고리즘보다 느린 문제점을 가지고 있다. 따라서, 본 논문에서는 공간 색인을 사용한 조인 알고리즘을 논의 대상으로 한다.

공간 색인을 사용하는 조인 알고리즘들은 사용되는 공간 색인의 종류에 따라 **원공간 색인 조인 알고리즘**(*original-space index join algorithm*)과 **변환공간 색인 조인 알고리즘**(*transform-space index join algorithm*)으로 나뉜다. 원공간 색인 조인 알고리즘은 원공간(original space) 상의 크기를 가지는 공간 객체들을 색인하여 공간 조인을 수행하는 알고리즘이다. 변환공간 색인 조인 알고리즘은 원공간 상의 크기를 가지는 공간 객체들을 변환공간(transform-space) 상의 크기를 갖지 않는 점들로 변환하여 색인한 후에 공간 조인을 수행하는 알고리즘이다.

원공간 색인 조인 알고리즘으로는 **깊이 우선 탐색 R 트리 조인**(*Depth-First Traversal R-Tree Join*)[1]과 **넓이 우선 탐색 R 트리 조인**(*Breadth-First Traversal R-Tree Join*)[2] 알고리즘이 있다. 이 두 알고리즘은 공간 색인으로서 원공간 색인은 R 트리[3-5]를 기반으로 한다. 깊이 우선 탐색 R 트리 조인은 공간 조인의 성능을 비교할 때 기준이 되는 대표적인 알고리즘으로서, 두 개의 R 트리를 깊이 우선 탐색(depth-first traversal)하면서 조인을 수행한다. 이 알고리즘은 local plane sweeping이나 local z-ordering과 같은 휴리스틱을 사용하는데 디스크 페이지 액세스 순서의 일부만을 최적화하기 때문에 전역 최적화를 못하고 지역 최적화만을 수행하는 단점을 가진다. 넓이 우선 탐색 R 트리 조인은 두 개의 R 트리를 넓이 우선 탐색(breadth-first traversal) 하여 전체 디스크 페이지 액세스 순서를 생성 한 다음에 이를 최적화하는 전역 최적화를 수행한다. 이 알고리즘은 조인을 수행하기 전에 디스크 페이지 액세스 순서를 디스크나 메인 메모리에 저장해야 하기 때문에 이를 관리하기 위한 디스크나 메모리 부담을 가지는 단점이 있다. 디스크 액세스 순서를 디스크에 저장하는 경우(*Combo1 strategy*)에는 깊이 우선 탐색 R 트리 조인과 비교하여 큰 버퍼 크기에서의 성능이 떨어지고, 메인 메모리에 저장하는 경우(*Combo2 strategy*)에는 작은 버퍼 크기에서의 성능이 떨어진다[2].

변환공간 색인 조인 알고리즘으로는 **변환 기반 공간 조인**(*Transform-Based Spatial Join*)[6] 알고리즘이 있다. 이 알고리즘은 공간 색인으로서 점을 저장하는 점 액세스 구조인 MLGF[7,8]를 사용한다. 이 알고리즘은 변환공간의 특성을 이용하여 전역 최적화를 하기 때문에 지역 최적화만을 수행하는 깊이 우선 탐색 조인 보다 항상 우수하거나 비슷한 성능을 보인다. 또한, 전역 최적화를 위해 디스크 페이지 액세스 순서를 디스크나 메인 메모리(즉, 버퍼)에 저장하지 않기 때문에 넓이 우선 탐색 조인 보다 작은 디스크 액세스 횟수와 버퍼를 필요로 한다. 그러나, 원공간 색인들을 조인하거나 원공간 색인과 변환공간 색인을 조인하는데 사용될 수 없다는 문제점을 가진다.

원공간 색인 조인 알고리즘과 변환공간 색인 조인 알고리즘은 각각의 장단점을 가진다. 원공간 색 인 조인 알고리즘은 R 트리와 같이 널리 사용되는 원공간 색인을 활용한다는 장점이 있는 반면, 크기를 가진 객체를 다루므로 공간 조인 알고리즘이 상대적으로 복잡하며, 특히 전역 최적화를 하기 위해서는 디스크나 메모리의 부담을 필요로 한다. 이에 반해 변환공간 색인 조인 알고리즘은 크기가 없는 점만을 다루므로 공간 조인 알고리즘이 상대적으로 단순하며 큰 부담 없이 전역 최적화를 수행하는 반면, R 트리와 같이 널리 사용되는 원공간 색인에 적용될 수 없는 단점을 가진다.

이 두 방법의 장점만을 살리는 방법으로서 본 저자들은 **변환공간 뷰**(*transform-space view*)와 **변환공간 뷰 조인 알고리즘**(*transform-space view join algorithm*)

을 제안하였다[9]. 변환공간 뷰는 원공간 색인에 대한 가상의 변환공간 색인이고, 변환공간 뷰 조인 알고리즘은 두 변환공간 뷰들을 조인하는 알고리즘이다. 변환공간 뷰를 통해 R 트리와 같이 원공간상에 이미 구축된 원공간 색인은 구조적 변경과 별도의 추가비용 없이 가상의 변환공간 색인으로 해석되며, 변환공간 뷰 조인 알고리즘을 통해 변환공간 상에서 서로 조인된다.

변환공간 뷰 조인 알고리즘은 다양한 공간 채움 곡선에 의해 결정되는 디스크 페이지 액세스 순서에 따라 성능이 크게 변화는 특징을 가진다. 본 저자들의 논문[9]에서는 공간 채움 곡선으로 널리 알려진 힐버트 순서(hilbert order)[10,11]를 선택하였다. 그러나, 이 논문[9]에서 저자들은 공간 채움 곡선과 성능간의 관계에 대한 정형적인 분석과 검증을 수행하지 못했기 때문에, 힐버트 순서가 변환공간 뷰 조인에 적합한지와 만약 적합하지 않다면 어떤 순서가 적합한지를 밝혀내지 못했다.

본 논문에서는 다양한 공간 채움 곡선들을 사용하는 경우의 성능에 대한 정형적인 분석을 수행하고, 그 결과로서 변환공간 뷰 조인의 성능을 대폭 향상시키는 새로운 공간 채움 곡선인 **적응형 행 기준 순서**(*adaptive row major order: ARM order*)를 제안한다. 성능의 척도로는 **원패스 버퍼 크기**(*one-pass buffer size*)와 **디스크 액세스 횟수**(*number of disk accesses*)를 사용한다. 여기서, 원패스 버퍼 크기란 공간 조인에 참여하는 각 페이지들이 단 한번만 읽히는 것을 보장하는 최소 버퍼 크기이다. 적응형 행 기준 순서는 주어진 버퍼 크기에 따라 디스크 페이지 액세스 순서를 적응적으로 조정한다. 실험을 통 해 제안된 방법이 기존의 다른 방법들에 비해 우수함을 보인다. 실험 결과, 다른 공간 채움 곡선을 사 용하는 변환공간 뷰 조인 알고리즘과 비교하여 적응형 행 기준 순서는 원패스 버퍼 크기를 최대 21.3배 줄이고, 디스크 액세스 횟수를 최대 74.6% 줄인다. 또한, 원공간 색인 조인 알고리즘들과 비교하여 원 패스 버퍼 크기를 최대 15.7배 줄이고, 디스크 액세스 횟수를 최대 65.3% 줄인다.

본 논문의 구성은 다음과 같다. 제2장에서는 관련 연구로서 변환공간 뷰와 변환공간 뷰 조인 알고리즘에 대해 설명한다. 제3장에서는 변환공간 뷰 조인 알고리즘에서 공간 채움 곡선과 원패스 버퍼 크기사이의 관계에 대해 논하고, 제4장에서는 원패스 버퍼 크기와 디스크 액세스 횟수를 동시에 최적화하는 새로운 공간 채움 곡선인 적응형 행 기준 순서를 제안한다. 제5장에서는 적응형 행 기준 순서를 사용하는 변환공간 뷰 조인 알고리즘의 성능 평가를 수행하고 그 결과를 제시한다. 마지막으로 제6장에서는 결론을 내린다.

2. 관련 연구

본 장에서는 변환공간 뷰와 변환공간 뷰 조인 알고리즘[9]에 대해 설명한다. 제2.1절에서는 변환공간 뷰에 대해 설명하고, 2.2절에서는 변환공간 뷰 조인 알고리즘에 대해 설명한다.

2.1 변환공간 뷰

변환공간 뷰는 원공간 색인에 대한 가상의 변환공간 색인으로, 변환기법을 사용하여 정의된다. 여기서 원공간 색인은 n차원 원공간상의 공간 객체를 색인하는 구조이고, 변환공간 색인은 $2n$차원 변환공간 상의 점 객체를 색인하는 구조이다. 변환공간 뷰의 장점은 성능 좋은 변환공간 색인 조인 알고리즘을 원공간 색인의 구조적인 변경 없이 원공간 색인에 직접 적용할 수 있게 한다는 점이다. 변환공간 뷰가 제안되기 전에는 원공간 색인에 변환공간 색인 조인 알고리즘이 적용될 수 없는 것으로 알려져 왔다. 그러나, 변환공간 뷰의 제안을 통해 이러한 적용이 가능케 되었다. 그림 1은 변환공간 뷰를 통해 원공간 색인에 변환공간 색인 조인 알고리즘을 적용하는 과정을 보인다. 그림에서 두 원공간 색인 R과 S는 변환공간 뷰 TV(R)과 TV(S)로 동적으로 해석되며,

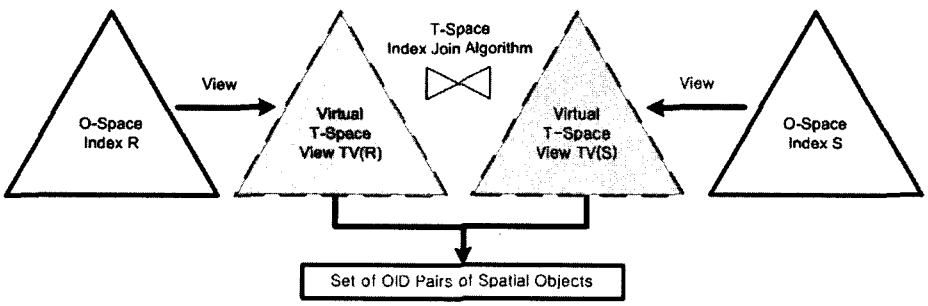


그림 1 변환공간 뷰를 통한 변환공간 조인 처리 과정

TV(R)과 TV(S)에 변환공간 색인 조인 알고리즘이 적용된다.

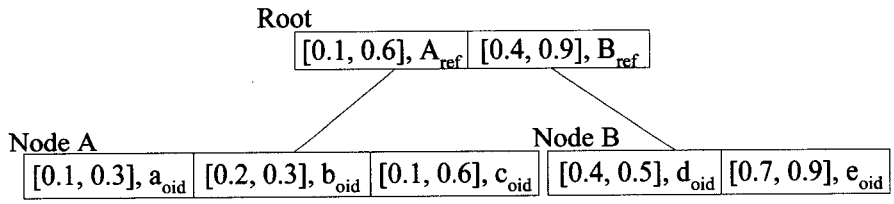
변환공간 뷰는 원공간 색인의 각 구조를 변환공간 색인의 구조로 해석함으로써 정의된다. 정의 1은 n차원 원공간 색인을 $2n$차원 변환공간 뷰로 해석하는 방법을 정형적으로 정의한다. 변환기법으로서는 구석점 변환기법을 사용한다. **구석점 변환기법**(*corner transformation*)이란 n차원 원공간상의 공간 객체를 $2n$차원 변환공간상의 점 객체로 변환하는 기법으로, 변환시 n차원 원공간 객체의 각 차원 축에 대한 최소값과 최대값을 사용하여 $2n$차원 점 객체의 좌표값을 구성한다.

정의 1 (변환공간 뷰). n차원 원공간 색인에서 원공간 객체와 원공간 영역이 각각$[l1, r1] \times [l2, r2] \times \ldots \times [ln, rn]$의 형태로 표현된다고 하자. 여기서 li는 원공간의 i번째 차원축에 대한 객체 혹은 영역의 최소값을 뜻하며, ri는 최대값을 뜻한다. n차원 원공간 색인에 대한 $2n$차원 변환공간 뷰는 다음과 같이 정의된다.

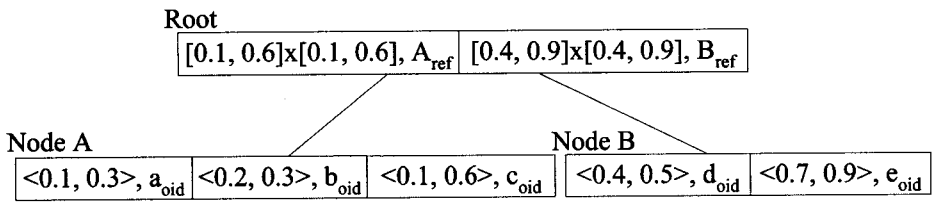
(a) **원공간 객체의 매핑**: *n차원 원공간 색인에서 원공간 객체*$[l1, r1] \times [l2, r2] \times \ldots \times [ln, rn]$은 $2n$차원변환공간 뷰의 점 $<l1, r1, l2, r2, \ldots, ln, rn>$으로 매핑된다.

(b) **원공간 영역의 매핑**: n차원 원공간 색인에서 원공간 영역$[l1, r1] \times [l2, r2] \times \ldots \times [ln, rn]$은 $2n$차원 변환공간 뷰의 영역$[l1, r1] \times [l1, r1] \times [l2, r2] \times [l2, r2] \times \ldots \times [ln, rn] \times [ln, rn]$으로 매핑된다.

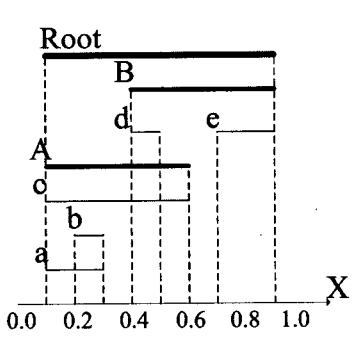
예 1: 그림 2는 1차원 원공간 색인인 R 트리에 대한 2차원 변환공간 뷰를 나타낸다. 그림 2(a)는 1차원 R 트리 구조를 표현한 것이고, 그림 2(b)는 이로부터 정의된 2차원 변환공간 뷰를 표현한 것이다. 그림 2(c)는 (a)의 객체들과 영역들을 1차원 원공간상에서 표현한 것이며, 그림 2(d)는 (b)의 점 객체들과 영역들을 2차원 변환공간상에서 표현한 것이다. 그림 2(a)의 원공간 색인은 두 개의 단말 페이지 A, B와 Root로 구성되고 단말 페이지들에는 원공간 객체 a, b, c, d, e가 저장된다.



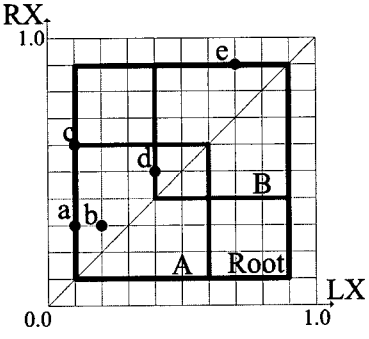
(a) 1차원 R 트리



(b) 1차원 R 트리로부터 정의된 2차원 변환공간 뷰.



(c) 1차원 원공간에서 R 트리 구조의 표현



(d) 2차원 변환공간에서 변환공간 뷰 구조의 표현.

그림 2 원공간 색인 구조에 대한 변환공간 뷰로의 매핑 예

원공간 객체와 영역의 변환공간 점과 영역으로의 매핑은 다음과 같다. 그림 2(c)에서 [0.1, 0.3]으로 표현된 원공간 객체 a는 정의 1(a)에 의해 그림 2(d)에서 변환공간 점 <0.1, 0.3>으로 매핑된다. 나머지 원공간 객체 b, c, d, e들도 이와 마찬가지 방법으로 매핑된다. 그림 2(c)의 [0.1, 0.6]으로 표현된 단말 페이지 A의 영역은 정의 1(b)에 의해 그림 2(d)에서 변환공간 영역 [0.1, 0.6] ×[0.1, 0.6]으로 매핑된다. [0.4, 0.9]로 표현된 단말 페이지 B의 영역은 변환공간 영역 [0.4, 0.9] ×[0.4, 0.9]으로 매핑된다. □

2.2 변환공간 뷰 조인 알고리즘

변환공간 뷰 조인 알고리즘은 두 원공간 색인 R과 S에 대한 변환공간 뷰 TV(R)과 TV(S)를 조인하는 알고리즘이다. 변환공간 뷰 TV(R)과 TV(S)의 변환공간을 각각 TS(R)과 TS(S)라 할 때, 알고리즘은 TS(R)의 영역과 서로 조인 관계에 있는 TS(S)의 영역들을 찾아 이들에 포함된 객체들을 서로 조인한다. 서로 조인관계에 있는 영역들은 공간 조인 윈도우[6]를 사용하여 찾는다. **공간 조인 윈도우**(*spatial join window*) *SJW(P)*란 TS(R)에 있는 사각형 영역 P에 포함될 수 있는 모든 객체들과 교차관계를 가지는 객체들이 존재하는 TS(S)내의 최소 영역이다. **공간 조인 윈도우 페이지**(*spatial join window pages*) *SJWP(P)*를 SJW(P)와 영역들이 겹치는 페이지들의 집합으로 정의할 때, TS(R)의 영역 P와 대 응되는 페이지는 SJWP(P)의 페이지들과 서로 조인관계를 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. □

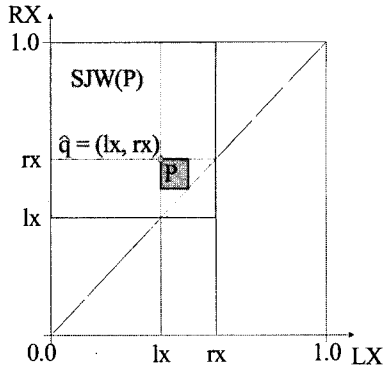
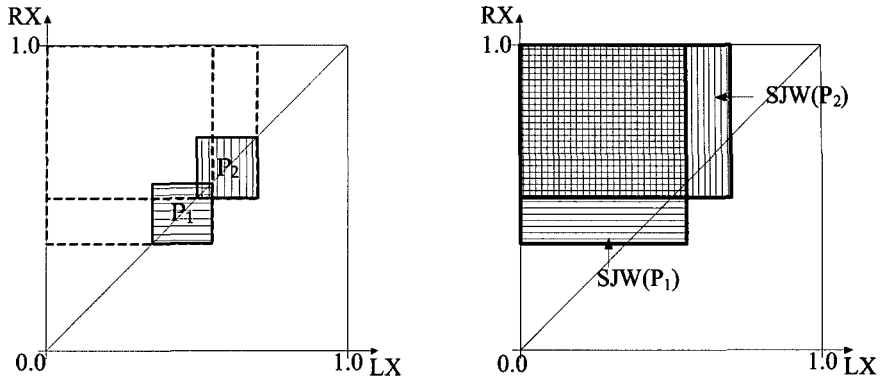


그림 3 공간 조인 윈도우

예 2: 그림 3은 TS(R)내의 사각형 영역 P에 대한 TS(S)내의 영역 SJW(P)를 나타낸다. 그림 3에서 영역 P는 짙은 사각형으로, SJW(P)는 옅은 사각형으로 표시된다. □

세로로 인접한 TS(R)의 두 영역 P_1과 P_2의 공간 조인 윈도우 SJW(P_1)과 SJW(P_2)사이에는 매우 큰 겹침이 존재하는데, 변환공간 뷰 조인 알고리즘은 이 특성을 사용하여 디스크 액세스 횟수를 크게 줄인다. 그림 4는 세로로 인접한 두 영역 P_1과 P_2의 공간 조인 윈도우 SJW(P_1)과 SJW(P_2)를 나타낸다. 그림에서 보듯이 이 두 공간 조인 윈도우는 서로 크게 겹친다. 만약, P_1과 대응되는 페이지를 SJWP(P_1)과 조인한 후에 P_2와 대응되는 페이지를 SJWP(P_2)와 조인하면, SJWP(P_2)의 대부분은 이미 버퍼에 있으므로 적은 수의 디스크 액세스로 조인을 처리할 수 있다.

이러한 공간 조인 윈도우의 특성을 이용하여 그림 5에서와 같이 **변환공간 뷰 조인**(*transform-space view join: TSVJ*) 알고리즘이 정의된다. 알고리즘은 변환공간 뷰 TV(R) 및 TV(S), 공간 채움 곡선 SFC를 입력으로 받는다. **공간 채움 곡선**(*space filling curve*)이란 공간상의 영역들을 선형적으로 순서화하는 방법으로, 여기서는 영역의 선택 순서를 결정하기 위해 사용된다. 2차원 변환공간의 경우, 세로축(RX)을 SFC로서 사용하여 영역들을 세로로 인접하게 선택한다. 세로로 인접하게 선택된 영역들은 공간 조인 윈도우들 간의 겹침을 크게 만들기 때문에, 적은 수의 디스크 액세스로 조인이 처리되도록 한다. *2n*차원 변환공간의 경우에는 *n*개의 세로축들의 카티션 곱(cartesian product)으로 구성된 *n*차 원공간에서의 공간 채움 곡선을 사용한다. 이는 2차원 변환공간 뷰의 조인을 위해 한 개의 세로축으로 구성된 1차원 공간에서의 공간 채움 곡선을 사용하던 것을 *2n*차원 변환공간 뷰의 조인을 위해 *n*차 원공간에



(a) 세로로 인접한 두 영역 P_1과 P_2 (b) SJW(P_1)과 SJWP(P_2)

그림 4 세로로 인접한 두 영역들의 공간 조인 윈도우들

```
Algorithm TransformSpaceViewJoin(TV(R), TV(S), SFC)
    /* create sfc_priority_queue , which is a priority queue whose elements
       are ordered by SFC */
1   SFCPriorityQueue sfc_priority_queue(SFC)
    /* enqueue the region of the root page of TV(R) into sfc_priority_queue */
2   sfc_priority_queue.Enqueue(GetRootPageRegion(TV(R)))
3   while not sfc_priority_queue.IsEmpty() do
        /* dequeue a region r_region from sfc_priority_queue */
4       r_region = sfc_priority_queue.Dequeue()
5       let r_page be the page corresponding to r_region
        /* if r_page is a leaf page, join r_page with every page in the set of
           pages in SJW(r_region) */
6       if r_page is a leaf page then
7           for each leaf page s_page in the set of pages in SJW(r_region) in TV(S) do
8               JoinPage(r_page, s_page)
9           end
        /* if r_page is a non-leaf page, and the set of pages in SJW(r_region) is not empty,
           enqueue all the regions of the children in r_page into sfc_priority_queue */
10      else if r_page is a non-leaf page and the set of pages in SJW(r_region) in TV(S) ≠ø then
11          for each region child_page_region in r_page do
12              sfc_priority_queue.Enqueue(child_page_region)
13          end
14      end
```

그림 5 Transform-Space View Join (TSVJ) 알고리즘

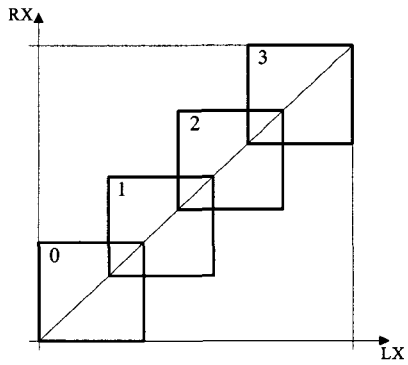
서의 공간 채움 곡선을 사용하는 것으로 일반화한 것이다. 힐버트 순서[10,11]는 공간을 가장 잘 인접하게 순서화하는 것으로 알려져 있기 때문에, 참고 문헌[9]에서는 공간 채움 곡선으로서 힐버트 순서를 사용한다. 알고리즘의 자세한 설명은 참고문헌[9]에 있으므로 생략한다.

3. 공간 채움 곡선과 원패스 버퍼 크기

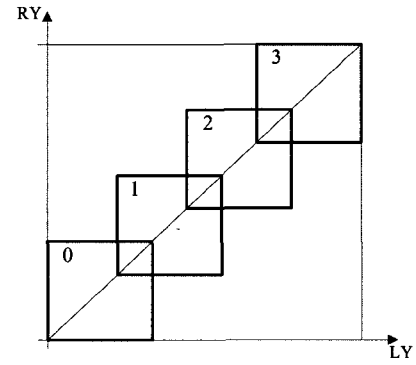
변환공간 뷰 조인 알고리즘에서 공간 채움 곡선에 의해 정해지는 디스크 액세스 순서는 성능에 큰 영향을 미친다[1,2,6]. 본 장에서는 널리 알려진 여러 공간 채움 곡선들에 대해 이들이 변환공간 뷰 조인 알고리즘의 성능에 미치는 영향을 분석한다. 성능의 척도로는 원패스 버퍼 크기(한 페이지 당 한번의 디스크 액세스를 보장하는 최소 버퍼 크기)와 디스크 액세스 횟수를 사용한다. 다음 장에서는 이러한 분석을 바탕으로 원패스 버퍼 크기를 최소화하고 동시에 디스크 액세스 횟수를 최소화하는 새로운 공간 채움 곡선을 제안한다.

분석을 용이하게 하기 위해 데이타 및 페이지 분포에 대해 다음과 같은 가정을 한다. 즉, 데이타 분포는 2차원 원공간에서 공간 객체들의 크기와 위치에 대해 균일하며, 이러한 분포를 가지는 객체들을 색인하는 원공간 색인의 단말 페이지들에 대응되는 영역들은 같은 크기를 가지며 그 중심점은 균일 분포를 가진다고 가정한다. 이 가정을 **균일 데이타 및 페이지 분포 가정**이라고 부르기로 한다. 제5장에서는 실험을 통해 균일 데이타 및 페이지 분포 가정을 만족하지 않는 분포와 실제 응용에 대하여도 유사한 결과를 가짐을 보인다. 마지막으로 상용 DBMS에서 널리 사용되는 LRU를 버퍼 교체 전략으로 사용한다.

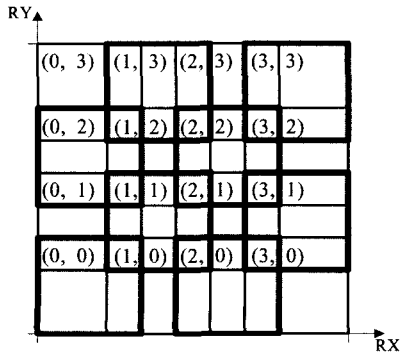
먼저, 4차원 변환공간 상의 페이지들을 2차원 좌표 값으로 식별하는 방법에 대해 설명한 다음에 이를 공간 채움 곡선으로 순서화하는 방법을 설명한다. 그림 6(a)와 (b)는 2차원 원공간의 X축과 Y축 에 대응되는 4차원 변환공간의 LX-RX 평면과 LY-RY 평면을 나타낸다. 변환공간 뷰 조인 알고리즘은 제2.2절에서 설명하였듯이 두 세로축 RX와 RY의 카티션 곱으로 구성된 2차원 공간에서 페이지들과 연관된 영역들을 순서화하므로, 그림 6(a)와 (b)의 4차원 변환공간을 RX-RY 평면에 투영을 한다. RX-RY 평면에서 RX 축의 좌표 값이 u이고 RY축의 좌표 값이 v인 영역에 2차원 좌표 값 (u, v)를 각 각 부여하면, 각 영역들은 2차원 좌표값을 통해 식별된다. 각 영역들은 페이지들과 대응되므로, 각 페이지들은 영역들에 부여된 좌표 값에 의해 식별된다. 그림 6(d)는 그림 6(c)의 영역들을 구별이 쉽도록 배열 형태



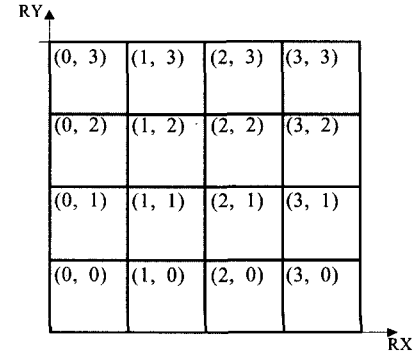
(a) LX-RX 평면



(b) LY-RY 평면



(c) RX-RY 평면



(d) 배열 행태로 표현된 RX-RY 평면의 영역들

그림 6 4차원 변환공간

로 표현한 그림이다. 앞으로 이 배열 행태로 표현된 영역들을 사용하여 설명을 진행한다.

공간 채움 곡선 C는 **선형화 함수**(*linearization function*) L_C를 통해 주어진 위치의 페이지가 몇 번째로 공간에 채워지는 지를 표현한다. 선형화 함수 L_C는 $L_C(u, v) = i$ 형태의 함수로서, (u, v)에 위한 페이지가 i번째 순서를 가짐을 의미한다. 선형화 함수는 페이지마다 고유 번호를 부여하므로, $(u, v) \neq (u', v')$이면 $L_C(u, v) \neq L_C(u', v')$이다.

대표적인 공간 채움 곡선의 예로서 **행 기준 순서**(*row major order: RM order*)를 들 수 있다. 그림 7은 페이지들의 총 수 n이 64인 경우에 페이지들을 행 기준 순서로 순서화한 것이다. 행 기준 순서에 대한 선형화 함수 L_{RM} 은 수식 (1)과 같다.

$$L(u,v) = v \times |\sqrt{n}| + u \quad (1)$$

다음으로, 주어진 공간 채움 곡선에 대한 겹침 페이지

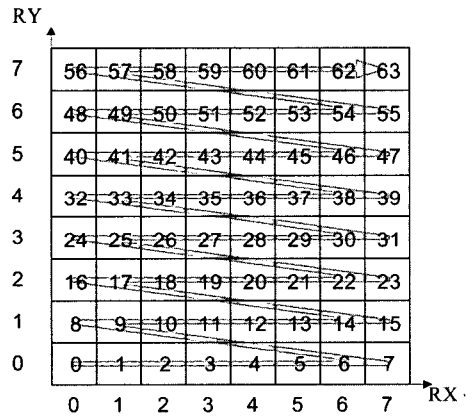


그림 7 행 기준 순서

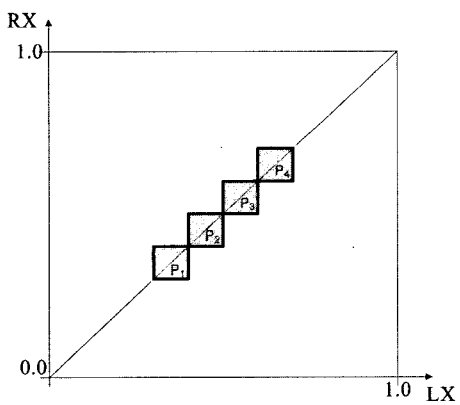
간 최대 선형 거리를 정의한 다음에 원패스 버퍼 크기를 계산하는 방법을 설명한다. **겹침 페이지간 최대 선형 거리**(*MAXLDIST*라 부른다)는 페이지들의 집합 SJWP(Ps)와 SJWP(Pe)가 겹치는 임의의 두 페이지 Ps와 Pe간의 선형 거리들 중의 최대값으로 정의된다. 여기서, SJWP(Ps)와 SJWP(Pe)는 각각 SJW(Ps)와 SJW(Pe)에 포함된 페이지들의 집합들이다. 두 페이지 Ps와 Pe간의 선형 거리는 Ps의 좌표 값이 (us, vs)이고 Pe의 좌표 값이 (ue, ve)일 때, $|LC(ue, ve) - LC(us, vs)| + 1$로 계산된다.

원패스 버퍼 크기는 Ps로부터 Pe까지 공간 조인을 하면서 읽는 페이지들의 합과 같다. 만약 이 보다 작은 크기의 LRU 버퍼가 주어지면, 버퍼 교체로 인해 Ps를 공간 조인할 때 읽었던 페이지들이 Pe를 조인할 때까지 버퍼에 있음을 보장할 수 없어, 원패스를 보장하지 못한다. 두 페이지 Ps와 Pe의 선형 거리가 MAXLDIST일 때, 수식 (2)는 원패스 버퍼 크기를 나타낸다.

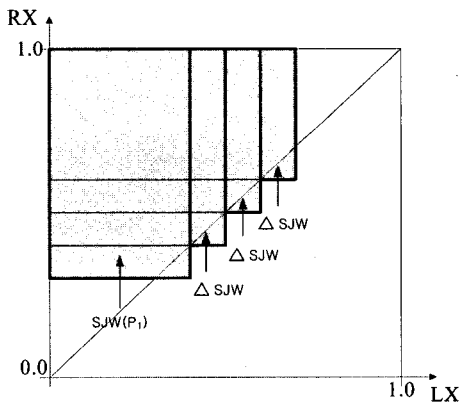
$$(1+|SJWP(P_s)|) + \sum_{i=1}^{MAXLDIST-1} (1+AVG(|\Delta SJWP|)) \quad (2)$$

수식 (2)에서 (1+|SJWP(Ps)|)는 페이지 Ps와 SJWP(Ps)에 포함된 페이지들의 개수이고, $(1+AVG(|\Delta$ SJWP|)는 페이지 Ps 이후에 선택된 페이지 P_i와 SJWP(P_i)에 포함된 페이지들 중에서 추가적으로 읽어 들인 페이지 개수의 평균값을 나타낸다.

예 3: 그림 8은 페이지 P_1, P_2, P_3, P_4와 대응되는 TS(R)의 영역들과 TS(S)에 있는 그들의 공간 조인 윈도우들을 나타낸다. 만약 P_1과 P_4의 선형 거리가 MAXLDIST일 때, 그림 8에 나타난 영역들에 포함 된 페이지들의 개수는 P_4에 대한 조인이 끝날 때까지 페이지 교체가 없음을 보장하는 버퍼 크기, 즉 원패스 버퍼



(a) TS(R)의 영역들



(b) TS(S)의 공간 조인 윈도우들

그림 8 P_1, P_2, P_3, P_4에 대한 공간 조인을 처리할 때, 읽어 들이는 페이지들이 존재하는 영역

크기를 나타낸다. 영역들 안에 포함된 페이지들의 개수, 즉 P_1, P_2, P_3, P_4를 공간 조인하기위해 읽어 들이는 페이지들의 개수는 $(1+|SJWP(P_1)|)+\sum_{i=1}^{3}(1+AVG(|\Delta SJWP|)$로 계산된다. 즉, P_1에 대한 공간 조인이 수행될 때, (1 + |SJWP(P_1)|)개의 페이지들이 읽히고, P_2, P_3, P_4에 대한 공간 조인이 수행될 때, (1 + AVG(|ΔSJWP|))개의 페이지들이 각각 읽힌다. □

다음으로, MAXLDIST의 크기에 영향을 미치는 겹침 페이지간 인접도를 정의한다.

정의 2 (겹침 페이지간 인접도). 색인에 대한 **겹침 페이지간 인접도**는 주어진 색인 안에서 SJWP가 서로 겹치는 모든 두 페이지 (u_1, v_1)와 (u_2, v_2)에 대해 $|u_1 - u_2| \le c$ and $|v_1 - v_2| \le c$을 만족하는 c의 최소값으로 정의된다. 즉, $|u_1 - u_2| > c$ or $|v_1 - v_2| > c$을 만족하는 두 페이지 (u_1, v_1)와 (u_2, v_2)의 SJWP는 서로 겹치지 않는다.

설명의 편의를 위해, 겹침 페이지간 인접도가 1일 때, 대표적인 공간 채움 곡선들에 대한 MAXLDIST의 계산 방법을 설명한 후에 겹침 페이지간 인접도가 1보다 클 때의 계산 방법을 설명한다. 본 논문에서는 대표적인 공간 채움 곡선들로는 행 기준 순서, Z 순서[12], 그리고 힐버트 순서[10]를 고려한다. 나머지 공간 채움 곡선들에 대해서도 이와 유사한 방법으로 계산할 수 있다.

보조정리 2 (행 기준 순서의 MAXLDIST). 균일 데이타 및 페이지 분포 가정 하에서 겹침 페이지간 인접도가 1이고 페이지의 총수 n이 4보다 크거나 같을 때, 행 기준 순서에 대한 $MAXLDIST_{RM}$은 $\lfloor\sqrt{n}\rfloor+2$이다.

증명: Appendix A 참고. □

보조정리 3 (Z 순서의 MAXLDIST). 균일 데이타 및 페이지 분포 가정 하에서 겹침 페이지간 인접도가 1이고 페이지의 총수 n이 4보다 크거나 같을 때, Z 순서에 대한 $MAXLDIST_Z$는 $\frac{n}{2}+2$ 이다.

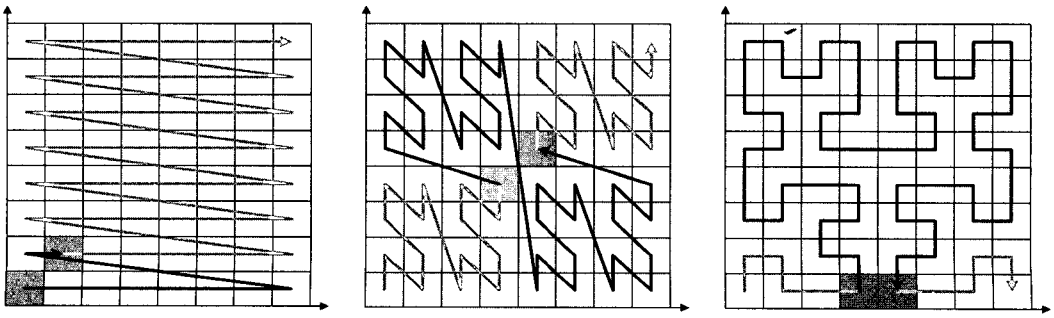
증명: Appendix B 참고. □

보조정리 4 (힐버트 순서의 MAXLDIST). 균일 데이타 및 페이지 분포 가정 하에서 겹침 페이지간 인접도가 1이고 페이지의 총수 n이 4보다 크거나 같을 때, 힐버트 순서에 대한 $MAXLDIST_{Hilbert}$는 $\frac{10n+8}{12}$이다.

증명: Appendix C 참고. □

예 4: 그림 9는 n이 64일 때의 행 기준 순서, Z 순서, 그리고 힐버트 순서의 MAXLDIST를 나타낸 것이다. 선형 거리가 MAXLDIST인 두 페이지들은 회색 사각형으로 나타나 있으며, 두 페이지를 연결하는 곡선은 짙은 선으로 표시되어 있다. 이 짙은 선의 길이가 MAXLDIST가 된다. 행 기준 순서의 경우, MAXLDIST는 $\sqrt{64}+2=10$이고, Z 순서의 경우, $\frac{64}{2}+2=34$이고, 힐버트 순서의 경우, $\frac{10\cdot 64+8}{12}=54$이다. □

겹침 페이지간 인접도가 c > 1인 경우의 MAXLDIST는 다음과 같이 계산된다. 행기준 순서의 경우, max($|u_1 - u_2|$)와 max($|v_1 - v_2|$)가 c 이므로, 수식 Appendix A의 식 (5)로부터 $MAXLDIST_{RM} = c\times\lceil\sqrt{n}\rceil+c+1$ 이 구해진다. Z 순서와 힐버트 순서의 경우, 우선 겹침 페이지간 인접도가 1이 되도록 인접한 페이지들(원 페이지들)을 하나의 가상 페이지로 묶는다. 정형적으로 설명하면, 겹침 페이지간 인접도 c에 대해, 모든 $4^{\lfloor\log_2 c\rfloor}$개의 가로와 세로로 인접하고 선형화 함수에 의해 연속적으로 번호를 부여 받은 원 페이지들을 하나의 가상의 페이지로 묶는다. 이와 같이 묶으면, 총 $\frac{n}{4^{\lfloor\log_2 C\rfloor}}$개의 가상 페이지들이 만들어진다. 그 다음, 가상 페이지들에서 MAXLDIST′을 구한 후에 구해진 값을 $4^{\lfloor\log_2 c\rfloor}$ 배



(a) 행 기준 순서 (b) Z 순서 (c) 힐버트 순서

그림 9 대표적인 공간 채움 곡선들에 대한 MAXLDIST들

하여 원 페이지들에서의 MAXLDIST를 구한다. 이때, 모든 $4^{\lfloor \log_2 c \rfloor}$개의 원 페이지들은 한 개의 가상 페이지로 묶이면서, 순서화 함수로부터 같은 값을 부여 받아 MAXLDIST′의 계산에 사용되기 때문에, 구해진 MAXLDIST에는 오차가 존재할 수 있다. 그러나, 이 오차는 하나의 가상 페이지로 묶인 원 페이지들의 개수, 즉 $4^{\lfloor \log_2 c \rfloor}$ 미만이다.

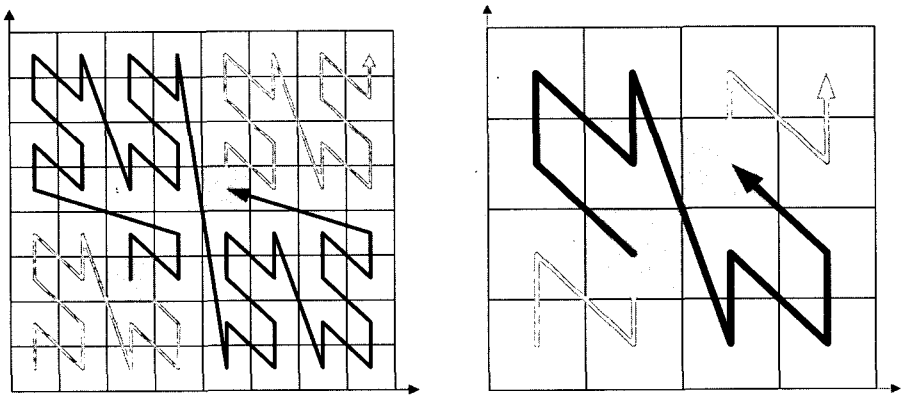
예 5: 그림 10은 겹침 페이지간 인접도가 2인 페이지들의 겹침 페이지간 인접도를 1로 변환하여 MAXLDIST를 구하는 과정을 보인다. 겹침 페이지간 인접도가 2인 그림 10(a)의 모든 가로와 세로로 인접하고 연속되게 순서화된 페이지들을 그림 10(b)에서와 같이 하나의 가상 페이지로 묶으면, 겹침 페이지간 인접도는 1이 되고, 총 가상 페이지들의 수는 16이 된다. 그 다음, 그림 10(b)에서 $MAXLDIST'_Z$를 구하면, $MAXLDIST'_Z$는 $\frac{16}{2}+2=10$이 된다. 구해진 $MAXLDIST'_Z$는 $MAXLDIST_Z$보다 4배 작은 값이므로, $MAXLDIST_Z$는 40이다. 이때, 오차가 4 미만으로 나타날 수 있기 때문에 실제 $MAXLDIST_Z$는 37에서 40사이의 값을 가진다. 그림 10(a)에서의 실제 $MAXLDIST_Z$은 예측된 범위 내의 값인 37이다. □

보조정리 2, 3, 그리고 4를 통해 지금까지 행 기준 순서, Z 순서, 그리고 힐버트 순서의 MAXLDIST를 구하였다. 구해진 MAXLDIST들은 각각 $\lceil \sqrt{n} \rceil + 2$, $\frac{n}{2}+2$, 그리고 $\frac{10n+8}{12}$이다. 이들을 살펴보면 행기준 순서는 $O(\sqrt{n})$이고 Z 순서와 힐버트 순서는 $O(n)$임을 알 수 있다. 따라서, $n \gg 1$의 경우, 행 기준 순서는 가장 작은 원패스 버퍼 크기를 가진다. 그러므로, 행 기준 순서로 공간 조인하는 것이 가장 작은 원패스 버퍼 크기를 가져 유리하다. 하지만, 버퍼가 원패스 버퍼 보다 작게 주어진 경우, 행 기준 순서는 급격히 많은 디스크 액세스를 유발하는 문제를 가지는데, 다음 장에서는 이 문제에 대한 분석과 해결책을 제시한다. 본 논문에서는 이 문제에 대한 해결책으로서 새로운 공간 채움 곡선인 적응형 행 기준 순서를 제안한다.

4. 적응형 행 기준 순서

제3장에서 살펴보았듯이 대표적인 공간 채움 곡선들 중에서 가장 작은 원패스 버퍼 크기를 필요로 하는 곡선은 행 기준 순서이다. 그러나, 행 기준 순서는 원패스 버퍼보다 작은 버퍼에서 급격히 많은 디스크 액세스를 유발하는 문제점을 가진다. 이에 대한 분석은 다음과 같다.

행 기준 순서에서는 주어진 버퍼가 원패스 버퍼 보다 조금이라도 작더라도, 모든 페이지들의 SJWP들을 다시 읽는 일이 발생할 수 있다. 겹침 페이지간 인접도가 c일 때, (u, v)를 i번째 페이지라 하면 $(u+c, v+c)$는 $(i+MAXLDIST_{RM})$번째 페이지가 된다. 이 두 페이지들 사이의 선형 거리는 $MAXLDIST_{RM}$이기 때문에, 수식 (2)와 $MAXLDIST_{RM}$에 의해 계산되는 버퍼 크기보다 작은 버퍼가 주어지는 경우, $(u+c, v+c)$의 조인을 처리할 때, (u, v)의 SJWP는 버퍼로부터 교체되어 나간다. 그런데, 겹침 페이지간 인접도가 c이기 때문에 $(u+c, v+c)$의 SJWP는 (u, v)의 SJWP와 겹치므로 (u, v)의 SJWP를 디스크로부터 다시 읽어야 한다. 따라서, 만약 주어진 LRU 버퍼가 $MAXLDIST_{RM}$에 의해 결정되는 원패스 버퍼 크기보다 작을 경우에는 모든 i에 대하여 $(i+MAXLDIST_{RM})$번째 페이지의 조인을 처리할 때마



(a) 원 페이지들에 대해 겹침 페이지간 인접도가 2인 경우 (b) 가상 페이지들에 대해 겹침 페이지간 인접도가 1인 경우

그림 10 가상 페이지들의 겹침 페이지간 인접도

다, i번째 페이지의 SJWP를 디스크로부터 다시 읽어야 한다.

페이지 (u, v)와 $(u+c, v+c)$사이의 행 기준 순서에 의한 선형 거리는 행 기준 순서의 가로 폭인 $\lfloor\sqrt{n}\rfloor$에 비례하는데, 이 가로 폭의 조정을 통해 행 기준 순서가 가지는 문제점을 해결할 수 있다. 본 논문은 행 기준 순서가 가지는 문제의 해결책으로 정의 3으로 정의되는 새로운 공간 조인 채움 곡선인 **적응형 행 기준 순서**(*adaptive row major order: ARM order*)를 제안한다.

정의 3 (적응형 행 기준 순서). 적응형 행 기준 순서에 대한 선형화 함수 L_{ARM}은 수식 (3)과 같다. 여기서 $\frac{\lfloor\sqrt{n}\rfloor}{k}$는 적응형 행 기준 순서의 가로폭이다.

$$L_{ARM}(u,v)=\begin{cases} v\times\frac{\lfloor\sqrt{n}\rfloor}{k}+u \\ \qquad \text{if } u<\frac{\lfloor\sqrt{n}\rfloor}{k} \text{ and } k>0 \\ \cdots \\ v\times\frac{\sqrt{n}}{k}+(u-\frac{i\lfloor\sqrt{n}\rfloor}{k})+\frac{i\cdot n}{k} \text{ for } 1\le i<k-1 \\ \qquad \text{if } \frac{i\lfloor\sqrt{n}\rfloor}{k}\le u<\frac{(i+1)\lfloor\sqrt{n}\rfloor}{k} \text{ and } k>2 \\ \cdots \\ v\times\frac{\sqrt{n}}{k}+(u-\frac{(k-1)\lfloor\sqrt{n}\rfloor}{k})+\frac{(k-1)n}{k} \\ \qquad \text{if } u\ge\frac{(k-1)\lfloor\sqrt{n}\rfloor}{k} \text{ and } k>1 \end{cases} \quad (3)$$

예 6 : 수식 (4)와 그림 11은 n = 36, k = 2인 경우의 적응형 행 기준 순서의 선형화 함수와 모양을 나타낸 것이다. 일반적으로, 적응형 행 기준 순서는 행 기준 순서로 순서화된 k개의 구성 요소들이 가로로 붙은 형태를 가지며, 각 구성 요소의 가로폭은 $\frac{\lfloor\sqrt{n}\rfloor}{k}$이다. 예에서의 적응형 행 기준 순서의 모양은 각각 행 기준 순서로 순서화된 두 개의 구성 요소들이 가로로 붙어 있는 형태를 갖는데, 왼쪽 부분(u < 3)은 선형화 함수 $v \times 3 + u$에 의해 가로폭이 3인 행 기준 순서이고, 오른쪽 부분($u \ge 4$)은 선형화 함수 $v \times 3 + (u - 3) + 18$에 의해 가로폭이 3인 행 기준 순서이다. □

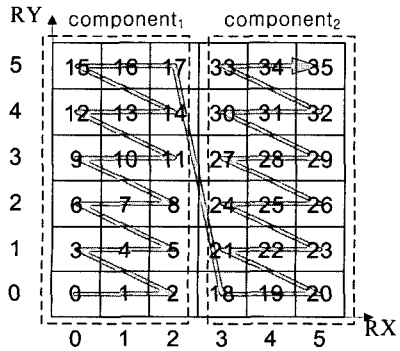


그림 11 n = 36, k = 2인 적응형 행 기준 순서

$$L_{ARM}(u,v)=\begin{cases} v\times 3+u & \text{if } u<3 \\ v\times 3+(u-3)+18 & \text{if } u\ge 3 \end{cases} \quad (4)$$

적응형 행 기준 순서의 k는 한 구성 요소내의 MAXLDIST(즉, $MAXLDIST_{component}$)에 의해 결정되는 원패스 버퍼 크기가 주어진 버퍼 크기 B보다 작거나 같은 값을 갖도록 결정된다. 여기서, $MAXLDIST_{component}$는 겹침 페이지간 인접도가 c 일 때, 수식 (5)에 대하여 $c\times\frac{\sqrt{n}}{k}+c+1$이다. 따라서, 수식 (2)와 $MAXLDIST_{component}$로부터, $(1+|SJWP(P_s)|)+(1+AVG(|\Delta SJWP|))\times(MAXLDIST_{component}-1) \le B$를 만족하는 최소의 k를 적응형 행 기준 순서의 k 값으로 선택한다.[1)]

5. 실험

본 장에서는 제 3장과 4장에서 소개한 여러 가지 공간 채움 곡선들을 변환공간 뷰 조인에 적용하였을 때, 그 성능을 비교하고, 분석한 결과와 실험 결과가 일치하는 지를 검증한다. 또한, 변환공간 뷰 조인과 원공간 조인 알고리즘들의 성능을 비교한다. 원공간 조인 알고리즘으로는 Brinkhoff 등의 깊이 우선 탐색 R 트리 공간 조인[1]과 Huang 등의 넓이 우선 탐색 R 트리 공간 조인 (Combo1과 Combo2)[2]을 선택하였다. 본 장에서 비교하는 알고리즘들을 정리하면 표 1과 같다.

표 1 실험에 사용된 알고리즘들

이 름	의 미
TSVJ-Z	Z 순서를 사용하는 변환공간 뷰 조인 알고리즘
TSVJ-Hilbert	힐버트 순서를 사용하는 변환공간 뷰 조인 알고리즘
TSVJ-RM	행 기준 순서를 사용하는 변환공간 뷰 조인 알고리즘
TSVJ-ARM	적응형 행 기준 순서를 사용하는 변환공간 뷰 조인 알고리즘
DFRJ	깊이 우선 탐색 R 트리 공간 조인 알고리즘[1]
BFRJ-Combo1	넓이 우선 탐색 R 트리 공간 조인 알고리즘: Combo1[2]
BFRJ-Combo2	넓이 우선 탐색 R 트리 공간 조인 알고리즘: Combo2[2]

1) 매개 변수 $|SJWP(P_s)|$, $AVG(|\Delta SJWP|)$, 그리고 c는 몇 개의 선택된 비단말 페이지들을 검색하여 추정한다. 비단말 페이지들의 선택은 비균일 분포에서의 최악의 경우도 다룰 수 있도록 하기 위해, 색인에서 가장 밀집된 영역안의 비단말 페이지들을 선택한다. 이들 선택된 비단말 페이지들은 색인으로부터 적은 비용으로 추출해 낼 수 있는데, 버퍼가 매우 작을 때를 제외하고는 이들 비단말 페이지들은 버퍼 안에 남아 있기 때문에, 이로 인한 추가 비용은 매우 작다. 매개변수 추정에 사용된 추가 비용은 제5장의 실험결과에 모두 포함되어 있다.

본 논문에서는 성능 평가의 척도로서 원패스 버퍼 크기와 디스크 액세스 횟수를 사용한다. 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(3.5 MBytes)로 구성된다. U1과 U2내의 MBR들의 중심점들은 전체 공간에 대해 균일 분포를 가지고, E1과 E2내의 MBR 들의 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 *tiger1*과 *tiger2*로 구성된다. 이 데이타는 Brinkhoff 등[1]과 Huang 등[2]이 사용한 실험 데이타와 동일한 것으로서, tiger1은 캘리포니아 지역의 도로 131,461개의 MBR들(3.8 MBytes)로 이루어진 데이타 집합이고 tiger2는 강과 철도 128,971개의 MBR들(3.4 MBytes)로 이루어진 데이타 집합이다. tiger1과 tiger2의 공간 조인 결과 수는 다른 데이타 분포에서와 비슷한 86,094개이다.

실험에서 사용한 공간 색인은 Brinkhoff 등[1]이 사용한 것과 동일한 R^* 트리를 사용하였다. 사용된 R^* 트리의 페이지 크기는 4 KBytes이다.

5.1 공간 채움 곡선에 따른 변환공간 뷰 조인 알고리즘의 성능 비교

그림 12는 균일 분포, 지수 분포, 그리고 실제 분포를 가지는 데이타에 대해, TSVJ-Z, TSVJ-Hilbert, TSVJ-RM, TSVJ-ARM의 원패스 버퍼 크기를 정리한 그림이다. 제3장에서 분석적으로 예측하였듯이, TSVJ-RM과 TSVJ-ARM은 같은 원패스 버퍼 크기를 가지며, 가장 작은 원패스 버퍼 크기를 가진다. 모든 데이타 분포에 대해, TSVJ-ARM과 TSVJ-RM은 TSVJ-Z와 비교하여 원패스 버퍼 크기를 최대 14.0배[2] 줄이고, TSVJ-Hilbert와 비교해서는 최대 21.3배 줄인다.

균일 분포 실험 결과에 의하면 TSVJ-RM, TSVJ-Z, 그리고, TSVJ-Hilbert의 원패스 버퍼 크기의 비율은 71:885:1480 = 0.04:0.59:1.00이다. 이 비율은 실험에 사용된 변환공간 뷰들의 몇몇 페이지들을 검색하여 구한 인수들인 $|SJWP(P_s)|$ = 4, $AVG(|\triangle SJWP|)$ = 1.12,[3]

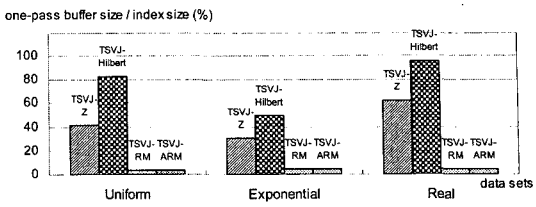


그림 12 공간 채움 곡선 및 데이타 분포에 따른 변환공간 뷰 조인 알고리즘들의 원패스 버퍼 크기

겹침 페이지간 인접도 c = 1, 총 페이지 수 n = 913을 수식 (2)와 보조정리 2, 3, 4에 대입함으로써 계산된 비율인 69:973:1615 = 0.04:0.60:1.00과 거의 같은 비율을 가진다. 따라서, 본 논문의 균일 분포에 대한 분석이 매우 정확한 것임을 알 수 있다. 또한, 균일 분포 이외의 실험 결과에서도 이와 유사한 비율이 계산되는데, 이는 본 논문의 분석이 다른 분포에서도 잘 적용됨을 나타낸다.

그림 13은 각각 균일 분포, 지수 분포, 그리고 실제 분포를 가지는 데이타에 대해 TSVJ-RM, TSVJ-Z, TSVJ-Hilbert, TSVJ-ARM의 버퍼 크기의 변화에 따른 디스크 액세스 횟수를 나타낸다. 그래프에서 맨 밑에 있는 수평선은 최적 즉, 원패스일 때의 디스크 액세스 횟수를 나타낸다.

그림 13에서 보듯이 TSVJ-ARM는 버퍼 크기에 상관없이 항상 가장 작은 디스크 액세스 횟수를 보장한다. TSVJ-RM는 원패스 버퍼 보다 큰 버퍼에서 TSVJ-ARM과 같이 가장 작은 디스크 액세스 횟수를 가지나, 원패스 버퍼 보다 작은 버퍼에서는 디스크 액세스 횟수가 많이 발생하는 문제점을 가진다. 이는 제3장과 4장의 분석 결과와 일치한다. 작은 버퍼를 조인 대상인 두 색인의 1~2% 크기로 정의하고, 큰 버퍼를 5~6%에 해당하는 버퍼로 정의할 때, 모든 데이타 분포에 대해 TSVJ-ARM은 다른 TSVJ 알고리즘들과 비교하여 작은 버퍼에서는 최대 74.6%[4] (3.95배[5]) 성능 향상을 보이고, 큰 버퍼에서는 최대 18.3% (1.31배)성능 향상을 보인다.

5.2 변환공간 뷰 조인 알고리즘과 원공간 색인 조인 알고리즘의 성능 비교

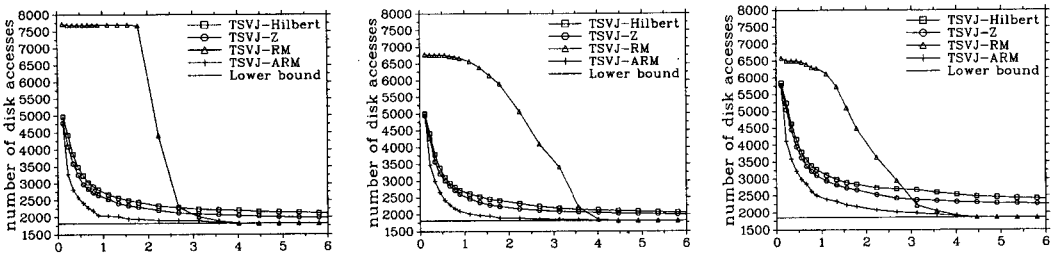
본 절에서는 TSVJ와 대표적인 두 원공간 색인 조인 알고리즘인 DFRJ와 BFRJ의 성능을 비교한다. 비교에 사용한 TSVJ는 성능이 제일 좋은 TSVJ-ARM을 사용

2) $\frac{one-pass\ buffer\ size(other\ algorithm)}{one-pass\ buffer\ size(TSVJ-ARM\ \text{or}\ TSVJ-RM)}$

3) 실험에 의하면 $AVG(|\triangle SJWP|)$는 공간 채움 곡선과 관계없이 거의 같은 값을 가진다.

4) $\frac{number\ of\ disk\ accesses(other\ algorithm)-number\ of\ disk\ accesses(TSVJ-ARM)}{number\ of\ disk\ accesses(other\ algorhtms)}\times 100$

5) $\frac{number\ of\ disk\ accesses(other)}{number\ of\ disk\ accesses(TSVJ-ARM)}$



(a) 균일 분포 (b) 지수 분포 (c) 실제 분포

그림 13 여러 공간 채움 곡선들을 사용한 TSVJ의 균일, 지수, 실제 분포 데이타에 따른 디스크 액세스 횟수

하였다.

5.2.1 DFRJ와 TSVJ-ARM의 비교

그림 14와 15는 각각 균일 분포, 지수 분포, 실제 분포를 가지는 데이타에 대해 TSVJ-ARM과 DFRJ의 성능을 비교한 것이다. 실험 결과, 제안하는 TSVJ-ARM은 모든 데이타 분포에 대해 원패스 버퍼 크기 측면과 디스크 액세스 횟수 측면에서 DFRJ 보다 좋은 성능을 보인다.

그림 14에 의하면, TSVJ-ARM은 DFRJ와 비교하여 모든 데이타 분포에서 원패스 버퍼 크기를 최대 15.7배 줄인다. 그림 15에 의하면, TSVJ-ARM은 작은 버퍼 크기에서 디스크 액세스 횟수를 DFRJ와 비교하여 최대 35.3% (1.55배) 줄이고, 큰 버퍼 크기에서는 최대 13.7% (1.16배) 줄인다.

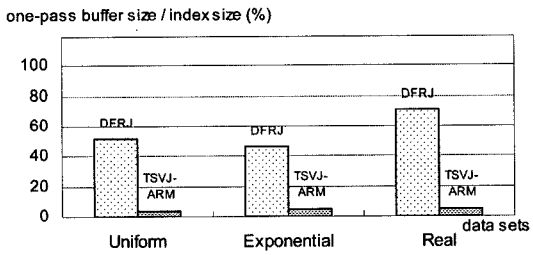
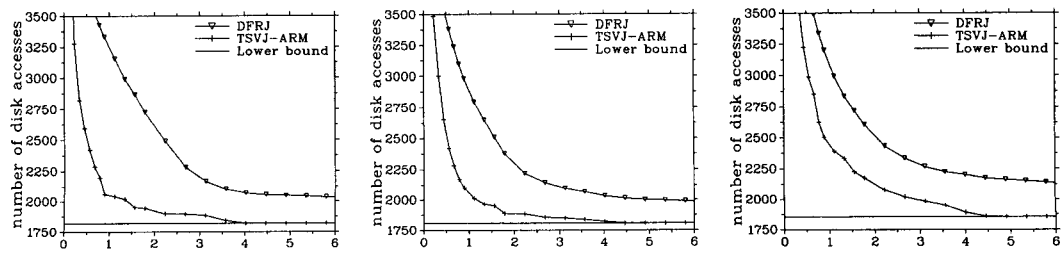


그림 14 DFRJ와 TSVJ-ARM의 원패스 버퍼 크기

TSVJ-ARM이 DFRJ 보다 좋은 이유는 다음과 같이 생각할 수 있다. TSVJ-ARM은 변환공간상의 특성을 활용하여 전역 최적화를 수행하는 알고리즘이다. 특히 디스크 페이지들이 읽히는 순서를 결정 하는 공간 채움 곡선의 정형적인 분석을 통해 가장 작은 원패스 버퍼 크기를 가지며, 주어진 버퍼의 크기에 따라 디스크 페이지들의 순서를 적응적으로 조정함으로써 가장 작은 디스크 액세스 비용을 가진다. 반면, DFRJ는 local plane-sweeping과 local Z-ordering등의 휴리스틱 기법들만을 사용하여 디스크 페이지들의 순서를 제어하는 지역적 최적화만을 수행하기 때문에 디스크 비용이 상대적으로 크다. 그 리고, 주어진 버퍼 크기에 적응하지 못하고 항상 같은 순서로만 페이지들을 버퍼에 읽어 들이기 때문에 버퍼 크기에 대한 적절한 대응을 하지 못한다.

5.2.2 BFRJ와 TSVJ-ARM의 비교

BFRJ-Combo2는 pin/unpin이라는 기법으로 LRU 버퍼를 제어하여 버퍼의 효율을 높이는 방법을 사용한다. Pin 연산은 버퍼에 다시 읽힐 것이라고 예상되는 페이지들을 다시 읽힐 때까지 버퍼에 고정하는 연산이고, unpin 연산은 pin 연산을 해제하는 연산이다. 전역 최적화를 수행하는 조인 알고리즘들은 어떤 페이지들이 읽히고 안 읽힐지를 예측할 수 있기 때문에 pin/unpin 기법을 사용할 수 있다. TSVJ-ARM에도 이 기법을 사용할 수 있으나, TSVJ-ARM은 이미 다시 읽힐 페이지들



(a) 균일 분포 (b) 지수 분포 (c) 실제 분포

그림 15 균일, 지수, 실제 분포 데이타에 대한 DFRJ와 TSVJ-ARM의 디스크 액세스 횟수

을 버퍼에 최대 한 남아 있도록 페이지들을 순서화하여 읽어 들이기 때문에, 사용하는 경우와 사용하지 않은 경우의 성능 차이가 거의 나타나지 않는다. 따라서, TSVJ-ARM에는 이 기법을 사용하지 않는다.

그림 16은 균일 분포, 지수 분포, 실제 분포를 가지는 데이타에 대해 TSVJ-ARM과 BFRJ-Combo2 원패스 버퍼 크기를 비교한 결과이다. BFRJ-Combo1은 외부 디스크 소트를 사용하기 때문에 항상 디스크 페이지들을 한번 이상 읽는다. 따라서, BFRJ-Combo1에서는 원패스 버퍼 크기가 존재하지 않으며, 이로 인해 비교를 수행하지 못했다. 그림 16에 의하면, TSVJ-ARM은 BFRJ-Combo2와 비교하여 모든 데이타 분포에서 원패스 버퍼 크기를 최대 1.14배 줄인다.

그림 17은 BFRJ-Combo1, BFRJ-Combo2, 그리고 TSVJ-ARM의 디스크 액세스 횟수를 비교한 것이다. 그림 17에 의하면, 디스크 액세스 횟수 측면에서 TSVJ-ARM은 BFRJ-Combo1과 BFRJ-Combo2 보다 항상 좋은 성능을 가진다. BFRJ-Combo1과 비교하여, TSVJ-ARM은 작은 버퍼 크기에서 디스크 액세스 횟수를 최대 46.4% (1.87배) 줄인다. 큰 버퍼 크기에서는 최대 23.4% (1.31배) 줄인다. BFRJ- Combo2와 비교하여, TSVJ-ARM은 작은 버퍼크기에서 최대 65.3% (2.89배) 줄인다. 큰 버퍼 크기는 최대 0.2%(1.002배) 줄인다.

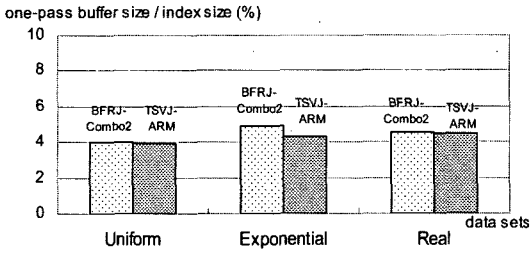


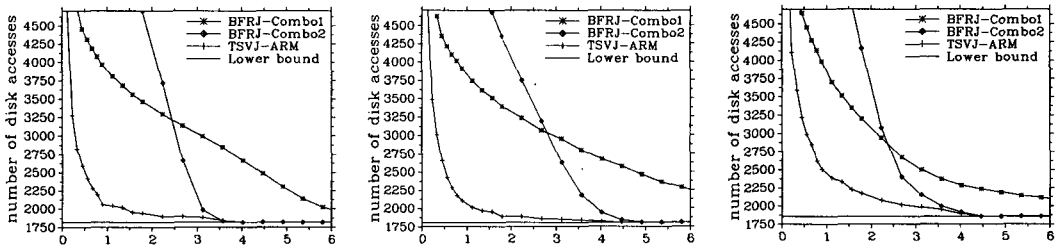
그림 16 BFRJ-Combo2와 TSVJ-ARM의 원패스 버퍼 크기

TSVJ-ARM이 BFRJ들 보다 좋은 이유는 별도의 디스크나 메인 메모리 오버헤드 없이 전역 최적화를 수행하기 때문이다. BFRJ-Combo1은 디스크 페이지 액세스 순서를 디스크에 저장하기 때문에 별도의 디스크 액세스 오버헤드를 가지며, BFRJ-Combo2는 메인 메모리에 저장하기 때문에 버퍼 크기가 줄어들어 디스크 액세스 횟수가 늘어나는 단점을 가진다. TSVJ-ARM은 또한 주어진 버퍼의 크기에 적응적으로 페이지 순서화 방법을 조정하기 때문에 BFRJ들 보다 더 좋은 성능을 보인다.

6. 결 론

본 논문에서는 변환공간 뷰 조인에서 디스크 페이지 액세스 순서를 결정하는 공간 채움 곡선과 성능 간의 정형적인 분석을 수행하고, 이를 기반으로 변환공간 뷰 조인의 성능을 향상시키는 새로운 공간 채움 곡선인 **적응형 행 기준 순서**를 제안하였다. 적응형 행 기준 순서는 주어진 버퍼 크기에 따라 디스크 페이지 액세스 순서를 적응적으로 조정하는 공간 채움 곡선으로 변환공간 뷰 조인 알고리즘의 원패스 버퍼 크기와 디스크 액세스 횟수를 크게 줄인다. 여기서, 원패스 버퍼 크기와 디스크 액세스 횟수는 본 논문에서 사용한 성능의 척도이다.

본 논문에서는 분석과 실험을 통해, 적응형 행 기준 순서를 사용하는 변환공간 뷰 조인 알고리즘의 우수성을 보였다. 적응형 행 기준 순서를 사용하는 변환공간 뷰 알고리즘은 다른 알고리즘들과 비교하여 항상 좋은 성능을 보였는데, 다른 공간 채움 곡선을 사용하는 알고리즘과 비교하여 원패스 버퍼 크기를 최대 21.3배 줄이고, 디스크 액세스 횟수를 최대 74.6% 줄인다. 또한, R 트리를 원공간에서 조인하는 알고리즘들과 비교하여 원패스 버퍼 크기를 최대 15.7배 줄이고, 디스크 액세스 횟수를 최대 65.3% 줄인다.



(a) 균일 분포 (b) 지수 분포 (c) 실제 분포

그림 17 균일, 지수, 실제 분포 데이타에 대한 BFRJ-Combo1, BFRJ-Combo2, 그리고 TSVJ-ARM의 디스크 액세스 횟수

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

[4] A. Guttman, "R-trees: a Dynamic Index Structure for Spatial Searching," In *Proc. Int'l Conf. on Management of Data*, ACM SIGMOD, pp. 47-57, 1984.

[5] T. Sellis, N. Roussopolus, and C. Faloutsos, "The R+-tree: A Dynamic Index for Multidimensional Objects," In *Proc. the Thirteenth Int'l Conf. on Very Large Data Bases*, pp. 507-518, 1987.

[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] J. Lee, Y. Lee, K. Whang, and I. Song, "A Region Splitting Strategy for Physical Database Design of Multidimensional File Organizations," In *Proc. the 23rd Int'l Conf. on Very Large Data Bases*, pp. 416-425, 1997.

[8] K. Whang and R. Krishnamurthy, Multilevel Grid Files, IBM Research Report RC 11516, 1985.

[9] 이민재, 한욱신, 황규영, "변환공간 뷰를 기반으로한 공간 조인", 한국정보과학회 논문지(데이타베이스), Vol. 30, No. 5, pp. 438-450, 2003년 10월.

[10] D. Hilbert, "Über die stetige Abbildung einer Linie auf Flächenstück," *Math Ann.*, Vol. 38, pp. 459-460, 1891.

[11] J. K. Lawder and P. J. H. King, "Querying Multi-Dimensional Data Indexed Using the Hilbert Space-filling Curve," *SIGMOD Record*, Vol.30, No.1, pp. 19-24, 2001.

[12] J. Orenstein, "Spatial Query Processing in an Object-Oriented Database System," In *Proc. Int'l Conf. on Management of Data*, ACM SIGMOD, pp. 326-336, May 1986.

Appendix A: 보조정리 2의 증명

$$
\begin{aligned}
MAXLDIST &= \max(|L(u_1,v_1)-L(u_2,v_2)|)+1 \\
&= \max(|(v_1\times|\sqrt{n}|+u_1)-(v_2\times|\sqrt{n}|+u_2)|)+1 \\
&= \max(|v_1-v_2|\times|\sqrt{n}|+|u_1-u_2|)+1 \\
&\quad (\text{since } page\ adjacency=1 \text{ and} \\
&\quad n\geq 4, \max(|u_1-u_2|)=\max(|v_1-v_2|)=1) \\
&= |\sqrt{n}|+2
\end{aligned}
\tag{5}
$$

□

Appendix B: 보조정리 3의 증명

주어진 좌표값 (u, v)에 대한 Z 순서[12]의 값은 이진수로 표현된 u, v의 각 비트(bit)들을 인터리빙(interleaving)함으로써 구해진다. Z 순서에 대한 순서화 함수 L_Z는 수식 (6)과 같다.

$$
\begin{aligned}
L_Z(u,v) = {} & u_{m-1}\cdot 2^{2m-1}+v_{m-1}\cdot 2^{2m-2}+u_{m-2}\cdot 2^{2m-3} \\
& +v_{m-2}\cdot 2^{2m-4}+\cdots+u_0\cdot 2^1+v_0\cdot 2^0, \\
& where\ m=\log_2\sqrt{n}
\end{aligned}
\tag{6}
$$

여기서, m은 u와 v를 이진수로 표현했을 때, 최대 자릿수를 뜻한다. 균일 데이타 및 페이지 분포 가정 하에서 u와 v는 0에서 n사이의 값을 가질 수 있으므로, m은 $\log_2 n$이다. u_i는 u를 이진수로 나타낼 때 i번째 bit의 값을 뜻하고, v_i는 v를 이진수로 나타낼 때, i번째 bit의 값을 뜻한다.

수식 (6)에 따라 $MAXLDIST_Z$는 다음과 같이 구해진다.

$$
\begin{aligned}
MAXLDIST_Z &= \max(|L_Z(u1,v1)-L_Z(u2,v2)|)+1 \\
&= \max(|u1_{m-1}\cdot 2^{2m-1}+v1_{m-1}\cdot 2^{2m-2}+ \\
&\quad \cdots+u1_0\cdot 2^1+v1_0\cdot 2^0- \\
&\quad (u2_{m-1}\cdot 2^{2m-1}+v2_{m-1}\cdot 2^{2m-2}+ \\
&\quad \cdots+u2_0\cdot 2^1+v2_0\cdot 2^0)|)+1 \\
&= \max(|(u1_{m-1}-u2_{m-1})\cdot 2^{2m-1}+ \\
&\quad (v1_{m-1}-v2_{m-1})\cdot 2^{2m-2}+ \\
&\quad (u1_{m-2}-u2_{m-2})\cdot 2^{2m-3}+ \\
&\quad (v1_{m-2}-v2_{m-2})\cdot 2^{2m-4}+\cdots \\
&\quad +(u1_0-u2_0)\cdot 2^1+(v1_0-v2_0)\cdot 2^0|)+1
\end{aligned}
$$

겹침 페이지간 인접도는 1이므로, $|u1-u2|\leq 1$과 $|v1-v2|\leq 1$의 관계가 존재한다. $|u1-u2|\leq 1$과 $|v1-v2|\leq 1$의 관계를 만족하는 $u1$, $u2$, $v1$, $v2$들 중에서, n이 4보다 크거나 같을 때, $u1$=10……0 (0의 개수는 m-1개), $u2$=01……1 (1의 개수는 m-1개), $v1$=10……0 (0의 개수는 m-1개), $v2$=01……1 (1의 개수는 m-1개)은 수식 (7)을 최대로 만든다. 왜냐하면, 수식 (7)의 가장 큰 값을 가지는 두 항인 2^{2m-1}과 2^{2m-2}을 1로 만드는 유일한 경우이기 때문이다. 따라서, 다음 수식이 성립한다.

$$
\begin{aligned}
MAXLDIST_Z &= |2^{2m-1}+2^{2m-2}-2^{2m-3}-2^{2m-4}- \\
&\quad \cdots-2^1-2^0|+1 \\
&= |2^{2m-1}+2^{2m-2}-(2^{2m-3}+2^{2m-4}+ \\
&\quad \cdots+2^1+2^0)|+1 \\
&= |2^{2m-1}+2^{2m-2}-(\sum_{i=0}^{2m-3}2^i)|+1 \\
&= |2^{2m-1}+2^{2m-2}-(\frac{1\cdot(2^{2m-2}-1)}{2-1})|+1 \\
&= |2^{2m-1}+2^{2m-2}-2^{2m-2}|+2
\end{aligned}
$$

$$= 2^{2m-1} + 2$$
$$= 2^{2\log_2 \sqrt{n}} \cdot 2^{-1} + 2 = 2^{\log_2 n} \cdot 2^{-1} + 2$$
$$= \frac{n}{2} + 2$$

□

Appendix C: 보조정리 4의 증명

힐버트 순서는 하나의 정사각형을 4개의 정사각형들로 재귀적으로 분할하여 순서화함으로써 정의된다[11]. 이때, 인접하게 순서화된 모든 두 정사각형들은 같은 면을 공유하도록 순서화된다. 그림 18(a)는 힐버트 순서의 첫 번째 step을 보여준다. 그림에서 각 정사각형들은 정의된 순서에 따라 숫자가 부여된다. 그림 18(b)는 두 번째 step으로 그림 18(a)의 각 정사각형들이 4개의 정사각형들로 분할된 후, 각각의 4개로 분할된 정사각형들에 크기가 작아진 첫 번째 step의 곡선이 부여된 후 연결되어 정의된다. 그림 18(c)은 세 번째 step을 나타낸다.

힐버트 순서에서 i번째 정사각형으로부터 4등분이 되어 정의된 정사각형들의 순서 값은 $4i$와 $4i+3$ 사이의 값을 가진다. 그림 18(a)의 0번 정사각형으로부터 정의된 그림 18(b)의 0, 1, 2, 3번 정사각형들의 순서 값은 0과 3사이의 값을 가지며, 그림 18(b)의 1번 정사각형으로부터 정의된 그림 18(c)의 4, 5, 6, 7번 정사각형들의 순서 값은 4와 7사이의 값을 가진다.

k번째 step에서 선형거리가 MAXLDIST를 이루는 두 정사각형을 $P_{s,k}$와 $P_{e,k}$라 하고 이들의 순서 값을 각각 s_k과 e_k라 하자. 그러면, 힐버트 순서의 재귀적 특성에 의해 다음과 같은 특성이 존재한다. $P_{s,k}$로부터 정의된 $P_{s,k+1}$의 순서값은 항상 $s_{k+1} = 4s_k + 1$이고, 유사하게, $P_{e,k}$로부터 정의된 $P_{e,k+1}$의 순서값은 항상 $e_{k+1} = 4e_k + 2$이다. 예를 들어, 2번째 step에서 순서값이 1인 $P_{s,2}$로부터 정의된 3번째 step의 $P_{s,3}$의 순서값은 $4 \cdot 1 + 1 = 5$이다. 이 특성을 사용하여 수식 (8)과 (9)를 다음과 같이 구한다.

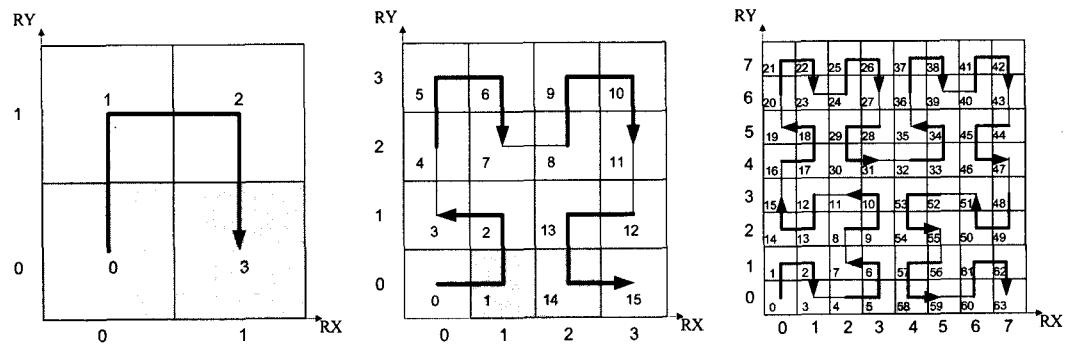
$$s_k = 4s_{k-1} + 1 = 4(4s_{k-2} + 1) + 1 = 4 \cdot 4s_{k-2} + 4 + 1$$
$$= 4^{k-1} s_1 + \sum_{i=0}^{k-2} 4^i$$
$$= \sum_{i=0}^{k-2} 4^i \qquad (\text{since } s_1 = 0)$$
$$= \frac{4^{k-1} - 1}{3}$$

$$e_k = 4e_{k-1} + 2 = 4(4e_{k-2} + 2) + 2 = 4 \cdot 4e_{k-2} + 2(4+1)$$
$$= 4^{k-1} e_1 + 2\sum_{i=0}^{k-2} 4^i$$
$$= 3 \cdot 4^{k-1} + 2\sum_{i=0}^{k-2} 4^i \qquad (\text{since } e_1 = 3)$$
$$= 3 \cdot 4^{k-1} + \frac{2(4^{k-1} - 1)}{3}$$

수식 (8)과 (9)을 사용하여 다음과 같이 MAXLDIST$_{\text{Hilbert}}$이 구해진다.

$$MAXLDIST_{Hilbert} = e_k - s_k + 1$$
$$= 3 \cdot 4^{k-1} + \frac{2(4^{k-1} - 1)}{3} - \frac{4^{k-1} - 1}{3} + 1$$
$$= 3 \cdot 4^{k-1} + \frac{4^{k-1} - 1}{3} + 1$$
$$= \frac{10 \cdot 4^{k-1} + 2}{3}$$
$$= \frac{10 \cdot 4^k + 8}{12}$$
$$= \frac{10n + 8}{12} \qquad (\text{since } n = 4^k)$$

□



(a) 첫 번째 step (b) 두 번째 step (c) 세 번째 step

그림 18 힐버트 순서의 첫 세 step들

이 민 재

2004년 3월~2004년 11월 포스트닥, 첨단정보기술연구센터, KAIST. 2004년 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.

황 규 영

정보과학회논문지 : 데이타베이스
제 32 권 제 3 호 참조