# 이동체 색인을 위한 KDB-Tree 의 분할 정책

이창헌° 임덕성 홍봉희
부 산 대 학 교 컴 퓨 터 공 학 과
{angel71, dsleem, bhhong}@pusan.ac.kr

## Splitting Policies of KDB-Tree for indexing of Moving Objects

Chang-Hun Lee° Duk-Sung Lim Bong-Hee Hong
Dept. of Computer Engineering, Pusan National University

요 약

최근 이동통신 및 GPS 기술의 발달로 위치기반서비스 요구가 점점 증가하고 있고, 대용량의 위치데이터가 저장되는 위치기반서비스의 구현을 위한 이동체의 저장 및 검색에 관한 연구가 활발하다. 이동체의 위치 정보를 점으로 모델링하여 색인 할 경우 KDB-Tree 의 성능이 우수하다. 그러나 KDB-Tree 는 시공간에서의 이동체 위치데이터 색인을 고려할 경우 시간 도메인의 특성으로 인해 성능 저하의 문제를 발생 시킨다. 본 논문에서는 이동체 위치데이터의 색인을 위한 KDB-Tree 의 사용에서 시간 도메인의 특성을 반영한 분할 도메인 선정 방법과 분할 정책을 제시한다. 새로운 분할 정책은 색인의 공간활용도를 높이고 색인의 크기를 작게 하여 검색의 성능을 높인 최근 시간 분할 기법과 LD(Last Division) 분할 정책이다. 본 논문에서는 KDB-Tree의 변경된 분할 정책을 구현하고 성능평가를 수행한다. 이 성능 평가 실험을 통해서 변경된 분할 정책을 사용한 KDB-Tree 에서 공간활용도가 높고 검색 성능이 우수함을 보인다.

## 1. 서론

무선 이동통신 기술의 발달로 휴대폰과 PDA(Personal Digital Assistants) 등의 무선 이동 기기가 보편화되면서 GPS(Global Positioning System) 기술을 사용한 시공간에서의 위치 기반 서비스(LBS : Location Based Service) 의 요구가 증대되고 있다.

다수의 이동체의 위치 정보를 저장하는 LBS 서버에는 대용량의 데이터가 저장된다. 이 경우 이동체의 현재 또는 과거 영역에 대한 효율적인 검색을 위해 색인이 필요하다. 위치 정보를 3차원 점으로 모델링 하는 경우 PAM(Point Access Method) 중에서 다차원 색인에서 성능이 우수한 KDB-Tree [1] 를 고려할 수 있다. 그러나 KDB-Tree 는 시간 도메인을 고려하고 있지 않기 때문에 시공간 데이터의 검색 시 성능이 저하되는 다음과 같은 3가지의 문제점이 있다.

첫째 KDB-Tree 의 순환적인 분할 도메인 선정 방법의 경우 분할 도메인이 시간 도메인으로 선정 되면 초기에 큰 과거 영역을 만들게 되므로 색인의 성능을 저하시키게 되는 문제를 발생시킨다.

둘째 포인트페이지 분할 방식인 균등분할 방식은 시간 도메인으로 분할이 될 경우 공간활용도를 저하시키는 문제를 발생 시킨다. 시공간에서 균등분할 방식은 공간 도메인으로 분할 할 때는 적합하지만 시간 도메인으로 분할 할 때는 분할 후 발생하는 과거 영역이 되는 포인트페이지의 공간활용도 저하 문제를 발생 시킨다. 그림 1 에서 P1, P2 의 포인트페이지는 공간 도메인으로 균등분할을 하고 P3, P4 의 포인트페이지는 시간 도메인으로 균등분할을 하게 된다. 포인트페이지 P3 의 경우는 과거 영역이 되므로 추가 삽입이 발생하지 않는데 균등분할을 하게 되면 포인트페이지의 공간 활용도가 떨어지게 된다.

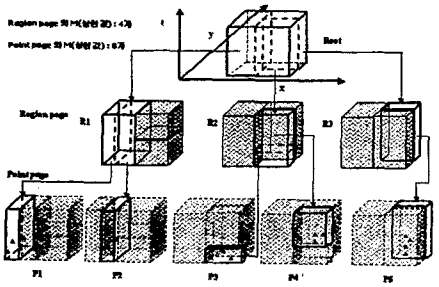


그림 1 KDB-Tree 의 예

마지막으로 KDB-Tree 의 영역페이지 분할 정책 중에서 FS(Forced Splitting)분할의 연쇄적 분할 전파 문제와 심각한 공간활용도 저하 문제를 해결한 FD(First Division) 분할 정책[1]은 시간 도메인의 특성을 고려하고 있지 않기 때문에 시간 도메인으로 분할을 하게 되면 영역페이지의 공간활용도를 저하시켜 결론적으로 전체 색인의 깊이를 증가 시키게 된다.

본 논문에서는 시공간 이동체 색인을 위한 KDB-Tree 의 변경된 동적 분할 정책으로서 공간 우선 기법을 사용한 분할 도메인 선정 방법과 시간 도메인을 고려한 포인트페이지의 최근 시간 분할 기법, 그리고 영역페이지 분할을 위한 LD(Last Division) 정책을 제시한다.

이 논문의 구성은 다음과 같다. 2장에서 관련연구를 기술하고 3장에서는 제안한 분할 정책을 기술하고 알고리즘을 보인다. 4장에서는 성능평가 결과를 분석하고 5장에서는 결론을 제시한다.

## 2. 관련연구

공간분할 방식인 KDB-Tree [2] 는 KD-Tree 와 B-Tree 의 특성을 조합한 것으로 노드들의 계층이 균형을 이룬 트리이고 영역페이지와 포인트페이지로 구성된다.

[6] 에서는 KDB-Tree 의 영역페이지 분할 정책이 강제적으로 분할을 전파하기 때문에 포인트페이지의 공간 활용도가 저하되는 문제를 지적했다. Buddy-Tree 와 LSD-Tree 에서는 강제적인 분할 전파 문제를 해결하기 위해 영역페이지가 처음 분할되었던 분할 요소에 의해서 분할되는 정책을 사용한다.

[1] 에서는 KDB-Tree 의 강제적인 분할 정책을 FS(Forced Splitting) 분할 정책이라고 정의하고 영역페이지가 처음으로 분할되었던 분할 축으로 분할 하는 방식을 FD(First Division) 분할 정책이라고 정의한다. 두 분할 정책을 비교 실험 하여 실험의 결과로 FD 분할이 FS 분할의 문제를 해결하여 공간활용도를 높이고 검색의 성능을 향상시킴을 보였다.

시간 도메인에 관한 연구로써 [3] 에서는 시간 도메인을 Transaction-Time, Valid-Time, Bitemporal-Time 으로 분류하여 정의하고 각각을 비교한다. Transaction-Time 의 경우는 데이터베이스에 저장되는 시점의 시간으로 정의 되고 Valid-Time 은 실제적으로 유효한 시점의 시간으로 정의 되며 Bitemporal-Time 은 Transaction-Time 과 Valid-Time 의 특징을 합쳐 놓은 것이다.

이동체의 생성에 대한 연구로써 GSTD(Generate Spatio-Temporal Data)[4] 알고리즘은 정의된 분포에 따라 이동체의 위치 보고 간격, 중점 이동 간격, 크기 변경 간격을 달리하여 이동체 데이터 집합을 생성한다. 네트웍 기반 이동체 생성 알고리즘은 네트웍 데이트를 기반으로 이동체의 최대속도, 출발지점, 도착지점, 네트웍 상의 최대 이동체 개수, 라우팅 정보들을 인자로 설정하여 이동체 데이터 집합을 생성한다.

KDB-Tree 의 분할 도메인 선정 방법과 분할 정책에 관한 연구[5]에서 이동체가 시공간에서 이동하는 특성과 시공간에서 시간

도메인이 선형적으로 그 값이 증가한다는 특징으로 인해 기존의 분할 정책의 문제점이 있음을 보였고 그 문제점을 해결하기 위해 시간 도메인을 고려한 분할 정책을 제시하였다.

## 3. 동적 분할 정책

이 장에서는 시공간에서의 이동체 색인을 위해 제시한 KDB-Tree 의 변경된 분할 도메인 선정 방법과 분할 정책에 대해 기술하고 그 알고리즘을 소개한다

### 3.1 분할 도메인 선정 방법과 분할 정책

이동체의 위치데이터는 응집과 확산의 패턴을 주기적으로 반복한다는 특징을 보인다. 공간 우선 분할 방식은 이동체의 이동 패턴이 주기를 가지고 반복한다는 것을 고려하여 공간을 우선적으로 분할하고 이후에 시간분할을 하는 분할 도메인 선정 방식이다. 공간분할에서 시간분할로의 전환시점은 데이터의 크기에 따라 크기가 다른 페이지당 평균면적($Area_{Avg}$)을 구하고 평균면적 이하가 되는 페이지들의 합이 전체 공간 영역의 특정한 비율이 되는 시점으로 한다. 평균면적의 크기는 이동체의 개수, 이동체의 보고 주기, 한 주기 동안의 위치보고 횟수, 한 페이지의 최대 Entry 수 의 상관 관계에 의해 구해진 값을 전체 공간 도메인에 적용하여 구한다.

최근 시간 분할 방법은 분할되는 두 개의 포인트페이지의 시간 도메인 특성을 이용하여 분할 후에 발생하는 과거 영역의 포인트페이지에는 상한 값이 되게 위치데이터를 저장하고 미래의 영역이 되는 포인트페이지에는 오버플로우를 발생시킨 위치데이터만 저장하는 방법이다. 이 방법은 그림 2 의 포인트페이지 P3 과 같이 분할 이후 과거 영역이 되는 포인트페이지에는 더 이상 위치데이터가 보고 되지 않는다는 특징을 고려해서 상한 값의 데이터를 저장한다. 그리고 포인트페이지 P4 에는 오버플로우를 발생 시킨 데이터만 저장한다. 최근 시간 분할 방식은 공간활용도를 최대한 높이고 새로 생성된 페이지에는 오버플로우를 발생 시킨 데이터만 존재 함으로 계속적으로 위치데이터가 보고 된다고 볼 때 오버플로우의 시점을 최대한 연기 시키는 장점을 갖는다.

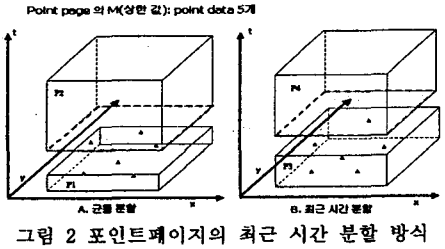


그림 2 포인트페이지의 최근 시간 분할 방식

LD(Last Division) 분할 정책은 시간분할이 시작된 이후에 영역페이지의 모든 Entry 들이 시간분할을 한 경우 영역페이지에서 마지막으로 분할이 이루어진 페이지로 분할을 하는 방식이다. 영역페이지가 공간 도메인으로 분할할 경우에는 FD 분할 정책을 사용하는 것이 강제 분할 전파를 줄이므로 효과적이고 분할이 시간 도메인으로 이루어질 때는 LD 분할을 사용하여 영역페이지의 용적률을 높이는 것이 효과적이다.

### 3.2 알고리즘

분할 도메인 선정 방식인 공간 우선 분할 기법 알고리즘은 공간 분할에서 시간 분할로의 전환점을 구하는 것이 선행된다. 전환 시점은 포인트페이지당 전체 공간 영역에 대한 평균 영역을 구하고 이 평균 영역보다 작아지는 데이터가 밀집된 포인트페이지를 계수하여 밀집 비율에 따라 전환 시점을 결정한다.

표 1 전환시점을 계산하기 위한 요소

| | |
|---|---|
| $Area_{Avg}$ | 페이지당 평균 면적의 크기 |
| $U_{Spatial}$ | 전체 공간 도메인 |
| $N$ | 주기 동안의 총 페이지 수 |
| $MO$ | 이동체의 개수 |
| $M$ | 포인트페이지의 최대 Entry 개수 |
| $T_{Cycle}$ | 이동체의 응집확산 주기 |
| $R$ | 한 주기 동안의 이동체의 보고 회수 |
| $Lx_{min}, Ly_{min}$ | 해당 포인트페이지의 x, y 도메인 하한 값 |
| $Lx_{max}, Ly_{max}$ | 해당 포인트페이지의 x, y 도메인 상한 값 |

표 1 에서는 전환 시점을 구하기 위해 사용되는 요소들을 보이고 있다. 식 1 에서는 이동 객체의 개수, 보고주기, 주기 당 보고횟수를 이용하여 데이터의 개수를 계산하고 이 값을 포인트페이지의 상한 값으로 나누어 주기 당 총 페이지의 개수를 구한다. 주기 당 총 페이지의 개수로 전체 공간 도메인을 나누면 페이지당의 평균 영역의 크기가 구해지고 이것을 식 2 에서 보이고 있다.

$$N = \frac{(MO * R) * T_{Cycle}}{M} \text{ --식①} \qquad Area_{Avg} = \frac{U_{Spatial}}{N} \text{ --식②}$$

식 3 에서는 공간 우선 분할을 하면서 계속적으로 생성되는 포인트페이지들의 영역을 계산하여 식 2 에서 구해진 페이지당 평균 영역보다 작은 영역인 데이터 밀집지역($Area_{Density}$)을 계수한다.

$$Area_{Density} = \sum_{Area \in Area_{Avg}} (Lx_{\_min} - Lx_{\_max})(Ly_{\_min} - Ly_{\_max}) \text{ --- 식③}$$

실험을 통해서 데이터 밀집지역이 전체 공간 도메인에 대해 어느정도의 비율이 되었을 때 검색 성능이 우수한가를 측정하여 그 값을 전환 시점의 비율로 선택한다.

그림 3 과 그림 4 는 제시한 공간 우선 분할 기법의 알고리즘과 공간분할에서 시간분할로 전환하는 전환 시점에 관한 알고리즘이다. 그림 3 에서 보면 Split_Domain_Check 를 호출하여 분할을 할 도메인을 검사하고 있다. 그림 4 의 Split_Domain_Check 에서는 식1 과 식2 를 사용하여 평균면적이 되는 포인트페이지를 계수하여 분할 도메인의 전환 시점을 계산하고 있다.

```
Set_Split_Domain(PointPage_Num){
  If (DEPTH-1 >= 0){
    Split_Domain_Check(PointPage_Num);
    If ( Split_Domain is Spatial)  div_dim  = Domain_SpatialLongest;
    Else      div_dim = Domain_Time ;
  }
  Else    div_dim = Domain_SpatialDefaul ;
Return  (div_dim);
}
```

그림 3 공간 우선 분할 기법 알고리즘

```
N_Page = (MO * Report_Num) * Cycle / Fanout;   Area_Avg = Spatial_Area / N_Page ;
Split_Domain_Check(PointPage_Num){
  If (Spatial_Domain_Size of  PointPage_Current < Area_Avg){
    Area_Density = Area_Density + Spatial_Domain_Size of  PointPage_Current;
    If (Area_Density > Density_Area_Maximum)   Split_Domain = Domain_Time ;
    Else    Split_Domain = Domain_Spatial ;

Return  (Split_Domain);
}
```

그림 4 공간분할에서 시간분할로의 전환점 알고리즘

다음으로 제시된 분할 정책은 두 가지로 나뉜다. 포인트페이지의 분할 정책으로 최근 시간 분할 기법을, 영역페이지의 분할 정책은 공간분할 일 경우 기존의 분할 정책 중 하나인 FD(First Division) 정책을 그대로 사용하고 시간분할 일 경우에는 LD(Last Division) 정책을 사용하는 것을 3.1 에서 제시하였다

아래의 그림 5 는 최근 시간 분할 기법의 알고리즘이다.

```
Recent_Time_Splitting(PointPage_Num, Entry, LBs, UBs){
    Set_Split_Domain(PointPage_Num);
    For(j = 0; j < (datafanout+1); j++){
        Sorting div_dim_value of all entry in PointPage_Current;
        Find Value_Max, Value_Min, Value_Mid;
    }
    If (Split_Domain is Time)   PointPage_Split_axis = Value_Max;
    Else        PointPage_Split_axis = Value_Mid;
}
```

그림 5 최근 시간 분할 기법 알고리즘

```
RegionPage_Splitting( RegionPage_Num, Branches, LBs, UBs){
  While(DEPTH_Current < DEPTH_Max){
    Read RegionPage_Current;
    If(RegionPage_Current is not overflow) {Create Entry_New;     break;}
    Else {
      If(Split_Domain is Time){
        If(All Entry is split by Time Domain)
          Last_Division_Splitting(RegionPage_Num, Branches, LBs, UBs);
        Else   First_Division_Splitting(RegionPage_Num, Branches, LBs, UBs);
      }
      Else   First_Division_Splitting(RegionPage_Num, Branches, LBs, UBs);
    }
  }
}
```

그림 6 영역페이지 분할 정책 알고리즘

그림 6 은 영역페이지 분할 정책에 대한 전체 알고리즘이고 그림 7 에서는 영역페이지 분할 정책 중에서 LD 분할 정책에 대한 알고리즘이다

```
Last_Division_Splitting( RegionPage_New, Branches, LBs, UBs){
    Find Entry_Lastest of RegionPage_Current ;   Create RegionPage_New ;
    If(Entry_Value >= Entry_(Lastest Value)) Insert Entrys RegionPage_New ;
    Else  Insert Entrys RegionPage_Current ;
}
```

그림 7 LD(Last Division) 분할 정책 알고리즘

## 4. 성능평가

### 4.1 전환시점 결정을 위한 실험 평가

공간 우선 분할 알고리즘을 적용하기 위해서는 우선 공간분할에서 시간분할로 전환되는 시점을 구하는 것이 선행된다. 그림 8 은 데이터의 크기(A : 55,000 B : 220,000 C : 440,000)와 페이지의 크기에 따라 전환시점을 달리 했을 때의 영역질의에 대한 디스크 접근 빈도를 측정한 실험이다.

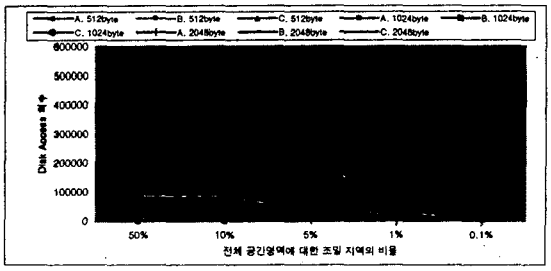


그림 8 데이터 밀집지역의 비율에 따른 Disk Access 수

### 4.2 실험 데이터

본 논문에서 사용한 실험 데이터는 데이터의 분포에 따라 균등 분포, 가우시안 분포, 주기적 응집 확산 분포의 3가지 유형이다. 이 실험 데이터들은 GSTD(Generate Spatio-Temporal Data) 알고리즘 [4]을 사용하여 이동체의 위치 보고 간격, 중점 이동 간격, 크기 변경 간격을 달리하여 이동체 데이터를 생성했다. 주기적 응집 확산 분포의 데이터는 이동체 데이터가 시간에 따라 주기를 가지고 집중과 확산을 반복하는 현상을 고려하여 생성한 데이터 분포이다. 즉 도심지에서 특정 시간에 반복적으로 이동체가 집중되는 현상이나 매일 유사한 시간에 도시의 외각 지역에 이동체가 집중되는 현상을 반영한 데이터의 분포이다.

### 4.3 실험 평가

이 실험에서는 각 실험 대상 방법을 동일한 환경에서 구현하여 성능평가를 수행하였다. 실험 환경으로는 CPU 속도는 266 MHZ 이고 메모리는 128MB 인 Linux Red hot 7.0 운영체제 환경의 Personal Computer 를 사용하였다. 4.1 에서 보인 전환시점에 대한 실험평가를 토대로 실험에서의 전환시점의 결정은 데이터 밀집지역이 1% 가 되는 시점을 전환시점으로 한다.

#### 4.3.1 이동체의 개수에 따른 색인구축 성능비교

이 실험에서는 이동체 데이터의 크기와 이동 분포에 따른 색인 성능 비교 평가를 위해 구축된 색인의 공간활용도를 비교하는 실험을 실시하였다. 그림 9 에서 3가지 데이터 분포 모두에서 제안한 방법을 사용한 KDB-Tree (Modify KDB-Tree : M-KDB) 의 공간활용도가 높은 것을 볼 수 있었다. 3가지 데이터 분포 유형 중에서 균등분포와 가우시안 분포의 경우 5% ~ 15% 가 성능향상이 되었고 주기적 분포의 경우는 10% ~ 30% 의 성능향상을 보였다.

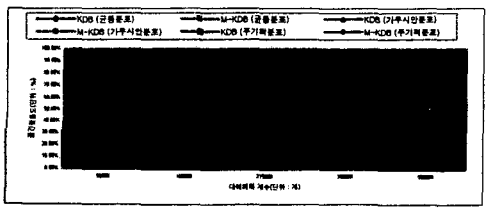

그림 9 데이터 크기에 따른 공간활용도

#### 4.3.2 이동체의 개수에 따른 검색 연산의 성능 비교

이 실험에서는 이동체의 수를 55,000 ~ 880,000 개로 증가시키면서 두 색인에 동일하게 임의의 크기의 영역질의를 1000 회 실행하여 디스크 접근 빈도를 비교 하였다. 그림 10 에서 구축된 색인에 대해 영역질의를 했을 때 제안한 분할 정책을 사용한 KDB-Tree 의 성능이 우수한 것을 볼 수 있다.

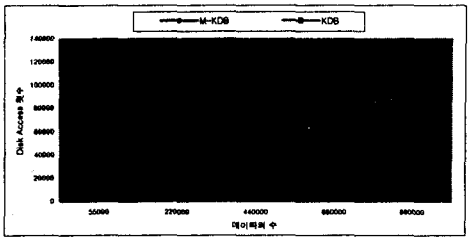


그림 10 이동체 수에 따른 검색 시 평균 Disk Access 수

#### 4.3.3 구축된 색인에 대한 검색 연산의 성능비교

이 실험에서는 페이지의 크기를 1024 byte 로 고정하고 데이터의 개수를 55,000 ~ 880,000 개로 증가시키면서 영역질의의 비율에 따라 1000 회 씩 실행하여 검색성능을 비교 하였다. 실험 결과 그림 11 에서 보듯이 영역질의의 비율이 작을 때에도 본 논문에서 제안한 분할 정책을 사용한 색인의 성능이 우수하고 영역질의의 비율이 증가하면서 성능의 차이가 커짐을 알 수 있었다.

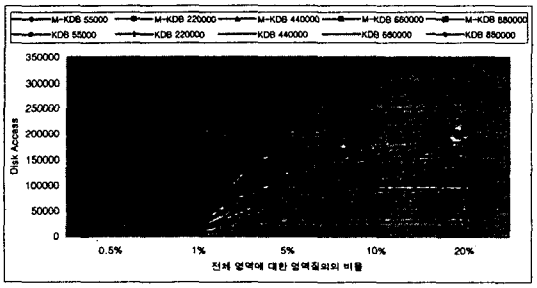


그림 11 질의영역별 검색 시 Disk Access 수

## 5. 결론

본 논문에서는 시공간에서의 이동체 색인을 위해 시간 도메인을 고려한 KDB-Tree 의 분할 정책들을 제안하고 제안한 방법을 구현하여 성능평가를 실시하였다. 성능평가 실험을 통해서 시간 도메인을 고려한 분할 정책을 사용한 KDB-Tree 가 공간활용도와 검색 성능에서 기존의 KDB-Tree 의 성능보다 우수한 것을 알 수 있었다. 제시한 분할 정책을 사용한 KDB-Tree 의 경우 색인의 구축비용이나 검색 비용에서 향상된 성능을 보였으며 특히 주기적 분포의 데이터 유형에서 월등한 성능을 나타내었다.

## 6. 참고 문헌

[1] Ratko Orlandic, Byunggu Yu: Implementing KDB-Trees to Support High-Dimensional Data. IDEAS 2001 : 58-67

[2] J.T. Robinson, "The K-D-B Tree : A Search Structure for Large Multidimensional Dynamic Indexes" Proc. ACM SIGMOD Int. Conf. On Management of Data, 10-18, 1981

[3] Betty salzberg, vassilis j. tsotras, " Comparison of Access Methods for Time-Evolving Data" ACM Computing Surveys, Vol.31, No.2, pp.158-221, 1999

[4] Y.Theodoridis, J.R.O Silva, and M.A Nascimento, " On the Generation of Spatio-Temporal Datasets" SSD 1999, Hong-Kong, LNCS 1651, Springer, p147-164

[5] 이창헌, 임덕성, 홍봉희, " KDB-Tree 를 사용한 이동체 색인의 동적 변경", 한국정보과학회 데이터베이스 연구회 2002 학술발표논문집, 제18권 2호, p117-124

[6] M. Freeston, " A General Solution of the N-dimensional B-tree Problem," Proc, ACM SIGMOD Int. Conf. On Management of Data, 80-91, 1995