다차원 색인구조 M-트리에서 노드 색인 공간의 중첩을 최소화하기 위한 효율적인 분할 알고리즘

임 상 혁[†] · 구 경 이[††] · 김 기 창[†††] · 김 유 성[††††]

요 약

멀티미디어 데이타를 위한 내용기반 검색 서비스의 속도를 증진하기 위해 다차원 색인 기법에 대한 연구가 활발하게 진행되고 있다. 다차원 색인 기법의 하나인 M-트리는 노드의 중심점과 객체간의 상대적 거리를 이용하여 색인을 구성하고, 검색 공간에 포함되는 객체를 액세스하는 기법으로서 노드들은 페이지 단위로 구성되며 하위 엔트리들을 포함할 수 있는 반경, 즉 유사도 거리에 의해 노드의 영역이 표현되어진다. 그러나 이와 같은 노드의 영역 표현에 있어서 노드 색인 공간의 중첩으로 인해 질의 시 검색해야 하는 노드수가 증가하고 이는 거리계산과 디스크 입출력의 횟수를 증가시킨다. 본 논문에서는 M-트리에서 문제가 되고 있는 노드 색인 공간 중첩을 최소화하는 노드 분할 정책을 제안한다. M-트리의 기존 분할 정책들과는 다르게 노드의 가상 중심점을 계산하여 라우팅 객체로 이용하여 노드 색인 공간의 중첩을 최소화하고 노드 안의 엔트리 재분배를 통해 노드의 색인 공간의 크기를 작게 유지하며 밀도 높은 노드를 구성하도록 한다. 실험으로부터 제안된 노드 분할 알고리즘이 라우팅 노드의 색인 공간의 반경을 작게 유지하며 결과적으로는 사용자 질의에 대해 개선된 응답 시간을 제공하는 것으로 판명되었다.

An Efficient Split Algorithm to Minimize the Overlap between Node Index Spaces in a Multi-dimensional Indexing Scheme M-tree

Sang-hyuk Im[†] · Kyong-I Ku[††] · Ki-chang Kim[†††] · Yoo-Sung Kim[††††]

ABSTRACT

To enhance the user response time of content-based retrieval service for multimedia information, several multi-dimensional index schemes have been proposed. M-tree, a well known multidimensional index scheme is of metric space access method, and is based on the distance between objects in the metric space. However, since the overlap between index spaces of nodes might enlarge the number of nodes of M-tree accessed for query processing, the user response time for content-based multimedia information retrieval grows longer. In this paper, we propose a node split algorithm which is able to reduce the size of overlap between index spaces of nodes in M-tree. In the proposed scheme, we choose a virtual center point as the routing object and entry redistribution as the postprocessing after node split in order to reduce the radius of index space of a node, and finally in order to reduce the overlap between the index spaces of routing nodes. From the experimental results, we can see the proposed split algorithm reduce the overlap between index space of nodes and finally enhance the user response time for similarity-based query processing.

키워드 : 다차원 색인 기법(Multi-dimensional Indexing Scheme), M-tree, 노드 분할(Node Split), 중첩 최소화(Minimizing Overlap), 라우팅 객체 선정(Selection Of Routing Objects)

1. 서 론

최근 멀티미디어 데이타베이스 응용에서 내용기반 검색을 위한 유사성에 기반한 검색기법에 대한 관심이 부각되고 있다. 유사성에 기반한 내용기반 검색을 멀티미디어 데이타베이스에 적용하기 위해서는 멀티미디어 데이타가 다차원, 대용량의 특성을 갖고 있기 때문에 기존의 색인 기법과는 다른 다차원 색인을 위한 효율적인 저장과 검색 기법이 필요하다[1, 2, 3, 4].

다차원 색인 구조의 대표적인 부류의 예로서 SAM(Spatial Access Methods)와 MAM(Metric Access Methods)를 들 수 있다[5, 6, 7]. SAM은 데이타 객체들을 다차원 벡터 공간에 매핑하여 객체의 절대적 위치좌표를 기준으로 색인을 구성하는 방법들을 의미한다. R-트리계열[8]의 트리들과 SS-트리[9, 10], SR-트리[10, 11], X-트리[12]등의 구조가 SAM이며 주로 데이타를 위한 영역 표현 방법 혹은 디렉토리 노드의 구조나 영역 분할 알고리즘이 이들 간의 차이점이라 할 수 있다. 반면, MAM은 다차원 데이타간의 상대적인 거리로

† 준 회 원 : 코난테크놀로지 연구원
†† 정 회 원 : 한국전자통신연구원 연구원
††† 정 회 원 : 인하대학교 정보통신공학부 교수
†††† 종신회원 : 인하대학교 정보통신공학부 교수
논문접수 : 2003년 9월 29일, 심사완료 : 2005년 2월 16일

서 메트릭 공간에 매핑하여 색인을 구성한다. MAM의 예로는 vp-트리[13], mvp-트리[14], gh-트리[13], M-tree[15] 등을 들 수 있다.

이러한 다차원 색인구조에서는 차원이 증가할수록 저장과 검색 효율이 떨어지는 차원의 저주(the curse of dimensionality)문제와 노드간의 중첩의 증가로 인해 거리계산이나 디스크 입출력이 증가하는 문제 등이 해결해야 할 문제점으로 지적되고 있다. 특히 SAM과는 다르게 두 객체간의 거리에 기반을 둔 색인 구조인 MAM에서는 차원의 증가에 따른 문제보다는 주로 질의 시 거리계산과 디스크 입출력의 횟수를 줄이기 위한 색인 구조들이 연구의 주안점이 되고 있다[16]. 하지만 대부분의 MAM방법들은 노드의 색인 공간 중첩으로 인해 질의 시 검색될 노드가 많아지므로 거리계산과 디스크 입출력이 크게 증가하는 문제를 안고 있다[7].

본 논문에서는 이러한 색인 공간 중첩의 감소를 위해 MAM의 대표적 색인 구조인 M-트리[15]를 위한, 새로운 노드 분할 알고리즘을 제안한다. 기존의 M-트리에서는 라우팅 객체를 오버플로우가 발생한 하부노드의 엔트리 중에서 선택함으로써 노드의 표현에 있어서 불필요하게 색인 공간이 커지고 이로 인해 다른 노드의 색인 공간과 중첩이 될 확률이 높아지기 때문에 질의 처리시 불필요한 디스크 접근을 필요로 하는 문제점을 가지고 있다. 그러나 본 논문에서 제안하는 분할 알고리즘은 기존 M-트리와는 다르게 노드의 중심점을 계산하여 그 중심점을 노드의 가상 라우팅 객체로 이용함으로서 노드 색인 공간의 크기를 최소화하고 노드간의 중첩을 감소시킬 수 있다. 이를 위해 메트릭 공간상의 데이타 분할과 중심점 선출을 위한 클러스터링 기법을 새롭게 제안하였다. 또한, 분할 후 처리 과정으로서 노드의 반경을 더욱 더 줄이면서 밀도 높은 노드들로 트리를 구성하는 엔트리 재분배 과정을 제안하였다. 제안된 알고리즘의 성능을 평가하기 위해 다차원 데이타 집합의 차원과 크기를 변화시키면서 생성된 노드의 개수와 반경 그리고 질의 응답 시간을 측정하는 실험을 하였다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구로서 기존의 MAM 기법들에 대해 간략하게 요약하고 장단점을 지적한다. 3장에서는 색인 공간의 중첩을 최소화하기 위해 본 논문에서 제안하는 노드분할 및 엔트리 재분배 정책을 설명하고, 4장에서는 다양한 실험을 통해 제안된 기법의 성능을 검증한다. 5장에서는 결론을 내리고 향후 연구방향을 제시한다.

2. 관련 연구

2.1 다차원 데이타 색인을 위한 MAM(Metric Access Method)

다차원 데이타의 색인구조로서 지리정보 시스템 등에서 널리 사용되고 있는 SAM에서 색인된 객체들은 단지 다차원 벡터 공간에서의 특성 값으로 표현되어야 하며 두 객체간의 유사도는 유클리드 거리로 측정되어야 한다는 제약점이 있다[15]. 예를 들어 유전자 데이타 집합(genome dataset)이나 영어 단어 집합과 같은 알파벳 순서의 경우, 데이타의 특성 표현은 벡터가 아니며 유사도의 측정 역시 유클리드 거리가 아닌 편집 거리를 사용해야 한다.

이러한 문제의 해결을 위해 MAM이 제안되었다. MAM은 거리함수를 사용하여 각 객체간의 상대적 거리를 측정한 후, 이를 바탕으로 메트릭 공간상에 매핑하게 된다. 메트릭 공간 M은 $M=(D,d)$(D는 특성 값들의 도메인 - 색인 키)와 같이 정의 되며, 거리함수 d의 특징은 다음과 같다. 여기서 O_x, O_y, O_z는 도메인 D상의 객체를 의미한다.

1. 대칭성(symmetry): $d(O_x, O_y) = d(O_y, O_x)$
2. 절대성(non-negativity): $d(O_x, O_y) > 0 (O_x \neq O_y)$과 $d(O_x, O_x) = 0$
3. 삼각부등식(triangle inequality): $d(O_x, O_z) \leq d(O_x, O_y) + d(O_y + O_z)$

특히 MAM은 위의 세 가지 특징 중 삼각부등식을 이용하여 검색 시 불필요한 노드를 제거함으로써 거리계산이나 디스크 입출력을 줄일 수 있는 필터링 능력을 가지고 있다[15].

MAM은 크게 정적인 색인구조와 동적인 색인구조로 나눌 수 있다. vp-트리[13], mvp-트리[14], gh-트리[13]등의 색인 구조는 우세점(vantage point)이라 불리는 특정 객체나 데이타 객체간의 상대적 거리에 따라 각각 데이타 영역을 분할하고 그를 색인하는 방법으로서 정적인 색인구조에 속한다. 이와 같은 경우, 하향식으로 균형 트리를 구성하며 트리를 구성하기 전에 데이타 집합에 대한 모든 분석이 요구되기 때문에 객체의 동적인 삽입과 삭제는 불가능하다. 반면에 M-트리[15]와 같은 색인구조는 각 데이타 객체간의 상대적 거리를 이용하여 상향식 균형 트리를 구성하게 된다. 즉, M-트리는 트리의 재편성을 최소화하며 삽입과 삭제가 가능한 SAM의 장점을 결합시킨 동적인 색인 구조이다.

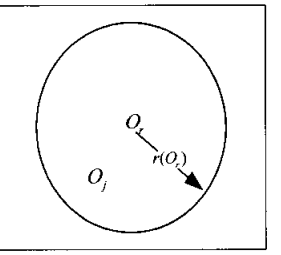
2.2 M-트리를 위한 노드 구조

M-트리는 객체간의 상대적 거리를 이용하여 주어진 검색 공간을 분할하는 MAM의 기본적인 원리를 따르면서 다음과 같은 특징을 갖는다.

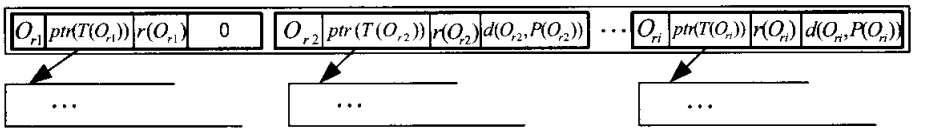
1. 동적 특성 : 저장과 검색 효율의 저하와 전체 트리의 재편성 없이 동적인 삽입과 삭제가 가능하다.
2. 페이지 특성 : 트리는 고정 크기의 노드(페이지 단위)로 구성된다.
3. 균형 특성 : 모든 단말노드들은 루트로부터 같은 레벨에 위치한다.

M-트리는 실제 객체를 저장하는 단말 노드(leaf node)와 라우팅 객체를 저장하는 내부노드(internal node)로 이루어진다. 객체들은 참조되는 라우팅 객체들로부터의 거리에 기반을 두어 재귀적으로 구성되며 라우팅 객체들은 노드의 분

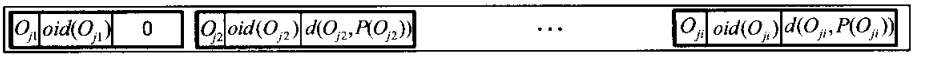
내부노드	O_r	라우팅 객체
	$ptr(T(O_r))$	부트리의 루트를 가리키는 포인터
	$r(O_r)$	O_r의 반지름
	$d(O_r,P(O_r))$	O_r과 O_r의 부모노드간의 거리
단말노드	O_j	데이타 객체
	$oid(O_j)$	객체 식별자
	$d(O_j,P(O_j))$	O_j와 O_j의 부모노드간의 거리



(그림 1) M-트리의 노드 종류와 엔트리 구조



(a) 내부노드 구조



(b) 단말노드 구조

(그림 2) M-트리 노드들의 도식적 표현

```
RS (N: node, Q: query object, r(Q): search radius)
1. {
2.   Let O_p be the parent object of node N;
3.   if N is not a leaf then                                  /* 내부 노드 : 라우팅 객체를 저장 */
4.   {   for ∀O_r in N do :                                   /* 노드 N의 모든 엔트리에 대해 */
5.         if | d(O_p, Q)  d(O_r, O_p) | ≤ r(Q)+r(O_r) then  /* 삼각 부등식을 적용 */
6.         {   Compute d(O_r, Q);                           /* 질의와 객체간의 거리를 계산 */
7.             if d(O_r,Q) ≤ r(Q)+r(O_r) then               /* 탐색범위 안의 엔트리에 대해 */
8.             {  RS( *ptr( T(O_r) ), Q, r(Q) );  } } }     /* 하위 트리로 재귀적 실행 */
9.   else                                                    /*  객체가 실제 저장된 노드 */
10.  {  for ∀O_j in N do :
11.       if | d(O_p, Q) - d(O_j, O_p) | ≤ r(Q) then        /* 삼각 부등식 적용 */
12.       {  Compute d(O_j, Q);
13.          if d(O_j, Q) ≤ r(Q) then                      /* 질의 범위내의 객체 */
14.          {  add oid(O_j) to the result;  } } }         /* 결과에 추가 */
15. }
```

(그림 3) M-트리의 범위질의 알고리즘

할 시 선택된 객체들로서 이루어진다. M-트리의 노드 종류와 엔트리 구조는 (그림 1)과 같고, 노드들의 도식적 표현은 (그림 2)와 같다.

(그림 2)에서 볼 수 있듯이 내부노드와 단말노드의 첫 번째 엔트리와 그 부모노드와의 거리($d(O_r,P(O_r))$ 혹은 $d(O_j,P(O_j))$)는 0으로서 이는 첫 번째 엔트리가 노드 색인 공간의 중심임을 의미한다. 실제로 M-트리에서는 노드안의 엔트리들과 노드의 중심까지의 거리에 따라 노드안의 엔트리들이 정렬되어 저장된다.

2.3 M-트리의 질의 처리

M-트리에서 지원하는 질의 형태는 범위질의(range query)와 K-근접질의(K-nearest neighbor query)가 있다. 범위질의는 질의 객체 Q에 대해 거리 $r(Q)$ 이하의 조건 $d(O_j,Q)\le r(Q)$을 만족하는 모든 객체 O_j를 반환하는데, 루트에서 시작하여 거리함수의 성질 중 삼각부등식을 만족하는 객체를 포함하는 모든 패스를 따라 재귀적으로 검색이 수행된다. $d(O_r,Q)>r(Q)+r(O_r)$를 만족하는 부트리 $T(O_r)$은 안전하게 검색 대상에서 제거되어질 수 있다. 하지만 실제로 M-트리에서는 거리계산을 줄이기 위해 미리 계산되어 저장된 거리인 $d(O_r,P(O_r))$를 사용한다. 즉, $|d(O_p,Q)-|d(O_r,O_p)||>r(Q)+r(O_r)$을 이용하여 노드안의 모든 엔트리와의 거리 계산 없이 불필요한 노드의 검색을 피할 수 있다. 범위질의 알고리즘을 나타낸 (그림 3)에서 5행과 11행은 이러한 방법을 나타내고 있다.

K-근접질의는 질의 객체 Q로부터 유사도 거리 값이 가장 가까운 k개의 객체들을 반환한다. 기존의 R-트리계열의 색인 구조에서 쓰인 분기 한정법(branch-and-bound)을 이

용하며, 필요한 자료구조로는 검색 대상이 될 부트리의 포인터를 저장하고 있는 우선순위 큐 PR과 k개의 결과를 저장하는 배열 NN이다. 중요한 파라미터와 알고리즘은 각각 (그림 4)와 (그림 5)와 같다.

d_{min}	$\max\{d(O_r,Q) - r(O_r),0\}$
d_{max}	$d(O_r,Q) + r(O_r)$
d_k	결과배열 NN에서 가장 큰 거리

(그림 4) 근접질의를 위한 파라미터

근접질의 처리를 위한 (그림 5)의 알고리즘에서 K-NN_Search() 메소드는 PR과 NN을 초기화하며 5행에서 ChooseNode() 메소드를 호출하여 검색 대상이 되는 부트리(노드)를 받아온다. 즉, ChooseNode() 메소드는 PR에서 가장 작은 d_{min}을 갖는 노드를 리턴하게 된다. 이 노드를 인자로 하여 7행에서 K-NN_NodeSearch() 메소드를 호출한다. K-NN_NodeSearch() 메소드는 내부노드의 경우, 16행에서 볼 수 있듯이 $d_{min} \le d_k$를 이용하여 검색할 부트리들을 찾아(branch) PR에 삽입하며, 18행의 $d_{max} < d_k$로 대상 부트리의 객체들을 안전하게 NN배열에 넣고 d_k를 갱신 한다. 갱신 후에는 $d_{min} \ge d_k$인 부트리들을 PR에서 제거하게 된다. 단말노드 역시 내부노드와 비슷한 처리과정을 거치며, 단지 27행과 같이 결과배열에 실제 데이타의 객체 식별자 $oid(O_j)$를 삽입하는 것이 다르다. 이러한 작업들은 d_k보다 작은 d_{min}을 갖는 부트리가 없을 때까지 계속된다. 이는 곧 우선순위 큐 PR에 더 이상 엔트리가 들어있지 않은 상태를 의미한다.

3. 색인 공간의 중첩을 최소화하기 위한 분할 알고리즘

3.1 색인 공간 중첩에 따른 문제 정의

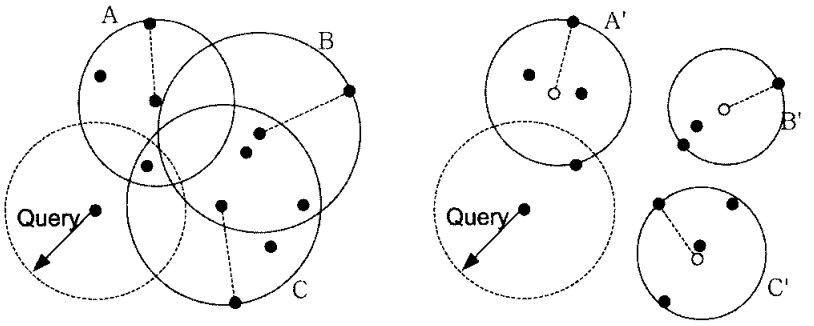
노드 색인 공간의 중첩은 다차원 색인의 성능에 영향을 미치는 매우 중요한 요소이다. 색인 공간의 중첩이 거리계산이나 디스크 접근 횟수와 같은 검색 효율에 미치는 영향은 (그림 6)을 보면 알 수 있다.

(그림 6)은 색인 공간 중첩의 문제점을 범위질의를 통해 보여 주고 있다. 중첩된 색인 공간이 큰 경우((그림 6(a)) 참조), 객체를 반환하기 위해 질의 범위와 겹치는 A, B, C 세 개의 공간을 탐색하기 위해 총 3회의 디스크 접근이 필요하다. 또한, 거리 계산 측면에서는 질의 객체와 세 노드의 라우팅 객체 간의 거리계산 3회와 실제 범위 안에 속한 객체와의 거리계산 1회를 합쳐 총 4회의 거리계산이 필요하다.

```
K-NN_Search( T: root node, Q: query object, k: integer)
1. {
2.    PR={ T, _ };                                   /*  initialize priority queue      */
3.    for i=1 to k do : NN[i] =[ _, ∞ ];             /*  initialize NN-array            */
4.     while PR ≠ Ø do :                             /*  until PR is empty              */
5.      {  Next_Node = ChooseNode(PR);               /*  return node which has          */
6.                                                   /*  the smallest  d_min from PR    */
7.          K-NN_NodeSearch(Next_Node, Q, k);
8.      }
9.  }

K-NN_NodeSearch (N: node, Q: query object, k: integer)
10. {
11.   Let O_p be the parent object of Node N;
12.   if N is not leaf then
13.    { for ∀O_r  in N do :
14.        if | d(O_p,Q)  - d(O_r,O_p) | ≤ d_k + r(O_r) then      /* avoid distance computation */
15.         {  Compute d(O_r ,Q);
16.           if d_min(T(O_r)) ≤ d_k  then
17.            {  Add [ ptr(T(O_r)), d_min(T(O_r)) ] to PR;
18.               if d_max(T(O_r)) < d_k  then
20.                {  d_k = NN_Update( [ _, d_max(T(O_r)) ] );
21.                     Remove from PR all entries for which d_min(T(O_r)) ≥ d_k ; }   }   }   }
22. else                        /*  if N is leaf node */
23.    {  for ∀O_j  in N do :
24.           if |d(O_p,Q) - d(O_j, O_p)| ≤ d_k then
25.            {  Compute d(O_j, Q);
26.              if d(O_j,Q) ≤ d_k then
27.               {   d_k =NN_Update( [ oid(O_j), d(O_j, Q) ] );
28.                    Remove from PR all entries for which d_min(T(O_r)) > d_k ;   }   }   }   }
30. }
```

(그림 5) M-트리의 근접질의 알고리즘



(a) 중첩된 색인공간의 증가 (b) 중첩된 색인공간의 최소화

(그림 6) 색인 공간 중첩의 증가와 최소화

```
Insert( N: node, entry(On): new M-tree entry )
1. {
2.   Let Ns be the set of entries in N;
3.   if N is not a leaf node then                                  /* 중간노드의 경우 */
4.    { Let Ns_in be the entries such that d(Or, On)≤r(Or)        /* 범위내의 라우팅 객체 */
5.        if Ns_in≠Ø then                                        /* 존재하는 경우 */
6.            let Or* ∈ Ns_in : d(Or*, On) is minimum;           /* 가까운 노드를 선택 */
7.        else                          /* if Ns_in is null */   /* 존재하지 않는 경우 */
8.         { let Or* ∈ Ns : d(Or*, On) - r(Or*) is minimum;      /* 반경 증가가 최소인 경우 선택 */
9.           let r(Or*) = d(Or*, On); }
10.       Insert( *ptr(T(Or*)), On ); }
11.  else                        /* if N is a leaf node */       /* 단말 노드의 경우 */
12.   { if N is not full then
13.       store On in N;
14.     else Split(N, On);      /* node overflow */ }            /* 오버플로우인 경우 분할 */
15. }
```

(그림 7) M-트리의 삽입 알고리즘

반면 (그림 6(b))의 경우, 임의의 객체를 라우팅 객체를 선정한 후 노드의 반경을 줄여 중첩 공간을 최소화한 경우이다. 이 경우 질의 범위와 겹쳐진 A′ 공간만을 탐색하기 위해 1회의 디스크 접근과 질의 객체와 라우팅 객체, 그리고 범위안의 객체 각각 1회씩 총 2회의 거리계산이 필요하게 된다. 이와 같이 같은 레벨의 중첩된 색인 공간을 감소시키면 검색 시 필요한 거리계산과 디스크 입출력을 크게 감소시킬 수 있다. 따라서 M-트리에서 색인 공간의 중첩을 감소시키기 위해서는 노드의 중심을 의미하는 라우팅 객체의 선정과 노드를 커버할 수 있는 반경을 최소로 만드는 것이 중요하다.

3.2 M-트리의 삽입

M-트리의 삽입 알고리즘은 새로 삽입될 객체 O_n이 들어갈 적당한 노드를 찾기 위해 하위레벨로 재귀적으로 검색이 이루어지며 도달한 단말 노드에 O_n을 저장한다. 여기서 말하는 적당한 노드란 $d(O_r, O_n) \le r(O_r)$을 만족하는 O_r 노드들 중 O_n에 가장 가까운 노드를 의미하며 이러한 부등식을 만족하는 노드가 없을 경우에는 새로운 객체를 추가했을 때 증가하는 반경이 가장 작은 노드를 선택한다. 의사코드로 표현된 삽입 알고리즘은 (그림 7)과 같다.

알고리즘의 6행과 8, 9행은 위에서 언급된 적당한 내부노드를 찾는 과정을 나타내며, 단말 노드에서는 12, 13행과 같이 실제 데이타를 저장하는 부분과 14행과 같이 노드에 오버플로우가 발생한 경우 분할 메소드를 호출하는 부분을 볼 수 있다.

3.3 M-트리의 노드 분할

오버플로우된 노드로부터 내부노드의 라우팅 객체가 될 두개의 객체를 선정하여 상위의 내부노드로 진급(promotion)시키게 된다. 이러한 진급 정책(promotion policy)의 몇 가지를 소개하면 <표 1>과 같다.

노드 분할을 위한 처리과정((그림 8) 참조)에서 새로운 라우팅 객체를 선출하기 위한 Promotion() 메소드를 호출하고 오버플로우 노드의 영역을 분할하기 위한 Partition() 메소드를 호출한다. 즉, 라우팅 객체의 진급 후 노드안의 나머지 객체들에 대해 각각 라우팅 객체를 중심으로 한 노드로 편입 시키게 되는데 이러한 과정은 Partition() 메소드를 통해 이루어진다. Partition() 메소드는 <표 2>와 같이 두 가지가 있다.

(그림 8)의 6, 7행에서 각각 진급과 파티션을 언급하고 있으며, 14행은 재귀적으로 상향식 분할이 이루어지고 있음을

〈표 1〉 라우팅 객체 선출을 위한 진급 정책

m_RAD	복잡도가 가장 큰 정책으로서 가능한 모든 객체 쌍간의 거리를 계산하여 $r(O_{p1})+r(O_{p2})$가 최소인 두 점을 라우팅 객체로 선출 한다.
mM_RAD	O_{p1}과 O_{p2}의 최대 반경이 최소가 되는 두 객체를 선출한다.
M_LB_DIST	$O_{p1}=O_p$가 되고 O_{p2}는 O_p로부터 가장 멀리 떨어진 객체가 된다. 미리 계산되어진 거리만을 사용한다.
RANDOM	O_{p1}과 O_{p2}를 랜덤하게 선출한다.
SAMPLING	분할될 노드의 엔트리 중 s개의 샘플을 랜덤하게 선출하고, 선출된 엔트리들에서 m_RAD방법을 이용한다. 예를 들어 s는 노드의 엔트리 최대 저장 개수의 1/10의 값으로 한다.

〈표 2〉 객체 파티션을 위한 파티션 정책

일반적 경계 파티션 (generalized hyperplane partition)	O_j는 O_{p1}, O_{p2}중 가장 가까운 쪽에 포함된다. 분할 후 두 노드에 포함된 엔트리 개수가 다를 수 있다.
균형 파티션 (balanced partition)	분할된 두 노드의 엔트리 개수를 같게 한다. 즉, O_{p1}에서 가장 가까운 객체들을 포함시키고(전체 엔트리의 1/2), 나머지 객체들은 O_{p2}로 포함시킨다.

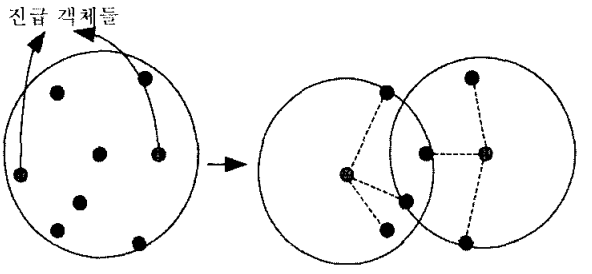
```
Split( N: node; E: new M-tree entry)
1. {
2.    Let N_s be entries of node N ∪ { E };
3.    if N is not the root then
4.       let O_p be the parent of N, stored in N_p node;
5.    Allocate a new node N';
6.    Promotion(N_s, O_p1, O_p2);                  /* select two routing objects O_p1, O_p2 from N_s */
7.    Partition(N_s, O_p1, O_p2, N_s1, N_s2);  /*  divide entries of N_s  into N_s1, N_s2          */
8.    Store N_s1's entries into N and N_s2's entries into N' ;
9.    if N is current root node then
10.   {  Allocate a new root node N_p;
11.      Store entry(O_p1) and entry(O_p2) in N_p;   }
12.   else
13.   {  Replace entry(O_p) with entry(O_p1) in N_p;        /* set the routing object          */
14.      if N_p is full then Split(N_p, O_p2);              /* Split is bottom-up process */
15.      else Store entry(O_p2) in N_p;   }                 /* set the routing object          */
16. }
```

(그림 8) M-트리의 분할 알고리즘

보여준다.

진급 정책과 파티션 정책에 대한 효율성의 비교는 [15]을 참조했다. 진급 정책에서 분할 시 생성되는 두 노드의 반경을 최소한으로 만드는 m_RAD 정책이 데이타 삽입시의 오버헤드($O(n^3)$)는 가장 크지만 검색 시의 효율 면에서는 가장 좋음을 알 수 있다. 그리고 파티션 정책면에서는 일반적 경계 파티션이 반경을 줄일 수 있는 객체 파티션에 훨씬 좋은 성능을 보이는 반면 한쪽 노드로만 데이타들이 몰리는 경우 잦은 분할과 저장 효율의 감소로 인해 트리의 높이가 증가할 가능성이 존재하게 된다. 본 논문에서 제안하는 분할 방법에서는 검색 시 디스크 입출력과 거리계산을 줄이는 것이 최대 목표이므로 일반적 경계 파티션을 데이타 파티션 방법으로 이용한다.

그러나 기존의 M-트리 구조와 분할 알고리즘은 노드의 엔트리 중 하나를 뽑아 라우팅 객체로 선정하므로 불필요하게 노드의 반경이 커지고 이로 인해 노드간의 중첩이 증가하는 문제점이 있다.



(a) 오버플로우 해결을 위해 진급 객체 선정 (b) 분할 후 노드 반경의 과도한 증가

(그림 9) 기존 분할 정책의 문제점

(그림 9)는 이와 같은 기존 분할 정책의 문제점을 나타내고 있다. 노드들의 중첩을 가장 줄일 수 있는 m_RAD 진급 정책을 이용하여 진급 객체를 선정하고 일반적 경계 파티션을 사용했음에도 불구하고 노드의 반경이 크고 이로 인해 오버랩이 발생할 가능성이 높다.

3.4 가상 중심점과 엔트리 재분배를 이용한 분할 알고리즘

3.4.1 가상 중심점

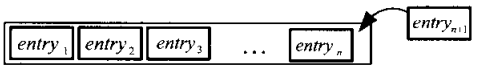
노드 안의 엔트리를 라우팅 객체로 선출한 기존의 분할 방법과는 달리 색인 공간의 중첩을 최소화하기 위해 본 논문에서 제안하는 방법은 노드 안의 모든 엔트리를 이용하여 가상 중심점을 계산한 후 중심점을 노드의 라우팅 객체로 선정한다. 새로운 분할 알고리즘을 위한 M-트리의 도식적 표현은 (그림 10)과 같다.

(그림 10(a))와 같이 노드에 오버플로우가 발생하면 노드 안의 데이타 파티션 과정을 통해 (그림 10(b))처럼 두개의 영역(노드)로 분할하게 되고, 각각 파티션된 영역의 엔트리들을 계산하여 가상의 중심점(virtual center point :*V_CP*)을 생성하며 생성된 가상 중심점을 상위 레벨의 부모 객체로 대체하게 된다((그림 10(c)) 참조). *V_CP*의 개념을 통해 내부 노드 엔트리들의 객체들은 모두 가상 점이 되며 단말 노드를 위한 라우팅 객체 역시 계산에 의해 얻어진 *V_CP*값으로 바뀌게 된다. 이러한 *V_CP*의 도입으로 인해 한 노드의 표현을 위한 최적의 라우팅 객체를 선출 할 수 있으며, 이로 인해 기존 분할 방법보다 더 작은 노드의 반경을 얻을 수 있다.

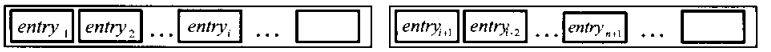
3.4.2 파티션을 위한 클러스터링 알고리즘

오버플로우가 발생한 노드를 분할하기 위해 노드 영역을 파티션하고 각 파티션영역의 중심점(*V_CP*)을 계산해야 한다. 이때 메트릭 공간상의 클러스터링 알고리즘을 이용할 수 있다.

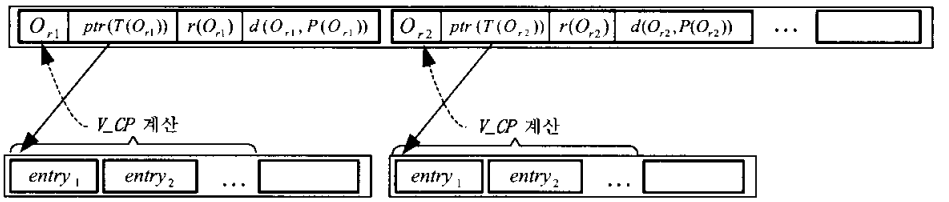
K-means 클러스터링 알고리즘[17]은 클러스터의 무게 중심점을 대표 값으로 분할해 나감으로서 메트릭 공간상의 데이타들을 분할하고 중심점을 찾는다. (그림 11(a))와 같이 임의로 데이타 집합을 두개의 클러스터로 파티션 한 후 (그



(a) 새로운 엔트리의 삽압과 오버플로우 발생

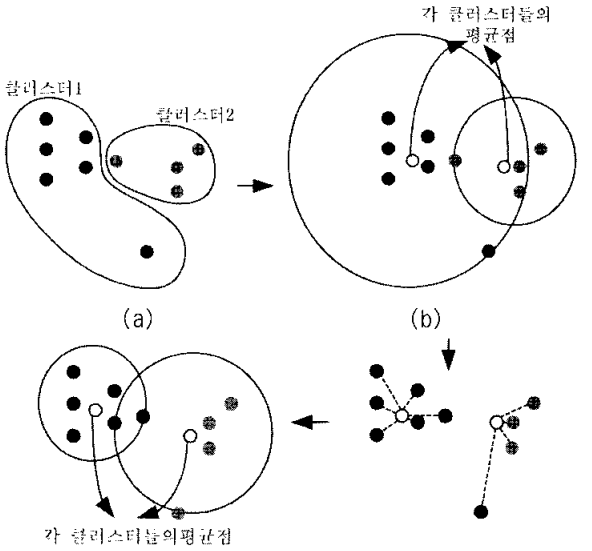


(b) 두 노드(영역)으로 파티션



(c) *V_CP*의 계산과 라우팅 객체 세팅

(그림 10) 새로운 분할 알고리즘을 위한 M-트리 구조

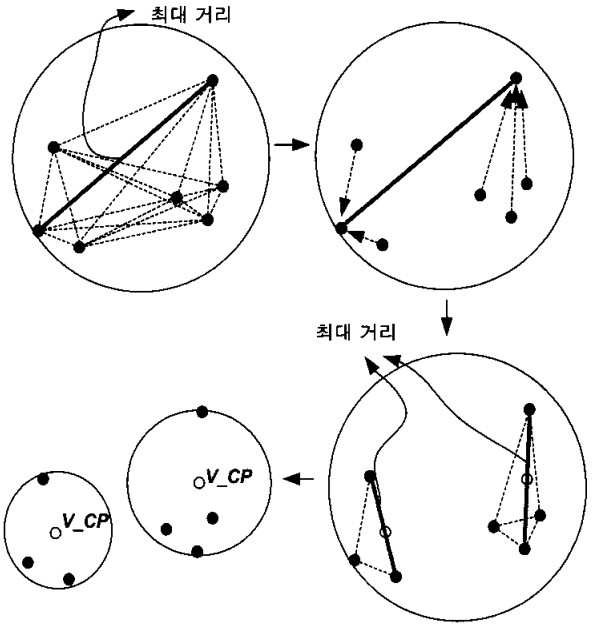


(그림 11) K-means 클러스터링

림 11(b))처럼 각 클러스터의 평균점(mean point)을 구한다. 그 후, 클러스터들의 평균점에 가까운 데이타들을 다시 묶어 (그림 11(c))과 같이 새로운 클러스터를 생성하고 이에 대한 새로운 평균점들을 구하게 된다((그림 11(d)) 참조).이와 같은 작업은 더 이상 평균점의 이동이 없을 때 까지 계속된다. 하지만 K-means 알고리즘은 무게 중심을 찾으므로 밀집도가 높은 곳으로 중심점이 이동하게 되고 또한 아웃라이어(outlier) 데이타에 민감하여 노드를 커버하는 반경의 증가를 가져오는 단점이 있다. K-medoids 클러스터링 알고리즘[17]은 시드(seed) 데이타와 각 데이타 간의 거리를 모두 더해 최소가 되는 객체를 찾아 클러스터의 대표점으로 한다. 이것은 기존 M-트리의 분할 방법인 m_RAD와 비슷한 방법으로서 모든 객체의 경우에 대해 실행을 해보아야 하므로 복잡도가 크게 증가하는 단점을 가지고 있다.

본 논문에서 제안하는 클러스터링 방법은 밀도와 관련된 K-means 클러스터링과는 달리 영역을 커버할 수 있는 최소의 반지름을 갖는 클러스터를 생성한다. 즉, 객체간의 거리를 각각 계산하고 최대 거리를 갖는 두 엔트리를 선정한 후 그 엔트리들에 각각 가장 가까운 객체들로 클러스터를 형성하게 된다. 클러스터의 중심점은 평균점이 아닌 클러스터의 엔트리쌍 중 최대 거리의 엔트리들의 중심점으로 한다.

(그림 12)와 같이 최대 거리쌍을 이용하여 분할을 하고 중심점을 찾아내므로 최소 반경을 갖는 두개의 클러스터(노드)와 클러스터간의 최소 중첩 공간을 얻을 수 있다. 알고리즘의 복잡도 역시 초기과정에서의 한 번의 엔트리 간 거리 계산 값을 이용함으로써 추가적인 오버헤드가 없다($O(n^2)$). 이는 최적의 클러스터를 찾기 위해 대표점과 모든 엔트리들 간의 계산을 반복하는 K-means, K-medoids등의 알고리즘보다 거리계산 및 복잡도가 감소한다는 것을 알 수 있다.



(그림 12) 노드분할을 위한 새로운 클러스터링 과정

새로운 클러스터링 방법을 사용한 분할 알고리즘은 (그림 13)과 같다.

(그림 13)에서 6, 7행은 가상 중심점을 계산하고 데이타들을 파티션하기 위해 사용될 각 엔트리들의 거리 행렬을 만드는 부분이다. 또한 8행에서 Pre_Partition() 메소드는 오버플로우 노드를 최대 거리를 갖는 엔트리 쌍을 기준으로 두 데이타 집합(클러스터)으로 나누는 것을 나타내며, 9행은 두 데이타 집합에서 각각의 엔트리 쌍 중 최대 거리 엔트리들의 중심점을 계산하는 부분이다. 계산되어진 V_CP는 각각

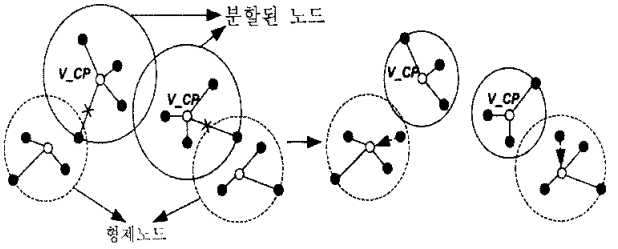
```
New_Split( N: node; E: new M-tree entry)
1. {
2.    Let Ns be entries of node N ∪ { E };
3.    if N is not the root then
4.       Let Op be the parent of N, stored in Np node;
5.    Allocate a new node N';
6.    Calculate distance matrix for all pairs of entry objects;
7.    Let (entry(Om), entry(On)) be max distance entry pair in distance matrix;
8.    Pre_Partition(Ns, Om, On, Ns1, Ns2);          /* divide entries of Ns into Ns1, Ns2  */
9.    Calculate V_CP1, V_CP2 on Ns1, Ns2 ;
10.    /* using medium on each max distance entry pair of Ns1, Ns2    */
11.  Store Ns1's entries into N and Ns2's entries into N' ;
12.  Tight_Node(N);                                  /* entry redistribution 1 */
13.  Tight_Node(N');                                 /* entry redistribution 2 */
14.  if N is current root node  then
15.   {   Allocate a new root node Np;
16.       Store entry(V_CP1)and entry(V_CP2) in Np ; }
17.  else
18.   {   Replace entry(Op) with entry(V_CP1) in Np;  /*set the routing object as v_cp*/
19.       if Np is full then Split(Np, V_CP2);        /* Split is bottom-up process      */
20.       else Store entry(V_CP2) in Np ;    }       /* set the routing object as v_cp */
21. }
```

(그림 13) 새로운 분할 알고리즘

18행과 20행에서 라우팅 객체로서 부모노드에 저장된다. 14~16행에서는 오버플로우 노드가 루트노드인 경우 새로운 루트 노드를 만들고 그 루트노드를 부모노드로서 *V_CP*들을 저장하며 12, 13행은 엔트리 재분배를 위한 메소드로서 다음 절에서 소개하기로 한다.

3.4.3 밀도 높은 노드의 생성을 위한 엔트리 재분배

분할의 후 처리과정으로 노드의 엔트리 재분배 기법을 소개한다. 보다 밀도 높은 노드를 만들기 위해 노드의 엔트리들을 형제노드로 재분배 한다. (그림 14(a))와 같이 분할된 노드의 중심에서 가정 먼 거리를 갖는 엔트리의 중심점까지의 거리가 다른 형제 노드의 중심점 보다 멀 때 (그림 14(b))처럼 그 노드에서 엔트리를 삭제한 후 형제 노드에 재삽입한다.



(a) 분할 후 노드 상태 (b) 엔트리 재분배

(그림 14) 엔트리 재분배 과정

엔트리 재분배를 위한 Tight_Node 알고리즘은 (그림 15)와 같다.

여기서 고려해야 할 사항은 재분배로 인한 형제 노드의 연속된 오버플로우를 막기 위해 형제 노드의 현재 엔트리 개수를 고려해야 한다는 것이다. (그림 15)의 7행에서 볼 수 있듯이 삽입될 수 있는 형제 노드 중 현재의 엔트리 개수가 노드가 최대로 수용할 수 있는 엔트리 개수 보다 1이상 작은 노드를 선택하게 된다. 이 기법을 통해 어떠한 노드들의 반경의 증가 없이 분할된 노드의 반경을 줄이고 밀도 높은 노드들로 트리를 구성할 수 있다.

5. 실험 및 평가

본 논문에서 제시한 알고리즘의 효율성을 다양한 실험을 통해 증명하였다. 실험을 위한 M-트리의 구현은 다차원 색인을 위한 GiST 0.9 패키지[18]를 기반으로 한 M-트리[15]를 이용하였다. 실험은 450MHz Pentium II PC(주기억 장치 256MB), Window2000 Professional 운영체제 환경에서 Visual C++6.0 컴파일러를 사용하여 수행되었다. 페이지 크기가 성능 비교에 영향을 끼치므로 모든 실험에서 4Kbyte로 페이지 크기를 고정 하였으며 실험 결과들의 평균치를 얻기 위해 10회의 질의를 통해 얻어진 값들의 평균치를 사용하였다.

실험 데이타는 난수 발생기를 이용하여 차원은 5차원에서 30차원까지 그리고 데이타 집합은 10,000개에서 100,000개까지 균등 분포와 불균등 분포를 갖는 [0,1] 범위의 실수 데이타 집합을 이용한다. 비교 대상으로서는 기존의 M-트리에서 질의 시 디스크 입출력과 거리계산에서 최고의 효율성을 지닌 m_RAD 진급 정책을 이용한 노드 분할 방법과 가장 간단한 진급 정책인 RANDOM 정책을 이용한 분할방법, 그리고 본 논문에서 제안한 가상 중심점과 엔트리 재분배를 이용한 분할방법의 성능 비교이다. 노드의 나머지 객체 파티션 정책은 공통적으로 일반적 경계 파티션 정책을 이용한다. 또한 성능 평가는 삽입과 근접질의, 범위질의 시 응답시간에 대해 수행하며, 생성된 트리의 노드(페이지) 개수와 평균 노드 반경을 통해 트리의 질을 평가한다.

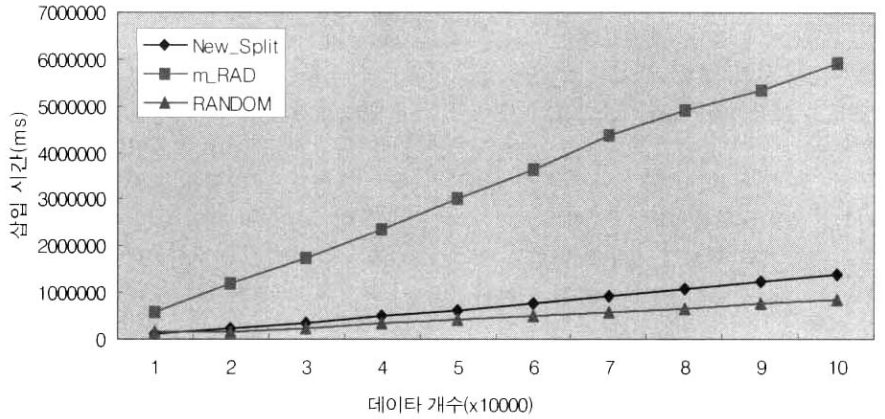
5.1 데이타 집합의 크기 증가에 따른 효율성 실험

데이타 집합의 증가에 따른 효율성을 실험하기 위해 본 실험에서는 균등 분포 데이타 집합의 차원을 10차원으로 고정하였으며 데이타의 크기는 10,000개에서 100,000개까지 증가 시켰다.

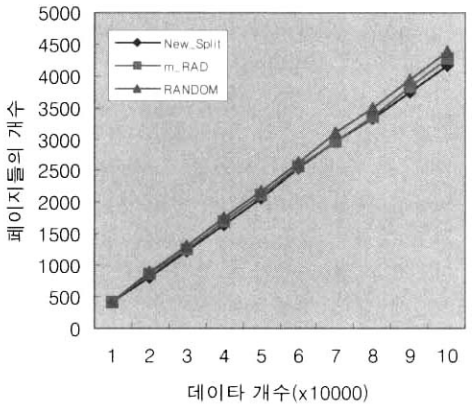
(그림 16)은 데이타 집합의 크기변화에 따라 전체 데이타 삽입 시 소요되는 삽입 시간의 변화를 나타내고 있다. 분할 시 진급 객체를 선정하기 위해 거리계산을 할 필요가 없는 RANDOM 분할 방법이 삽입 시간이 가장 작으며 *V_CP*와 엔트리 재분배를 이용한 New_Split 분할 방법이 그 다음으

```
Tight_Node( N: node )
1. {
2.   Let  O_max  be object which has max distance from center point(V_CP) in node N;
3.   Let Overflow_NO be the maximum entry number of node;
4.   Let N_sibling is N's sibling nodes;
5.   for ∀ N_sibling do :
6.     if d( N_sibling 'sV_CP, O_max ) < d( N's V_CP, O_max ) then
7.       {   if N_sibling 's entry number < Overflow_NO - 1 then  /*  prevent chain split  */
8.         {   Delete entry(O_max) in node N;
9.             Insert entry(O_max) into N_sibling ;
10.            break;                          /*   loop process terminated   */
11.        }
12.      }
13. }
```

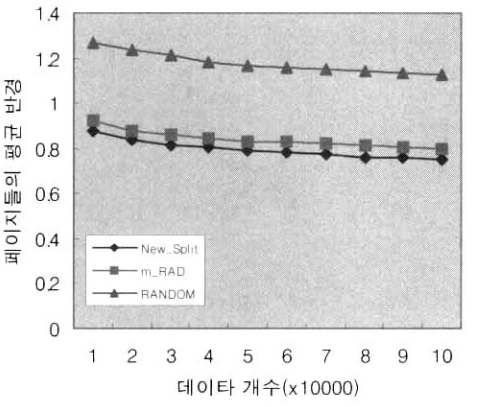
(그림 15) 엔트리 재분배를 위한 Tight_Node 알고리즘



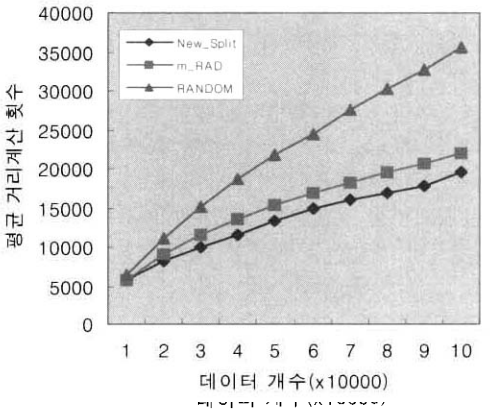
(그림 16) 데이타 집합의 크기변화에 따른 삽입시간



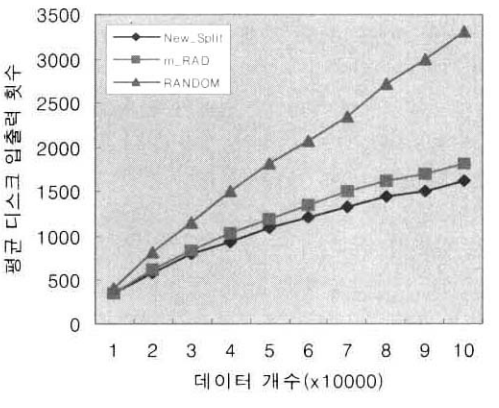
(그림 17) 생성된 페이지들의 개수



(그림 18) 생성된 페이지들의 평균반경



(그림 19) 평균 거리계산 횟수

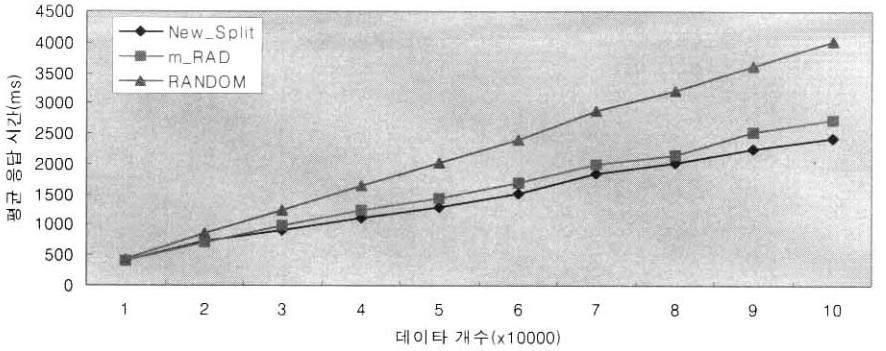


(그림 20) 평균 디스크 입출력 횟수

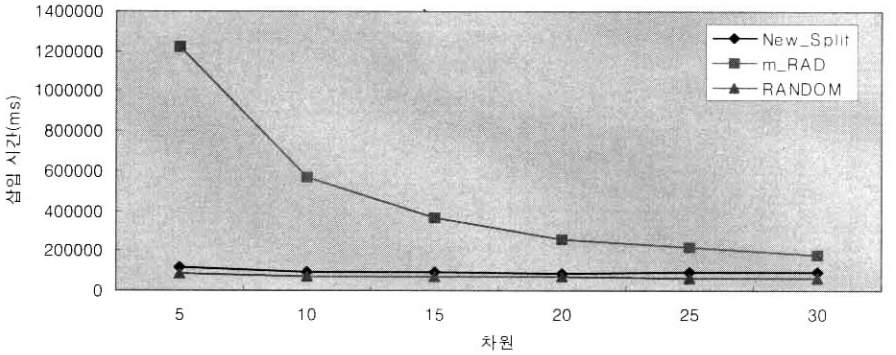
로 작은 삽입시간을 나타낸다. m_RAD 분할 방법이 다른 분할 방법들보다 약 5배 이상 삽입시간이 오래 걸리는 이유는 진급 객체의 선정을 위해 노드의 모든 객체 쌍에 대해 각 객체 쌍을 진급 객체로 만들어본 후 나머지 객체들을 파티션 시켜 가장 작은 반경의 합을 갖는 객체 쌍을 진급 객체로 선정하기 때문이다. 노드안의 모든 객체 쌍에 대해 각각 파티션을 시켜보아야 하므로 분할의 오버헤드가 다른 방법들에 비해 매우 큼을 알 수 있다. 즉, m_RAD의 경우 실행 시간은 $O(n^3)$인 반면 New_Split은 $O(n^2)$이다.

(그림 17)과 (그림 18)은 각각 트리가 생성된 후 노드들의 개수와 평균 반경을 나타내고 있다. 그래프에서 볼 수 있듯이 본 논문에서 제안하는 New_Split 분할 방법이 노드들의 수와 평균반경에서 다른 방법들에 비해 작은 것을 알 수 있는데, 이는 V_CP와 엔트리 재분배로 인한 노드들의 중첩 감소와 저장 효율의 증가를 의미한다.

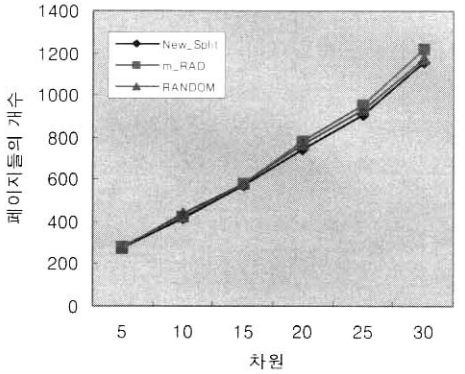
(그림 19)와 (그림 20)은 10개의 근접 객체를 찾는 근접질



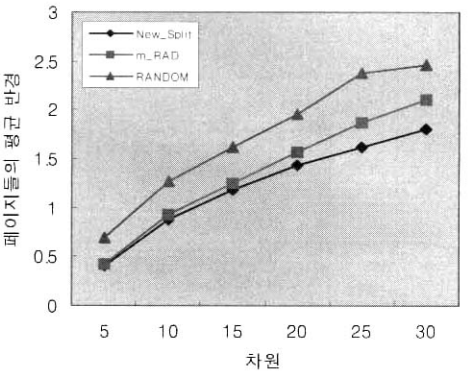
(그림 21) 근접 질의 시 평균 응답 시간



(그림 22) 데이타 집합의 차원 변화에 따른 삽입시간



(그림 23) 생성된 페이지들의 개수



(그림 24) 생성된 페이지들의 평균반경

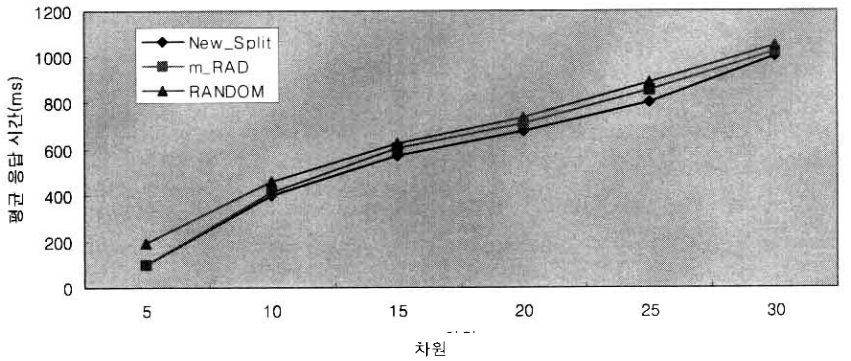
의를 10회 실시했을 경우 평균 거리계산과 디스크 입출력의 횟수를 나타낸다. New_Split 분할 방법이 노드 색인 공간 중첩의 감소로 인해 거리계산과 디스크 입출력이 가장 적음을 알 수 있는데, 이로 인해 평균 응답 시간을 나타낸 (그림 21)에서와 같이 질의 시 응답 시간이 가장 적게 걸린다.

5.2 데이타 집합의 차원 증가에 따른 효율성 실험

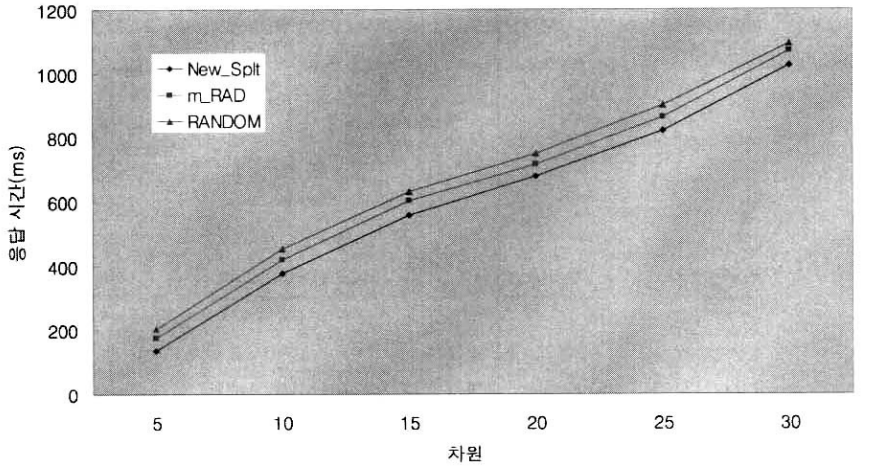
데이타 집합의 차원 증가에 따른 효율성을 실험하기 위해 본 실험에서는 균등 분포 데이타 집합의 크기를 10,000개로 고정하고 차원을 5차원에서 30차원까지 증가 시켰다. 차원의 증가는 곧, 노드가 가질 수 있는 엔트리의 개수를 감소시키는 효과가 있다.

(그림 22)는 차원이 증가함에 따라 전체 데이타 삽입 시 소요되는 삽입시간을 나타내고 있다. New_Split 분할 방법과 RANDOM 분할 방법은 차원변화에 따라 삽입시간의 변화가 크지 않지만 m_RAD 분할 방법의 경우 앞의 두 방법보다 큰 삽입시간을 갖으며 점차 감소되고 있음을 볼 수 있다. 이는 차원증가에 따라 노드의 저장능력이 감소하여 분할 시 진급될 엔트리를 선출하는데 고려해야 할 엔트리들의 개수가 줄기 때문이다. m_RAD의 경우 노드안의 엔트리 개수가 m이라면 노드의 분할 시 m/2번 노드를 분할 시켜본 후 가장 작은 반경을 갖는 경우의 엔트리를 진급 한다.

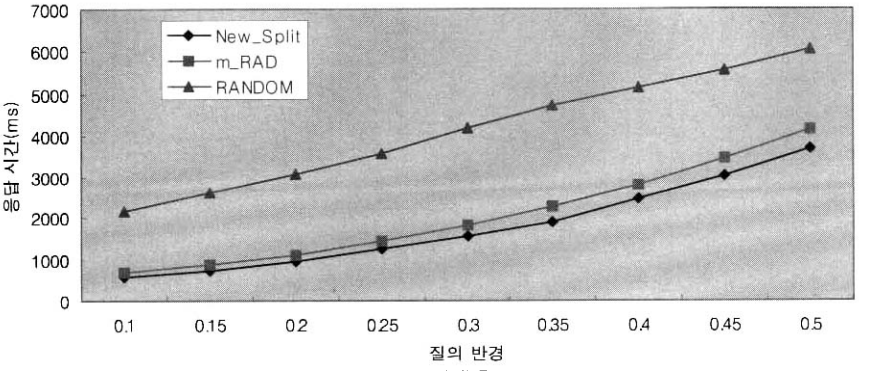
생성된 트리의 질적인 측면 역시 (그림 23)과 (그림 24)에서 볼 수 있듯이 차원 증가 시에도 New_Split 분할 방법이



(그림 25) 근접 질의 시 평균 응답 시간(균등 분포 데이타 집합)



(그림 26) 근접 질의 시 평균 응답 시간(불균등 분포 데이타 집합)



(그림 27) 질의 반경 증가에 따른 응답시간

가장 작은 노드 개수와 평균 노드반경을 얻으므로 중첩 색인 공간과 저장 효율 면에서 가장 좋은 성능을 나타낸다.

(그림 25)는 근접질의 시 평균 응답 시간을 나타내며 이를 통해 크기 증가 실험과 마찬가지로 차원증가 실험에서도 역시 New_Split 분할 방법이 노드들의 중첩 공간 감소로 인해 질의 효율 면에서 가장 좋은 성능을 보여주고 있다. 또한 불균등 분포 데이타 집합을 사용한 (그림 26)의 실험에서 볼 수 있듯이 불균등 분포에서도 균등 분포와 유사한 결과를 얻음을 보여주고 있다.

5.3 검색 범위에 따른 효율성 실험

범위질의에 대한 효율성 실험을 위해 데이타 집합의 크기는 100,000개 그리고 차원은 10차원으로 고정한 후 검색 범위를 늘리며 범위질의에 대한 응답시간을 측정하였다.

(그림 27)의 결과에서 볼 수 있듯이 고정된 질의점에서 반경을 증가 시켰을 경우에도 New_Split 분할 방법이 응답시간이 가장 빨랐음을 알 수 있다. 이는 본 논문에서 제안하는 분할 방법이 색인 공간의 중첩을 줄이는 측면에서 더 좋은 효율을 갖는다는 직관적 평가가 될 것이다.

실제 데이타에 이용한 실험은 [19]에서 쓰인 방법처럼, 한 MP3 데이타로부터 약 2초 간격으로 슬라이딩 방식으로 겹치면서 15~50차원으로 데이타들의 차원을 달리하여 추출하였다. 실험을 위해 100곡의 MP3데이타로부터 약 12,000개의 데이타를 추출하여 사용하였으며, 난수 발생기를 이용하여 추출된 데이타와 동일한 방식으로 거리 계산의 수, 디스크 입출력 횟수, 그리고 근접 질의 시 평균 응답 시간을 비교하였다. 실험 결과는 실험 데이타를 이용한 방식과 동일하게 본 논문에서 제안하는 분할 방법이 색인 공간의 중첩을 줄이는 측면에서 더 좋은 효율을 가짐을 알 수 있었다. 실제 데이타를 이용한 실험 결과는 지면상 생략한다.

6. 결 론

멀티미디어 데이타는 다차원, 대용량의 특성을 갖고 있기 때문에 이에 대한 내용기반 검색을 위해서는 기존의 색인 기법과는 다른 다차원 색인을 위한 효율적인 저장과 검색 기법들이 필요하다. 다차원 데이타의 색인 기법들은 주로 데이타의 영역 표현이나 노드의 구조, 영역분할 알고리즘을 통해 질의 시 발생되는 거리계산이나 디스크 접근의 횟수 등을 줄여 검색의 효율성을 높이는 것이 주된 목표이다.

본 논문에서는 다차원 데이타를 색인하고 질의하기 위한 기법 중 M-트리를 소개하고 성능 향상을 위해 노드의 색인 공간 중첩을 감소시키는 새로운 분할 방법을 제안하였다. 기존의 M-트리와는 다르게 분할 시, 엔트리간의 최대거리를 이용하여 오버플로우 노드를 파티션하고 노드들의 가상 중심점을 계산하여 라우팅 객체로 선정하므로서 노드를 포함하는 반경을 최소화하고 이를 통해 중첩 공간을 감소시킨다. 또한, 분할의 후처리 과정으로서 노드의 엔트리들 중, 중심점에서 가장 먼 엔트리를 형제노드로 삽입 시키는 엔트리 재분배를 통해 노드의 반경을 더욱 줄이면서 밀도 높은 노드들로 트리를 재구성 할 수 있었다. 실험에서는 기존의 분할 방법들과의 다양한 비교를 통해, 트리의 구성과 질의 시 응답 시간 측면에서 제안하는 알고리즘의 우수성을 증명하였다. 기존의 분할 방법과 비교하였을 때 삽입시간과 검색시간에서 성능 향상을 볼 수 있었다.

앞으로의 연구 방향은 오버플로우 노드의 데이타 객체 파티션 시, 분할된 노드의 불균등한 엔트리 개수로 인해 발생할 수 있는 잦은 분할 문제와 노드의 최소 엔트리 개수를 조정하여 트리의 팬아웃을 증진시키는 알고리즘의 연구가 필요하다.

참 고 문 헌

[1] C. Faloutsos, W. Equitz, M. Flickner, W. Niblack, D. Petkovic, and R. Barber, "Efficient and Effective Querying by Image Content," Journal of Intelligent Information System, 3(3/4), pp.231-262, 1994.

[2] C. Faloutsos and K. I. Lin, "FastMap: A Fast Algorithm for Indexing, Data-mining and Visualization of Traditional and Multimedia Databases," Proc. of Int. Conf. on ACM SIGMOD, pp.163-174, 1995.

[3] E. Wood, T. Blum, D. Keislar, and J. Wheaton, "Content-based Classification, Search, and Retrieval of Audio," IEEE Multimedia, 3(3), pp.27-36, 1996.

[4] R. Agrwal, C. Faloutsos, and A. Swami, "Efficient Similarity Search in Sequence Databases," Lecture Note in Computer Science, Springer-Verlag, Vol.730, pp.69-84, 1993.

[5] V. Gaedes and O. Gunther, "Multidimensional Access Methods," Technical Report TR-96-043, International Computer Science Institute, Berkely, CA, pp.36-59, 1996.

[6] C. Traina Jr., A. Traina, C. Faloutsoss, and B. Seeger, "Fast Indexing and Visualization of Metric Data Sets using Slim-Trees," IEEE Transaction on Knowldege and Data Engineering, 14(2), pp.244-259, 2002.

[7] C. Traina Jr., A. Traina, B. Seegar, and C. Faloutsos, "Slim-tree: High Performance Metric Trees Minimizing Overlap Between Nodes," Proc. of Int. Conf. on Extending Database Technology(EDBT), pp.51-65, 2000

[8] N. Beckmann, H. P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: An Efficient and Robust Access Method for Point and Rectangles," Proc. of Int. Conf. on ACM SIGMOD, pp.322-331, 1990.

[9] D. A. White and R. Jain, "Similarity Indexing with the SS-tree," Proc. 12th Int. Conf. on Data Engineering(ICDE), pp.516-523, 1996.

[10] N. G. Colossi and M.A Nascimento, "Benchmarking Access Structure for High-Dimensional Multimedia Data", TR-99-05, Univ. of Alberta, 1999.

[11] N. Katayama and S. Satoh, "The SR-tree: An Index Structure for High-Dimensional Nearest-Neighbor queries," Proc. of Int. Conf. on ACM SIGMOD, pp.28-29, 1996.

[12] S. Berchtold, D. A. Keim, and H. P. Kriegel, "The X-tree: An Index Structure for High-Dimensional Data," VLDB Journal, 3(4), pp.517-542, 1996.

[13] J. K. Uhlmann, "Satisfying General Proximity/Similarity Queries with Metric Trees," Information Processing Letters, 40(1), pp.175-179, 1991.

[14] T. Bozkaya and M. Ozsoyoglu, "Distance-Based Indexing for High-Dimensional Metric Spaces," Proc. of Int. Conf. on ACM SIGMOD, pp.357-368, 1997.

[15] P. Ciaccia, M. Patella, and P. Zezula, "M-tree: An Efficient Access Method for Similarity Search in Metric spaces," Proc. of Int. Conf. on Very Large Databases(VLDB), pp.426-435, 1997.

[16] E. Chavez, G. Navarro, R. Baeza-Yates, and J. Marroquin, "Searching in Metric Spaces," ACM Computing Surveys, 33(3), pp.273-321, 2001.

[17] J. Han, M. Kamber, Data Mining: *Concepts and Techniques*, Morgan Kaufmann Publishers, pp.349-354, 2001.

[18] J. M. Hellerstein, J. F. Naughton, and A. Pfeffer, "Generalized Search Trees for Database Systems," Proc. of Int. Conf. on Very Large Databases(VLDB), pp.562-573, 1995.

[19] 박만수, 박철의, 김희린, 강경옥, "Pitch 히스토그램을 이용한 내용기반 음악 정보 검색," 한국방송공학회, 방송공학회논문지, 제 9권, 1호, pp.2-8, 2004.

임 상 혁

e-mail : sanha97@hotmail.com
2002년 인하대학교 전자전기컴퓨터공학부 (공학사)
2004년 인하대학교 정보통신대학원 (공학석사)
2004년~현재 코난테크놀로지 연구원
관심분야 : 멀티미디어 데이타베이스, 정보 검색, 다차원 색인 기법

구 경 이

e-mail : kiku@etri.re.kr
1997년 인하대학교 전자계산공학과(공학사)
1999년 인하대학교 전자계산공학과 (공학석사)
2004년 인하대학교 컴퓨터 · 정보공학과 (공학박사)
2005년~현재 한국전자통신연구원 연구원
관심분야 : 음악 데이타베이스, 이동 데이타베이스, S/W 스트리밍

김 기 창

e-mail : kchang@inha.ac.kr
1984년 California State Polytechnic University at Pomona, 전산학(학사)
1988년 University of California at Irvine, 전산학(석사)
1992년 University of California at Irvine, 전산학(박사)
1992년~1994년 IBM T.J. Watson 연구실, 연구원
1994년~현재 인하대학교 정보통신공학부 교수
관심분야 : 컴퓨터 시스템보안, 실시간 운영체제, 병렬화 컴파일러

김 유 성

e-mail : yskim@inha.ac.kr
1986년 인하대학교 전자계산학과(이학사)
1988년 한국과학기술원 전산학과(공학석사)
1992년 한국과학기술원 전산학과(공학박사)
1996년~1997년 미국, 퍼듀대학교 전산학과 방문연구원
1992년~현재 인하대학교 정보통신공학부 교수
관심분야 : 멀티미디어 정보검색, 다차원 색인 기법, 바이오인포메틱스