RFID 태그의 추적을 위한 시간매개 변수간격 색인 기법

(A Time Parameterized Interval Index Scheme for RFID Tag Tracing)

반 재 훈 † 홍 봉 희 ††
(ChaeHoon Ban) (BongHee Hong)

요 약 RFID 시스템의 태그를 추적하기 위해서는 태그의 궤적을 모델링하고 색인으로 구성해야 한다. 궤적은 태그가 판독기의 인식영역으로 들어갈 때와 나갈 때 보고되는 두 개의 시공간 위치를 연결한 선분으로 표현될 수 있다. 만약 태그가 판독기의 인식영역에 들어와 나가지 않는 경우에 태그의 시공간 위치는 인식영역에 들어올 때만 보고된 점으로 표현된다. 따라서 판독기에 머물고 있는 태그는 궤적을 표현할 수가 없으므로 질의 시 이러한 태그를 검색할 수 없다. 이러한 문제를 해결하기 위하여 이 논문에서는 태그의 궤적을 시간 매개변수 간격으로 정의하고 새로운 색인인 TPIR-tree(Time Parameterized Interval R-tree)를 제안한다. 또한 효율적인 질의처리를 위한 새로운 삽입 및 분할 알고리즘을 제안하여 노드가 차지하는 영역을 최소화한다. 마지막으로 제안된 색인을 구현하여 다양한 데이타에서 기존 색인과 성능을 비교한다.

키워드 : RFID, 태그색인, 태그추적, 간격

Abstract For tracing tag locations, the trajectories should be modeled and indexed in radio frequency identification (RFID) systems. The trajectory of a tag can be represented as a line that connects two spatiotemporal locations captured when the tag enters and leaves the vicinity of a reader. If a tag enters but does not leave a reader, its trajectory is represented only as a point captured at entry. Because the information that a tag stays in a reader is missing from the trajectory represented only as a point, it is impossible to find the tag that remains in a reader. To solve this problem we propose the data model in which trajectories are defined as time-parameterized intervals and new index scheme called the Time Parameterized Interval R-tree. We also propose new insert and split algorithms that reduce the area of nodes to enable efficient query processing. We evaluate the performance of the proposed index scheme and compare it with previous indexes on various datasets.

Key words : RFID, tag indexing, tag trace, interval

1. 서 론

RFID(Radio Frequency IDentification)는 무선 주파수를 이용하여 태그(tag)를 장착한 객체를 판독기(reader)가 자동으로 인식하고 확인하는 기술로서 태그를 장착한 객체의 위치 추적이나 객체의 현재의 상태를 감시하는 자동화 생산(automated manufacturing), 재고관리(inventory tracking), 공급망 관리(supply chain manage-ment) 등과 같은 다양한 응용분야에서 사용된다[1].

태그는 이동체(moving object)와 유사하게 시간에 연속적으로 이동하므로 태그의 궤적을 추적하기 위해서 이동체를 위한 시공간 색인을 적용할 수 있다. 즉, 이동체는 이동하면서 일정한 시간에 위치를 보고하므로 보고된 두 개의 시공간 위치를 연결하는 선분으로 궤적을 표현하고 색인을 구성할 수 있다[2-5]. 마찬가지로 태그가 판독기에 들어올 때와 나갈 때 보고되는 시공간 위치를 이용하여 태그의 궤적을 선분으로 표현하고 색인을 구성할 수 있다.

그러나 이러한 궤적의 모델링 기법을 사용하는 경우

· 이 논문은 교육인적자원부 지방연구중심대학육성사업(차세대물류IT기술연구사업단)의 지원에 의하여 연구되었음
† 정 회 원 : 경남정보대학 인터넷응용계열 교수
chban@kit.ac.kr
†† 종신회원 : 부산대학교 컴퓨터공학과 교수
bhhong@pusan.ac.kr
논문접수 : 2005년 8월 3일
심사완료 : 2005년 10월 27일

에 판독기의 인식영역에 들어와 머무는 태그를 찾을 수 없는 문제가 발생한다. 태그의 궤적은 태그가 판독기에 들어올 때와 나갈 때 보고하는 두 개의 시공간 위치를 연결한 선분으로 표현된다. 만약 태그가 판독기에 머무는 경우에 궤적은 태그가 판독기에 들어갈 때 보고하는 시공간 점으로만 구성된다. 따라서 태그가 판독기에 머문다는 정보를 시공간 점으로는 표현할 수 없으므로 판독기에 머무는 태그를 찾을 수 없는 문제가 발생한다. 따라서 판독기에 머무는 태그를 표현할 수 있는 새로운 방법이 제시되어야 한다.

이 논문에서는 위와 같은 문제를 해결하기 위하여 RFID 태그의 궤적을 위한 시간 매개변수(time parameterized) 간격 데이타 모델을 정의한다. 이 모델에서는 태그의 궤적을 시간에 종속적인 함수로 표현되는 시간 매개변수 간격으로 표현하여 판독기에 머무는 태그를 찾을 수 있게 한다. 또한 시간 매개변수 간격 데이타 모델에 적합한 R-tree 기반 색인 구조인 TPIR-tree (Time Parameterized Interval R-tree)를 제시하며 효율적인 질의처리를 위해 새로운 삽입 및 분할 알고리즘을 제안한다. 마지막으로 제안된 색인과 기존 알고리즘을 사용하는 색인과의 성능비교를 통하여 색인의 우수성을 입증한다.

이 논문에서 제시하는 태그의 궤적을 시간 매개변수 간격 데이타로 모델링하여 TPIR-tree를 구성하는 경우에 다음과 같은 장점이 있다. 첫째, 판독기에 머무는 태그를 검색하는 질의 처리를 수행할 수 있다. 둘째, 시간 매개변수 간격은 시간에 종속적이므로 이러한 특징을 고려한 삽입, 분할 알고리즘을 사용하면 색인을 구성하는 노드의 면적과 겹침이 줄어들어 질의의 수행 속도를 향상 시킬 수 있다. 마지막으로 태그를 장착한 객체의 위치 추적이나 객체의 현재의 상태를 감시하는 자동화 생산(automated manufacturing), 재고 관리(inventory tracking), 공급망 관리(supply chain manage-ment) 등과 같은 유비쿼터스 응용분야에서 태그의 과거 이력 정보인 궤적을 탐색할 수 있다.

이 논문의 구성은 다음과 같다. 2장에서는 관련연구를 기술하며 3장에서는 대상환경 및 태그의 궤적 표현으로 인해 발생되는 문제를 정의한다. 4장에서는 RFID 시스템에서 태그의 위치추적을 위해 사용되는 질의를 분류하며 시간 매개변수 간격을 정의한다. 5장에서는 시간 매개변수 간격을 이용한 TPIR-tree의 데이타 구조를 제시하며 질의 수행 방법과 새로운 삽입, 분할 알고리즘을 제시한다. 6장에서는 제안된 TPIR-tree를 구현하고 기존 알고리즘을 사용하는 색인과의 성능 비교를 통하여 그 우수성을 입증한다. 마지막으로 7장에서는 결론 및 향후 연구를 기술한다.

2. 관련 연구

2.1 간격 데이타를 위한 색인 구조에 관한 연구

간격(interval)은 [l, r](l≤r)인 형태를 가지며 어떤 속성의 기간(duration)을 나타내는 순서쌍이다. 간격을 표현하는 대표적인 색인은 R-tree 계열이 있다. R-tree[6]는 데이타 분할 방법의 대표적인 공간 색인으로서 공간 객체를 최소 경계 사각형(Minimum Bounding Rectangle)으로 표현하며 단말 노드에 모든 데이타를 저장한다. 색인에 데이타를 삽입하기 위한 ChooseSubtree알고리즘은 최소 영역 확장 정책(Least Area Enlargement Policy)을 사용하여 삽입할 노드를 선택한다. 또한 노드 오버플로우 시에 분할되는 2개의 노드가 최소 영역을 갖도록 분할한다.

R*-tree[7]는 R-tree가 삽입 시에 영역만을 고려하는 단점을 보완하기 위하여 겹침(overlap)과 가장자리(margin)를 추가적으로 고려한 색인 구조이다. R*-tree는 삽입을 위해서 단말 노드를 선택할 시에, 최소 겹침 확장 정책(Least Overlap Enlargement Policy)을 사용한다. 그리고 오버플로우가 발생한 노드의 분할은 노드의 가장자리가 최소가 되는 분할 축을 선택하며, 형제 노드와의 겹침 값이 최소가 되도록 엔트리를 분배한다. 또한 재삽입 정책을 사용하여 R-tree의 공간 활용도가 낮은 문제점을 보완하였다.

SR-tree[8]는 R-tree[6]와 동일한 구조를 가지며 기존의 삽입 정책과 분할 정책을 그대로 사용한다. 차이점은 자식 노드를 완전히 걸치는 긴 간격 데이타가 삽입될 경우, 데이타를 단말 노드에 저장하지 않고 걸쳐지는 노드의 부모 노드에 저장한다는 점이다. SR-tree는 비단말 노드에도 데이타를 저장하므로, 노드의 크기가 루트 노드로 갈수록 커지는 단점이 있다. 그리고 아주 긴 간격 데이타가 삽입될 경우, 선분을 잘라서 다시 삽입해야 하고, 삽입과 분할로 인하여 노드의 크기가 변할 경우 비단말 노드에 저장되어 있는 간격 데이타를 다른 노드로 이동해야 한다. 태그 객체의 매우 긴 성장 간격은 비단말 노드와 단말 노드에 절단 되어서 저장되기 때문에 SR-tree는 태그 객체를 위한 색인으로 적합하지 않다.

2.2 이동체의 색인 구조에 관한 연구

이동체는 시간에 따라 위치가 변화하기 때문에 시공간 객체로 볼 수 있으며, 시공간 데이타를 저장하는 색인 중에서 3DR-tree[2], 2+3R-tree[3], HR-tree[4]를 이동체를 위한 색인으로 사용할 수 있다. 또한 이동체의 복합질의를 위한 색인으로는 STR-tree[5]와 TB-tree [5] 등이 있다.

기존 2차원 공간 데이타를 저장하는 R-tree에 기반한

3DR-tree[2]는 시간을 또 다른 하나의 축으로 고려한다. 이러한 원리를 이용하여, 객체가 시간 $[t_i,t_j)$ 동안 위치 (x_i,y_i)에 있고, 시간 $[t_j,t_k)$ 동안 위치 (x_j,y_j)에 있다면, 3차원 시공간에서 선분 $\overline{[(x_i,y_i,t_i),(x_i,y_i,t_j))}$와 선분 $\overline{[(x_j,y_j,t_j),(x_j,y_j,t_k))}$으로 표현할 수 있으며, 3차원 R-Tree를 사용하여 색인 할 수 있다. 3DR-tree는 두 끝점이 알려져 있는 선분만 저장할 수 있기 때문에 현재 위치를 저장할 수 없다. 특정 시간과 공간 영역이 주어지는 영역 질의에 높은 성능을 보이지만, 노드 분할 시에 시간 도메인에 대한 고려가 없기 때문에 공간 활용도 크게 떨어지는 단점이 있다.

2+3 R-tree[3]는 3D R-tree[2]의 문제점을 2차원 점과 3차원 선분에 대해 각각 따로 저장하는 두 개의 다른 R-tree를 사용함으로써 해결한다. 2차원 점은 현재 시점에서 객체의 공간 상의 점을 표현하며, 3차원 선분은 객체의 과거 이력을 표현한다. 2+3 R-tree에서 객체의 위치에 대한 끝 시간을 아직 모른다면, 그 데이타는 2차원 R-tree에서 객체의 위치에 대한 시작 시간과 oid를 유지하면서 저장된다. 열린(opened) 객체의 현재 상태에서 끝 시간을 알게 될 경우, 2차원 R-tree에서 해당 엔트리를 삭제하고, 3차원 시공간 상의 선분을 구성하여 3차원 R-tree에 삽입한다. 2+3 R-tree는 두 개의 R-tree에서 질의를 수행해야 하는 문제점이 있다.

HR-tree[4]는 R-tree에 트랜잭션 시간 개념을 추가하여 객체의 이력 정보를 표현하며, 타임스탬프마다 다른 색인 인스턴스를 구성한다. 기본 원리는 원래의 트리는 유지하고 상태가 변한 루트와 가지를 대체함으로써 현재와 과거를 유지한다. 상태가 변화하지 않은 가지들은 복제되지 않는다. HR-tree는 이동체의 이동이 빈번하지 않은 경우에는 효율적이지만, 이동이 많이 발생하는 경우에는 단말 노드와 비단말 노드를 새로 생성해야 하는 문제점을 가진다.

STR-tree[5]는 이동체의 궤적을 선분의 집합으로 나타내고, 보존 파라미터를 사용하여 같은 객체의 궤적을 근접한 페이지에 저장하도록 유도한다. STR-tree의 부분적인 궤적 보존 정책은 3DR-tree에 비해 궤적 질의의 성능 향상 효과가 낮을 뿐만 아니라, 삽입 정책에서 공간 근접성과 같은 공간 구별이 낮아지기 때문에 영역 질의의 성능이 좋지 않다.

TB-tree[5]는 궤적 질의 및 복합 질의의 성능 향상을 위하여 극단적인 궤적 보존 정책을 사용한다. 하나의 단말 노드는 오직 동일한 궤적에 속하는 선분들만 저장할 수 있으며, 양방향 연결 리스트를 구성하여 궤적에 대한 질의 성능을 향상 시킬 수 있다. 그러나 특정 객체의 이동 궤적이 단절 되어 있는 경우 새로운 단말 노드를 생성하여 저장하기 때문에 RFID 태그 객체를 위한 색인 구조로는 적합하지 않다.

3. 대상 환경 및 문제정의

이 장에서는 RFID 시스템과 태그의 이동으로 인해 발생되는 이벤트를 소개한다. 또한 태그의 궤적을 정의하고 기존 연구에서 궤적을 표현하는 방법을 사용하는 경우에 발생되는 문제점을 제시한다.

3.1 대상환경

RFID 시스템은 태그와 판독기 그리고 호스트서버로 구성된다. 태그는 컨테이너나 팔레트와 같은 실 개체에 부착되어 판독기 사이를 이동한다. 판독기는 전략적으로 중요한 위치에 고정되며 태그를 인식할 수 있는 인식영역에 들어오거나 나가는 태그의 정보를 수집한다. 태그가 인식영역에 들어오면 *Enter* 이벤트가 발생하며 이 정보를 서버에 전송한다. 반대로 태그가 인식영역을 빠져나가면 *Leave* 이벤트가 발생하며 이 정보를 서버에 전송한다[1].

RFID 시스템의 태그는 이동체와 유사한 특징을 가진다. 즉, 이동체가 이동하면서 시간에 연속적으로 자신의 위치를 변경하듯이 태그도 판독기의 인식범위에 들어가거나 빠져나가며 자신의 위치를 변경한다. 따라서 태그의 위치 추적을 위해 이동체의 궤적 모델링 기법을 적용할 수 있다.

이동체에 대한 기존 연구에서는 궤적을 두 개의 시공간 위치를 연결한 선분들로 표현한다. 이동체가 t_i의 시간에 (x_i,y_i)의 위치를 보고하고 t_{i+1} 시간에 (x_{i+1},y_{i+1})의 위치를 보고하는 경우에 궤적을 구성하는 단일궤적은 두 개의 시공간 위치를 연결하는 선분인 $\overline{(x_i,y_i,t_i),(x_{i+1},y_{i+1},t_{i+1})}$로서 3차원 시공간 좌표에 표현된다[2,3,5].

마찬가지로 태그의 단일궤적은 태그가 판독기를 들어갈 때와 빠져나갈 때 획득한 시공간 위치를 연결한 선분으로 표현될 수 있다. 차이점은 이동체가 자신의 실제 위치를 보고하는 반면에 태그 경우에는 태그를 인식한 판독기가 자신의 위치를 태그의 위치로 보고한다는 것이다. 이 논문에서는 판독기 r에서의 태그의 단일궤적을 다음과 같이 정의한다. 정의에서 x, y, t는 3차원 공간의 각 축을 의미하며 x_r, y_r은 판독기의 위치, t_{enter}, t_{leave}는 각각 *Enter*와 *Leave* 이벤트의 발생 시간을 의미한다.

정의 1. 판독기 r에서의 태그의 단일궤적 $tr = \{(x,y,t)\in R^3 \mid x = x_r,\ y = y_r,\ t_{enter} \le t \le t_{leave}\}$

예를 들어 그림 1과 같이 *Enter* 이벤트가 t_{enter}시간에 발생하고 *Leave* 이벤트가 t_{leave} 시간에 발생했다고 가정하자. 이 경우에 판독기 r의 위치를 $(x_r,\ y_r)$이라하면 *Enter* 이벤트에는 $(x_r,\ y_r,\ t_{enter})$의 시공간 위치가 보고

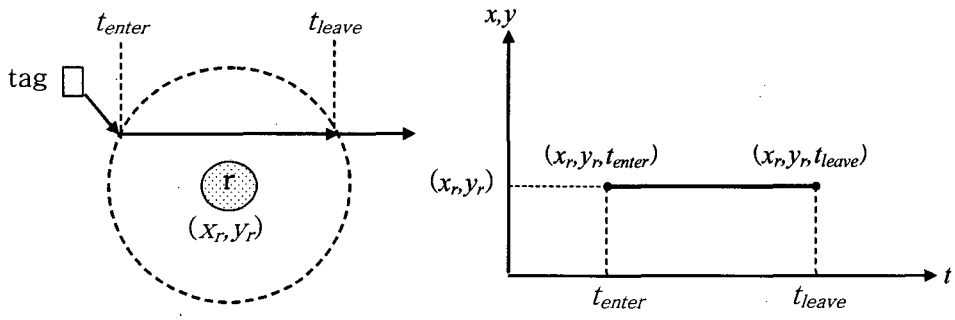


그림 1 태그의 궤적

되며 *Leave* 이벤트에는 (x_r, y_r, t_{leave})의 시공간 위치가 보고된다. 따라서 태그의 단일궤적은 이 두 개의 시공간 위치를 연결한 선분이 된다.

3.2 문제정의

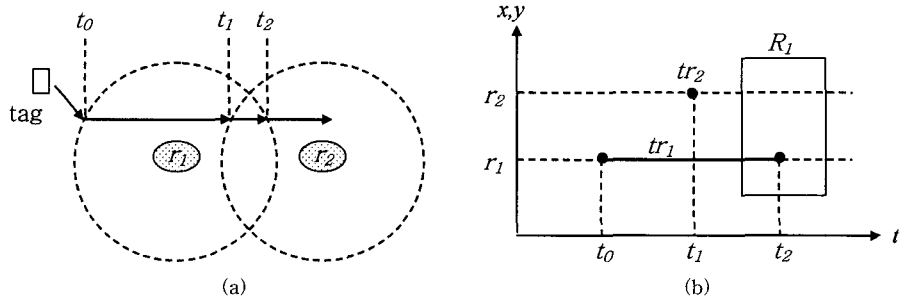
태그를 위해 기존 연구에서 사용한 궤적의 표현 방법을 사용하면 판독기에 머무는 객체를 찾을 수 없는 문제가 발생한다. 만약 태그가 판독기의 인식범위에 들어와 나가지 않고 머무는 경우에는 질의 수행 시에 질의의 결과임에도 불구하고 질의의 후보가 될 수 없다. t_{now}를 현재시간이라고 가정하고 t_{enter}와 t_{leave}를 *Enter* 이벤트와 *Leave* 이벤트의 발생시간이라 가정하자. 만약 $t_{enter} \leq t_{now} < t_{leave}$이면 *Enter* 이벤트만 발생하고 *Leave* 이벤트는 발생하지 않게 된다. 따라서 궤적은 *Enter* 이벤트 시에 보고된 시공간 위치인 점으로만 표현되므로 판독기에 머물러 있다는 정보를 표현할 수 없다. 따라서 질의 수행 시 이러한 태그를 찾을 수 없는 문제가 발생한다.

예를 들어 그림 2(a)와 같이 판독기 r_1에 태그가 시간 t_0에 들어와 t_2에 빠져나간다고 가정하자. 정의1에 따라 궤적은 그림 2(b)의 tr_1과 같이 두 개의 시공간 위치를 연결한 선분으로 표현된다. 만약 태그가 판독기 r_2에 시간 t_1에 들어와 빠져나가지 않으면 태그의 *Leave* 이벤트는 발생하지 않게 된다. 따라서 궤적은 tr_2와 같이 *Enter* 이벤트 발생시에 보고된 시공간 위치인 점으로 표현되며 이것이 색인에 삽입되게 된다. 그림 2(b)의 R_1과 같이 특정 시간에 판독기 r_1과 r_2에 위치하는 태그를 찾는 질의를 수행한다고 가정하자. 질의가 수행되면 R_1에 포함되는 궤적을 찾는데, tr_1은 포함되나 tr_2는 포함되지 않으므로 태그가 판독기 r_2에 머물러 있음에도 불구하고 그 정보가 표현되지 않으므로 tr_2는 질의의 결과에서 제외된다. 따라서 판독기에 들어와 *Enter* 이벤트만 발생되고 머무는 태그를 표현할 수 있는 방법이 필요하다.

이 논문에서는 이러한 문제를 해결하기 위하여 태그의 궤적을 시간 매개변수 간격으로 정의한다. 시간 매개변수 간격은 시간에 따라 시간 축 길이가 변하는 선분이다. 따라서, *Enter* 이벤트만 발생한 태그의 궤적은 시간에 종속적인 선분으로 표현되므로 위와 같은 질의를 처리할 수 있다.

4. 태그의 궤적 표현을 위한 시간 매개변수 간격 데이타 모델

이 장에서는 태그를 위한 질의를 정의하고 판독기에 머무는 태그를 표현하기 위한 시간 매개변수 간격을 정의한다. 또한 질의의 효율적인 수행을 위해 태그 식별자를 색인의 차원으로 추가할 때 발생되는 문제점 및 해결책을 제시한다.



(a) (b)

그림 2 판독기에 머무는 태그의 문제발생

4.1 태그를 위한 질의 정의

RFID 시스템에서 태그의 위치를 추적하기 위해서는 다양한 질의가 필요하다. 질의는 특정 시공간 지역에 포함되는 궤적을 추출하는데 다음과 같이 네 종류로 분류된다. 다음에서 질의의 분류를 위하여 $[a^{\vdash}, a^{\dashv}]$의 표현을 사용하는데 이 의미는 좌표 축에 투영한 값의 범위를 나타낸다.

Type 1

FIND 질의: Q = (*tid*, $[t^{\vdash}, t^{\dashv}]$) - 시간 $[t^{\vdash}, t^{\dashv}]$에 태그 식별자 *tid*가 이동한 판독기의 위치 반환.

예: *"18:00~23:00까지 #13 태그가 위치한 판독기는?"*

Type 2

LOOK 질의: Q = ($[x^{\vdash}, x^{\dashv}]$, $[y^{\vdash}, y^{\dashv}]$, $[t^{\vdash}, t^{\dashv}]$)- 시간 $[t^{\vdash}, t^{\dashv}]$에 특정 위치 $[x^{\vdash}, x^{\dashv}]$, $[y^{\vdash}, y^{\dashv}]$에 위치한 판독기를 지나간 태그의 식별자 반환.

예: *"18:00~23:00까지 판독기 #1-3, #1-4에 위치한 태그는?"*

Type 3

HISTORY 질의: Q = (*tid*) - 태그 식별자 *tid*가 지나간 모든 판독기의 위치 반환.

예: *"태그 #13이 거쳐간 모든 판독기의 위치를 찾아라."*

Type 4

WITH 질의: Q = (*tid*, $[t^{\vdash}, t^{\dashv}]$) - 시간 $[t^{\vdash}, t^{\dashv}]$에 태그 식별자 *tid*와 같이 이동한 모든 태그들의 식별자 반환.

예: *"18:00~23:00까지 #13 태그와 같이 있었던 모든 태그를 찾아라."*

질의에서 주목할 만한 것은 HISTORY질의는 시간 범위가 시간 축 전체인 FIND 질의이며 WITH질의는 FIND질의를 수행하고 난 뒤에 LOOK질의를 수행하여 처리할 수 있다. 따라서 RFID 시스템에서는 FIND 질의와 LOOK 질의가 기본 질의이다.

4.2 시간 매개변수 간격

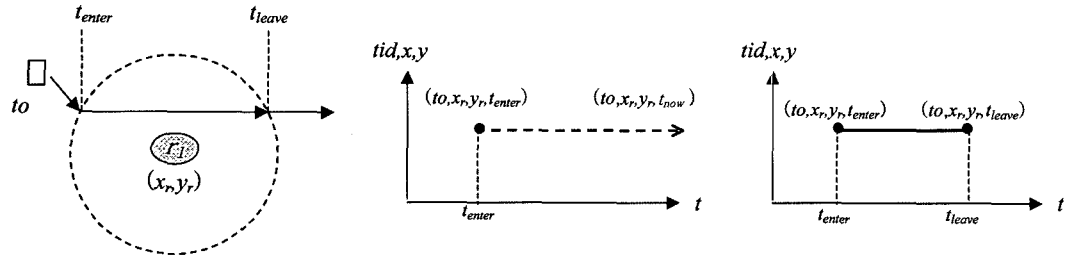
판독기에 머무는 태그를 표현하기 위하여 [11]에서는 판독기에 들어간 태그의 궤적을 시간 도메인의 최대값을 시간 구간의 끝 값으로 가지는 긴 선분으로 표현하였다. 그러나 시간 도메인의 최대값은 응용에 따라 틀리므로 그 값이 모호하며 현재시간 이후의 미래 질의에 대하여 질의 처리시 항상 특정 판독기에 머문다는 잘못된 질의 결과를 반환하게 된다.

이 논문에서는 판독기에 머무는 태그를 표현하기 위하여 태그의 궤적을 시간 매개변수 간격으로 정의한다. 시간 매개변수 간격은 태그가 판독기의 인식범위에 들어오는 경우에 생성된다. 시간이 변경되면서 시간 매개변수 간격도 자신의 시간 도메인의 값을 변경한다. 다음의 시간 매개변수 간격의 정의를 위하여 이 논문에서는 태그 식별자 *tid*, 공간 좌표 *x*, *y*, 및 시간 *t*를 4차원의 각 축으로, *to*는 태그의 식별자, x_r, y_r은 태그의 공간 위치 t_{enter} 를 *Enter* 이벤트의 발생시간, $f(t)$를 시간 함수로 가정한다. 판독기 r에서의 시간 매개변수 간격의 정의는 다음과 같다.

정의 2. 시간 매개변수 간격 = $\{(tid,x,y,t)\in R^4 \mid tid = to,\ x = x_r,\ y = y_r,\ t_{enter} \le t \le f(t)\}$.

태그가 판독기에 들어오면 시간 매개변수 간격은 생성되며 $f(t)$는 현재시간을 나타내는 t_{now}로 변경된다. 따라서 시간 매개변수 간격의 시간 축 길이는 $t_{enter} \le t \le t_{now}$가 되며 t_{now}에 의해 시간 축 길이가 계속해서 변하게 된다. 판독기에 들어와 아직 빠져나가지 않은 태그의 궤적이 시간에 종속적인 선분으로 표현되므로 질의발생 시간을 t_{query}라 하면 $t_{enter} \le t_{query} < t_{leave}$에 발생되더라도 질의를 처리할 수 있다. 만약 태그가 판독기를 빠져나가게 되면 $f(t)$는 t_{leave}로 변경되며 시간 매개변수의 시간 축 길이는 $t_{enter} \le t \le t_{leave}$로 변경되고 $t_{query} \ge t_{leave}$에 발생하는 질의를 처리할 수 있다.

예를 들어, 그림 3(a)와 같이 태그가 판독기에 t_{enter}시간에 들어와 t_{leave}시간에 빠져 나간다고 가정하자. $t_{enter} \le t_{query} < t_{leave}$인 경우에 태그의 궤적은 그림 3(b)와 같이 태그가 아직 판독기를 빠져 나오지 않았으므로 시간 축 범위가 $[t_{enter}, t_{now}]$인 시간 매개변수 간격이 되어 질



(a) (b) $t_{enter} \le t_{now} < t_{leave}$ (c) $t_{now} \ge t_{leave}$

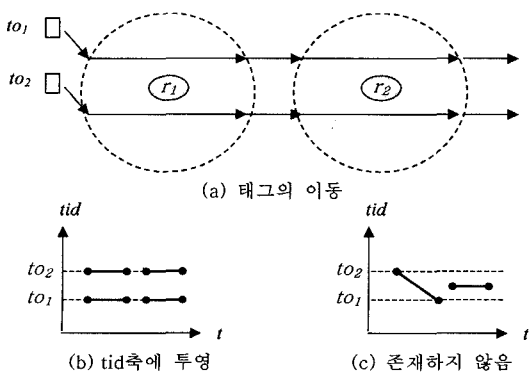
그림 3 시간 매개변수 간격의 예

의 처리가 가능하다. $t_{query} \geq t_{leave}$인 경우에는 태그가 그림 3(c)와 같이 판독기를 빠져 나왔으므로 궤적은 시간 축 범위가 $[t_{enter}, t_{leave}]$인 시간 매개변수 간격이 되어 질의 처리가 가능하다. 따라서 위와 같이 태그의 궤적을 시간 매개변수 간격으로 정의하면 언제나 질의처리가 가능하다.

4.3 태그식별자의 차원 추가

정의 1에서 태그의 궤적은 3차원 시공간 좌표에 정의하였다. 즉, 2차원 공간에 시간 차원을 추가하여 3차원 공간에 표현하였다. 정의 1과 다르게 시간 매개변수 간격은 4차원 공간에 정의된다. 즉, 3차원 시공간에 태그의 객체 식별자 차원을 추가한 4차원 시공간에 표현한다. 그 이유는 앞에서 정의된 RFID 시스템의 질의 중에서 LOOK, HISTORY, WITH 질의가 태그의 식별자를 매개변수로 하기 때문이다. 만약 태그의 식별자를 차원에 추가하지 않으면 질의 처리를 위해 구축된 색인을 모두 검색해야 하므로 효율적인 질의 처리를 위해 태그의 식별자를 차원으로 추가한다.

x, y, t 축이 연속적인(continuous) 데이타를 나타내는 축인 것과 달리, 태그의 식별자는 연속적인 데이타를 나타내는 축이 아니다. 즉, 태그의 식별자 사이의 데이타는 의미를 가지지 않으며 존재할 수 없다. 그러나 태그의 시간 매개변수 간격은 항상 시간 축 t에 평행하므로 태그의 식별자가 비연속적이더라도 표현에는 문제가 발생하지 않는다. 예를 들어 그림 4(a)와 같이 태그가 판독기를 이동한다고 가정하자. 이동으로 인해 발생되는 시간 매개변수 간격을 tid 축에 투영시키게 되면 모든 궤적은 시간 축에 평행하게 생성된다. 따라서 그림 4(b)와 같이 태그의 식별자 값을 가지는 선분이 시간 축에 평행하며 그림 4(c)와 같이 각 식별자 사이를 걸치는 선분이 존재하지 않으므로 태그 식별자가 비연속적이라도 시간 매개변수 간격을 표현하는데 문제가 발생하지 않는다.

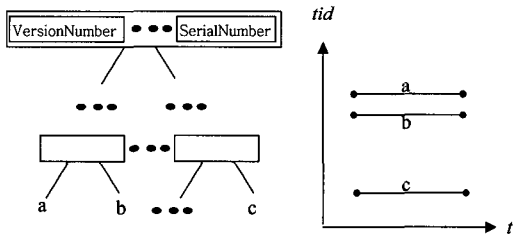


(a) 태그의 이동

(b) tid축에 투영 (c) 존재하지 않음

그림 4 태그 식별자의 비연속성

x, y, t 축에 존재하는 데이타들간에는 관련성을 가진다. 예를 들면 시간 축의 경우 시간적으로 가까운 데이타들끼리 축에서도 가깝게 위치한다. 그러나 태그 식별자의 경우 축에서 서로 가깝게 위치한다고 해서 식별자끼리 관련성이 크다고는 할 수 없다. 그러나 식별자를 계층적으로 구성하면 관련성을 부여할 수 있다. 예를 들어, EPC(Electronic Product Code)의 경우 계층적으로 구성하여 같은 품목끼리는 EPC를 그룹핑하여 표현한다[9]. 따라서 이러한 방식으로 태그의 식별자를 구성하면 식별자 축에서 가까운 것들끼리는 같은 품목일 관련성이 높아진다. 따라서 태그 식별자 tid축도 위치하는 데이타 간에 관련성을 가진다고 할 수 있다. 예를 들어 그림 5에서와 같이 유사한 품목의 a, b 객체를 EPC 로 구성하게 되면 식별자 또한 유사하게 되고, 이를 식별자 축에 대응시키면 가깝게 위치한다. 따라서 이와 같이 식별자를 계층적으로 구성하면 데이타간의 관련성을 표현할 수 있다.



(a) EPC의 구성 (b) tid 축에서의 관련성

그림 5 태그 식별자의 관련성

5. TPIR-tree(Time Parameterized Interval R-tree)

이 장에서는 태그의 궤적을 위한 R-tree 기반의 TPIR-tree를 소개한다. 먼저 4장에서 제안된 시간 매개변수 간격을 이용하여 색인에서 사용되는 데이타구조와 탐색 알고리즘을 제시한다. 또한 질의의 효율적인 처리를 위해 시간 매개변수 간격의 특성을 고려한 새로운 삽입과 분할 알고리즘을 제시한다.

5.1 데이타 구조

TPIR-tree의 단말노드는 $<MBB>$의 형태의 시간 매개변수 간격을 엔트리로 포함한다. MBB는 4차원 최소 경계박스로서 기존 R-tree 계열의 색인에서는 $<[tid^{\vdash}, tid^{\dashv}],[x^{\vdash},x^{\dashv}],[y^{\vdash},y^{\dashv}],[t^{\vdash},t^{\dashv}]>$의 형태를 갖는다. 그러나 TPIR-tree에서는 시간 구간 $[t^{\vdash}, t^{\dashv}]$에서 식별자와 공간 위치가 변하지 않으므로 $<tid, x, y, [t^{\vdash},t^{\dashv}]>$의 형태를 가진다. 만약 태그가 판독기에 들어와 머무는 경우

에는 시간 매개변수 간격은 <*tid*, *x*, *y*, t_{enter}, *now*>의 형태로 단말노드에 삽입되며 현재간격(now interval)이라고 하며 *nowI*로 표기한다. 만약 태그가 판독기에 들어와 나가는 경우에는 시간 매개변수 간격은 <*tid*, *x*, *y*, t_{enter}, t_{leave}>의 형태로 단말노드에 삽입되며 고정간격(fixed interval)이라고 하며 *fixedI*라 표기한다.

비단말노드는 <*child-pointer*, *state*, *MBB*>의 형태를 갖는 엔트리를 포함하며 *child-pointer*는 자식노드를 가리키는 포인터, *state* 는 엔트리의 상태를 나타내며 , *MBB* 는 $<[tid^{\vdash},tid^{\dashv}],[x^{\vdash},x^{\dashv}],[y^{\vdash},y^{\dashv}],[t^{\vdash},t^{\dashv}]>$ 형태의 4차원 최소경계박스를 나타낸다. 엔트리의 상태 *state*는 *fixedEntry*와 *nowEntry*의 두 가지로 분류되는데, 자식노드가 포함하는 엔트리가 모두 *fixedEntry*이거나 *fixedI*인 경우에는 *state*의 값으로 *fixedEntry*를 가지며 이 경우에 *MBB*는 기존 방법과 동일하게 자식노드의 모든 엔트리들을 포함하는 두 개의 4차원 시공간 점으로 구성된다.

자식노드가 포함하는 엔트리가 *nowEntry*나 *nowI*인 경우에 *state*는 *nowEntry*가 된다. 이 경우 최고깊이 -1 레벨에서의 *MBB*는 *nowI*를 하나 이상 포함하게 되는데 시간 구간이 정해지지 않았으므로 시작점만을 포함하며 *fixedI*의 경우에는 선분 전체를 포함하는 영역이 된다. 그 이상의 레벨에서의 *MBB*는 *nowEntry*와 *fixedEntry*를 포함하는 영역이 된다. 이러한 *MBB*는 *nowEntry*나 *nowI*를 포함하므로 시간에 종속적으로 증가하며 질의처리 시에 노드를 접근시간까지 확장해야 한다. 따라서 영역의 면적에 기반을 둔 기존의 삽입, 분할 정책을 사용하면 할 수 없게 만든다. 이것은 다음 절에서 설명한다.

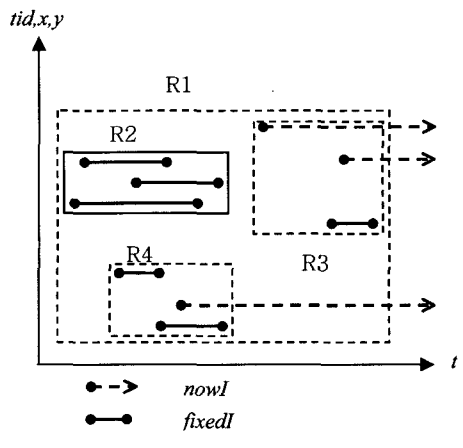
예를 들어 그림 6(a)와 같이 4차원 공간에 *fixedI*와 *nowI*가 존재한다고 가정하자. 단말노드 R2는 그림 6(a)와 같이 세 개의 *fixedI*를 포함하는 *MBB*를 가지며 그림 6(b)와 같이 부모노드 R1에서 R2를 가리키는 엔트리의 *state*는 *fixedEntry*가 된다. R3는 하나의 *fixedI*와 두 개의 *nowI*를 포함하므로 *nowEntry*가 되며 이것의 *MBB*는 *fixedI*와 각 *nowI*의 시작점만을 포함하게 된다. 이와 같은 방식으로 R4와 R1은 모두 *nowEntry*가 된다.

5.2 탐색

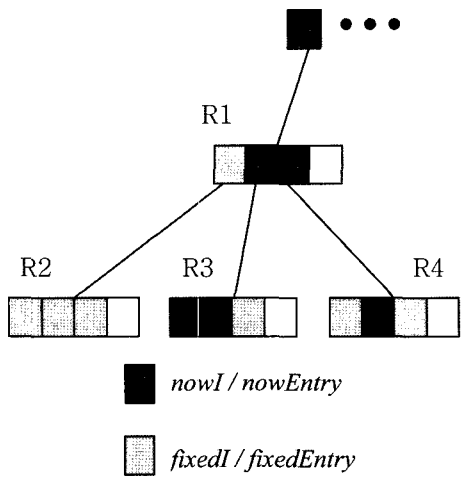
탐색은 기존의 R-tree와 유사하게 색인의 루트에서부터 하위방향으로 탐색해 나간다. 그러나 TPIR-tree에서는 노드의 타입에 따라 탐색 방법이 달라지게 된다. 단말노드가 발견될 때까지 질의영역과 교차하는 *MBB*를 가진 *fixedEntry*를 검사한다. *nowEntry*의 경우에는 먼저 *MBB*의 시간 축을 *now*까지 늘이고 나서 질의영역과 교차하는지를 검사한다. 그 이유는 *MBB*가 포함하는 *nowI* 또는 *nowEntry*의 시작점으로만 구성되었기

```
TPIR_Search( root, q )
TPIR_S1 Set N to be the root
TPIR_S2 If N is a leaf
        for each entry e
            if e = fixedI AND e∩q then return e
            else if e = nowI AND e_extend ∩q then return e
    else
        for each entry e
            if e = fixedEntry AND e∩q then TPIR_Search( e, q )
            else if e = nowEntry AND e_extend ∩q then TPIR_Search( e, q )
    endif
```

그림 7 탐색 알고리즘



(a) 4차원 공간의 시간매개변수 간격



(b) TPIR-tree의 구조

그림 6 TPIR-tree의 예

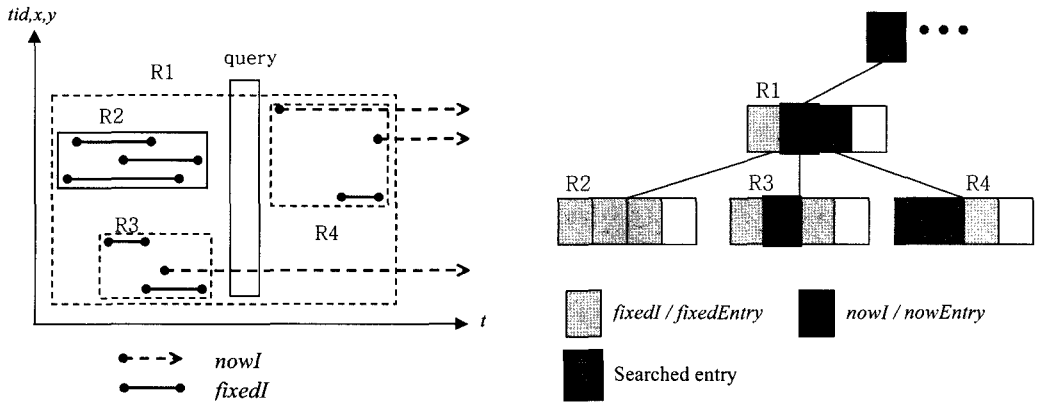


그림 8 탐색 알고리즘의 예

때문이다. 단말노드가 발견되며 포함된 *fixedI*와 *nowI*를 검사하여 질의의 결과로 반환하게 된다. 다음은 탐색 알고리즘이다.

그림 8과 같이 TPIR-tree가 구성되었다고 가정하자. 노드 R1은 질의영역과 교차하므로 하위노드로 탐색을 수행하게 된다. R3는 *MBB*를 확장한 후에 질의영역과 교차하므로 하위노드로 탐색하게 된다. 최종적으로 R3에 포함되는 세 개의 엔트리를 검사하게 되며, 하나의 *nowI*가 확장되어 질의영역과 교차하게 되므로 질의의 결과로 반환되게 된다.

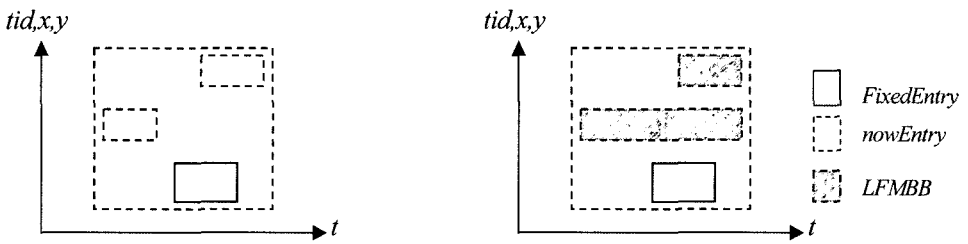
5.3 삽입

이 논문에서는 태그를 위한 질의처리를 효율적으로 수행하기 위하여 기존의 R-tree 계열[6,7]에서 적용하는 삽입방법을 개선한 알고리즘을 제시한다. R-tree에서의 삽입은 최소영역확장(least area enlargement) 정책을 이용하여 수행하게 된다. 그러나 TPIR-tree에 이러한 정책을 사용하게 되면 *nowEntry*가 질의수행시 확장되는 특성을 고려하지 않았기 때문에 겹침이 증가되어 비효율적이다. 따라서 TPIR-tree의 *nowEntry*의 특성을 고려하기 위해 먼저 *Local Fixed MBB(LFMBB)*를 아래와 같이 정의한다.

정의 3. *nowEntry*의 *LFMBB*는 부모노드 *MBB*의 최대 시간 값으로 시간 간격을 늘인 *MBB*

*LFMBB*의 예는 그림 9와 같다. 그림 9(a)와 같이 비단말노드에 포함된 모든 엔트리들의 *MBB* 중에서 *nowEntry*들의 *MBB*를 그림 9(b)와 같이 부모노드 *MBB*의 시간 축 최대값으로 늘인 것을 *LFMBB*라고 한다. 이것은 *nowEntry*가 확장되는 성질을 일정으로 보장해 주므로 삽입 수행시 이러한 *LFMBB*를 사용하여 최소영역확장 방법을 사용한다.

표 1은 삽입되는 간격의 타입에 따라 삽입시 선택되는 엔트리들의 영역확장 계산 방법을 제시하고 있다. 표



(a) 비단말노드에 포함된 모든 엔트리들의 *MBB* (b) *nowEntry*의 *LFMBB*

그림 9 *LFMBB*의 예

표 1 엔트리들의 영역확장

삽입되는 간격	삽입시 선택되는 엔트리	영역확장
fixedI	*fixedEntry*	Area(*MBB'*) - Area(*MBR*)
	nowEntry	Area(*LFMBB'*) - Area(*LFMBB*)
nowI	*fixedEntry*	Area(*LFMBB'*) - Area(*MBB*)
	nowEntry	Area(*LFMBB'*) - Area(*LFMBB*)

```
TPIR_Insert( root, I )
TPIR_I1  Set N to be the root
TPIR_I2  If N is a leaf
           return N
         else
           Choose the entry e by least new area enlargement. Resolve ties by least overlap enlargement.
             If I = nowI and chosen e = fixedEntry then e.state = nowEntry
             endif
         endif
TPIR_I3  Set N to be the childnode pointed to by the childpointer of e and repeat from TPIR_I2
```

그림 10 삽입 알고리즘

에서 Area( )는 영역을 계산하는 함수이며 *MBB'*와 *LFMBB'*는 각각 삽입되는 간격을 포함한 *MBB*와 *LFMBB*를 나타낸다. *fixedI*가 삽입되는 경우 *fixedEntry*의 영역확장 계산은 기존 알고리즘과 동일하게 계산되어진다. 그러나 *nowEntry*의 경우에는 앞에서 언급한 것처럼 *LFMBB*를 사용하여 계산하게 된다. *nowI*가 삽입되는 경우에 *fixedEntry*는 삽입되는 간격으로 인해 상태가 변하게 되므로 *LFMBB'*와 *MBB*를 사용하여 영역확장을 계산하게 된다. *nowEntry*의 경우에는 *fixedI*가 삽입되는 경우와 동일하다. 위의 영역확장 계산 방법을 적용한 삽입 알고리즘은 그림 10과 같다.

5.4 분할

노드에 오버플로우가 발생하게 되면 기존 방법에서는 마진을 사용하여 분할 축을 설정하고 겹침이 최소화되도록 노드를 분할하였다[7]. 그러나 TPIR-tree에 이러한 정책을 사용하게 되면 *nowEntry*와 *nowI*가 질의수행시 확장되는 특성을 고려하지 않았기 때문에 겹침이 증가되어 비효율적이다. 따라서 TPIR-tree의 특성을 고려하기 위해 먼저 *Local Fixed nowI(LFnowI)*를 아래와 같이 정의하고 이를 이용하여 기존 R*-tree의 분할 방법과 동일한 방법을 사용하여 분할한다.

정의 4. *LFnowI*는 단말노드의 최대 시간 값으로 *nowI*의 시간 간격을 고정시킨 것

*LFnowI*의 예는 그림 11과 같다. 그림 11(a)와 같이 단말노드에 *fixedI*와 *nowI*가 있다고 가정하자. 이 노드를 분할하는 경우에 포함되는 모든 *nowI*를 그림 11(b)와 같이 *LFnowI*로 만들고 난 뒤에 분할을 수행하게 된다. 이것은 *nowI*가 확장되는 성질을 일정으로 보장해 주기 때문이다.

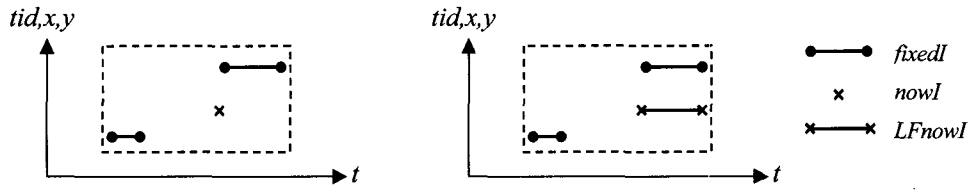
비단말노드의 분할은 앞에서 정의한 *LFMBB*를 이용하여 분할한다. 즉, *fixedEntry*의 *MBB*와 *nowEntry*의 *LFMBB*를 이용하여 분할을 수행한다. 분할 수행 후, 분할된 노드를 가리키는 부모노드의 엔트리들의 *state* 값은 변경되어야 되며 이것은 루트까지 전달되어야 한다. 분할 알고리즘은 그림 12와 같다.

6. 성능 평가

이 장에서는 논문에서 제안한 TPIR-tree와 기존 삽입 및 분할 알고리즘을 사용하는 색인들을 다양한 데이타와 질의를 통해 성능평가를 수행하여 우수성을 입증한다. 비교 평가되는 기존의 색인은 시간 매개변수 간격 데이타 모델을 적용하되, R-tree와 R*-tree의 삽입 및 분할 알고리즘을 사용한 색인이다. 즉, 비교 대상이 되는 두 색인 모두 4차원 색인으로 태그의 간격을 색인으로 구성한다. 비교 대상인 R-tree는 삽입 시 Choose-Subtree 알고리즘과 분할 시 quadratic 알고리즘을 사용하며, R*-tree는 분할 축을 선택하여 분할하고 재삽입을 수행한다.

6.1 실험 환경

이 논문에서 제안한 TPIR-tree는 Microsoft MFC



(a) 단말노드에서의 *fixedI*와 *nowI* (b) *fixedI*와 *LFnowI*

그림 11 *LFnowI*의 예

TPIR_Split(N_{old}, N_{new1}, N_{new2})
If N_{old} is leaf node then
  change all *now*Is to *LFnow*Is in N_{old} and invoke R*_Split(N_{old}, N_{new1}, N_{new2})
  set state of entries which indicate N_{new1} and N_{new2} in the parent node
else
  change all *MBB*s of *now*Entrys to *LFMBB*s in N_{old} and invoke R*_Split(N_{old}, N_{new1}, N_{new2})
  set state of entries which indicate N_{new1} and N_{new2} in the parent node
endif

그림 12 분할 알고리즘

라이브러리와 Visual C++ 6.0을 사용하여 구현하였고, Windows 2003 서버에서 512MB 메인 메모리, CPU Pentium IV 2.6Ghz를 장착한 PC를 이용하여 실험하였다.

색인의 성능을 검증하기 위해서 다양한 환경에서 색인의 성능을 평가할 수 있는 실험 환경이 필요하다. 이동체를 위한 대부분의 색인은 GSTD[10]를 이용하여 다양한 조건에서 실험을 수행한다. 그러나 태그 객체의 경우, 이동체와 다른 위치 정보를 요구하므로 새로운 실험 환경이 필요하다. 이 논문에서 제안하는 새로운 색인의 실험을 위하여 태그 객체의 위치 데이타를 생성하기 위한 태그 데이타 생성기(Tag Data Generator, TDG)를 개발하여 사용하였다. 기본적인 알고리즘은 GSTD알고리즘과 유사하지만, 차이점은 위치 보고 시점이 호출 지역 안으로 들어갈 때와 밖으로 나갈 때만 데이타를 생성한다. 태그 객체가 2개 이상의 호출 지역이 중첩된 지역 안으로 들어갈 경우, 동일 시간에 하나 이상의 위치를 가지도록 설계하였다. 그림 13은 이 논문에서 제시한 색인의 실험을 위해 개발한 태그 객체 위치 생성기이다. 그림에서 원은 판독기의 호출 지역을 나타내며, 공간상으로 근접한 다수의 호출 지역이 중첩되어 나타날 수도 있다. 호출 지역은 태그 객체가 이동할 수 있는 지역에 설치 된다. 그림에서 격자 모양이 나타나지 않는 곳은 건물의 벽이나 잔디밭과 같은 지역이며 태그 객체가 이동할 수 없는 지역이다.

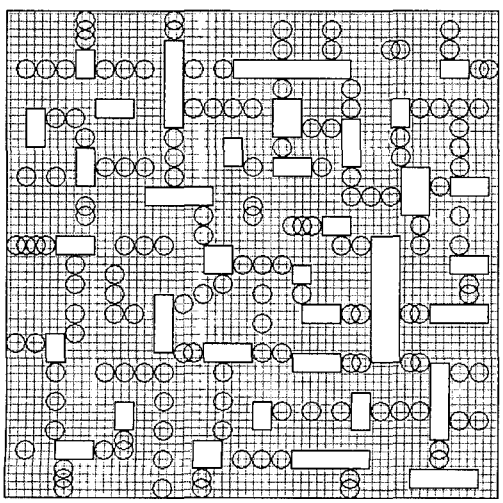


그림 13 태그 객체 데이타 생성기

실험에 사용되는 데이타는 총 5개의 종류로서 태그 수는 각각 50, 100, 200, 500, 1000개이며 각 태그는 *Enter*와 *Leave* 이벤트를 각각 1000번씩 보고한다. 즉, 태그 수가 50개인 데이타세트에는 총 50 × 1000개의 태그의 궤적을 포함한다.

6.2 실험 결과

TPIR-tree의 성능을 평가하기 위하여 FIND 질의와 LOOK 질의를 1, 2, 5, 10, 20%로 질의영역을 변경하며 실험을 수행하였다. 각 질의는 1000번을 수행하였고 그 결과를 평균으로 나타내었다. 그림 14는 질의영역이 1%인 FIND와 LOOK 질의에 대한 각 색인들의 노드 접근을 나타낸 그래프이다. 그림과 같이 두 질의 모두에서 TPIR-tree가 R-tree, R*-tree에 비해 좋은 성능을 나타내며 태그의 수가 증가함에 따라 그 차이는 증가하게 된다. 또한 FIND 질의가 LOOK 질의에 비해 더 많은 노드를 접근하는데 그 이유는 FIND 질의가 공간 차원 x, y 축을 모두 탐색하며 LOOK 질의는 식별자 차원인 *tid* 축만을 탐색하기 때문이다.

그림 15는 질의영역이 2%인 FIND와 LOOK 질의에 대한 각 색인들의 노드 접근을 나타낸 그래프이다. 앞 결과와 동일하게 두 질의 모두에서 TPIR-tree가 R-tree, R*-tree에 비해 좋은 성능을 나타내며 태그의 수가 증가함에 따라 그 차이는 증가하게 된다.

그림 16은 질의영역이 5%인 FIND와 LOOK 질의에 대한 각 색인들의 노드 접근을 나타낸 그래프이다. 앞 결과와 동일하게 두 질의 모두에서 TPIR-tree가 R-tree, R*-tree에 비해 좋은 성능을 나타내며 태그의 수가 증가함에 따라 그 차이는 증가하게 된다. 주목할만

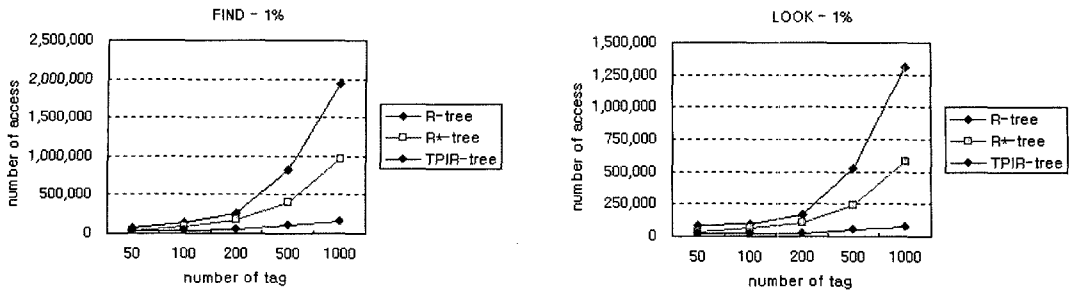


그림 14 1% FIND 및 LOOK 질의

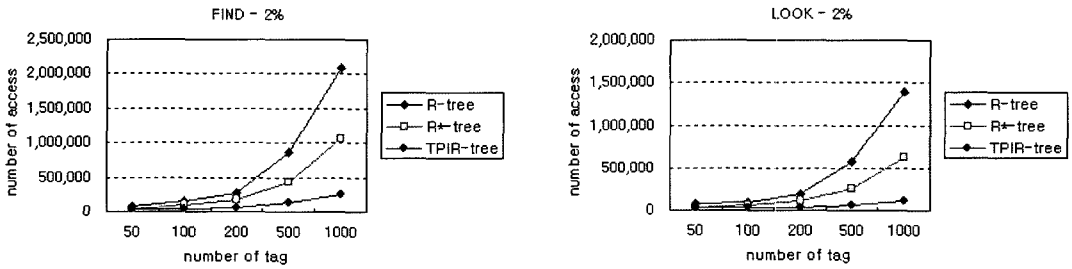


그림 15 2% FIND 및 LOOK 질의

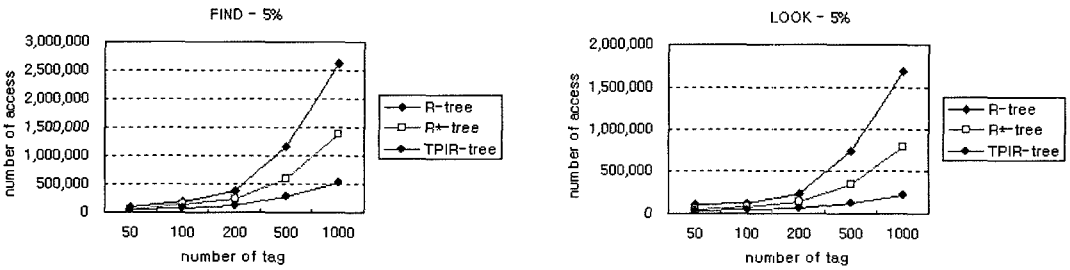


그림 16 5% FIND 및 LOOK 질의

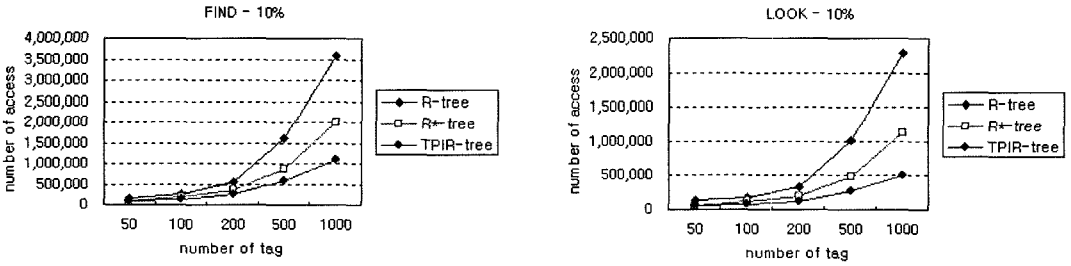


그림 17 10% FIND 및 LOOK 질의

한 점은 FIND 질의가 LOOK 질의보다 질의영역이 증가하면 노드의 탐색도 더욱 증가한다는 것이다. 그 이유는 FIND 질의의 경우 공간 축인 x, y 두 축을 대상으로 모두 탐색하므로 질의영역이 증가할수록 더 많은 탐색이 요구되기 때문이다.

그림 17, 18은 질의영역이 각각 10, 20%인 FIND와 LOOK 질의에 대한 각 색인들의 노드 접근을 나타낸 그래프이다. 앞 결과와 동일하게 두 질의 모두에서 TPIR-tree가 R-tree, R*-tree에 비해 좋은 성능을 나타내며 태그의 수가 증가함에 따라 그 차이는 증가하게 된다. 또한 FIND 질의가 LOOK 질의에 비해 많은 탐색을 요구하며 질의영역의 증가로 인해 노드 탐색도 더욱 증가하게 된다.

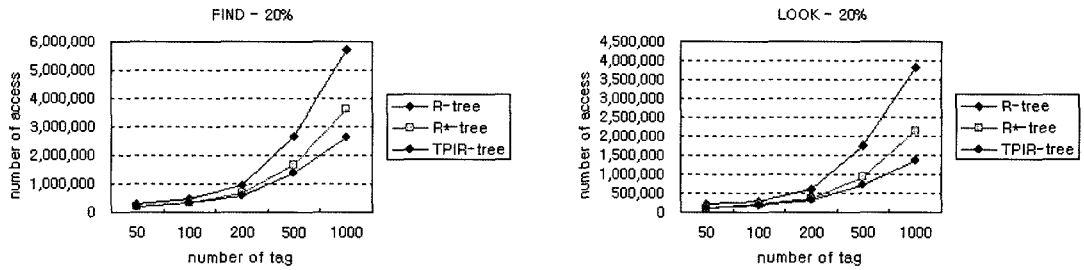


그림 18 20% FIND 및 LOOK 질의

7. 결론 및 향후 연구

태그와 이동체는 시간에 따라 위치가 변하는 공통된 특성을 가지므로 태그의 궤적을 위한 색인은 이동체의 색인을 사용하여 구축할 수 있다. 그러나 판독기에 머물고 있는 태그는 궤적을 표현할 수가 없으므로 질의 시 이러한 태그를 검색할 수 없다.

이 논문에서는 이러한 문제를 해결하기 위하여 태그의 궤적을 시간 매개변수 간격으로 정의하고 새로운 색인인 TPIR-tree(Time Parameterized R-tree)를 제안하였다. 시간 매개변수 간격은 시간에 종속적인 선분으로 태그가 판독기에 들어온 것만을 보고하더라도 질의를 처리할 수 있다. 또한 효율적인 질의처리를 위한 새로운 삽입 및 분할 알고리즘을 제안하였다.

실험을 위하여 태그 객체의 위치 데이타 생성기를 GSTD 알고리즘에 기반하여 설계 및 구현하였으며, 생성된 데이타를 사용하여 성능 평가를 수행하였다. 이 논문에서 제안하는 TPIR-tree가 기존 색인들보다 FIND 질의와 LOOK 질의에서 우수한 질의 성능을 보였다.

향후 연구로서 삽입 알고리즘과 분할 알고리즘의 개선을 통해 성능의 향상이 필요하다. 또한 다양한 질의 예를 들어, 복합질의 또는 최소근접질의 등의 수행에 관한 연구가 필요하다.

참 고 문 헌

[1] K. Romer, T. Schoch, F. Mattern and T. Dubendorfer,: Smart Identification Frameworks for Ubiquitous Computing Applications. Proceedings of the First IEEE International Conference on Pervasive Computing and Communications (2003) 256-262.

[2] Y. Theodoridis, M. Vassilakopoulos, and T. Sellis. "Spatio-temporal indexing for large multimedia applications," In Proc. of the 3rd IEEE Conf. on Multimedia Computing and Systems, pp.441-448, June 1996.

[3] M. A. Nascimento, J.R.O. Silva and Y, Theodoridis. "Evaluation of Access Structures for Discretely Moving Points," In Proc. of the Intl. Workshop on Spatiotemporal Database Management (STDBM'99), pp. 171-188. Edinburgh, UK, Sep/99.

[4] M. A. Nascimento and J.R.O. Silva. "Towards historical R-Trees," In Proc. of the 1998 ACM Symposium on applied Computing, pp.235-240, February 1998.

[5] D. Pfoser, C. S. Jensen, and Y. Theodoridis. "Novel Approaches to the Indexing of Moving Objects," In Proc. of the 26th VLDB Conf, pp.395-406, 2000.

[6] A. Guttman, "R-trees: A dynamic index structure for spatial searching," In Proc. ACM SIGMOD, pp.47-54, 1984.

[7] N. Beckmann and H. P. Kriegel, "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles," In Proc. ACM SIGMOD, pp.332-331, 1990.

[8] C. Kolovson and M. Stonebraker. "Segment Indexes: Dynamic Indexing Techniques for Multi-Dimensional Interval Data," Proc. ACM SIGMOD, pp.138-147, 1991.

[9] EPC Tag Data Standard Work Group, "EPC Tag Data Standards Version 1.23," EPC Global, 2005.

[10] Y. Theodoridis, J. R. Silva and M. A. Nascimento, "On the Generation of Spatiotemporal Datasets," SSD, Hong Kong, LNCS 1651, Springer, pp.147-164, 1999.

[11] 이기형, 반재훈, 김동현, 홍봉희, "RFID 태그 객체의 간격 데이타 색인," 한국정보과학회 가을 학술발표 논문집 제31권 제2호, pp.82-84, 2004.

반 재 훈

1997년 부산대학교 컴퓨터공학과 공학사 1999년 부산대학교 컴퓨터공학과 공학석사. 2001년 부산대학교 컴퓨터공학과 박사수료. 2002년~현재 경남정보대학 인터넷응용계열 조교수. 관심분야는 데이타베이스, 공간 데이타베이스, 이동체 데이타베이스, RFID

홍 봉 희

1982년 서울대학교 전자계산기공학과 졸업(학사). 1984년 서울대학교 전자계산기공학과 졸업(석사). 1988년 서울대학교 전자계산기공학과 졸업(박사). 1987년~현재 부산대학교 공과대학 컴퓨터공학과 교수/부산대학교 컴퓨터 및 정보통신연구소 책임연구원. 관심분야는 이동체 데이타베이스, 모바일 데이타베이스, 공간 데이타베이스