韓國컴퓨터情報學會 論文誌
第15卷 第5號, 2010. 5.

2010-15-5-1-3

속도 변화가 빈번한 이동 객체의 미래 시점 위치 추정에 적합한 색인 기법

임 성 채*

An Indexing Scheme for Predicting Future-time Positions of Moving Objects with Frequently Varying Velocities

Sung Chae Lim*

요 약

정보 기술과 이동 통신망의 발전에 따라 이동 객체의 위치 추적에 필요한 효과적인 색인 기법과 이를 통한 다양한 응용 서비스에 대한 사용자 요구가 커지고 있다. 이에 따라, 이동 객체의 현재 위치 추적과 미래 위치 추정에 사용되는 TPR*-트리와 같은 색인 기법이 관심을 받고 있다. TPR*-트리는 정적인 물체를 색인하기 위해 고안된 R-트리를 기본 구조로 하기 때문에, 갱신 비용이 크다는 단점이 있다. 따라서 이동 객체가 자주 속도나 위치 정보를 갱신하는 경우 트리 유지비용이 빠르게 증가할 수 있다. 특히, 이동 객체가 빠른 속도로 이동하는 경우 넓은 범위에 걸쳐 불필요한 노드 갱신이 발생할 수 있다는 문제점이 있다. 이런 문제점을 피하기 위해 논문에서는 루트노드의 자식노드에 이런 빠른 이동 속도의 객체만을 따로 색인하도록 하여 노드 갱신 비용을 크게 줄이는 방법을 제안한다. 제안된 방법을 통해 노드 갱신 비용을 최소화 하면서도 TPR*-트리의 장점을 유지할 수 있었다. 이런 성능상의 장점을 보이기 위해 시뮬레이션 기법을 사용한 성능 비교를 수행하였다.

Abstract

With the advances in the information technology and mobile communications, we now face increasing demands for various services based on both of position tracking of moving objects and their efficient index scheme. Accordingly, the TPR*-tree, which were proposed for efficiently tracking moving objects and predicting their positions in the future time, has drawn much intention. As the TRP*-tree came from the R-tree that is suitable for indexing static objects, it does not support cheap update costs. Therefore, it seems to be very costly to index moving objects if there are frequent occurrences of node updates caused by continuously changing velocities and positions. If some moving objects with high velocities have node updates, in particular, then the TPR*-tree may suffer from many unnecessary updates in the wide range of tree regions. To avoid such a problem, we propose a method that can keep fast-moving objects in the child nodes of the root node, thereby saving node update costs in the TPR*-tree. To show our performance advantages and retaining TPR*-tree features, we performed some performance experiments using a simulation technique.

• 제1저자 : 임성채
• 투고일 : 2010. 03. 17, 심사일 : 2010. 04. 13, 게재확정일 : 2010. 04. 23.
* 동덕여자대학교 컴퓨터학과 교수
※ 이 논문은 2009년도 동덕여자대학교 학술연구비 지원에 의하여 수행된 것임.

▸ Keyword : 색인(indexing), 이동 객체(moving objects), TPR*-트리(TPR*-tree)

Ⅰ. 서 론

정보통신기기 및 이동통신망의 발전과 함께 이동 객체에 대한 위치 추적 및 위치 검색 기능이 점차 중요시 되고 있다[1, 2]. 이러한 추세는 향후 보편화 될 자동차 텔레매틱스, GPS 장착 무선기기들의 확대로 계속될 것이라 예상된다. 또한 최근 국내에서도 크게 관심을 모으고 있는 스마트 폰이 빠르게 보급된다면 좀 더 다양한 형태의 위치 정보 서비스들이 개발되고 사용될 것이다. 이런 이동 객체의 위치 추적 및 검색과 관련된 다양한 서비스의 활성화를 위해서는 매우 효율적인 위치 정보 색인 기술이 요구된다[1-5].

이동 객체의 위치 정보와 관련된 질의는 크게 세 가지 정도로 분류된다. 첫째는 이동 물체의 현재 위치 정보와 관련된 질의이다 [5-7]. 예를 들어, 특정 이동 객체의 현 위치나 그와 인접한 이동 객체들을 식별하는 질의가 있을 수 있다. 두 번째로는 이동 객체의 이동 이력과 관련된 질의이다. 예를 들어 과거 어떤 시점에 특정 지역을 지나간 이동 객체를 검색하는 질의가 있을 수 있다. 이런 질의에 대해 가급적 작은 색인 공간을 사용해서 과거의 이동 내력을 표현할 수 있는 기술이 요구된다[8-10]. 마지막으로는, 미래의 특정 시점이나 시간 구간에 대한 위치를 예측해 보는 질의가 있다. 이런 질의를 위해서는 객체의 이동 속도를 효과적으로 저장하고 미래 위치를 낮은 비용으로 계산할 수 있어야 한다[4, 11-13]. 이런 세 가지 분류 중 미래 위치 관련 질의에 대한 연구는 교통량 예측이나 교통 통제 시스템 등에서 다양하게 응용될 수 있기 때문에 활발히 연구되었다. 또한 미래 위치 질의는 미래 시점을 매우 가깝게 하면 현재 위치 질의로 사용될 수 있기에 그 활용도가 높다. 이런 이유로 본 논문에서는 미래 위치 추정 및 현재 위치 추적이 가능한 TPR*-트리[13]를 개선한 색인 기법을 연구한다.

이동 객체 색인에 사용되는 TPR*-트리는 R-트리[14, 15]를 그 기본 구조로 했다고 할 수 있다. 공간 데이터 색인에 사용된 R-트리는 색인된 객체들의 공간 밀도에 적응하여 적절히 색인 공간을 나눌 수 있음으로 해서 저장 공간을 효과적으로 사용할 수 있다는 장점이 있다. R-트리에서는 MBR(Minimum Bounding Rectangle)이라고 하는 사각형을 하나의 단위로 삼아 MBR이 표현하는 공간에 속한 객체들을 하나의 단말노드에서 저장한다. 그리고 이런 단말노드에 저장된 객체를 싸는(enclosing) MBR에 대한 정보는 해당 단말노드의 부모노드에 저장되어 경로 탐색에 사용된다. MBR들을 싸는 상위 MBR도 계산되어 부모노드들에 bottom-up 방식으로 저장된다. R-트리는 고정된 객체의 위치 정보나 값의 변화가 적은 다차원 데이터 색인에 사용되는 것이기 때문에 위치 정보나 다차원 데이터 값이 자주 변하는 상황에서는 효과적이지 못하다. 또한 속도 정보를 효과적으로 표현하는 방법이 없기에 이동 객체 색인에 그대로 적용하기는 어렵다[4, 13].

이에 비해 TPR*-트리는 R-트리의 MBR에 최대 확장 속도 정보를 합친 형태의 CBR(Conservative Bounding Rectangle)을 사용한다. CBR 사용을 통해 이동 객체가 위치할 수 있는 최대 공간을 예측하고 실제 이동 객체 개개의 속도 정보는 내부노드(internal node)에 두지 않음으로써 색인 데이터 크기를 줄인다. 하지만 TPR*-트리 역시 R-트리의 특성인 갱신 비용이 크다는 단점을 가고 있다[11, 12]. 특히, 빠른 속도의 이동 객체가 빈번히 노드 갱신을 유발하는 경우 트리 유지비용이 매우 커질 수 있다. 이런 문제 인식에 따라 노드 갱신이 자주 발생하는 상황에서 낮은 빈도의 노드 갱신만으로도 트리를 유지할 수 있는 방법을 연구한다. 이를 위해 안정화 노드(cooling-off node) 개념을 도입하였다. 이런 안정화 노드는 메모리에 저장될 수 있고 단말노드에 비해 매우 넓은 색인 공간을 표현하기 때문에 노드 갱신 비용을 최소화 할 수 있다.

논문의 구성은 다음과 같다. 2장에서는 논문 관련된 기존의 이동 객체 색인 기법에 대해 알아본다. 3장에서는 논문에서 제안하는 아이디어 및 세부 알고리즘을 기술한다. 4장에서는 성능 평가를 통해 제안된 방법의 우수성을 보인다. 마지막으로 5장에서 결론을 맺는다.

Ⅱ. 관련 연구

2.1 TPR*-트리

TPR*-트리는 미래 특정 시점이나 시간구간에 대한 미래 위치 질의 처리을 위해 고안된 구조이다[4, 13]. 이 트리의 단말노드 N에는 하나의 CBR(Conservative Bounding Rectangle)에 속한 이동 개체들의 위치 및 이동 속도 정보가 저장되고, N의 부모노드 P에는 노드 N에 대응되는 CBR 데이터가 저장된다. CBR은 색인할 이동 객체들의 최소 외각을 싸는 MBR과 이동 객체들의 최대 속도만을 기록한 최대 속도

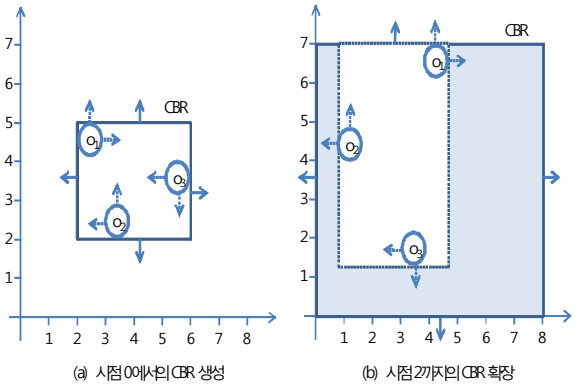

그림 1. CBR 생성 및 그 확장 예
Fig. 1. Creation and expansion of a CBR

벡터로 이뤄진다. MBR을 최대 속도로 확장시킨 공간은 색인된 모든 이동 객체의 미래 위치를 항상 포함한다. 즉, CBR의 확장을 통해 색인된 이동 객체의 미래 위치를 감쌀 수 있으며, 대부분의 경우 CBR 영역은 이동 객체들의 실제 위치 공간보다 크다. 이런 CBR 사용을 통해 중간노드에는 이동 객체의 개별 속도를 저장하니 않아도 됨으로써 저장 공간을 크게 줄일 수 있는 장점이 있다.

그림 1은 CBR의 개념을 간단히 보인다. 그림은 3개의 이동 객체 o1, o2, o3를 시점 0에 색인한 후, 시간이 2 흐른 후의 모습을 보인다. 그림 1.(a)은 시점 0에서 CBR이 생성될 때의 모습이며, 사각형은 객체들을 싸는 MBR이고 MBR 바깥의 실선 화살표는 확장 속도를 표현하고 있다. 각 이동 객체의 속도는 두 개의 점선 화살표로 나타냈고, 편의상 이동 객체의 속도는 두 방향에 대해서 단위 시간당 1로 동일하다고 가정했다. 이런 속도로 이동하는 경우 시간 2가 흐른 후에는 그림 1.(b)와 같은 위치를 가진다. 만약 시점 2에 CBR을 다시 생성한다면(혹은 갱신한다면) 그림 1.(b)의 점선 표시 사각형이 새로운 MBR로 사용될 것이다. 따라서 그림 1.(b)의 회색 칠해진 부분은 CBR의 확장에 의해 불필요하게 커진 색인 공간이며 이를 사장영역(dead space)이라 부른다. CBR의 확장이 색인된 객체들의 최대 이동 속도에 따라 확장되기 때문에 이런 사장 영역의 발생은 피할 수 없다. CBR의 생성(혹은 갱신)이 없다면 사장영역은 지속적으로 커지며, 이에 따라 질의 처리 시 불필요한 단말노드 검색이 다수 발생할 수 있다. 여기서 불필요한 노드 검색이란 실제 이동 객체가 실제로는 존재할 수 없는 색인 구역임에도 사장 영역이 발생하여 트리 탐색이 일어날 때를 의미한다. 예를 들어 그림 1.(b)의 회색 구역은 이동 객체가 위치할 수 없음에도 해당 회색 구역에 대한 검색 질의가 있다면 해당 노드를 읽어 들여야 한다. 이런 경우 전혀 불필요한 노드 읽기가 발생할 것이다[11].

TPR*-트리는 앞서 설명한 TPR-트리[4]의 기본 구조에 이동 객체 삽입 시 스위핑(sweeping) 영역의 개념을 사용함으로써, 확장해 가는 CBR의 스위핑 영역 크기를 고려해 질의 처리 성능에 미칠 영향을 좀 더 정확히 계산했다. 이런 계산을 위해서 TPR*-트리는 CBR이 시간의 흐름과 CBR 크기 변화 요소를 모두 고려해야 한다. 즉, 적분(integral)의 개념을 사용하여 시간 축에 따른 스위핑 역역 크기를 구해야 한다. 이에 따라 기존 TPR-트리에 비해 계산 비용이 커지는 특징이 있다. 하지만 미래 시간 동안에 대한 보다 정확한 위치 추정으로 인해 기존의 TPR-트리에 비해 좋은 질의 처리 성능을 가지면 이런 성능 향상을 통해 CPU 비용 추가분을 상쇄하는 것으로 알려져 있다[13].

2.2 TPR*-트리의 능동적 재조정

앞서 설명했듯이 CBR은 자신이 싸고 있는 이동 객체의 최대 속도만을 저장함으로 저장 데이터 크기를 줄이고 계산을 단순화시킨다. 하지만 이로 인해 그림 1.(b)와 같은 사장영역 발생이 불가피하다. 이런 사장영역을 줄이기 위해서는 자주 CBR 데이터를 갱신해야 한다. 이런 CBR 갱신은 단말노드 갱신이 발생할 때 수행되며, 단말노드 갱신은 색인된 이동 객체가 속도를 변경함으로써 생긴다. 즉, 단말노드에 갱신 시 관련된 CBR 데이터도 함께 상향식(bottom-u)p 방식으로 갱신되며 이런 갱신 작업은 루트노드까지 전파될 수 있다. R-트리 계열의 색인 에서와 같이 이런 상향식 재구성 기법은 가급적 적은 노드 수정으로 트리 일관성을 지키기 위해 필요하다.

기존의 R-트리 구조에서는 노드 갱신이 자주 일어나지 않지만 TPR*-트리에서는 속도 및 위치 변화가 빈번할 수 있기에 갱신이 자주 일어 날 수 있고 이에 따라 성능 저하가 초래될 수 있다는 문제가 있다. 이런 문제점을 해결하기 위해서 [4]와 같이 노드 갱신 시에 단말노드에 직접 접근하여 노드 갱신을 수행하는 방식이 제안된바 있다. [4]에서는 노드 갱신 시 트리 탐색 없이 별도의 해시 테이블을 사용하여 해당 단말노드에 접근할 수 있다는 장점이 있다. 이런 방식은 논문에서 제안한 방식과 상호 배제적인 방식은 아니다. 즉, 초기 단말 노드 접근 시 별도의 해시 테이블을 사용하며 이후 노드 갱신이 발생할 때 제안한 방식이 적용될 수 있다. 이외에 능동적 재조정을 통한 TPR*-트리 성능 향상 기법이 있다[11, 12].

능동적인 CBR 재조정 기법에서는 질의 처리 프로세스가 질의 처리 과정 중 읽어 들인 트리 경로에 기록된 CBR 데이

터와 단말노드의 최종 갱신 시점 정보를 사용하여, 자신이 능동적으로 CBR 갱신을 수행할지 정하는 방법이다. 즉, 다음 단말노드 갱신이 발생할 것으로 예상하는 시간 Tnext를 확률적으로 계산하고 현재 시점을 t라 할 때, 자신이 수행할 CBR 갱신으로 구간 [t , Tnext] 동안에 사장영역을 탐색해 들어오는 불필요한 노드 검색을 얼마나 줄일 수 있는지를 계산한다. 그리고 이런 성능 향상 기대치와 CBR 갱신 비용을 비교함으로써 능동적 CBR 갱신의 수행 여부를 결정하는 방식이다. 이 방식은 단말노드에 타임스탬프에 해당하는 공간을 할당해야 하고, 질의 처리 프로세스가 성능 향상 기대치와 비용을 계산하는 CPU 비용이 있다. 하지만 두 비용 모두 전체 질의 처리 비용에 비해 매우 작은 비용이라 할 수 있다.

이런 능동적인 CBR 재조정 기법은 기존 TPR*-트리의 두 가지 문제점을 피할 수 있다. 첫 번째는 노드 갱신이 적절한 빈도로 생기지 않을 경우 사장영역이 계속 커져 트리 성능이 크게 악화될 수 있다는 점이다. 두 번째는 단말노드 갱신으로 CBR 갱신이 자주 일어나도 이로 인해 콤팩트해진 색인 영역과 실제 질의되는 색인 영역이 불일치 할 수 있다는 점이다. 즉, 노드 갱신이 자주 발생하는 트리 부분과 자주 질의 되는 트리 부분이 서로 다를 경우 CBR 갱신이 성능 향상에 큰 영향을 주지 못할 수 있다[11, 12]. 앞서 언급한 기존 TPR*-트리의 두 가지 문제점을 효과적으로 해결한다. 즉, 노드 갱신이 자주 발생하지 않을 경우에도 질의 처리 프로세스가 CBR 갱신을 수행할 수 있기에 지나친 사장 영역의 확장을 막을 수 있다. 또한, 질의 처리 과정에서 접근되는 트리 경로에 대해 CBR 갱신이 수행되는 것이기 때문에 CBR 갱신으로 인한 트리 처리 성능 향상으로 이어질 개연성이 매우 크다. [12]에서는 이런 방법 적용으로 최적의 경우 질의 처리 시 2-3배의 성능 향상을 보임을 알 수 있다.

III. 제안하는 미래 위치 색인 기법

3.1 기본 아이디어

논문에서는 빠르게 움직이는 이동 객체가 빈번히 속도를 변경하는 상황에서 TPR*-트리의 노드 갱신 비용을 최소화시키는 방법을 연구한다. 이를 위해 상대적으로 넓은 범위에 속한 이동 물체를 색인할 수 있는 안정화(cooling-off) 노드를 사용하며, 안정화 노드는 빠른 속도를 가지고 여러 노드를 지나쳐 가는 이동 객체를 매우 작은 노드 갱신 비용으로 색인할 수 있다. 제안하는 안정화 노드는 TPR*-트리의 루트노드 바로 아래 단에 존재하며, 제안 기법은 루트노드를 포함해 최소

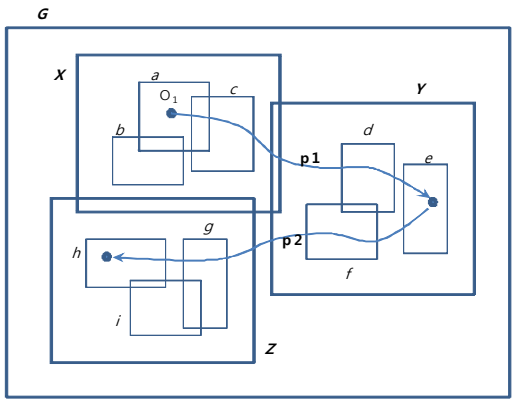

그림 2. 이동객체 *o1*의 빠른 이동 예
Fig. 2. Fast movement of object *o1*

3계층의 높이를 가지는 TPR*-트리에 적용 가능하다. 이런 조건은 우리가 성능 향상을 위해 관심을 갖는 색인 데이터가 다수의 이동 객체 추적에 사용되는 것이기 때문에 실제적 제약조건은 되지 않는다.

안정화 노드의 유용성을 직관적으로 설명하기 위해 그림 2를 사용한다. 그림에서 점선으로 표시된 사각형들(a부터 i)은 모두 CBR G를 조상으로 갖는 CBR들이며, CBR a, b, c는 CBR X에 의해 싸여있고, d, e, f는 Y에 CBR g, h, i는 Z에 의해 싸여 있다. 그림은 CBR a에 속한 이동 객체 o1이 이동 경로 p1을 따라 CBR e의 공간으로 이동한 후, 다시 이동 경로 p2를 따라 CBR h로 이동한 상황을 보인다. 만약 이때 o1이 빠른 속도를 가속하여 이동하였다면 이동 경로에 있는 a, c, d, e, f, g, h와 상위 계층의 CBR인 X, Y, Z에 갱신이 필요할 수 있다. 하지만 매우 짧은 시간 동안 빈번이 발생하는 노드 갱신 비용은 CBR을 콤팩트하게 함으로써 얻는 이익에 비해 매우 클 수 있다. 이런 현상은 이동 객체가 빠른 속도로 매우 넓은 색인 구간을 지날 때 더욱 커질 수 있다. 논문에서는 이처럼 빠른 가속이 발생하여 여러 노드를 불필요하게 갱신하는 상황은 이동 객체의 전체 이동 시간에 비해 크지 않을 것임을 고려해, 이런 이동 객체는 좀더 넓은 공간을 색인하는 안정화 노드에 둠으로써 갱신 비용을 줄인다.

이런 안정화 노드의 사용 형태는 그림 3에서 보인다. 그림 3은 그림 2의 상황을 트리 형태로 보인 것으로, 가장 아래 사각형은 단말노드들이며 루트노드의 자식노드인 노드 N에 안정화 노드(그림에서 CoN(Cooling-off Node))가 추가되어 있다. 노드 N에 들어있는 안정화 노드 CoN은 CBR G에 속한 이동 객체의 속도 및 현재 위치를 저장할 수 있다. 즉, 단말노드에 저장되는 데이터가 CoN에 저장되며, 기존 내부노드의 크기를 확장함으로써 만들어 진다. 그림 2에서와 같이 o1이

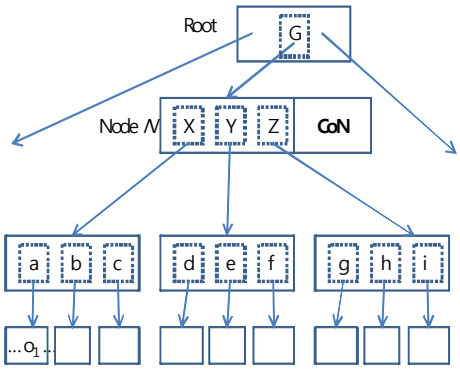

그림 3. 노드 N에 추가된 안정화노드 CoN
Fig. 3. Cooling-off node CoN on node N

빠른 속도로 움직일 때는 o1을 노드 N의 CoN에 저장하여 둔다면 이런 빠른 이동에도 노드 N의 CoN에만 노드 갱신이 발생할 것이다. 물론 이런 급격한 속도 변화가 안정화된 후에는 다시 그 시점의 위치에 따라 단말노드로 다시 내려가 색인된다. 안정화 노드는 상위 계층에 있기 때문에 표현하고 색인 공간이 크고 확장 속도 역시 크다. 따라서 이곳에 저장된 이동 객체가 빠르게 이동하여도 해당 CBR에 머물게 될 확률이 크므로, 노드 갱신 범위를 최소로 할 수 있다.

또한, 안정화 노드는 트리의 루트노드를 제외한 상위 계층에 속하므로 그 숫자가 작다. 따라서 메인 메모리에 위치시키는 것이 일반적이다. 이런 메인 메모리 사용은 B-트리나 R-트리에서와 같이 트리 상위 계층에 있는 노드 숫자가 작고 상대적으로 빈번히 접근되는 색인 구조에서는 성능 향상을 위해 널리 적용되는 방식이다[8, 12]. 이런 상황에서, o1이 자주 갱신 작업을 유발한다고 할 때, 제안하는 방식은 메모리에 존재하는 노드 N을 참조함으로써 해당 작업을 수행할 수 있다. 즉, 별도의 디스크 노드 접근 없이도 작업을 수행할 수 있다. 이에 대한 세부 알고리즘은 다음 절에서 기술하기로 한다.

3.2. 상세 알고리즘

앞에서 논문은 기존 TPR*-트리 구조 및 검색 알고리즘을 대부분 동일하게 사용할 수 있다. 단지 안정화 노드가 루트노드 자식노드 단에 존재하는 점을 고려해 이곳에 색인된 이동 객체 위치 정보를 참조하는 과정이 있다는 것이 차이점이다. 따라서 탐색 알고리즘은 별도 설명을 하지 않고, 갱신 알고리즘에 대해서만 설명한다.

제안하는 기법 설명을 위해 이동 객체의 속도 변경 시 수행되는 TPR*-트리 갱신 연산에 대해 간단히 기술한다. 특정 이동 객체 o1이 시간 t에 속도를 기존 v에서 v'로 변경했다면, 시점 t에서의 위치 p와 바뀐 속도 정보 (p, v')가 TPR*-트리에 반영되어야 한다. 이를 위해 위치 정보인 p를 이용하여 트리 탐색이 수행되고 이를 통해 o1에 대한 위치 및 속도 데이터가 있는 단말노드가 읽혀진다. 이 후, 갱신 작업이 수행된다. 즉, o1의 정보 수정으로 관련 CBR 데이터가 수정되어야 한다면 o1이 단말노드에서 삭제된 후 다시 트리에 재삽입된다. 그렇지 않고 CBR 데이터 수정이 필요치 않다면 단말노드 수정으로 갱신 작업을 완료한다. 이런 갱신 작업은 알고리즘의 라인 12에 있는 함수 호출을 통해 수행되며 이는 기존 [13]의 알고리즘에 따른다.

그림 4는 제안하는 안정화 노드를 사용한 갱신 알고리즘이다. 알고리즘에 주어진 매개변수 *Root*와 *Ctime*은 각각 갱신될 TPR*-트리의 루트노드 포인터와 갱신 시점이다. *Obj*는 (id, v, p)의 데이터로 이루어지며 id는 식별자, v는 갱신된 속도, p는 위치이다. 그리고 마지막의 Threshold는 안정화 노드에 저장할 지를 판별하는 최저 속도 값이다. 그림 4의 알고리즘은 식별자 *Obj.id*를 가진 이동 객체의 속도 변경에 따른 갱신 알고리즘이다.

라인 1에서는 갱신될 *Obj*가 안정화 노드에 저장되어 있는지를 체크한다. 이는 메모리 상에 위치한 안정화 노드를 참조함으로써 수행되며, 존재한다면 해당 노드의 포인터를 CoN에 저장한다. 만약 *Obj*가 CoN에 존재하고 CoN의 CBR 정보가 수정될 상황이 아니라면 라인 4에서 CoN 데이터가 수정된 후 갱신 작업이 완료된다. 그렇지 않고, CoN의 CBR이 갱신되어야 될 상황이라면 라인 7-8이 수행된 후 나머지 단계들이 수행된다. 루트노드 수정이나 CoN 데이터 수정은 모두 메모리에서 발생하기 때문에, 이에 대한 갱신 비용은 거의 없다고 할 수 있다. 라인 3에서 사용한 "contains"는 CoN의 CBR이 *Obj*를 완전 포함하여 CBR 데이터가 수정될 수 없음을 위미한다. 즉, 현재 CBR의 영역이 *Obj.p*를 포함하고 있고, *Obj.v*가 CBR의 확장 속도보다 작거나 같음을 의미한다. 라인 11 이후의 단계들은 안정화 노드를 통해 갱신이 완료되지 않은 경우에 대해 수행된다. 라인 11에서 Threshold 값 보다 작은 속도의 이동 객체는 기존 TPR*-트리 삽입 알고리즘에 따라 처리되며 그렇지 않은 경우에는 라인 14-26을 통해 안정화 노드에 저장된다. 라인 14-18에서는 *Obj*를 포함하는 안정화 노드 목록이 Candidate에 저장된다. Candidate에 속한 노드가 1개 이상이라면 라인 20에서와 같이 *Obj*와 가장 멀리 떨어진 외곽을 가진 CBR에 *Obj*가 삽입된다. 이 경우 CBR 데이터는 갱신될 필요가 없기에 단지 CoN 데이터만 갱신되면 된다. 이와 다르게 Candidate에 노드가 하나도 없다면 *Obj*를 삽입한 안정화 노드 N을 라인 23에서와 같이 선택하고 라인 25에서 루트노드를

```
알고리즘 Update Tree Obj( Root, Ctime, Obj , Threshold )
    Root: TPR*-트리의 루트노드 포인터
    Ctime: 현재 갱신 시간
    Obj: 삽입할 이동 객체 정보. (id, p, v)로 구성됨
    Threshold : 안정화 노드 저장 이동 물체의 최저 속도
Begin
1.  CoN ← GetCoolingNode(Root, Ctime, Obj). // Obj를 색인하고 있는 안정화 노드 탐색
2.  if CoN != Nil then // Obj는 이미 안정화 노드에 존재
3.      if CBR(CoN) contains Obj then // CoN의 CBR 수정이 필요한지 체크
4.          CoN에 저장된 위치 및 속도 정보를 Obj.p와 Obj.v로 변경. // 메모리 내 연산
5.          알고리즘 종료. // 안정화 노드의 데이터 갱신으로 작업 완료
6.      else // CoN의 CBR 수정이 요구됨
7.          CoN에서 Obj.id를 식별자로하는 객체의 데이터 삭제. // 삭제 후 다시 삽입됨
8.          루트노드를 수정하여 N의 CBR 데이터를 갱신. // 메모리 내 연산
9.      endif
10. endif
11. if |Obj.v| < Threshold then // 안정화 노드에 삽입되지 않음
12.     함수 UpdateTPRTree(Root, Ctime, CoN, Obj)를 호출. // TPR*-트리 갱신 알고리즘
13. else
14.     Candidate ← ∅. // 초기화
15.     foreach child node c ∈ Root do
16.         if CBR(c) contains Obj then
17.             Candidate ← Candidate ∪ {c}.
18.     endfor
19.     if |Candidate| > 0 then
20.         Select node N among the set Candidate such that the boundaries of CBR(N) are farthest
        from Obj. // Obj에서 가장 멀리 떨어진 boundary를 가진 CBR에 삽입됨.
21.         함수 InsertObjCoN( Root, Ctime, N, Obj )를 호출. // 안정화 노드에 삽입
22.     else
23.         루트노드의 CBR 중 Obj 삽입으로 가장 작게 확장될 자식노드 N을 찾음.
24.         함수 InsertObjCoN( Root, Ctime, N, Obj )를 호출. // 안정화 노드에 삽입
25          루트노드를 수정하여 N의 CBR 데이터를 갱신. // 메모리 내 연산
26.     endif
27. endif
End.
```

그림 4. 갱신 알고리즘 *UpdateTreeObj()*
Fig. 4. Update algorithm for *UpdateTreeObj()*

갱신하여 N의 CBR 데이터를 수정한다.

사용한 함수 중 *InsertObjCoN()*은 몇 가지 고려 사항이 있다. 안정화 노드에 삽입하려 할 때, 해당 노드에 빈 공간이 존재할 경우는 빈 공간을 사용하여 저장하면 되지만 빈공간이 없는 상태도 존재한다. 즉, 여러 개의 이동 객체들이 안정화 노드로 이미 많이 이동해 있는 경우이다. 이런 경우 기존에 있는 이동 객체 중에서 하나를 삭제하여 빈 공간을 생성해야 한다. 즉, 안정화 노드 CoN에 저장된 기존 이동 객체 중 가장 속도가 작은 이동 객체를 삭제하고 해당 희생자 객체는 TPR*-트리 알고리즘에 따라 트리에 다시 삽입된다.

IV. 성능 평가

성능 평가를 위해 기존 객체 이동 시뮬레이션 프로그램인 GSTD(Generator for SpatioTemporal Data)[12, 16, 17]를 사용한다. GSTD를 통해 12,000 x 12,000 크기의 2차원 공간 상에 250,000 개의 점(point)를 가상 이동시킴으로써 성능평가를 수행한다. 이동 객체가 최초 생성될 때의 위치분포는 균등분포, 스큐 분포, 가우시안 분포 등을 사용할 수 있고 이에 따라 초기 분포 형태를 변화시킬 수 있다. TPR*-트리는 색인 물체의 밀도에 잘 적응하고, 갱신 연산의 경우는 물체의 밀도 특성에 큰 영향을 받지 않기에 이런 초기 밀도 변화는 성능에 큰 영향을 미치지 않는다. 이는 기존 성능 평가[12]에서 알 수 있었고 이런 이유로 본 논문에서는 균등 분포에 대해서 실험한다. 그리고 가상공간을 벗어난 이동 객체는 다시 공간의 반대쪽으로 재등장하도록 한다. 실험은 윈도우 서버 2003에서 수행되었고 사용된 메모리 크기는 2GB이다. 안정화 노드에 저장되는 이동 객체 수는 단말노드에 저장되는 이동 객체 수의 두 배로 하여 안정화 노드의 메모리 사용을 제한했다. 또, TPR*-트리의 단말노드와 내부노드에는 각각 40개와 20개의 엔트리를 저장할 수 있게 했다.

그림 5는 이동 객체들의 평균속도 변화에 따른 트리 갱신 비용을 측정한 결과이다. 결과 그래프에서 CoN는 제안한 방식을, TPR은 TPR*-트리 방식에 따른 결과를 보인다. 트리 갱신 비용은 갱신 연산 동안 접근되는 노드 수(읽기 및 수정을 위한 노드 접근)를 측정한 값이다. 뒤에 S로 표시된 그래프는 트리에 발생하는 갱신 수가 단위 시간당 10회인 경우를, S는 40회인 경우를 나타낸다. 실험 결과는 속도가 높은 이동 객체가 많을수록 상대적으로 좋은 성능 향상을 보인다. 이는 속도가 높을수록 보다 많은 확률로 노드 갱신이 단말노드에서 완료되지 않음을 의미한다. 기존 방식의 경우 높은 속도를 가진 이동 객체가 여러 단말노드를 갱신하고 이를 위해 노드 탐색을 함께 동반하는 것에 의해 노드 접근수가 빠르게 증가할 수 있는 반면, 제안된 방식에서는 메모리에 위치한 안정화 노드로 이런 갱신 및 노드 접근이 상당 부분 제한되기 때문에 성능 향상을 기대할 수 있다. 또한 트리의 하단 계층에 존재하는 CBR의 확장속도를 불필요하게 증가시키지 않기 때문에 노드 갱신 연산 시 불필요하게 접근되는 노드 수를 줄일 수 있다.

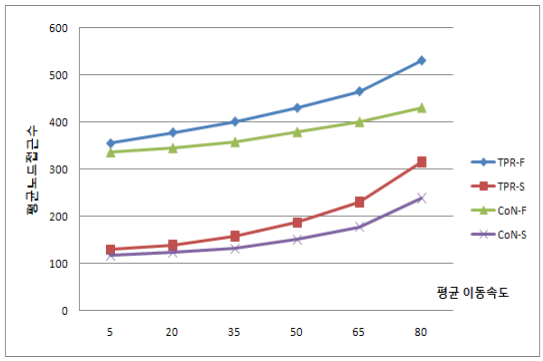

그림 5. 이동속도 증가에 따른 성능 변화
Fig. 5. Performance w.r.t the varying velocity

그림 6은 갱신 속도에 따른 노드 접근 회수를 실험한 결과이다. 그래프에서 CoN는 제안한 방식을, TPR은 TPR*-트리 방식에 따른 결과를 보인다. 뒤에 F로 표시된 그래프는 객체들의 평균 이동 속도가 40인 경우를, S로 표시한 그래프는 객체들의 평균 이동 속도가 10인 경우를 보인다. 트리 갱신 비용은 앞서와 같이 갱신 연산 동안 접근되는 노드 수(읽기 및 수정을 위한 노드 접근)를 측정한 값이다. 실험 결과에서 보이듯이 전체 갱신 속도 영역에서 좋은 성능을 보임을 알 수 있다. 실험된 범위에서는 최대 23% 정도의 성능 향상을 보였으며, 갱신 비도가 높은 상황에서 보다 좋은 성능 향상을 기대할 수 있었다. 하지만 앞의 실험에서 보이듯이 이동 속도에 비해서는 그 영향이 다소 적었다. 이를 통해 이동 속도가 빠름으로 인해서 발생하는 추가적인 노드 갱신 비용이 큼을 알 수 있다. 인식률이 떨어지는 현상도 볼 수 있었다.

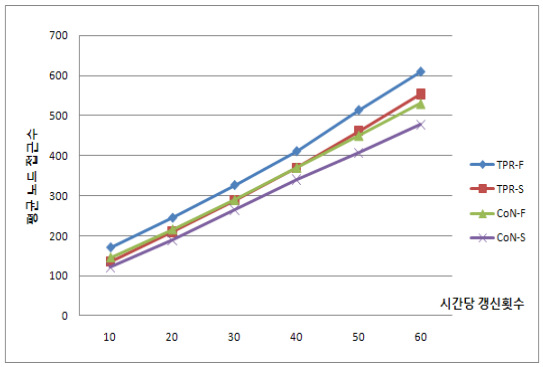

그림 6. 갱신 속도 증가에 따른 성능 변화
Fig. 6. Performance w.r.t the varying update speed

V. 결 론

본 논문에서는 짧은 시간 안에 여러 개의 CBR을 거쳐 이동하는 이동 객체가 발생시킬 수 있는 지나친 노드 갱신 비용을 줄일 수 있는 방법을 연구했다. 연구된 방식은 루트노드의 자식노드 단을 안정화 노드로 사용함으로써 단말노드에서 표현할 수 있는 색인 공간보다 매우 큰 공간에 빠른 속도의 이동 객체를 머물게 함으로써 갱신 작업이 메모리에 존재하는 안정화 노드에서 일어나도록 한다. 이런 안정화 노드 사용으로 추가적인 공간 사용이 요구되지만 루트노드의 자식노드 개수가 전체 노드 개수에 비해 극히 일부임을 감안한다면 그 비용은 무시할 수준이라고 할 수 있다.

향후 연구과제로 이동 객체 이동을 시뮬레이션 하는 기법을 좀 더 현실성 있게 개선할 필요가 있다. 본 연구에 사용한 GSTD는 매우 정형화된 이동 형태만을 표현할 수 있기 때문에 제안된 기법의 장점이 적절히 평가되고 있지 못한 점이 있다. 따라서 시뮬레이션 기법을 좀 더 정교화 시킬 수 있도록 하는 추가 연구가 요구된다. 또한, 이를 바탕으로 현재 알고리즘의 여러 성능에 영향을 미치는 요소들, 예를 들어 안정화 노드의 크기, 안정화 노드에 수용되는 이동 객체의 조건 등에 대한 추가 연구가 필요하다.

참고문헌

〔1〕 O. Wolfson, B. Xu, S. Chamberlain, and L. Jiang, "Moving Objects Databases: Issues and Solutions," In Proc. Int'l. Conf. on Scientific and Statistical Database Management(SSDBM), pp. 111-122, 1998.

〔2〕 A. P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao, "Modeling and Querying Moving Objects," In Proc. IEEE Int'l Conf. on Data Engineering (ICDE), pp. 422-432, 1997.

〔3〕 D. Pfoser, C. S. Jensen, and Y. Theodoridis, "Novel Approaches in Query Processing for Moving Objects," In Proc. Int'l. Conf. on Very Large Data Bases (VLDB), pp. 395-406, 2000.

〔4〕 S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez, "Indexing the Positions of Continuously Moving Objects," In Proc. ACM Int'l. Conf. on Management of Data(ACM SIGMOD), pp. 331-342, 2000.

〔5〕 M. F. Mokbel, T. M. Ghanem, and W. G. Aref, "Spatio-Temporal Access Methods," Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, Vol. 26, No. 2, pp. 40-49, 2003.

〔6〕 권동섭, 이상준, 이석호, "R-트리에서 빈번한 변경 질의 처리를 위한 효율적인 기법," 한국정보과학회 논문지: 데이타베이스, 제 31권, 제 3호, 261-273쪽, 2004년 6월.

〔7〕 Kyriakos Mouratidis, Marios Hadjieleftheriou, Dimitris Papadias, "Conceptual Partitioning: An Efficient Method for Continuous Nearest Neighbor Monitoring," In Proc. of SIGMOD, pp. 634-645, 2005.

〔8〕 Christian S. Jensen , Dan Lin , Beng Chin Ooi, "Query and Update Efficient B+-Tree Based Indexing of Moving Objects," In Proc. of VLDB, pp. 768-779, 2004.

〔9〕 S. Prabhakar, Y. Xia, D. V. Kalashnikov, W. G. Aref, and S. E. Hambrusch, "Query Indexing and Velocity Constrained Indexing: Scalable Techniques for Continuous Queries on Moving Objects," IEEE Trans. on Computers, Vol. 51, No. 10, pp. 1124-1140, 2002.

〔10〕 Mohamed F. Mokbel, Xiaopeng Xiong, Walid G. Aref, "SINA: Scalable Incremental Processing of Continuous Queries in Spatio-temporal Databases," In Proc. of SIGMOD, pp. 623-634, 2004.

〔11〕 김상욱, 장민희, 임성채, "능동적 재조정: TPR*-트리의 검색 성능 개선 방안," 정보처리학회 논문지 D, Vol. 15-D, No. 4, 451-462쪽, 2008년 4월.

〔12〕 Sang-Wook Kim, Min-Hee Jang, and Sungchae Lim, "Adjustment: An Approach for Improving the Performance of the TPR*-Tree," In Proc. Database and Expert Systems Applications (DEXA), pp. 834-843, 2007.

〔13〕 Y. Tao, D. Papadias, and J. Sun, "The TPR*-Tree: An Optimized Spatio-Temporal Access Method for Predictive Queries," In Proc. Int'l. Conf. on Very Large Data Bases (VLDB), pp. 790-801, 2003.

〔14〕 Antonin Guttman, "R-trees: a dynamic index structure for spatial searching," In Proc. of ACM SIGMOD, pp. 606-615, 1989.
〔15〕 N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles," In Proc. ACM Int'l. Conf. on Management of Data(ACM SIGMOD), pp. 322-331, 1990.
〔16〕 Y. Theodoridis, R. Silva, and M. Nascimento, "On the Generation of Spatiotemporal Datasets," In Proc. Int'l. Symp. on Spatial Databases, pp. 147-164, 1999.
〔17〕 Mario A. Nascimento, Dieter Pfoser, Yannis Theodoridis, "Synthetic and Real Spatiotemporal Datasets," IEEE Data Eng. Bull. 26(2), pp. 26-32, 2003.

저 자 소 개

임 성 채

1994 : KAIST 공학석사.
2003 : KAIST 공학박사.
2000 - 2005 : 코리아 와이즈넛 R&D수석연구원 및 이사
2005 - 현재 : 동덕여자대학교 컴퓨터학과 조교수
관심분야 : 이동통신 응용, 시맨틱 웹, 데이터마이닝