이동객체 데이터베이스에서의 밀집 영역 연속 탐색☆

Continuous Discovery of Dense Regions in the Database of Moving Objects

이 영 구* 김 원 영**
Young-Koo Lee Won-Young Kim

요 약

일상 생활에서 널리 사용 중인 핸드폰, PDA 등 소형 이동장치들의 밀집된 영역들을 구하는 것은 매우 중요한 문제들 중의 하나로서, 군대의 집결, 차량 이동의 모니터링 등 다양한 분야에 사용될 수 있다. 본 논문에서는 대량의 이동객체 데이터베이스 상에서 밀집 영역 탐색을 연속적으로 수행하기 위한 새로운 알고리즘을 제안한다. 본 논문에서는 이동객체들이 전원 절약 등의 이유로 자신들의 위치를 주기적으로 서버에 보고하는 대신, 기대되는 위치로부터 멀리 떨어지게 되는 경우에만 새로운 위치를 보고하는 환경을 가정한다. 이러한 경우 서버에서 관리되는 이동객체 위치는 정확하게 지정할 수 없고 확률적으로 표시되는데, 대량의 이동객체에 대해 확률적인 분포를 고려하여 밀집 영역을 찾기 위해서는 커다란 비용이 요구된다. 본 논문에서는 근접한 위치에 있는 이동객체들을 하나의 그룹으로 묶고, 동일한 그룹에 속한 이동객체들은 동일하게 취급함으로써 계산의 복잡도를 줄인다. 최종 결과에서 밀집 영역 판단이 모호해지는 경우에만 개별 이동객체들이 자세히 조사된다. 여러 데이터 집합들을 대상으로 다양한 실험을 수행하여 제안된 알고리즘의 우수성을 보이고, 민감성 및 확장성 분석 결과를 제시한다.

Abstract

Small mobile devices have become commonplace in our everyday life, from cellular phones to PDAs. Discovering dense regions for the mobile devices is one of the problems of grate practical importance. It can be used in monitoring movement of vehicles, concentration of troops, etc. In this paper, we propose a novel algorithm on continuously clustering a large set of mobile objects. We assume that a mobile object reports its position only if it is too far away from the expected position and thus the location data received may be imprecise. To compute the location of each individual object could be costly especially when the number of objects is large. To reduce the complexity of the computation, we want to first cluster objects that are in proximity into a group and treat the members in a group indistinguishable. Each individual object will be examined only when the inaccuracy causes ambiguity in the final results. We conduct extensive experiments on various data sets and analyze the sensitivity and scalability of our algorithms.

☞ KEYWORD : 이동 객체, 밀집 영역, 연속 질의

1. 서 론

이동객체들은 PDA같은 휴대용 장치들로부터 센서 같은 임베디드 장치까지 일상 생활에서 널리 사용되고 있다. 가까운 미래에 이러한 이동장치들은 자신의 위치를 파악하는 장치를 내장할 것으로 예상된다. 예를 들면, GPS 기능을 탑재한 휴대폰들이 점차 늘어나고 있다. 이동장치에는 세 가지 특성이 있다.

- 저전력 : 이동장치의 배터리 에너지는 매우 제한적이므로, 에너지 소모량은 모바일 컴퓨팅에

* 종신회원 : 경희대학교 컴퓨터공학과 조교수
yklee@khu.ac.kr
** 정 회 원 : 한국전자통신연구원 선임연구원
wykim@etri.re.kr
[2008/04/01 투고 - 200/04/03 심사 - 2008/04/22 심사완료]
☆ 이 연구는 2004년도 경희대학교 연구비지원에 의한 결과임. (KHU-20040454)

서 가장 큰 관심사이다.

- 제한된 컴퓨팅 자원 : 대부분의 이동장치에서 메모리, 디스크, 대역폭 등의 크기는 데스크탑 PC 등과 비교할 때 작은 수치이다.
- 비동기식 통신 : 일반적으로 서버는 강력한 기계장치이고, 상대적으로 높은 대역폭으로 먼 거리에 메시지를 송신할 수 있다. 반면에 이동장치는 소형 기계장치이고, 작은 대역폭으로 단지 가까운 수신자에게 메시지를 보낼 수 있다.

종종 이동장치가 새로운 정보를 필요로 하는 경우가 발생한다. 예를 들면, 사용자가 자신이 현재 위치한 지역의 지도를 이용하고 싶지만 그 지도가 이동장치에 저장되지 않은 경우가 있다. 이 경우에 필요한 정보를 검색하기 위해 두 가지 방법을 생각해 볼 수 있다. 첫번째 방법은 서버로부터 얻는 것으로서, 서버가 이동장치로 정보를 전달하기 위해서 푸쉬-기반과 풀-기반의 두 가지 방법이 널리 사용되고 있다[13]. 푸쉬-기반의 경우 방송의 주기가 길어지면 대기 시간이 길어지는 문제가 있다. 반면에 풀-기반은 이동장치가 메시지를 서버로 보낼 수 없으면 (서버가 너무 멀리 있는 경우) 사용이 불가능하다. 이동장치가 정보를 검색하는 두번째 방법은 다른 이동장치와 정보를 공유하는 것이다. 정보의 전파와 공유는 이동 컴퓨팅에서 중요하다. 이동장치가 근처의 다른 장치로부터 필요한 데이터를 획득할 확률을 높이기 위해서는, 다른 이동장치들이 많이 밀집해 있는 지역으로 이동해야 한다. 밀집 영역들의 추적을 유지하는 서버가 있다면 정보 획득에 큰 도움이 될 것이다. 따라서, 밀집 영역들의 식별은 중요한 문제이다. 이동장치 또는 이동객체들은 동적으로 움직이므로, 밀집 지역 역시 시간에 따라 변하게 된다.

지금까지 클러스터링에 관한 많은 연구가 오랫동안 폭넓게 수행되어 왔다[6]. 초기의 연구는 대부분 정적 데이터를 대상으로 한 것이었으며, 동적 또는 스트림 데이터에 대한 연구도 수년 전부터 시작되었다 [5, 12]. 그런데, 기존의 공간 데이터 클러스터링 알고리즘들은 이동객체 위치의 부정확성, 밀집영역의 형태, 대량 이동객체의 이동 다양성 등의 이동객체 환경 특징을 제대로 반영하지 못해 직접 적용하기가 쉽지 않다. 이러한 이동객체 환경의 특징과 이를 고려한 본 논문의 접근 방법은 다음과 같다.

첫째, 에너지 소모로 인해 각 이동장치가 자신의 위치를 자주 보고하는 요청이 불가능 할 수 있다. 이 경우, 어떤 순간에 이동객체의 정확한 위치를 알지 못할 수도 있다. 이동객체들의 전력 소모를 줄이기 위해, 장치가 전송하는 메시지들을 최소화해야 한다. 만약 이동장치가 서버에 자신의 현재 위치를 계속 알리고 있으면, 매우 비효율적일 것이다. 결과적으로, 이 논문에서는 확률 모델 [16]이 적용된다. 최종적으로 알려진(보고된) 위치와 객체의 이동 속도, 방향을 바탕으로 계산된 기대되는 위치가 있다. 만약 객체의 실제 위치와 기대되는 위치의 차이가 어떤 임계값 U보다 작으면, 객체는 자신의 위치를 보고할 필요가 없다. 그렇지 않으면, 서버에 위치를 보고할 것이다. 본 논문에서는 이동객체의 실제 위치가 기대되는 위치를 평균값으로 하는 정규 분포를 따른다고 가정한다.

둘째, 밀집 영역은 임의의 형태가 될 수 있으며, 정확한 밀집 영역 계산은 매우 어려울 것이다. 그리고, 앞에서 언급한 응용에서는 밀집 영역의 정확한 형태를 알 필요가 없다. 또한, 확률 모델을 사용하기 때문에, 주어진 시간에 객체의 위치는 명확하지 않으므로, 밀집 영역의 정확한 형태의 계산은 유용하지 않을 것이다. 따라서, 본 논문에서는 그리드(grid) 방법을 사용한다. 서버가 관리하는 전체 영역은 작은 그리드들로 분할된다. 만약 그리드 내에서 객체들의 기대되는 수가 어떤 임계값 이상이면, 그것을 *밀집 그리드*라고 한다. 이제 문제는 밀집 그리드들의 집합을 연속적으로 식별하는 것이다.

셋째, 서버에서 관리하는 객체들의 개수가 많고 객체들이 각기 다른 방향과 속도로 움직일 수도 있다. 매 순간에 각각의 객체에 대한 확률 분포를 계산하는 것은 많이 시간이 소요된다. 이 시간을 줄이기 위하여, 본 논문에서는 객체들을 공간적 근접성에 따라 분할하는 *코로케이션 그룹*(collocation group)방법을 제안한다. 코로케이션 그룹은 시간적으로 한 순간에 인접한 객체들의 집합을 말한다. 인접한 객체들을 그룹으로 묶음으로써, 커다란 부정확성을 야기하지 않고 그룹 내 객체들을 하나처럼 다룰 수 있다. 코로케이션 그룹에 속한 객체들은 동일 위치에 배치된 객체들의 집합처럼 간주되는 것이다. 결과적으로, 하나의 코로케이션 그룹이 k개의 객체들을 포함한다고 했을 때, k개의 객체들이라기 보다는 단지 하나의 객체에 대한 확률분포만 계산하게 됨으로 수행시간을 단축시킨다. 그러나, 밀집 그리드 식별 프로세스가 수행될 때마다 그룹화 작업을 수행해야 한다. 매번 그룹화 작업을 반복하는 오버헤드를 줄이기 위해 *코무브먼트 그룹*(co-movement group)을 사용한다. 코무브먼트 그룹의 개념은 고속도로에서 자동차들이 오랫동안 함께 이동하는 것처럼, 인접한 객체들이 상당 시간 비슷한 속도로 함께 움직인다는 고찰에서 고안되었다. 이런 타입의 그룹멤버들은 오랜 기간 동안 멤버쉽이 바뀌지 않을 것이다. 따라서, 이 객체들은 일정 시간 동안 다시 클러스터 할 필요가 없다. 코무브먼트 그룹은 일정 시간 동안 가깝게 있을 객체들로 구성하고, 그 그룹의 멤버들은 그 기간 동안 반복해서 클러스터 되지 않을 것이다. "코로케이션 그룹"과 "코무브먼트 그룹"의 차이점은 코로케이션 그룹은 지금 현재 가까이 있는 객체들을 포함하고 있는 반면, 코무브먼트 그룹은 긴 기간 동안 가까이 있을 객체들을 포함하는 것이다. 모든 객체들 개개마다 확률 분포를 계산할 필요가 없고, 단지 어느 그룹에도 속하지 않은 객체들과 그룹들에 대해서만 계산을 한다. 코무브먼트와 코로케이션 근사법으로 인하여 결과는 100% 정확하지 않게 될 것이다. 그러나, 이 결과에 대한 에러 한계를 계산할 수 있다. 그러므로, 하나의 그리드 내에 있을 것으로 기대되는 객체들의 수는 하나의 값이 아니라, 범위로 표현된다. 만약 그 범위가 밀집 영역을 판단하는 기준인 임계값을 포함하지 않으면, 그 그리드가 밀집인지 아닌지 안전하게 결론지을 수 있다. 밀집 그리드인지 안전하게 파악할 수 있는 그리드에 대해서만 자세히 조사하면 된다. 이 세 번째 기법을 *밀집도 정제*(density refinement)라고 한다.

논문의 구성은 다음과 같다. 제 2절에서는 문제를 정형적으로 정의한다. 제 3절에서는 기초적인 방법인 NaïveCluster를 기술하고, 제 4절에서는 제안하는 방법인 MobiCluster를 논의한다. 제 5절에서는 실험 및 분석 결과를 제시하고, 제 6절에서는 관련연구를 설명한다. 마지막으로, 제 7절에서는 결론을 내린다.

2. 문제 정의

본 논문에서는 이동장치에 대한밀집 영역들을 식별하는 문제를 논한다. 서버와 이동장치들의 집합이 있고, 이동장치들은 GPS 등을 이용하여 자신의 위치를 알 수 있다고 가정한다. 그리고 이동장치들은 비주기적으로 그들의 위치를 무선 네트워크를 통하여 서버에 보고하고, 서버가 마지막 스냅샷 후에 수신한 정보를 기초로 이동장치의 현재 위치 스냅샷을 생성한다고 가정한다. 이동객체는 자신의 이전 위치, 속도, 방향들로부터 현재 위치가 유도될 수 있을 때는, 오랜 시간 동안 자신의 현재 위치를 서버에게 통보하지 않을 것이다.

서버가 이동장치의 위치를 예측하기 위해 다양한 방법을 사용할 수 있으나, 본질적으로, 그 방법들은 대부분 확률적 방법을 사용한다는 공통점을 갖는다. 즉, 이동장치의 위치는 영역에 대한 확률 분포를 따른다. 일반적으로, 참고문헌 [16]에 제안된 방법이 널리 사용되고 있다. 주어진 장치

o에 대하여, *last_loc*를 객체의 최종적으로 알려진 위치라고 하자. 그리고, v는 객체의 속도 벡터라고 하자. o의 예상된 현재위치(*predict_loc*)는 다음과 같이 정의한다.

$$predict\_loc = last\_loc + v * t \quad (1)$$

여기서, t는 o의 최종적으로 알려진 위치로부터 경과된 시간이다. 이것은 단지 예측이므로 정확한 o의 위치는 예상된 위치와 차이가 있을 것이다. o의 정확한 위치는 k차원의 다변량정규분포 $N_k(\mu, \Sigma)$를 따른다. 여기서 k는 공간의 차원과 같고, 평균 μ = *predict_loc*, 그리고 Σ는 분산-공분산 행렬이다. 분산-공분산 행렬은 주변분포 분산[7]과 같은 대각원소들 (diagonal elements) σ^2을 가지고 있는 대칭적인 $k*k$ 행렬이다. σ는 $(1/c)U$로 정의된다. 여기서 U는 객체의 불확실성(uncertainty)이고, c는 상수이다. 파라미터 U와 c를 할당하는 여러 가지 방법들이 있다. U는 상수이거나, 경과시간 t 또는 이동 거리 기대치(expected traversed distance) d에 대한 함수가 될 수 있다. 본 논문에서는, 데이터베이스의 모든 객체들이 같은 불확실성을 갖게하기 위하여 U를 상수로 가정한다. 이 가정은 각 객체에 대한 불확실성을 찾기 어려울 때 자주 사용되어 왔다. c는 네트워크 신뢰성 등에 의존적인 상수이다. c가 커질수록 확률은 더 높아진다. 예를 들면, 이동장치가 μ로부터 U만큼의 거리에 있을 확률은 c = 1, 2, 3에 대하여 각각 확률 0.68, 0.95, 0.997이 된다.

서버가 관리하는 공간은 작은 그리드들로 분할된다. (일반성에 영향을 주지 않고, 설명을 단순화하기 위하여 그리드는 이차원 공간이라고 가정한다. 이 논문의 모든 결과는 삼차원 공간으로 쉽게 일반화 될 수 있다.) 서버는 공간에서 밀집된 그리드들을 발견하려고 시도한다. 만약 그리드 내의 장치의 (기대되는) 수가 일정한 임계값 τ 이상이면, *밀집 그리드*(dense grid)라고 한다. 그렇지 않으면, *성긴 그리드*(sparse grid)라고 한다. 그리드에서 이동장치의 기대되는 수는 스냅샷 이후에 변할 것이다. 요약하면, 본 논문에서는, 임계값 z가 주어졌을 때, 각각의 스냅샷 이후에 밀집 그리드들을 식별하는 프로세스를 계속적으로 반복한다.

3. NaïveCluster: 초보 알고리즘

제안하는 알고리즘을 설명하기 전에, 본 절에서는 밀집 그리드 식별을 위한 초보적인 알고리즘으로 NaïveCluster를 설명한다. 이 방법은 밀집 영역의 계산이 필요할 때마다 각 그리드에서 객체 수의 기대치를 계산한다. 먼저 그리드에서 객체 수의 기대치를 계산하는 방법을 설명한 후, NaïveCluster 알고리즘을 기술한다.

각 그리드에서 객체 수의 기대치는 해당 그리드 영역에 대한 각 객체의 존재 기대치들의 합으로 계산된다. 여기서, 각 객체의 존재 기대치를 *참여도*(contribution)라고 정의한다. 한 객체의 참여도는 그리드 영역에 대한 객체의 확률 밀도 함수의 적분으로써 계산된다. 수식 (2)는 그리드 G에서 객체 수의 기대치 N_G를 계산하는 방법을 보여준다. 여기서, $f_i(x,y)$는 객체 "O_i"의 확률 밀도함수이다. 그리고, *region*(G)는 그리드 G의 영역이다.

$$N_G = \sum_{o_i} \int_{region(G)} f_i(x, y)\,dxdy \quad (2)$$

본 논문에서 사용하는 모델에서, $f_i(x,y)$는 이변량(bivariate) 정규 분포가 되는데, 이변량 정규 분포에 대한 확률 적분 계산은 비용이 매우 큰 연산이다. 그런데, 모바일 환경에서는, 이변량 정규 분포는 분포의 중심에 가까운 그리드에 대한 적분들은 큰 값을 갖는 반면에, 멀리 떨어진 그리드에 대한 적분들은 매우 작다는 특징이 있다. 정규 분포가 종(bell) 모양을 갖음에 주목하기 바란다. 이 특징을 이용하여, 참고문헌 [8]에서 기술된 것처럼, 정확성은 거의 손상 없이 계산 시간을 크게 줄일 수 있다. 그리드에서 멀리 떨어진 객체들의

적분은 무시하는 반면에 가까운 객체들의 적분은 정확히 계산하는 것이다. 이 경우, 계산에서 발생한 에러가 아주 작은 범위에 있음을 보장할 수 있다. 객체 수의 기대치에 대한 근사값을 정의하기 위하여, 함수 *near*(*G*)를 *인접 객체*(near object)들의 집합이라고 정의한다. 여기서, 인접 객체란 $d(G,O_i) = \sigma_{near}$를 만족하는 객체로서, $d(G,O_i)$는 O_i의 확률 분포의 중심과 *region*(*G*) 내의 가장 가까운 점과의 거리이고 σ_{near}는 인접 객체를 정의하기 위해 사용하는 임계치(threshold)이다. 그리드 *G*에 대한 객체 수의 기대치의 근사값 AN_G는 다음과 같다.

$$AN_G = \sum_{o_i \in near(G)} \iint_{region(G)} f_i(x,y)\,dxdy \qquad (3)$$

수식 (2)의 실제 값 대신에 이 근사값을 사용할 때 발생하는 에러의 상한(upper bound)은 참고문헌 [8]에 나와 있다.

이제 알고리즘 1에 보인 NaïveCluster 알고리즘을 설명한다. 입력은 이동객체들의 데이터베이스 *MDB*, 밀도 임계치 τ, 그리고 보고 주기(reporting period) t_p이다. 이동 데이터 베이스에서는 각 객체들에 대하여 예상 위치 계산을 위하여, 제 2절에서 설명한 대로 (velocity, last_time, last_location)으로 된 레코드를 유지한다고 가정한다. 보고 주기는 밀집 그리드들의 식별 작업이 수행되는 시간 간격이다.

NaïveCluster는 매 식별 작업 때마다 다음의 세 단계를 반복한다. (1) 현재 시점에서 각 그리드에 대한 객체 수의 기대치를 계산한다(Line 2). (2) 각 그리드가 밀집 영역인지를 검사(객체 수의 기대치가 τ보다 작지 않으면 된다)하고, 밀집 영역

(알고리즘 1) NaïveCluster: 밀집 영역의 식별.

```
Input:
    database of mobile objects MDB
    density threshold τ
    reporting period t_p
Output:
    the set of dense regions
Method:
 1:  while true do
 2:      call ComputeExpectedNumObj(MDB, current-time);
 3:      Result = ∅;
 4:      for each grid g_j do
 5:          if (the expected number of g_j) ≥ τ then
 6:              Result = Result ∪ {g_j};
 7:          end if
 8:      end for
 9:      Output Result;
10:      sleep(t_p);
11:  end while
Procedure ComputeExpectedNumObj(MDB, t_cur)
    for each o_i in the database MDB do
        compute the predicted location at time t_cur by plugging the o_i.velocity, o_i.last_time, and o_i.last_location into Equation (1);
        for each near grid g_j in near(o_i) do
            compute the expected number objects contributed by o_i to g_j using the object probability density function;
        end for
    end for
```

인 경우 *Results*에 삽입한다(Lines 3-9). (3) 밀집 영역의 다음 식별 작업이 있을 때까지 수면(sleep)에 들어간다(Line10). 단계 (1)에서, 각 그리드에 대한 객체 수의 기대치 계산은 ComputeExpectedNumObj 프로시져에 의해 수행된다. ComputeExpectedNumObj 프로시져는 데이터베이스에서 각 객체 O_i에 대하여, O_i가 인접 그리드들에 대해주는 영향을 계산한다. 먼저 제 2절에서 설명한 수식 (1)을 이용하여 O_i의 예상되는 위치를 계산한다. 그 후에, O_i의 각 인접 그리드에 대하여, O_i에 의해 참여되는 객체 수의 기대치를 계산한다. 매 식별 작업 때마다 각 그리드에 대한 모든 객체들의 참여도(contribution)를 계산하기 때문에 NaïveCluster 알고리즘은 수행 시간이 길어지는 단점을 갖는다. 다음 절에서는, 객체들의 이동 지역성(mobile locality)의 이점을 이용하여 수행 시간을 효과적으로 줄이는 방법을 기술한다.

4. MobiCluster: 고급 알고리즘

본 절에서는 코무브먼트, 코로케이션, 및 밀집도 정제에 기반한 MobiCluster라는 새로운 알고리즘을 기술한다. MobiCluster는 이동객체들을 함께 움직이는 객체들의 그룹으로 나눈다. MobiCluster는 주어진 그리드에 대한 인접객체 수의 기대치를 계산할 때, 객체들의 그룹을 단위로 취급한다. 따라서, 모든 객체들의 확률 밀도 함수의 영향을 개별적으로 계산한 후 합산하는 NaïveCluster 알고리즘에 비교하여, MobiCluster 알고리즘은 계산 오버헤드를 줄일 수 있다.

MobiCluster 알고리즘은 (1) 코무브먼트 그룹의 식별, (2) 코무브먼트 그룹의 영향 계산, (3) 밀집도 정제의 세 단계로 구성된다. MobiCluster는 먼저 코무브먼트 객체들의 그룹들을 식별하고, 그 그룹 내의 모든 멤버들을 통합하여 하나의 객체처럼 취급한다. 다음으로, 코무브먼트 그룹에 속하지 않는 객체들을 코로케이션 그룹으로 묶는다. 코로케이션 그룹은 주어진 시점에서 서로 가까이 있는 객체들로 구성된다. (코무브먼트 그룹과 코로케이션 그룹의 차이점은 코로케이션 그룹은 가까운 위치만 요구하는 반면에, 코무브먼트 그룹은 가까운 위치와 함께 비슷한 속도도 요구한다는 것이다.) 다음으로, MobiCluster는 각 그리드에 대하여 그룹이 미치는 기대치 참여(contribution)와 어떤 그룹에도 속하지 않는 개별 객체의 기대치 참여를 계산한다. 기대치 참여는 범위로 표현된다. 따라서, 하나의 그리드내의 객체 수의 기대치도 역시 범위로 표현된다. 마지막으로, 밀집 영역인지의 판별이 모호한 그리드를 대상으로 개별 객체들 레벨에서 정밀 조사함으로써 모든 밀집 그리드들을 찾는다.

4.1. 코무브먼트 그룹의 식별

먼저 코무브먼트의 개념을 정의하고, 다음으로 코무브먼트 그룹의 식별 방법을 논의한다. 코무브먼트 객체들을 정의하기 위하여 "속도"와 "위치"라는 연관성이 있는 서로 다른 두 개의 개념이 사용될 수 있다. 속도에 기반하게 되면, 서로 인접한 위치에서 출발하여 비슷한 속도를 가지고 비슷한 방향으로 움직이는 객체들을 코무브먼트 객체들로 정의할 수 있다. 반면에, 위치에 기반하게 되면, 주어진 기간 동안 임의의 두 객체 간의 거리가 항상 어떤 임계값 내에 있는 객체들을 코무브먼트 객체들로 정의할 수 있다. 본 논문에서는, 위치 기반의 정의를 사용한다. 다음에서 논의하는 바와 같이, 위치 기반 정의는 밀집 영역을 찾는데 사용될 수 있는 유용한 성질들을 갖는다.

정의 **1.** *(코로케이션 객체)* $O=\{o_1, o_2, \cdots, o_n\}$를 이동객체들의 집합이라고 하자. 그리고, B를 $d * d$ 의 박스(box)라고 하자. 여기서 d는 주어진 거리 임계치이다. 객체들이 시간 t에서 주어진 박스 B 범위 내에 있으면, o_i들을 시간 t에서의 코로케이션 객체라고 정의한다.

정의 **2.** *(코무브먼트 객체, 코무브먼트 그룹)*

$O=\{o_1, o_2, \cdots, o_n\}$를 이동객체들의 집합이라고 하자. 그리고, B를 주어진 거리의 임계값 $d * d$의 박스라고 하자. 만약 객체 o_i들이시간 t_b에서 t_e까지 계속해서 코로케이션 객체들이면, o_i들을 시간 t_b부터 t_e까지 *코무브먼트 객체*들이라고 하고, *CO-MOVE(O, [t_b, t_e])*로 표현한다. 또한 객체들의 집합 O를 *코무브먼트 그룹*이라 한다.

예제 1. 그림 1은 코무브먼트 객체들의 예를 보인다. 객체들 *O1, O2, O3, O4* 는 주어진 거리 임계값 박스 B 이내로 유지되며 움직이기 때문에, 시간 t_2부터 t_4까지 코무브먼트 객체이다. 시간 t_5에서, *O4*는 B의 범위에서 벗어난다. 그러므로, 시간 t_2부터 t_5*까지*는 *O4*를 제외한 *O1, O2, O3* 만이 코무브먼트 객체들이다.

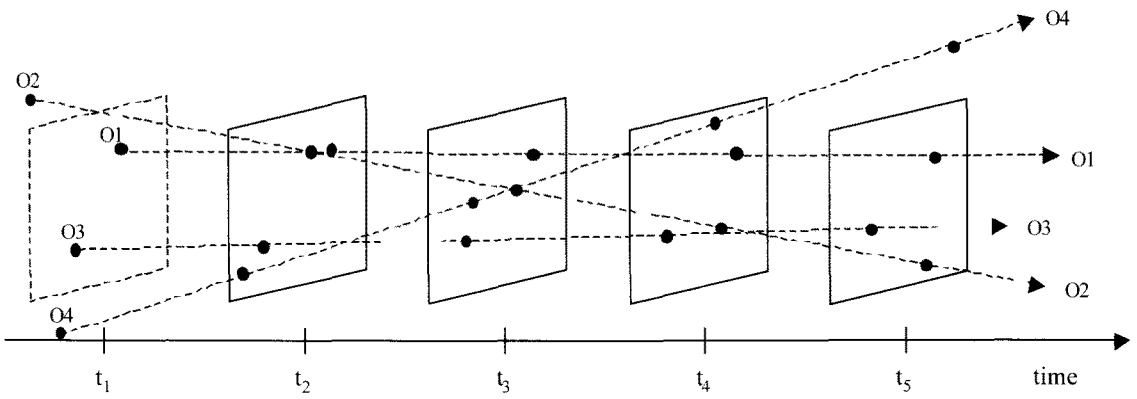
코무브먼트 객체들은 아래 보조정리 1과 같은 특징을 가지며, 이 특징은 코무브먼트 객체들을 식별하기 위해 사용될 수 있다. 이 보조정리는 제 4절에서 설명한 위치예측 모델이 선형 방정식이라는 사실에 기초를 두고 있다.

보조정리 1. $O=\{o_1, o_2, \cdots, o_n\}$ 를 이동객체들의 집합이라고 하자. 그리고 B를 거리 임계값으로 주어진 박스라고 하자. 만약, o_i들이 시점 t_s와 t_e에서 코로케이션 객체들이면, o_i들은 시간 t_b와 t_e 사이에서 코무브먼트 객체들이다(*CO-MOVE(O, [t_b, t_e])*).

이제 보조정리 1을 이용하여 코무브먼트 객체를 식별하는 방법을 서술한다.

정의 **3.** (*시간도약(time leap)*) 시간도약 t_l은 코무브먼트의 범위로 사용될 수 있는 시간의 범위이다.

보조정리 1에 따라, 어떤 주어진 시간 t_c에서 주어진 시간도약 t_l에 대한 코무브먼트 그룹을 식별하는 문제는, 시점 t_c와 $t_c + t_l$에서의 코로케이션 객체들의 집합을 찾는 문제로 바꿀 수 있다. 어떤 시점에서의 코로케이션 객체들의 집합을 찾는 것은 클러스터링 문제(clustering problem)이다. 정적 객체들(static objects)의 환경에서 클러스터들을 찾는 연구는 광범위하게 제안되어 왔었다. 그런데, 본 연구에서는 모든 코로케이션 객체들의 집합을 완벽하게 찾을 필요가 없다는 것에 주목하기 바란다. 여기서는 코무브먼트 그룹을 이용하여 밀집 그리드들을 식별할 때 발생하는 계산의 오버헤드를 줄이는 것을 목표로 하기 때문에, 본 연구에서는 다른 기법에 비하여 적은 계산 오버헤드를 가지고 클러스터를 찾을 수 있는 그리드 기반 클러스터링 기법을 사용한다. 그리드 기반 클러스터링 방법은 시간 복잡도가 객체들 개수에 선형적으로 비례한다는 장점이 있다.



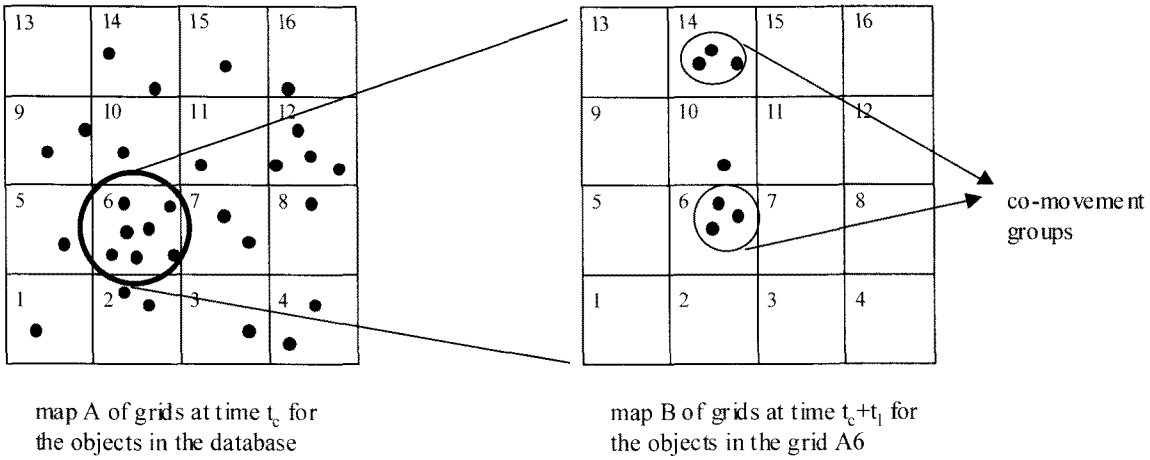
(그림 1) 코무브먼트 객체들의 예.

(알고리즘 2.) FindComovementGroup: 코무브먼트 그룹의 식별.

Function FindComovementGroup
Input:
database of mobile objects *MDB*
group size threshold S_g
box representing the distance threshold *B*
current time t_c
time leap t_l
Output:
the co-movement groups
Method:
1: Divide the data space grids with the same edge-length with *B*
2: Hash objects in *MDB* into the map *A* of the grids based on the location at t_c
3: Let *LargeGridSet* be the grids whose number of objects contained is no less than S_g
4: **for** each grid a_j in *LargeGridSet* **do**
5: Hash the objects in a_j into the map *B* of grids based on the location at $t_c + t_l$
6: Let *ComovementGridSet* be the grids in *B* whose number of objects contained is no less than S_g
7: **for** each grid b_j in *ComovementGridSet* **do**
8: Add to *Results*a group consisting of the objects in b_j
9: **end for**
10: **end for**
11: Output *Results*

이동객체들이 존재하는 데이터 공간은 에지(edge) 길이 *d*의 그리드들로 분할된다. 여기서 *d*는 박스 B의 에지 길이이다. 해쉬 테이블을 사용하면, 이동객체들을 한번 스캔함으로써, 객체들의 예상 위치에 따라 현재 시간 t_c에서 각 그리드 내에 있는 객체들을 식별할 수 있다. 그룹 크기 임계값 S_g 이상의 객체 수를 포함하는 각 그리드에 대하여, 이 그리드에 포함된 객체들을 시점 $t_c + t_l$에서의 (예측된) 위치를 기반하여 그리드들의 맵으로 한번 더 분할한다. 만약 시점 $t_c + t_l$ 에서의 어떤 그리드가 S_g 이상의 객체들을 포함한다면, 이 그리드에 포함된 객체들의 집합은 시간 t_c에서부터 $t_c + t_l$까지의 코무브먼트 그룹이 된다. 다음 알고리즘 2의 FindComovementGroup은 지금 설명한 내용을 의사코드로 나타낸 것이다.

예제 2. 그림 2는 코무브먼트 객체들을 식별하는 기법의 동작을 예시한 것이다. 이 예에서, 데이터공간은 4*4 크기로, 그룹 사이즈 임계값 S_g는 3으로 가정한다. 그림 왼쪽의 그리드 맵 A에서와 같이, 먼저 데이터베이스의 이동객체들을 현재 시점 t_c에서의 (예측) 위치에 따라 그리드 맵 상으로 분할한다. A6과 A12는 적어도 S_g 만큼의 멤버들을 포함하는 그리드들이다. 이제, A6 그리드에 속한 객체들을 시점 $t_c + t_l$에서의 위치를 기반으로 하여 그림 오른쪽에 있는 그리드 맵 B 상으로 분할한다. 이 때, B6과 B14는 적어도 S_g만큼의 멤버들을 포함하고 있는 그리드들로 남는다. 따라서, B6내의 객체들은 코무브먼트 그룹을 형성하고, B14내의 객체들은 또 다른 코무브먼트 그룹을 형성한다. 비슷한 방식으로 A14의 코무브먼트 그룹들을 식별한다.



(그림 2.) 코무브먼트 그룹을 식별하는 예 (그룹 크기 임계값 S_g=3).

코무브먼트 객체들을 식별하는 작업은 최초 식별 작업 때부터 시작하여 매 시간도약 때마다 수행된다. 그런데, 코무브먼트 그룹을 식별하는 간격이 보고주기 보다 짧게 되면 성능 향상에 도움이 되지 않기 때문에, 시간도약을 보고주기의 배수로 설정하는 것이 바람직하다. 시간도약 값은 시작할 때 고정될 수도 있고, 응용의 특성에 따라 동적으로 조절될 수도 있다. 본 논문에서는, 초기에 보고주기의 두 배로 설정한다. 만약 코무브먼트 그룹의 개수가 임계값보다 크면, 다음 처리 시점에 시간도약을 두 배로 증가시킨다. 반면에, 코무브먼트 그룹의 개수가 임계값보다 작으면, 시간도약을 반으로 줄인다.

만일 시간도약의 기간이 길다면, 코무브먼트 그룹을 식별하는 작업을 주어진 시간도약의 중간 시점에 수행할 수도 있다. 이렇게 하면, 중간 시점부터 형성된 코무브먼트 객체들을 놓치지 않게 된다. 중간 시점에 처리할 때는, 어떤 코무브먼트 그룹에도 속하지 않은 객체들만을 고려한다.

이동객체들은 예측위치로부터 벗어날 때 새로운 위치를 보고하므로, 코무브먼트 그룹에 속한 객체들의 수는 시간에 따라 변하게 된다. 객체가 예측위치와 다른, 자신의 실제 위치를 보고할 때, 만일 그 실제 위치가 객체가 속한 코무브먼트 그룹의 영역을 벗어나게 되면 객체는 그룹에서 제거된다. 그룹으로부터 제거된 객체는 다른 그룹의 새로운 멤버가 될 수 있거나, 어떤 그룹에도 속하지 않는 다른 객체들과 새로운 그룹을 형성할 수 있다. 그러나, 제거된 객체를 새로운 그룹에 포함시키는 작업은 너무 많은 오버헤드를 요구하기 때문에, 이 작업을 수행하지 않는다. 따라서, 코무브먼트 그룹내의 객체 수는 시간에 따라 작아질 수 있다. 만약 코무브먼트 그룹의 객체 수가 그룹 크기 임계값보다 작게 되면, 그 그룹을 분해한다.

4.2. 객체 수의 기대치 계산

4.2.1. 코무브먼트 그룹에 대한 계산

코무브먼트 객체들의 그룹을 식별한 후, 각 그리드에서 그룹들의 참여도(그룹에 속한 객체들의 참여도의 합)를 계산한다. 다중 객체 밀도함수 자체 또는 그 함수들의 그리드 영역에 대한 적분들의 합은 닫힌 공식(closed formula)으로 표현될 수 없기 때문에, 그리드에 대한 그룹의 참여도 계산은 그룹에 속한 객체들의 참여도 계산과 같은 오버헤드를 갖는다. 따라서, 본 논문에서는 그리드에서 그룹의 참여도로 하나의 값을 계산하기보다

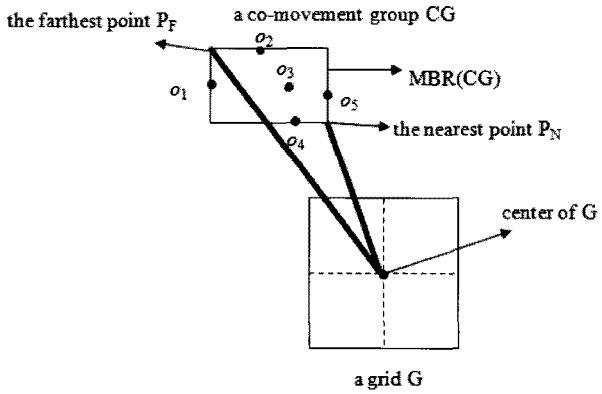
는, 참여도의 상한과 하한으로 구성된 범위를 계산한다. 객체 밀도 함수가 이변량 정규 분포를 가지므로, 이 함수는 분포의 중심(=객체의 예측위치)에 가까울수록 높은 값을 갖는다. 따라서, 그룹 내의 모든 객체의 예측위치들이 주어진 그리드에 가장 가까운 지점일 때, 그리드에 대한 코무브먼트 그룹 CG의 참여도는 최대이다. 그 지점은 MBR(CG) 내에 위치한다. 여기서, MBR(CG)은 코무브먼트 그룹 CG의 모든 객체들을 포함하는 최소 포함 사각형으로 좌하점(left-bottom point)과 우상점(right-top point)의 쌍으로 표시한다. 하한은 가장 먼 지점을 이용하여 유사하게 계산된다.

예제 **3.** 그림 3은 그리드 G에 대한 코무브먼트 그룹 CG의 참여도가 어떻게 계산되는지 보여준다. 코무브먼트 그룹 CG가 다섯 개의 객체 $O1$, $O2$, $O3$, $O4$, $O5$로 구성되어 있다고 하자. 이 예제에서, MBR(CG)의 오른쪽 하단 코너 P_N이 G의 중심에 가장 가까운 지점이고, MBR(CG)의 왼쪽 상단 코너 P_F가 G의 중심으로부터 가장 먼 지점이다. CG의 정확한 참여도는 $\sum_{i=1}^{5} \iint_{region(G)} f_i(x,y)\,dxdy$로 계산되고, 이것은 다섯 번의 적분 계산을 요구한다. 반면에, 하한과 상한은 P_F와 P_N을 이용하여, 각각 $5\times \iint_{region(G)} f_F(x,y)\,dxdy$ 과 $5\times \iint_{region(G)} f_N(x,y)\,dxdy$으로 계산된다. 이것은 단지 두 번의 적분 계산을 요구한다.

정의 4는 위에서 설명한 내용을 정형적으로 기술한다.

정의 **4.** (*그룹 참여도의 상한과 하한*) m개의 객체 o_1, o_2, ..., o_m와 $N(\mu_i,\sigma)$로 구성되는 코무브먼트 그룹 CG를 객체 $o_i(1\le i\le m)$에 대한 객체 밀도 함수라고 하자. 그때, 그리드 G에 대한 그룹 CG의 참여도의 상한선은 $|CG| \times N(\mu_i,\sigma)$로 정의된다. 여기서, μ_i는 G의 중심에 대해 MBR(CG)내의 가장 가까운 지점이다. 반면에, 그리드 G에 대한 그룹 CG의 참여도의 하한선은 $|CG| \times N(\mu_u,\sigma)$로 정의된다. 여기서 μ_u는 G의 중심으로부터 MBR(CG) 내의 가장 먼 지점이다.

객체로부터 그리드까지의 거리를 계산할 때, 그리드의 중심점을 사용한다. 그 이유는 코무브먼트 그룹의 그리드에 대한 참여도는 그룹에 속한 모든 객체의 예측위치가 그리드의 중심점에 가장



$$\text{정확한 값} = \sum_{i=1}^{5} \iint_{region(G)} f_i(x,y)\,dxdy$$

$$\text{값의 하한} = 5\times \iint_{region(G)} f_F(x,y)\,dxdy$$

$$\text{값의 상한} = 5\times \iint_{region(G)} f_N(x,y)\,dxdy$$

(그림 3) 코무브먼트 그룹이 그리드에 미치는 객체 수의 기대치 계산 예.

가까울 때 최대값을 갖고, 그리드의 중심위치에서 가장 멀 때 최소값을 갖기 때문이다.

4.2.2. 개별 객체에 대한 계산

앞 절에서는 코무브먼트 그룹의 참여도를 계산하는 방법을 설명하였다. 이제 어떤 그룹에도 속하지 않은 개별 객체들의 참여도를 계산하는 방법을 설명한다. NaïveCluster 알고리즘은 모든 (인접) 객체들의 적분의 합으로 각 그리드에서의 객체 수의 기대치를 계산하였다. 그러나, MobiCluster는 *코로케이션 그룹* 기법을 이용함으로써, 개별 객체의 참여도를 계산하는 시간을 줄인다. 주요 아이디어는 객체들을 코로케이션에 기반한 그룹들로 나눈 후, 각 객체들이 아니라, 이 그룹들의 참여도를 계산하는 것이다. 코로케이션 객체들을 식별할 때, 코무브먼트 객체들을 식별하기 위해 사용한 동일한 크기의 박스를 사용한다. 코로케이션 객체들의 그룹이 식별되면, 그리드에 대한 이 그룹들의 참여도는 코무브먼트 그룹의 참여도를 계산할 때와 유사한 방식으로 계산될 수 있다.

4.3. 밀집도 정제

이전 단계들의 결과로서, 각 그리드의 객체 수의 기대치는 정확한 값 대신에 하한 값과 상한 값으로 표시된 범위를 갖는다. 이 경계 값들과 밀도 임계값 τ 사이의 관계에 따라서, 그리드들은 세가지 클래스로 분류될 수 있다.

1. 밀집 그리드: $lower\_bound \geq \tau$
2. 성긴 그리드: $upper\_bound \leq \tau$
3. 결정되지 않은 그리드: $lower\_bound < \tau \geq upper\_bound$

첫 두 클래스는 각각 $exact\_value \leq lower\_bound$ 와 $exact_value \geq upper_bound$인 경우로서, 밀집 그리드인지 성긴 그리드인지 간단하게 결정할 수 있다. 여기서, *exact_value*는 대응되는 그리드에서 객체 수의 기대치이다. 이 두 클래스들은 더 이상 정제를 할 필요가 없다. 반면에, 세 번째 클래스는 τ와 *exact_value* 사이의 관계를 정확히 유도할 수 없으므로, 밀집 그리드인지의 여부를 결정할 수 없다. 이 클래스에 속한 그리드들은 밀집 영역이 될 수도 있고 성긴 영역이 될 수 있다. 따라서, 세 번째 클래스에 속한 그리드들은 정제를 해야 한다.

그리드에 대한 *exact_value*를 계산하기 위해, 그리드에 영향을 주는 그룹들을 풀어서, 그 그룹에 속한 각 객체의 참여도들의 합을 구하면 된다. 그러나, 한 번에 *exact_value*를 계산하는 것보다는 점차적으로 상한과 하한의 차이를 줄이는 것이 계산 오버헤드를 줄일 수 있다. 따라서, 그룹들을 하나씩 처리해 가면서, 상한과 하한의 차이를 점차적으로 좁혀가고, 그리드가 밀집되었는지, 아닌지를 결정할 수 있을 때, 남아있는 그룹들을 더 이상 처리하지 않고 계산을 종료한다.

이 방법은 그리드에 영향을 주는 그룹들의 처리 순서와 관계없이 단순히 점차적으로 범위를 좁혀갈 뿐이다. 그러나 처리된 그룹들의 순서는 성능에 영향을 미친다. 범위를 빨리 좁히게 되면, 처리해야 하는 그룹의 수가 줄어들 것이다. 이를 위하여, 본 논문에서는 그리드에 영향을 주는 그룹들 중에서 상한과 하한의 차이가 가장 큰 그룹을 우선적으로 선택한다. 그 이유는, 하나의 그룹을 처리할 때, 그리드의 객체 수 기대치의 하한과 상한 사이의 차이가 그룹이 미치는 영향의 하한과 상한의 차이만큼 감소하게 되기 때문이다.

5. 성능 평가

본 절에서는 MobiCluster 알고리즘의 성능 평가 분석 결과를 제시한다. 먼저 제 5.1절에서 실험 환경을 기술한 후, 제 5.2절에서 실험 결과를 논의한다.

5.1. 실험 환경

실험은 512MB 주기억장치가 장착된 2.2GHz Pentium IV PC 에 Windows 2000을 설치하여 수행하였다. 알고리즘들은 Visual C++로 구현하였다. 성능 비교를 위하여, MobiCluster 알고리즘을 NaïveCluster 알고리즘과 비교하였다.

워크로드 모델: 이동객체들에 대한 실제 데이터셋을 수집하는데 어려움이 있기 때문에, 성능 평가는 합성적으로 생성된 워크로드에 기반한다. 본 논문에서는 참고문헌 [14]의 워크로드 모델을 채택하되, 클러스터의 개념을 적용하기 위해 수정하여 사용하였다. 워크로드의 생성을 위해, 1,000,000 × 1,000,000 평방미터의 공간에서 N개의 객체가 이동하는 것을 시뮬레이션한다. 주어진 공간은 일 킬로미터의 에지 길이를 갖는 1,000 × 1,000개의 그리드들로 분할된다. 밀도 임계값 τ는 객체들의 총 개수 N에 대한 비율로 주어진다. τ는 0.1%부터 1.0%까지 변화한다. 객체들이 여러 클러스터들을 형성하고 있는 실제 데이터 모습을 시뮬레이션하기 위해, NC개의 다변량 정규 분포를 중첩시키는 데이터 분포를 사용한다. 다변량 정규분포의 각 축(axis)은 $N(\mu_i,\sigma^2_c)$의 정규분포를 갖는다. 여기서, 평균 μ는 공간 영역 내에서 랜덤하게 선택되고, 표준편차 σ_c는 2,000이다. σ_c 값은 이 분포를 따르는 46.6%(=약 50%)의 객체들이 4×4 그리드내에 있고 91.1%의 객체들이 8×8 그리드 내에 있게 하도록 선택되었다. 각 클러스터들은 초기에 N/NC의 객체들을 갖는다. 본 워크로드에서는 시뮬레이션 기간 동안, 객체가 사라지거나 새로운 객체가 생성되지 않는다고 가정한다.

이제, 워크로드 모델에서 객체들의 이동에 대해 기술한다. 각 객체는 자신의 클러스터 내에서, 또는 다른 클러스터로 이동할 수 있다. 객체들의 전체 집합은 두 개의 그룹으로 분할된다. 하나는 이동중인 객체들이고 다른 하나는 정지된 객체들이다. 각 클러스터들은 $RS*(N/NC)$ 개수만큼의 정지된 객체들의 수를 갖는다. 매 시간 단위 시점에, 만약 어떤 클러스터가 더 많은 정지된 객체들을 갖는다면, 클러스터의 정지된 객체들의 수를 만족시키도록 정지된 객체들을 무작위로 선택하여 이동하게 한다. 각 선택된 정지된 객체들에 대하여, 목적지와 속도를 할당한다. 먼저 목적지는 다음과 같이 결정된다. (1) 목적지 클러스터는 NC 클러스터들 중에 무작위로 선택된다. (2) 목적지 클러스터 내에서의 위치는 클러스터의 다변량 정규 분포에 의해서 할당된다. 만약 목적지 클러스터가 객체가 원래 속한 클러스터와 같다면, 객체는 같은 클러스터 내에서 이동하는 것을 나타낸다. 다음으로, 속도는 0.75, 1.5, 3 Km/min의 최대 속도를 갖는 세 객체들의 그룹 중 하나에 같은 비율로 할당한다. 전체 이동 경로는 동일 길이의 여섯 개의 부분경로로 나누어진다. 이중에서, 첫 번째 부분경로동안, 속도를 0에서 최대까지 등가속으로 끌어올리고, 중간의 네 부분경로 동안에 최대속도로 이동한다. 마지막 여섯 번째 부분경로 동안에 등감속으로 속도를 0으로 줄인다. 목적지에 도착한 이동객체는 목적지 클러스터의 멤버가 된다.

객체의 정확한 위치가 $N(\mu,\sigma_0)$의 객체 밀도 함수를 따른다고 가정한다. 여기서 σ_0는 100이다. 객체들이 예측위치로부터 σ_0보다 더 멀리 이탈할 때, 그들의 새로운 위치를 서버에 보고한다고 가정한다. 실험에서, 보고 주기 t_P를 30분으로 설정한다. 이것은 30분마다 식별된 밀집그리드를 보고하는 것을 의미한다.

실험 방법: 위에서 기술한 워크로드 모델에 따라 합성 데이터를 생성하여 다음과 같이 다양한 실험을 수행하였다. 성능 평가 지수로는 실행 시간을 사용한다.

1) 실험 **A:** 정지객체 비율의 영향을 분석하기 위해 비율을 20%에서 80%까지 변화시키며 실험한다.

2) 실험 **B:** 클러스터 수의 영향을 분석하기 위해, 총 객체 수는 고정하고, 클러스터 수를 30에서 100까지 변화시키며 분석된다.

3) 실험 **C:** 밀도 임계값과 관련된 민감도 분석을 위하여, 그 값을 0.1%에서 1.0%까지 변화시키며 실험한다.

4) 실험 **D:** MobiCluster 알고리즘의 확장성을 테스트하기 위하여, 객체의 수를 100,000부터 1,000,000까지 변화시키며 실험한다.

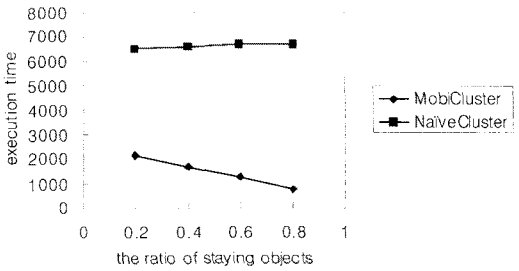
표 1은 실험에서 그 값이 변경되는 워크로드 생성 파라메터들을 보여준다. 표에서 굵은 글씨로 표시된 값들은 디폴트 값들이다.

(표 1) 실험에서 사용되는 파라메터들.

파라메터	설명	값
N	총 객체 개수	100,000, **500,000**, 1,000,000
NC	클러스터 개수	30, **50**, 70, 100
RS	정지객체 비율	20%, 40%, **60%**, 80%
τ	밀도 임계값	**0.05%**, 0.1%, **0.5%**, 1.0%
t_P	보고주기 (분)	**30**

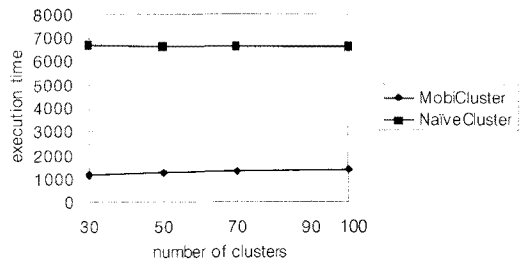
5.2. 실험 결과

그림 4는 실험 A의 결과를 보여준다. 가로축은 정지 객체들의 비율을, 세로축은 실행 시간을 나타낸다. 실험 결과들은 모든 정지객체 비율에 대하여 Mobicluster가 NaïveCluster보다 성능이 우수함을 보인다. 예를 들면, 비율이 80%일 때, MobiCluster가 NaïveCluster보다 약 9배 빠르다. 또한, 이 결과는 NaïveCluster의 실행시간이 변화가 거의 없는 것에 반해, MobiCluster의 실행시간은 비율이 증가함에 따라 감소하는 것을 보인다. 정지 객체들의 비율이 증가할수록, 더 많은 객체들이 클러스터들에 머물게 된다. 즉, 이동객체들이 감소하는 것이다. NaïveCluster 알고리즘이 이동객체들의 수를 고려하지 않는 반면에, MobiCluster 알고리즘은 객체별 계산을 정지객체들을 그룹화하여 계산함으로써 더 나은 성능을 얻는다.



(그림 4) 정지객체 비율의 변화에 따른 실행시간.

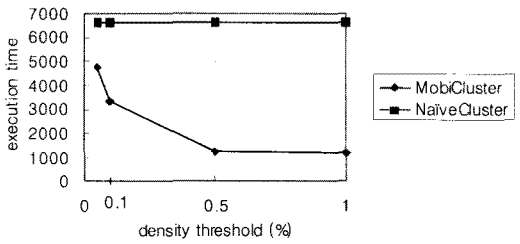
그림 5는 실험 B의 결과를 나타낸다. 실험 B는 클러스터들의 수 *NC*의 영향에 대해 분석한다. 그림 5에서 가로축은 클러스터들의 수이다. 실험결과에 따르면 클러스터들의 수에 관계없이 항상 MobiCluster 알고리즘이 NaïveCluster 알고리즘 보다 더 나은 성능을 갖는 것을 보인다. 그림 5는 클러스터들의 수가 증가할수록 실행시간이 상승하는 것을 보인다. 그 이유는 다음과 같다. 첫째, 클러스터들의 수가 많아질수록, 처음 시작할 때부터 객체들은 더 흩어진 상태에 있게 된다. 둘째, 객체들이 이동하고 있을 경우에, 객체들이 이동하는 경로의 수가 *NC*(*NC*-1)이므로, 객체들은 더 흩



(그림 5) 클러스터 수 변화에 따른 실행시간.

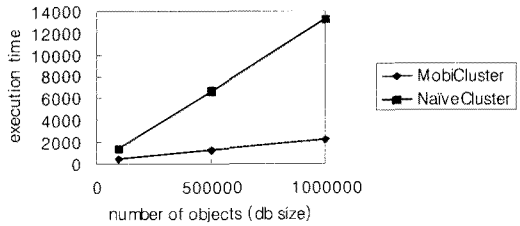
어지게 된다. (각 클러스터가 이변량 정규분포를 따르기 때문에, 경로가 선형 라인이 아니고, 폭이 넓은 도로와 같다는 것에 유의한다.) 그러므로, 클러스터들의 수가 증가할수록, 코무브먼트 그룹의 수는 점점 많아지고, 그룹내의 객체들의 수는 점점 적어진다. 이것은 실행시간의 증가를 만든다.

그림 6은 밀도 임계값에 따른 변화를 분석하는 실험 C의 결과를 나타낸다. 가로축은 밀도 임계값이다. 결과들은 MobiCluster의 실행시간이 밀도 임계값이 증가함에 따라 감소하는 것을 보여준다. 그러나, 여전히 MobiCluster가 밀도 임계값 전체에 걸쳐서 NaïveCluster 보다 좋은 성능을 가진다. 밀도 임계값을 제외한 다른 파라미터들을 고정시켰기 때문에, 식별된 코무브먼트 그룹들은 밀도 임계값과 관계없이 같다. 낮은 밀도 임계값에서 실행시간이 큰 이유는 밀집영역으로 구분이 애매한 그리드의 수가 증가하기 때문이다. 따라서, Mobicluster는 밀집도 정제 단계에서 많은 시간을 소비한다.



(그림 6) 밀도 임계값 변화에 따른 실행시간.

그림 7은 데이터베이스 크기에 따른 알고리즘의 확장성을 분석하기 위한 실험 D의 결과를 나타낸다. 가로축은 데이터베이스 크기로서, 데이터베이스에 존재하는 객체들의 개수로 표현된다. 실험결과는 NaïveCluster와 MobiCluser 모두 데이터베이스 크기에 대해 선형인 것을 보여준다. 그러나, 그림 7에서와 같이 MobiCluster 알고리즘은 NaïveCluster 알고리즘보다 매우 빠르다. 이것은 MobiCluster가 대용량 이동 데이터베이스에 실제적으로 사용될 수 있음을 나타낸다.



(그림 7) 데이터베이스 크기 변화에 따른 실행시간.

6. 관련 연구

모바일 컴퓨팅의 응용들의 등장 이후, 모바일 객체 관리에 관한 많은 연구가 수행되어 왔다. 이들 중에서, 모바일 객체를 대상으로 한 공간 클러스터링, 공간 질의, 연속적인 공간 질의 등의 연구가 본 논문과 밀접한 관계가 있다. 본 절에서는 이들 연구들에 대해서 토의한다.

6.1. 공간 클러스터링

최근에 공간 클러스터링[6]은 상당한 주목을 받아왔고, 많은 알고리즘들이 개발되었다. 대부분의 연구들은 정적 데이터들에서 정확한 클러스터들을 찾는 것이 목적이었다. 스트림 데이터를 대상으로 한 공간 클러스터링 알고리즘은 참고문헌 [12]에서 제안하였다. 이 논문에서, 저자들은 객체들이 연속적으로 새로 추가되고, 객체들의 위치는 알려져 있다고 가정한 후, 이 객체들을 점진적 기법으로 클러스터링하는 방법을 사용한다. 이 논문에서 연구된 시나리오는 본 논문과는 다르다. 본 논문에서는, 객체들의 수가 변하지 않는 대신에, 객체들의 위치가 시간에 따라 변한다. 따라서, 참고문헌 [12]의 방법은 본 문제를 해결하는데 직접적으로 적용하기 어렵다.

6.2. 이동객체에 대한 공간 질의

Saltenis의 알고리즘[14]은 이동객체들에 대한 범위 질의들을 해결하는 새로운 인덱싱 구조 TPR-트리를 제안하였다. TPR-트리는 최소 포함 사각형(minimum bounding box)을 저장하는 대신에, 시간을 고려한 포함 사각형(time-dependent bounding box)을 저장한다. 시간을 고려한 포함 사각형은 주어진 시간 주기 동안 모든 이동객체들을 포함하는 것을 보장하도록 정해진다.

참고문헌 [16]에서, 저자들은 이동객체들의 위치를 정하는 새로운 방법을 제안하였다. 각 객체는 시간에 따른 예측위치를 갖는다. 예측위치는 객체의 최종 보고 위치, 속도, 방향을 기반으로 계산된다. 객체는 예측위치로부터 일정한 거리 U 내에 있으면, 현재 위치를 보고하지 않는다. 객체들이 자신의 위치를 오랜 기간 동안 보고하지 않게 되어, 정확한 위치가 알려지지 않으므로, 객체의 위치는, 예측위치를 기대값으로 갖는 정규 분포를 따른다고 가정한다. 이 모델을 기반으로, 저자들은 참고문헌 [3]에서, 부정확 데이터를 대상으로 하는 최근접 질의 처리를 위한 새로운 방법을 제안하였다.

6.3. 연속 공간 질의

연속 공간 질의는 최근에 상당한 주목을 받고 있다. 참고문헌 [15]에서는 연속 KNN 질의 처리 방법이 제안되었다. 이 논문에서 저자들은 주어진 이동객체에 대하여, 임의의 주어진 순간에 K개의 최근접 객체들을 찾는 문제를 다룬다. 여기서, 데이터셋들은 정적 데이터이고, 질의 점인 이동객체는 매 순간 변하게 된다. Iwerks 등은 연속 윈도우를 사용하는 효율적인 KNN 알고리즘을 제안하였다[9]. 이 논문에서는, 대부분의 시간 동안, 최근접 객체들이 질의 지점으로부터 일정 범위 내에 있다고 가정하고, 이 범위에 속한 모든 객체를 공간 영역 질의로 구한다. 질의 결과에 포함된 객체 수가 K 이상이라면, K개의 최근접 객체들은 이 객체들의 질의 결과로부터 구할 수 있게 된다.

6.4. 이동 객체 클러스터링

클러스터링 연구가 폭 넓게 진행되어 왔음에도 이동장치의 밀집 지역을 찾아내는 일은 상대적으로 주의를 받지 못했다. 참고문헌 [17]에서는 히스토그램에 기반한 이동 객체 클러스터링 방법을 제안했으나, 이 방법은 갱신이 자주 일어나는 이동 환경에서 매번 히스토그램을 변경하는 문제점이 있다. 참고문헌 [11]에서는 클러스터를 과거 이동 경로가 비슷한 이동 객체로 정의하고 이를 찾는 방법을 제안하였는데, 비슷한 이동 경로를 구할 때 비용이 크다는 단점이 있다. 참고문헌 [10]에서는 이동 객체의 향후 움직임과 갱신 주기 기대값 등을 고려한 효율적인 점진적 클러스터링 방법을 제안하였다. 그런데 제안된 방법들은 이동 객체의 위치를 추정하기 위한 예측 모델을 사용함에도 불구하고 그 위치에 대한 확률 모델을 제대로 고려하지 못하였다. 반면에 본 논문에서는 확률적 모델에 기반한 밀집 영역 연속 식별 알고리즘을 새롭게 제안하였다.

7. 결론

본 논문에서는 이동객체에 대한 밀집영역 식별 문제를 논의하였다. 에너지 절약 정책으로 인해, 많은 환경에서 이동객체의 정확한 위치를 유지하기가 어렵다. 따라서, 객체의 위치 정보들이 정확하지 않게 된다. 이러한 환경에서 밀집 영역을 효율적으로 식별하기 위하여, 코무브먼트 그룹, 코로케이션 그룹, 밀집도 정제라는 세 개의 새로운 기법들을 제안하였다. 이 기법들의 주된 목적은 객체 개별 단위의 정확한 정보보다는 그룹 단위의 개략적인(coarse) 정보를 사용하여 계산 시간을 줄이고, 개략적인 정보가 결과에서 모호성을 일으키는 경우에만, 모호성을 제거한 정확한 정보를

사용하는 것이다. 본 논문에서는 다양한 실험을 통하여 이 기법들을 사용한 MobiCluster 알고리즘의 성능이 우수함을 보였다.

참 고 문 헌

[1] P. Castro, B. Greenstein, R. Muntz, C. Bisdikian, P. Kermani, and M. Papadopouli. Locating application data across service discovery domains. *In Proceedings of SIGMOBILE*, 2001.

[2] S. Chandrasekaran and M. J. Franklin. Streaming queries over streaming data. *In Proceedings of VLDB*, 2002.

[3] R. Cheng, D. Kalashnikov, and S. Prabhakar. Evaluating Probabilistic Queries over Imprecise Data, *In Proceedings of SIGMOD*, 2003.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. 2nd ed., The MIT Press, 2001.

[5] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. *In Proceedings of FOCS*, 2000.

[6] J. Han and M. Kamber. *Data Mining Concepts and Techniques*, 2001.

[7] A. J. Hayter. *Probability and Statistics for Engineers and Scientists*, 2nd ed., 2002.

[8] A. Hinneburg and D. Keim. An Efficient Approach to Clustering in Large Multimedia Databases with Noise. *In Proceedings of KDD*, 1998.

[9] G.S. Iwerks, H. Samet, and K. Smith. Continuous k-nearest neighbor queries for continuously moving points with updates. *In Proceedings of VLDB*, Sep. 2003.

[10] P. Kalnis, N. Mamoulis, and S. Bakiras. On Discovering Moving Clusters in Spatio-Temporal Data. *In Proceedings of SSTD*, pp. 364 - 381, 2005.

[11] C.S. Jensen, D. Lin, and B.C. Ooi. Continuous Clustering of Moving Objects. *IEEE Trans. on Knowledge and Data Engineering*, pp. 1161 - 1174. Sept. 2007.

[12] L. O'Callaghan, N. Mishra,A. Meyerson, S. Guha, and R. Motwani. Streaming-data algorithms for high-quality clustering. *In Proceedings of ICDE*, 2002.

[13] E. Pitoura and G. Samaras. *Data Management for Mobile Computing*. Kluwer, 1999.

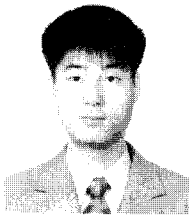
[14] S. Saltenis, C.S. Jensen, S.T. Leutenegger, and M.A. Lopez. Indexing the positions of continuously moving objects, *In Proceedings of SIGMOD*, pp. 331 - 342, Dalas, TX, May 2000.

[15] Y. Tao, J. Sun, and D. Papadias. Selectivity estimation for predictive spatio-temporal queries. *In Proceedings of ICDE*, pp. 417 - 428, Bangalore, India, Mar. 2003.

[16] O. Wolfson, P. Sistla, S. Chamberlain, and Y. Yesha. Updating and query databases that track mobile unites. *Distributed and Parallel Databases*, 7(3), 1999.

[17] Q. Zhang and X. Lin. Clustering Moving Objects for Spatio-Temporal Selectivity Estimation. *In Proceedings of ADC (Australian Database Conf.)*, pp. 123 - 130, 2004.

◐ 저 자 소 개 ◑



이 영 구 (YOUNG-KOO LEE)

1992년 한국과학기술원 전산학과 학사
1994년 한국과학기술원 전산학과 석사
2002년 한국과학기술원 전산학과 박사
2002. 3 ~ 2004. 2 한국과학기술원 박사 후 연구원
2002. 9 ~ 2004. 2 미국 University of Illinois at Urbana-Champaign, Postdoctoral Research Associate
2004. 3 ~ 현재 경희대학교 전자정보대학 조교수
관심분야: 유비쿼터스 데이터베이스, 데이터 마이닝, 데이터베이스 시스템, Bioinformatics
E-mail : yklee@khu.ac.kr

김 원 영 (WON-YOUNG KIM)

1989년 이화여자대학교 전산학과 학사
1991년 한국과학기술원 전산학과 석사
1998년 한국과학기술원 전산학과 박사
1999년 ~ 현재 한국전자통신연구원 선임연구원
관심분야: 데이터베이스 시스템, 소프트웨어 스트리밍, 데이터 마이닝
E-mail : wykim@etri.re.kr