

(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2020-0024074

(43) 공개일자 2020년03월06일

(51) 국제특허분류(Int. Cl.)

G06Q 50/00 (2018.01) G06F 16/29 (2019.01)

G06Q 50/10 (2012.01)

(52) CPC특허분류

G06Q 50/01 (2013.01)

G06F 16/29 (2019.01)

(21) 출원번호 10-2019-0072650

(22) 출원일자 2019년06월19일

심사청구일자 2019년06월19일

(30) 우선권주장

1020180100510 2018년08월27일 대한민국(KR)

(71) 출원인

고려대학교 산학협력단

서울특별시 성북구 안암로 145, 고려대학교 (안암동5가)

(72) 발명자

정연돈

서울특별시 중구 청계천로 400, 105-812(황학동, 롯데캐슬베네치아)

심창범

경상남도 창원시 마산합포구 서성로 43, 3동 1009호

김우일

경기도 과천시 별양로 42, 309동 1403호

(74) 대리인

김홍석

전체 청구항 수 : 총 9 항

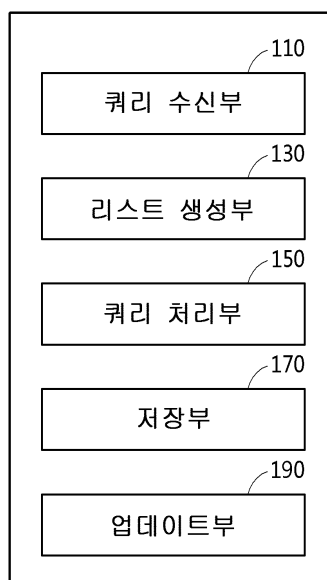
(54) 발명의 명칭 최근접 친구 검색 장치

(57) 요약

친구 검색 장치가 개시된다. 상기 친구 검색 장치는 셀들(cells)로 구성되고, 사용자들 각각을 나타내는 정점(vertex, $v \in V$)의 집합(V)과 상기 사용자들 사이의 친구 관계를 나타내는 엣지(edge, $e \in E$)의 집합(E)으로 구성된 비방향성 그래프(undirected graph, $G = \{V, E\}$)가 반영된 그리드(grid) 상에서 쿼리를 처리하고, 질(뒷면에 계속)

대표도 - 도18

100



의자(query issuer, u), 질의 지점(query point, p_q), 검색 결과의 개수(k , k 는 자연수), 및 친밀도(ℓ , ℓ 은 자연수)를 포함하고 상기 질의자(u)의 ℓ -홉 친구들 중 상기 질의 지점(p_q)으로부터 가장 가까운 k 명에 대한 검색 요청인 $k\ell$ -NCF 쿼리(k -Nearest ℓ -Close Friends query, k 와 ℓ 은 자연수)를 수신하는 쿼리 수신부, 상기 질의자(u)의 ℓ -홉 친구 리스트(V_u^ℓ)와 상기 질의자(u)의 ℓ -홉 친구가 포함된 셀들의 리스트인 ℓ -홉 개인 셀 리스트(P_u^ℓ)를 생성하는 리스트 생성부, 및 상기 ℓ -홉 친구 리스트(V_u^ℓ)와 상기 ℓ -홉 개인 셀 리스트(P_u^ℓ)를 이용하여 상기 $k\ell$ -NCF 쿼리에 대한 응답을 생성하는 쿼리 처리부를 포함한다.

(52) CPC특허분류

G06Q 50/10 (2013.01)

이 발명을 지원한 국가연구개발사업

과제고유번호 2018019262

부처명 과학기술정보통신부

연구관리전문기관 한국연구재단

연구사업명 (이공)중견연구자-핵심(후속)

연구과제명 CCTV 빅데이터 기반 지능형 범죄추적/예방 및 사회안전시스템

기 여 율 1/1

주관기관 고려대학교 산학협력단

연구기간 2018.03.01 ~ 2019.02.28

명세서

청구범위

청구항 1

셀들(cells)로 구성되고, 사용자들 각각을 나타내는 정점(vertex, $v \in V$)의 집합(V)과 상기 사용자들 사이의 친구 관계를 나타내는 �지(edge, $e \in E$)의 집합(E)으로 구성된 비방향성 그래프(undirected graph, $G = \{V, E\}$)가 반영된 그리드(grid) 상에서 쿼리를 처리하는 친구 검색 장치에 있어서,

질의자(query issuer, u), 질의 지점(query point, p_q), 검색 결과의 개수(k , k 는 자연수), 및 친밀도(ℓ , ℓ 은 자연수)를 포함하고 상기 질의자(u)의 ℓ -홉 친구들 중 상기 질의 지점(p_q)으로부터 가장 가까운 k 명에 대한 검색 요청인 $k\ell$ -NCF 쿼리(k -Nearest ℓ -Close Friends query, k 와 ℓ 은 자연수)를 수신하는 쿼리 수신부;

상기 질의자(u)의 ℓ -홉 친구 리스트(V_u^ℓ)와 상기 질의자(u)의 ℓ -홉 친구가 포함된 셀들의 리스트인 ℓ -홉 개인 셀 리스트(P_u^ℓ)를 생성하는 리스트 생성부; 및

상기 ℓ -홉 친구 리스트(V_u^ℓ)와 상기 ℓ -홉 개인 셀 리스트(P_u^ℓ)를 이용하여 상기 $k\ell$ -NCF 쿼리에 대한 응답을 생성하는 쿼리 처리부를 포함하는 친구 검색 장치.

청구항 2

제1항에 있어서,

상기 쿼리 처리부는 상기 그리드 상에서 상기 질의 지점(p_q)이 포함된 쿼리 셀(c_q)과 상기 쿼리 셀(c_q)를 둘러싸는 4 개의 직사각형들(rectangles)을 초기화된 힙(heap, $\mathcal{H}$)에 삽입하고,

상기 그리드 상의 직사각형은 위(up), 아래(down), 왼쪽(left) 또는 오른쪽(right)을 포함하는 방향(direction)과 해당 직사각형과 상기 쿼리 셀(c_q) 사이에 존재하는 직사각형의 개수인 레벨 넘버(level number)로 표현되고,

동일한 레벨 넘버를 갖는 직사각형들은 동일한 개수의 셀을 포함하는,

친구 검색 장치.

청구항 3

제2항에 있어서,

상기 힙($\mathcal{H}$)의 엔트리(entry)들은 상기 질의 지점(p_q)과의 최소 거리($mindist$)의 오름 차순으로 정렬되는,

친구 검색 장치.

청구항 4

제2항에 있어서,

상기 쿼리 처리부는 상기 힙($\mathcal{H}$)의 엔트리들 중 상기 질의 지점(p_q)과의 최소 거리가 가장 작은 엔트리를 디-힙(de-heap)하는,
친구 검색 장치.

청구항 5

제4항에 있어서,

상기 쿼리 처리부는,

디-힙된 엔트리(e)가 상기 ℓ -홉 개인 셀 리스트(P_u^ℓ)에 포함되는 셀이면, 디-힙된 엔트리(e) 내에 포함된 정점 중 상기 ℓ -홉 친구 리스트(V_u^ℓ)에 포함되는 정점을 결과 집합($\mathcal{R}$)에 삽입하고,

디-힙된 엔트리(e)가 직사각형이면, 상기 직사각형 내의 모든 셀들과 동일한 방향에서 다음 레벨의 직사각형을 상기 힙($\mathcal{H}$)에 삽입하는,

친구 검색 장치.

청구항 6

제5항에 있어서,

상기 쿼리 처리부는,

상기 힙($\mathcal{H}$)에 포함된 엔트리가 없거나 상기 결과 집합($\mathcal{R}$)의 k 번째 원소(component)와 상기 질의 지점(p_q) 사이의 최소 거리(kNN_dist)가 디-힙된 엔트리(e)와 상기 질의 지점(p_q)과의 최소 거리($mindist$) 보다 작아질 때까지,

상기 질의 지점(p_q)과의 최소 거리가 가장 작은 엔트리를 디-힙하는 동작, 상기 ℓ -홉 친구 리스트(V_u^ℓ)에 포함되는 정점을 결과 집합($\mathcal{R}$)에 삽입하는 동작, 및 상기 직사각형 내의 모든 셀들과 동일한 방향에서 다음 레벨의 직사각형을 상기 힙($\mathcal{H}$)에 삽입하는 동작을 반복적으로 수행하는,

친구 검색 장치.

청구항 7

제6항에 있어서,

상기 쿼리 처리부는 상기 결과 집합($\mathcal{R}$)을 상기 응답으로 출력하는,

친구 검색 장치.

청구항 8

제7항에 있어서,

상기 그리드에 포함된 복수의 셀들 중 적어도 하나의 정점을 포함하는 셀들 각각에 대응하는 매트릭스(matrix, M_c)를 생성하는 업데이트부를 더 포함하고,

상기 매트릭스(M_c)는 상기 그리드의 크기와 동일한 크기를 갖고,

상기 매트릭스(M_c)의 성분(element, $m_c^{c'}$)은 상기 매트릭스(M_c)에 대응하는 셀(c)과 임의의 셀(c') 내의 정점들 사이의 엣지(edge)의 개수를 나타내는,

친구 검색 장치.

청구항 9

제8항에 있어서,

상기 업데이트부는 임의의 사용자의 위치 변경 정보와 친구 관계 변동 정보 중 적어도 하나를 수신하고 수신된 정보에 대응하도록 상기 매트릭스를 업데이트하는,

친구 검색 장치.

발명의 설명

기술 분야

[0001] 본 발명은 최근접 친한 친구 검색을 요청하는 쿼리를 처리하는 장치와 방법에 관한 것이다.

배경 기술

[0003] 스마트 디바이스(smart device)의 급격한 보급으로 인하여, LBS(location-based services)가 주목받고 있다. 스마트 디바이스를 통해 사용자는 자신의 위치 정보를 위치 기반 소셜 네트워크(location-based social network), 즉 지오-소셜 네트워크(Geo-Social Network, GeoSN) 서비스(예; Foursquare, Facebook, Twitter, Yelp 등)에 활용할 수 있게 되었다. 예컨대, Facebook의 질의자들(query issuers)은 'Nearby Friends'라 불리는 기능을 사용하여 인접한 친구들(neighboring friends)을 검색할 수 있다. 또한, Foursquare의 사용자들은 체크인 정보(check-in information)를 그들의 친구들에게 리포트(report)하고, 서비스(Foursquare)는 그들의 친구 관계(friend relationships)를 향상시키기 위한 몇몇 장소를 추천한다.

[0004] 그럼에도 불구하고, 종래의 공간 질의 처리 기법들은 사회적 관계(social relationships)를 고려하지 않고 오직 위치 데이터(location data)만을 고려하기 때문에 활용도가 떨어진다. 일 예로, 서비스의 사용자가 인접한 친구들 또는 그들의 지인들(acquaintances)을 검색하고자 할 때, 서비스는 사용자의 요구를 효율적으로 만족시키지 못한다. 따라서, 사용자들은 순차적으로 그리고 직접 그들의 친구 리스트(friend lists)를 검토하여야 한다.

선행기술문헌

특허문헌

[0006] (특허문헌 0001) 대한민국 공개특허 제2014-0026305호 (2014.03.05. 공개)

(특허문헌 0002) 대한민국 등록특허 제1760600호 (2017.07.24. 공고)

비특허문헌

[0007] (비특허문헌 0001) K. Mouratidis, D. Papadias, M. Hadjieleftheriou, Conceptual partitioning: an efficient method for continuous nearest neighbor monitoring, in: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data, ACM, 2005, pp. 634-645.

(비특허문헌 0002) C. Shim, W. Kim, W. Heo, S. Yi, C.Y. Dohn, l-Close range friends query on social

grid index, IEICE Trans. Inf. Syst. 100 (4) (2017) 811-812.

발명의 내용

해결하려는 과제

[0008] 본 발명이 이루고자 하는 기술적인 과제는 최근접 친한 친구 검색 요청을 처리할 수 있는 장치 및 방법을 제공하는 것이다.

과제의 해결 수단

[0010] 본 발명의 일 실시예에 따른 친구 검색 장치는 셀들(cells)로 구성되고, 사용자들 각각을 나타내는 정점(vertex, $v \in V$)의 집합(V)과 상기 사용자들 사이의 친구 관계를 나타내는 엣지(edge, $e \in E$)의 집합(E)으로 구성된 비방향성 그래프(undirected graph, $G = \{V, E\}$)가 반영된 그리드(grid) 상에서 쿼리를 처리하는 친구 검색 장치로써, 에 있어서, 질의자(query issuer, u), 질의 지점(query point, p_q), 검색 결과의 개수(k , k 는 자연수), 및 친밀도(ℓ , ℓ 은 자연수)를 포함하고 상기 질의자(u)의 ℓ -홉 친구들 중 상기 질의 지점(p_q)으로부터 가장 가까운 k 명에 대한 검색 요청인 $k\ell$ -NCF 쿼리(k -Nearest ℓ -Close Friends query, k 와 ℓ 은 자연수)를 수신하는 쿼리 수신부, 상기 질의자(u)의 ℓ -홉 친구 리스트(V_u^ℓ)와 상기 질의자(u)의 ℓ -홉 친구가 포함된 셀들의 리스트인 ℓ -홉 개인 셀 리스트(P_u^ℓ)를 생성하는 리스트 생성부, 및 상기 ℓ -홉 친구 리스트(V_u^ℓ)와 상기 ℓ -홉 개인 셀 리스트(P_u^ℓ)를 이용하여 상기 $k\ell$ -NCF 쿼리에 대한 응답을 생성하는 쿼리 처리부를 포함한다.

발명의 효과

[0012] 본 발명의 실시 예에 따른 친구 검색 장치 및 방법에 의할 경우, 시스템에 부하를 주지 않고, 질의 지점과 가장 근접한 친구를 검색할 수 있는 효과가 있다.

도면의 간단한 설명

[0014] 본 발명의 상세한 설명에서 인용되는 도면을 보다 충분히 이해하기 위하여 각 도면의 상세한 설명이 제공된다.

도 1은 본 발명이 적용 가능한 하우스 파티 시나리오를 설명하기 위한 도면이다.

도 2는 $k\ell$ -NCF 쿼리의 일 예를 설명하기 위한 도면이다.

도 3은 CPM(conceptual partitioning)을 설명하기 위한 도면이다.

도 4는 친구 셀의 예를 도시한다.

도 5는 V_0 의 3-홉 친구 리스트의 예를 도시한다.

도 6a는 NCS의 구체적인 알고리즘을 도시하고, 도 6b는 도 6a에 도시된 FindCellData() 함수의 구체적인 알고리즘을 도시한다.

도 7은 $k=2$ 이고 $\ell=2$ 일 때, NCS의 예를 도시한다.

도 8은 c_1 의 3-홉 친구 셀 리스트의 예를 도시한다.

도 9a는 FCS의 수도 코드(pseudo code)를 도시하고, 도 9b는 도 9a에 도시된 MakeFriendCell() 함수의 알고리즘을 도시한다.

도 10은 FCS의 예를 도시한다.

도 11은 3-홉 개인 셀 리스트의 예를 도시한다.

도 12는 SGI 기반 PCS의 수도 코드(pseudo code)를 도시한다.

도 13은 PCS의 예를 도시한다.

도 14는 기본 업데이트의 세부 내용이 포함된 알고리즘을 도시한다.

도 15는 SGI 업데이트의 5 가지 예를 도시한다.

도 16a는 FAM을 이용한 SGI 업데이트의 상세 알고리즘을 도시하고, 도 16b는 도 16a에 도시된 CheckFAM() 함수의 상세 알고리즘을 도시한다.

도 17은 FAM을 이용한 업데이트의 일 예를 도시한다.

도 18은 본 발명의 일 실시 예에 의한 친구 검색 장치의 기능 블록도를 도시한다.

발명을 실시하기 위한 구체적인 내용

- [0015] 본 명세서에 개시되어 있는 본 발명의 개념에 따른 실시 예들에 대해서 특정한 구조적 또는 기능적 설명들은 단지 본 발명의 개념에 따른 실시 예들을 설명하기 위한 목적으로 예시된 것으로서, 본 발명의 개념에 따른 실시 예들은 다양한 형태들로 실시될 수 있으며 본 명세서에 설명된 실시 예들에 한정되지 않는다.
- [0016] 본 발명의 개념에 따른 실시 예들은 다양한 변경들을 가할 수 있고 여러 가지 형태들을 가질 수 있으므로 실시 예들을 도면에 예시하고 본 명세서에서 상세하게 설명하고자 한다. 그러나, 이는 본 발명의 개념에 따른 실시 예들을 특정한 개시 형태들에 대해 한정하려는 것이 아니며, 본 발명의 사상 및 기술 범위에 포함되는 모든 변경, 균등물, 또는 대체물을 포함한다.
- [0017] 제1 또는 제2 등의 용어는 다양한 구성 요소들을 설명하는데 사용될 수 있지만, 상기 구성 요소들은 상기 용어들에 의해 한정되어서는 안 된다. 상기 용어들은 하나의 구성 요소를 다른 구성 요소로부터 구별하는 목적으로만, 예컨대 본 발명의 개념에 따른 권리 범위로부터 벗어나지 않은 채, 제1 구성 요소는 제2 구성 요소로 명명될 수 있고 유사하게 제2 구성 요소는 제1 구성 요소로도 명명될 수 있다.
- [0018] 어떤 구성 요소가 다른 구성 요소에 "연결되어" 있다거나 "접속되어" 있다고 언급된 때에는, 그 다른 구성 요소에 직접적으로 연결되어 있거나 또는 접속되어 있을 수도 있지만, 중간에 다른 구성 요소가 존재할 수도 있다고 이해되어야 할 것이다. 반면에, 어떤 구성 요소가 다른 구성 요소에 "직접 연결되어" 있다거나 "직접 접속되어" 있다고 언급된 때에는 중간에 다른 구성 요소가 존재하지 않는 것으로 이해되어야 할 것이다. 구성 요소들 간의 관계를 설명하는 다른 표현들, 즉 "~사이에"와 "바로 ~사이에" 또는 "~에 이웃하는"과 "~에 직접 이웃하는" 등도 마찬가지로 해석되어야 한다.
- [0019] 본 명세서에서 사용한 용어는 단지 특정한 실시 예를 설명하기 위해 사용된 것으로서, 본 발명을 한정하려는 의도가 아니다. 단수의 표현은 문맥상 명백하게 다르게 뜻하지 않는 한, 복수의 표현을 포함한다. 본 명세서에서, "포함하다" 또는 "가지다" 등의 용어는 본 명세서에 기재된 특징, 숫자, 단계, 동작, 구성 요소, 부분품 또는 이들을 조합한 것이 존재함을 지정하려는 것이지, 하나 또는 그 이상의 다른 특징들이나 숫자, 단계, 동작, 구성 요소, 부분품 또는 이들을 조합한 것들의 존재 또는 부가 가능성을 미리 배제하지 않는 것으로 이해되어야 한다.
- [0020] 다르게 정의되지 않는 한, 기술적이거나 과학적인 용어를 포함해서 여기서 사용되는 모든 용어들은 본 발명이 속하는 기술 분야에서 통상의 지식을 가진 자에 의해 일반적으로 이해되는 것과 동일한 의미를 가진다. 일반적으로 사용되는 사전에 정의되어 있는 것과 같은 용어들은 관련 기술의 문맥상 가지는 의미와 일치하는 의미를 갖는 것으로 해석되어야 하며, 본 명세서에서 명백하게 정의하지 않는 한, 이상적이거나 과도하게 형식적인 의미로 해석되지 않는다.
- [0021] 이하, 본 명세서에 첨부된 도면들을 참조하여 본 발명의 실시 예들을 상세히 설명한다. 그러나, 특허출원의 범위가 이러한 실시 예들에 의해 제한되거나 한정되는 것은 아니다. 각 도면에 제시된 동일한 참조 부호는 동일한

부재를 나타낸다.

- [0023] 본 발명은 프로모션 광고(promotion advertisement), 공간적인 크라우드소싱(spatial crowdsourcing), 바이럴 마케팅(viral marketing), 블라인드 데이트(blind date) 등과 같이 실생활에 활용이 가능하다. 이하에서, 본 발명이 적용 가능한 시나리오에 대하여 설명하기로 한다.
- [0025] 도 1은 본 발명이 적용 가능한 하우스 파티 시나리오(house party scenario)를 설명하기 위한 도면이다.
- [0026] 여섯 명의 사람들(James, Emma, Mia, Lucas, Luke, David), 이들 각각의 친구 리스트(friend lists), 및 파티 장소(party location)가 주어진 상황에서, 제임스(James)는 새로운 친구들을 사귀기 위한 하우스 파티(house party)를 개최하고 가능한 많은 참석자를 초대하고자 한다. 제임스(James)는 파티 장소로부터 가장 근접한 사람들이 파티에 참석할 것으로 생각한다. 그러나, 제임스(James)의 친구는 단 두 명(즉, 루크(Luke)와 미아(Mia)) 뿐이고, 루크(Luke)는 파티 장소로부터 아주 먼 곳에 위치한다. 이러한 상황에서, 파티를 성공적으로 개최하기 위해, 제임스(James)는 친구인 미아(Mia)의 도움을 받아 엠마(Emma)와 루카스(Lucas)를 초대할 수 있다. 엠마(Emma)와 루카스(Lucas)는 제임스(James)의 친구가 아님에도 불구하고, 지리적인 근접(proximity)과 적절한 사회적 교우 관계(social friendships)로 인하여 새로운 친구가 될 수 있다.
- [0028] 본 발명에서 k -최근접 ℓ -친한 친구 쿼리(k -Nearest ℓ -Close Friends ($k\ell$ -NCF) query, k 와 ℓ 은 자연수)라 불리는 새로운 유형의 지오-소셜 네트워크 쿼리(GeoSN query)를 정의한다. $k\ell$ -NCF 쿼리 $q = (p_q, u, k, \ell)$ 는 쿼리 사용자(즉, 질의자) u 의 ℓ -홉(ℓ -hop) 친구들 중에서 쿼리 포인트(query point, 질의 지점) p_q 와 가장 근접한 k 개의 데이터 객체들(data objects)을 묻는 질의를 의미할 수 있다. 즉, $k\ell$ -NCF 쿼리는 ℓ -홉 친구들 중에서 p_q 와 가장 근접하게 위치한 k 명의 친구들에 대한 추천(또는 검색) 요청을 의미할 수 있다. 여기서, ℓ -홉 친구란 ℓ 번 이하의 직접적인 친구 관계 내에 연결될 수 있는 관계를 의미할 수 있다. 따라서, ℓ -홉 친구는 1-홉 친구, 2-홉 친구, ..., 및 $(\ell-1)$ -홉 친구를 포함한다.
- [0030] **Example 1.** 도 2는 $k\ell$ -NCF 쿼리의 일 예를 설명하기 위한 도면이다. 7 개의 데이터 오브젝트들 $\{d_0, d_1, d_2, d_3, d_4, d_5, \text{ and } d_6\}$, 결과(즉, 쿼리의 결과)의 개수 $k=1$, 사회적 제약(social constraint) $\ell=2$, 쿼리 사용자(질의자) d_0 , 및 쿼리 포인트(질의 지점) p 가 주어졌을 때, $k\ell$ -NCF 검색(search)은 전통적인 공간적 쿼리(traditional spatial query)와는 다르게 d_4 를 응답으로 선택한다. 이는 d_5 와 d_6 이 d_4 보다 쿼리 포인트 p 에 근접하지만 d_0 와 관련성이 없기 때문이다. 다시 말해, d_5 와 d_6 은 d_0 의 2-홉 친구가 아니기 때문이다.
- [0032] 본 발명의 기여를 요약하면 다음과 같다.
- [0033] · 새로운 유형의 지오-소셜 네트워크 쿼리(GeoSN query), 즉 $k\ell$ -NCF 쿼리를 정의하였다.
- [0034] · $k\ell$ -NCF 쿼리를 처리하기 위한 일련의 알고리즘들을 개발하였다.
- [0035] · 효율적인 인덱스 업데이트 방법을 제안하였다.
- [0036] · 제안하는 방식의 효율성을 평가하기 위하여 인위적인 데이터 셋과 실제 데이터 셋으로 다양한 실험을 수행하였다.

[0038] 이하에서는, CPM(Conceptual partitioning)과 SGI(Social grid index)를 설명한 후 $k\ell$ -NCF 쿼리에 대하여 상세하게 설명하기로 한다.

[0040] **Conceptual partitioning (CPM)**

[0041] 주어진 쿼리 포인트(query point, 질의 지점)에 가장 근접한 k 개의 데이터 객체들을 찾는 k -최근접 이웃(k -Nearest Neighbor, k -NN) 쿼리와 이의 변형 예들은 지난 수십년 동안 꾸준히 연구되었다. 이들 중 Mouratidis 등은 가장 널리 차용되고 있는 기법 중 하나이고 CPM(conceptual partitioning)이라 불리는 k -NN 검색 알고리즘을 소개하였다(K. Mouratidis, D. Papadias, M. Hadjieleftheriou, Conceptual partitioning: an efficient method for continuous nearest neighbor monitoring, in: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data, ACM, 2005, pp. 634-645.). CPM의 주된 개념은 전체 그리드(whole grid)를 셀들(cells)의 집합(set)으로 구성되는 다양한 크기의 직사각형들(various-sized rectangles)로 분할하는 것이다(도 3의 (a)와 (b) 참조). 직사각형은 방향(direction(up, down, right, or left))과 레벨 넘버(level number, 해당 직사각형과 쿼리 포인트(query point) p_q 를 포함하는 쿼리 셀(query cell) c_q 사이에 존재하는 직사각형의 개수)의 두 가지 요소(elements)에 의해 표현된다. 예컨대, 도 3의 (b)에서 U_0 와 L_2 는 각각 c_q 로부터 왼쪽 첫번째(first up-side) 직사각형과 왼쪽 세번째(third left-side) 직사각형을 나타낸다. 또한, 동일한 레벨 넘버를 갖는 직사각형들은 동일한 개수의 셀을 포함한다.

[0042] k -NN 쿼리 $q = (k, p_q)$ 가 주어졌을 때, CPM은 레벨의 오름차순(ascending level-order)으로 직사각형들과 셀들을 유지하기 위해 공집합인 결과 집합(empty result set) $\mathcal{R}$ 과 비어 있는 민-힙(empty min-heap) $\mathcal{H}$ 를 초기화한다. 즉, $\mathcal{R}$ 과 $\mathcal{H}$ 가 초기화된다. $mindist(p_q, p_o)$ 를 p_q 와 타겟 오브젝트(target object) o (타겟 오브젝트는 직사각형, 셀, 또는 데이터 오브젝트가 될 수 있음)의 위치 사이의 최소 거리(minimum distance)라고 하고, kNN_dist 를 $\mathcal{R}$ 의 k 번째 원소(k^{th} component)와 p_q 사이의 $mindist$ 의 하자. $\mathcal{H}$ 의 엔트리들(entries)은 항상 $mindist$ 의 오름차순으로 유지된다. 쿼리에 대한 처리가 시작되면, CPM은 c_q 와 c_q 를 에워싸는(surrounding) 직사각형들(U_0 , D_0 , R_0 및 L_0)을 $\mathcal{H}$ 에 삽입한다. 그리고, $\mathcal{H}$ 의 첫번째 엔트리(c_q , 즉 $c_{3,3}$)는 삭제된다(de-heaped). 이때, 삭제된 엔트리(즉, 디-힙된 엔트리)가 셀이면, CPM은 셀을 조사하고(examines) $\mathcal{R}$ 과 kNN_dist 를 업데이트한다. 삭제된 엔트리가 직사각형이면, CPM은 직사각형에 포함되는 모든 셀들과 다음 레벨의 직사각형(next-level rectangle)을 $\mathcal{H}$ 에 삽입한다. 이러한 단계들은 $\mathcal{H}$ 의 엔트리가 없을 때까지 또는 kNN_dist 가 체크될 다음 셀(또는 엔트리)의 $mindist$ 보다 작아질 때까지 반복적으로 수행된다. 요컨대, CPM은 아래 과정을 이용하여 민-힙의 첫번째 엔트리를 반복적으로 삭제(디-힙)한다.

[0043] · 엔트리가 셀이면, 셀을 체크함(check)

[0044] · 엔트리가 직사각형이면, 현재 직사각형에 포함되는 셀들과 동일 방향에서 다음 레벨의 직사각형을 삽입함

[0046] **Example 2.** 도 3의 (c)는 CPM의 검색 예를 도시하고, 표 1에는 검색 예의 세부 내용인 힙의 상태(heap state)가 기재되어 있다. 쿼리 포인트가 p_q 이고 k 가 1이라 가정하자. 우선, CPM은 p_q 가 포함된 셀, 즉 쿼리 셀 $c_{3,3}$ 을 검색한다. 다음으로, 힙의 콘텐츠스(heap contents)는 $mindist$ 의 오름차순으로 정렬된 p_q 주위의 4 개의 초기 직사각형들(즉, L_0 , U_0 , D_0 , 및 R_0)이다. $c_{3,3}$ 에는 어떠한 데이터 오브젝트(data object)도 존재하지 않기 때문에, 결과 집합(result set)은 공집합이다. 다음으로, L_0 가 디힙(de-heaped)된 후 $c_{2,3}$ 과 $c_{2,4}$ 가 힙(heap)에 삽입된다. 데이터 오브젝트 d_0 가 발견될 때까지 검색(search)은 동일한 방식으로 진행된다. CPM은 d_0 를 결과 집

합(result set)에 삽입하고 kNN_dist 를 $mindist(d_0, p_q)$ 로 설정한다(즉, $kNN_dist = mindist(d_0, p_q)$). 그런 다음, $mindist(U_0, p_q)$ 가 kNN_dist 보다 작기 때문에(대시 기호(-)로 된 큰 원(large dashed circle)과 겹침) CPM은 U_0 를 방문한다(visit). $c_{3,2}$ 를 방문한 후 CPM은 최종 해(final answer), 즉 d_1 을 반환하고(return), 어떤 직사각형이나 어떤 셀의 $mindist$ 도 kNN_dist 보다 작지 않기 때문에(작은 실선 원(small, solid circle)과 겹치지 않음) 절차를 중단한다.

표 1

Access (iteration)	Heap contents (ordered by mindist)	Result (data object)
$c_{3,3}$	L_0, U_0, D_0, R_0	ϕ
L_0	$c_{2,3}, U_0, c_{2,4}, D_0, R_0, L_1$	ϕ
$c_{2,3}$	$U_0, c_{2,4}, D_0, R_0, L_1$	ϕ
U_0	$c_{3,4}, c_{2,4}, D_0, R_0, c_{4,4}, L_1, U_1$	ϕ
$c_{3,4}$	$c_{2,4}, D_0, R_0, c_{4,4}, L_1, U_1$	ϕ
$c_{2,4}$	$D_0, R_0, c_{4,4}, L_1, U_1$	d_0
D_0	$c_{3,2}, c_{2,2}, R_0, c_{4,4}, L_1, U_1, D_1$	d_0
$c_{3,2}$	$c_{2,2}, R_0, c_{4,4}, L_1, U_1, D_1$	d_1

Social Grid Index (SGI)

SGI 구조(Social Grid Index structure)는 그리드 셀들(grid cells) 사이의 공간적인 관계(spatial relationships)에 사회적 관계(social relationships)를 더하여 나타내는 그리드 인덱스 구조(grid index structure)이다(C. Shim, W. Kim, W. Heo, S. Yi, C.Y. Dohn, l-Close range friends query on social grid index, IEICE Trans. Inf. Syst. 100 (4) (2017) 811-812.). SGI는 셀들에 포함된 데이터 오브젝트들(data objects)의 친구 관계를 고려하여 셀들의 친구 관계를 형성한다. 예컨대, 동일한 셀(들) 또는 상이한 셀(들)에 포함된 어떤 2 개의 데이터 오브젝트들이 친구이면 셀들은 친구 셀(friend-cell(s))이다.

Definition 1 (friend-cell, 친구 셀). 셀 c_i 내의 데이터 오브젝트 v_i 와 셀 c_j 내의 데이터 오브젝트 v_j 가 친구라면, c_i 와 c_j 는 친구 셀(friend-cells)이다.

Observation 1. 모든 데이터 오브젝트가 적어도 하나의 친구를 갖는다고 가정하자. 동일한 셀 내의 2 개의 서로 다른 데이터 오브젝트들이 친구라면 셀 역시 친구 셀이 될 수 있다.

Example 3. 도 4는 친구 셀의 예를 도시한다. 9 개의 데이터 오브젝트들(v_0 내지 v_8)을 포함하는 8 개의 그리드 셀들(grid cells)이 존재한다. 친구 셀의 정의에 따라, 셀 c_1 의 친구 셀들은 c_0, c_1, c_2, c_5 및 c_6 (음영으로 표시된 셀들)이다.

$G = \{V, E\}$ 를 각 정점(vertex, $v \in V$)이 데이터 오브젝트(사용자)에 대응하는(쿼리 사용자 $u \in V$ 역시 데이터 오브젝트들 중 하나임) 비방향성 소셜 그래프(undirected social graph)라 하고, 각 엣지(edge) $e \in E$ 가 데이터 오브젝트(사용자) 사이의 친구 관계를 나타낸다고 하자. 다시 말해, 엣지 $e(v_i, v_j)$ 는 v_i 와 v_j 가 친구임을 나타낸다. 모든 데이터 오브젝트들은 유클리드 공간(Euclidean space)에서 포인트 p_v 로 표현되는 그들 자신의

위치를 갖고, 공간적인 거리(spatial distance) $d(p_{v_i}, p_{v_j})$ 는 2 개의 데이터 오브젝트들(즉, 사람들) 사이의 지리적인 거리(geographical distance)를 가리킨다. 또한, 친구 관계의 정도(friendship degree) ℓ 은 그래프 내에서 2 개의 데이터 오브젝트들 사이의 엣지들(edges 또는 hops)의 최소 개수를 나타낸다.

Definition 2 (ℓ -hop friend list, ℓ -홉 친구 리스트). $V_v^\ell \subseteq V$ 은 아래 수학적 식 1과 같이 v 의 ℓ -홉 친구 리스트(ℓ -hop friend list)를 나타낸다.

수학적 식 1

$$V_v^\ell = \begin{cases} \{v' \mid \exists e(v, v') \in E\} & (\ell = 1), \\ V_v^{\ell-1} \cup \{v' \mid \exists e(v', v'') \in E \wedge v'' \in V_v^{\ell-1}\} \setminus \{v\} & (\ell > 1). \end{cases}$$

Example 4. 도 5는 v_0 의 3-홉 친구 리스트의 예를 도시한다. 각 정점(point 또는 vertex)과 각 선분(line segment 또는 edge)은 데이터 오브젝트(data object)와 데이터 오브젝트들 사이의 친구 관계에 대응한다. 엣지들에 따르면, v_0 의 1-홉 친구 리스트는 (v_1, v_2, v_4) 이다. 순차적으로, v_5 는 v_4 의 친구이기 때문에 v_0 의 2-홉 친구 리스트는 v_5 와 v_0 의 1-홉 친구들로 구성된다. 동일한 방식으로, v_5 와 v_7 의 친구 관계로 인하여 3-홉 친구 리스트는 v_7 을 포함한다.

Definition 3 (nearest ℓ -close friend query, 최근접 ℓ -친한 친구 쿼리). 비방향성 그래프(undirected graph) $G = \{V, E\}$, 쿼리 포인트(query point) p_q , 쿼리 사용자(query user) u , 그리고 친밀도 또는 친구 관계 정도(friendship degree) ℓ 이 주어졌을 때, 최근접 ℓ -친한 친구(ℓ -NCF) 쿼리 $q = (p_q, u, \ell)$ 는 아래 수학적 식 2와 같은 v 를 반환(return)한다.

수학적 식 2

$$v \in V_u^\ell \wedge v' \in V_u^\ell \setminus \{v\} \wedge d(p_q, p_v) \leq d(p_q, p_{v'}).$$

Definition 4 (k -nearest ℓ -close friends query, k -최근접 ℓ -친한 친구 쿼리). 비방향성 그래프(undirected graph) $G = \{V, E\}$, k -최근접 ℓ -친한 친구(k -Nearest ℓ -Close Friends, $k\ell$ -NCF) 쿼리 $q = (p_q, u, k, \ell)$, 쿼리 포인트(query point) p_q , 쿼리 사용자(query user) u , 결과 원소의 개수(the number of result elements) k , 그리고 친구 관계 정도(friendship degree) ℓ 이 주어졌을 때, k -최근접 ℓ -친한 친구($k\ell$ -NCF) 쿼리는 아래 수학적 식 3과 같은 결과 리스트(result list 또는 result set) $R = (v_1, v_2, \dots, v_k) (1 \leq i \leq k)$ 을 찾는다.

수학적 식 3

$$v_i \in V_u^\ell \wedge d(p_q, p_{v_i}) \leq d(p_q, p_{v_{i+1}}) \wedge v' \in V_u^\ell \setminus R \wedge d(p_q, v_i) \leq d(p_q, v').$$

[0064] **Example 5.** 도 5에서 u , k 및 ℓ 이 각각 v_0 , 1 및 3인 $k\ell$ -NCF 쿼리를 가정하자. 그러면, 쿼리 응답(query answer, 쿼리에 대한 해)은 v_7 이다(쿼리 포인트 p_q 에 가장 근접한 3-홉 친구). v_6 가 p_q 에 가장 근접한 데이터 오브젝트이나, v_6 는 v_0 의 3-홉 친구가 아니기 때문이다.

[0066] $k\ell$ -NCF 쿼리를 처리하기 위한 2 가지의 접근법이 존재한다. 첫번째 방법은 1) 모든 데이터 오브젝트들을 각 데이터 오브젝트와 주어진 쿼리 포인트 사이의 최소 거리인 *mindist*의 오름차순(ascending order)으로 정렬하는 단계와 2) 각 데이터 오브젝트가 쿼리 사용자의 ℓ -홉 친구에 포함되는지 아닌지를 반복적으로 체크(check)하는 단계의 2 단계로 구성된다. 이러한 단계들은 쿼리 결과로써 k 개의 데이터 오브젝트들이 발견될 때까지 반복된다. 다른 방법은 첫번째 방법의 단계를 역으로 수행하는 것이다. 즉, 1) 쿼리 사용자의 ℓ -홉 친구들을 찾는 단계와 2) ℓ -홉 친구들을 *mindist*의 오름 차순으로 정렬하는 단계로 구성된다. 결과는 최초의 k 개 데이터 오브젝트들이다.

[0067] 그러나, 이러한 접근 방식들은 2 가지의 주된 문제점을 갖는다. 첫째, k 가 증가하면 각 데이터 오브젝트에 대하여 스크래치(scratch, 예컨대 그래프에서 최소 경로 계산(shortest path computation))로부터 매우 많은 ℓ -홉 친구 체크를 수행하여야 하고, 이는 쿼리 처리에 상당한 오버헤드를 발생시킨다. 게다가, 친구 관계 정도 ℓ 의 값이 증가함에 따라 정렬되어야 하는 데이터 오브젝트들의 개수가 증가하게 된다.

[0069] 이하에서는, $k\ell$ -NCF 쿼리를 처리하기 위한 일련의 방법들에 관해 기술한다.

[0071] 1. Neighboring cell search(NCS) over CPM

[0072] NCS의 주된 개념은 ℓ -홉 친구 리스트(ℓ -hop friend list)를 이용하여 불필요한 ℓ -홉 친구 조사를 감소시키는 것이다. NCS는 각 데이터 오브젝트가 쿼리 사용자의 ℓ -홉 친구인지 아닌지를 수월하게 체크할 수 있다. 또한, NCS는 다음 최근접 데이터 오브젝트(next nearest data object)를 점진적으로 찾기 위하여 CPM(conceptual partitioning)을 이용한다.

[0073] NCS는 대안적으로 각 데이터 오브젝트 v 의 공간적인 그리고 사회적인 네트워크 정보를 체크함으로써 $k\ell$ -NCF 쿼리의 결과를 찾는다. NCS의 제1 단계는 소셜 그래프(social graph)에 너비 우선 탐색 기법(breadth-first traversal method)을 적용하여 쿼리 사용자 u 의 ℓ -홉 친구 리스트(ℓ -hop friend list)를 구축하는 것이다. 다음 단계는 CPM을 이용하여 최근접 데이터 오브젝트를 재귀적으로 찾는 것이다. 마지막 단계는 ℓ -홉 친구 리스트를 이용하여 각 최근접 데이터 오브젝트가 u 의 ℓ -홉 친구인지 아닌지를 체크하는 것이다. NCS의 개략적인 과정은 다음과 같이 요약될 수 있다.

[0074] 1) 쿼리 사용자(query user, 또는 질의자) u 의 ℓ -홉 친구 리스트(ℓ -hop friend list) V_u^ℓ 를 생성

[0075] 2) 쿼리 셀 C_q 와 C_q 에 인접한 4 개의 직사각형들(C_q 를 둘러싸는 4 개의 직사각형들)을 비어 있는(empty) 민-힙(min-heap) $\mathcal{H}$ 에 삽입

[0076] 3) $\mathcal{H}$ 의 첫번째 엔트리 e 를 추출하고, 다음을 체크

[0077] · e 가 셀이면, V_u^ℓ 를 사용하여 e 내의 각 데이터 오브젝트 v 가 u 의 ℓ -홉 친구인지 아닌지를 체크

[0078] · e 가 직사각형이면, 직사각형 내에 포함된 모든 셀들과 동일한 방향에서의 다음 레벨 직사각형(next level

rectangle)을 $\mathcal{H}$ 에 삽입

4) 본 알고리즘이 모든 쿼리 결과들(즉, k)을 찾거나 $\mathcal{H}$ 의 원소(요소 또는 콘텐츠)가 없어질 때까지 단계 3)을 반복적으로 수행

물론, 단계 4)에서 체크될 다음 셀(또는 엔트리)의 $mindist$ 가 kNN_dist 보다 큰 경우, 검색은 중단된다.

NCS의 구체적인 알고리즘은 도 6a에 도시되어 있고, 도 6a에 도시된 FindCellData() 함수의 구체적인 알고리즘은 도 6b에 도시되어 있다.

Example 6. 도 7에서, 쿼리 사용자 u , $k=2$, $\ell=2$, 및 쿼리 포인트 p_q 가 주어졌을 때, 쿼리 응답(query answer)은 $list(v_{10}, v_{11})$ 이다. 표 2는 대응하는 힙 콘텐츠(heap contents)와 검색(search) 반복에 따른 결과를 도시한다.

표 2

Access (iteration)	Heap contents (ordered by mindist)	Result (data object)
$c_{2,2}$	L_0, U_0, D_0, R_0	ϕ
L_0	$c_{1,2}, U_0, c_{1,3}, D_0, R_0, L_1$	ϕ
$c_{1,2}$	$U_0, c_{1,3}, D_0, R_0, L_1$	ϕ
U_0	$c_{2,3}, c_{1,3}, D_0, R_0, c_{3,3}, L_1, U_1$	ϕ
$c_{2,3}$	$c_{1,3}, D_0, R_0, c_{3,3}, L_1, U_1$	ϕ
$c_{1,3}$	$D_0, R_0, c_{3,3}, L_1, U_1$	ϕ
D_0	$c_{2,1}, c_{1,1}, R_0, c_{3,3}, L_1, U_1, D_1$	ϕ
$c_{2,1}$	$c_{1,1}, R_0, c_{3,3}, L_1, U_1, D_1$	ϕ
$c_{1,1}$	$R_0, c_{3,3}, L_1, U_1, D_1$	v_{10}
R_0	$c_{3,2}, c_{3,3}, c_{3,1}, L_1, U_1, D_1, R_1$	v_{10}
$c_{3,3}$	$c_{3,1}, L_1, U_1, D_1, R_1$	v_{10}
$c_{3,1}$	L_1, U_1, D_1, R_1	v_{10}, v_7
L_1	$c_{0,2}, c_{0,3}, c_{0,1}, U_1, D_1, R_1, c_{0,4}, L_2$	v_{10}, v_7
$c_{0,2}$	$c_{0,3}, c_{0,1}, U_1, D_1, R_1, c_{0,4}, L_2$	v_{10}, v_{11}

2. Friend-cell search on SGI(Social Grid Index)

$k\ell$ -NCF 쿼리를 처리하기 위하여, NCS는 각 데이터 오브젝트의 ℓ -홉 친구 리스트를 이용하여 불필요한 ℓ -홉 친구 계산(ℓ -hop friend computations)을 생략할 수 있다. 그럼에도 불구하고, NCS는 쿼리 사용자의 ℓ -홉 친구들과 무관하게 매우 많은 수의 셀들을 체크한다(예컨대, 도 7에서 셀들 $c_{2,3}$, $c_{3,4}$, 및 $c_{2,4}$). 따라서, 셀 검색(cell search)과 관련하여 성능(performance) 향상을 위한 보다 치밀한 $k\ell$ -NCF 쿼리 처리 기법이 필요하다.

이하에서는, $k\ell$ -NCF 쿼리를 효율적으로 풀이기 위하여, FCS(Friend-Cell Search)라 불리는 신규한 검색 알고리즘(search algorithm)을 소개한다. FCS의 주된 전략은 셀들의 사회적 관계를 고려하여 불필요한 검색(즉, 셀 방문(cell visits))을 피하는 것이다. 그 목적을 달성하기 위하여, SGI에 대한 분기 한정 접근(branch-and-

bound approach)을 차용하였다.

Definition 5 (ℓ -hop friend-cell list, ℓ -홉 친구 셀 리스트). 셀 c 가 자신의 친구 셀들을 가지고 있고 c 의 모든 친구 셀들 또한 그들 자신의 친구 셀들을 가지고 있다고 가정하자. c 의 ℓ -홉 친구 셀 리스트(ℓ -hop friend-cell list) C_c^ℓ 는 아래 수학적 식 4와 같이 정의된다.

수학적 식 4

$$C_c^\ell = \begin{cases} \left\{ c' \mid \exists v \text{ in } c \in C \wedge \exists v' \text{ in } c' \in C \wedge \exists e(v, v') \in E \right\} & (\ell = 1), \\ C_c^{\ell-1} \cup \left\{ c' \mid \exists v \text{ in } c' \in C \wedge \exists v' \text{ in } c'' \in C_c^{\ell-1} \wedge \exists e(v, v') \in E \right\} & (\ell > 1). \end{cases}$$

Example 7. 도 8은 c_1 의 3-홉 친구 셀 리스트의 예를 도시한다. Definition 5에 의해, c_1 의 1-홉 친구 셀 리스트는 $(c_0, c_1, c_2, c_5, c_6)$ 이다. 마찬가지로, c_6 과 c_7 의 친구 관계로 인하여 2-홉 친구 셀 리스트는 $(c_0, c_1, c_2, c_5, c_6, c_7)$ 이다. 유사하게, 3-홉 친구 셀 리스트에는 c_3 가 추가된다.

$k\ell$ -NCF 쿼리가 주어졌을 때, FCS는 우선 쿼리 사용자(query user) u 의 ℓ -홉 친구 리스트와 u 를 포함하는 사용자 셀(user cell) c_u 의 ℓ -홉 친구 셀 리스트 $C_{c_u}^\ell$ 를 생성한다. 그런 다음, FCS는 c_u 의 최근접 ℓ -홉 친구 셀(nearest ℓ -hop friend-cell)이 u 의 ℓ -홉 친구들을 포함하는지 여부를 회귀적으로 체크한다. FCS의 모든 과정이 NCS의 과정과 유사함에도 불구하고, FCS는 불필요한 셀 방문(cell visits)을 감소시킴으로써 상당수의 불필요한 친구 관계 체크 동작을 피할 수 있다. SGI에 기반한 FCS의 과정은 다음과 같다.

- 1) 쿼리 사용자(query user) u 의 ℓ -홉 친구 리스트 V_u^ℓ 과 사용자 셀(user cell) c_u 의 ℓ -홉 친구 셀 리스트 $C_{c_u}^\ell$ 를 생성
 - 2) 쿼리 셀(query cell) c_q 와 c_q 를 둘러싸는 4개의 직사각형들을 비어있는 민-힙(empty min-heap) $\mathcal{H}$ 에 삽입
 - 3) $\mathcal{H}$ 의 첫번째 엔트리(entry) e 를 추출하고 아래 사항을 체크
 - e 가 $C_{c_u}^\ell$ 에 포함되는 셀이면, V_u^ℓ 를 이용하여 e 내의 각 데이터 오브젝트 v 가 u 의 ℓ -홉 친구인지 여부를 체크
 - e 가 직사각형이면, 직사각형에 포함되는 모든 셀들과 동일한 방향에서 다음 레벨의 직사각형을 $\mathcal{H}$ 에 삽입
 - 4) 쿼리 결과를 찾거나 $\mathcal{H}$ 의 엔트리가 없어질 때까지 단계 3)을 반복 수행
- 물론, 단계 4)에서 체크될 다음 셀(또는 엔트리)의 *mindist*가 *kNN_dist*보다 큰 경우, 검색은 중단된다.
- FCS의 수도 코드(pseudo code)는 도 9a에 도시되어 있고, 도 9a에 도시된 MakeFriendCell() 함수의 알고리즘은 도 9b에 도시되어 있다.

Example 8. 도 10은 FCS의 예를 도시한다. 정사각형들(squares)은 SGI 셀들을 나타내고, 특히 옅은 회색(light gray)의 정사각형들은 사용자 셀 c_u 의 친구 셀들을 나타낸다. 쿼리 사용자 u , $k = 2$, $\ell = 2$, 및 쿼리 포인트

pq 가 주어진 경우, 정답은 Example 6.과 같이 (v_{10}, v_{11}) 이다. 표 3은 각 단계의 결과와 상세 내역을 보여준다.

표 3

Access (iteration)	Heap contents (ordered by mindist)	Result (data object)
L_0	$c_{1,2}, U_0, c_{1,3}, D_0, R_0, L_1$	Φ
U_0	$c_{1,3}, D_0, R_0, c_{3,3}, L_1, U_1$	Φ
D_0	$c_{2,1}, c_{1,1}, R_0, c_{3,3}, L_1, U_1, D_1$	Φ
$c_{2,1}$	$c_{1,1}, R_0, c_{3,3}, L_1, U_1, D_1$	Φ
$c_{1,1}$	$R_0, c_{3,3}, L_1, U_1, D_1$	v_{10}
R_0	$c_{3,3}, c_{3,1}, L_1, U_1, D_1, R_1$	v_{10}
$c_{3,1}$	L_1, U_1, D_1, R_1	v_{10}, v_7
L_1	$c_{0,2}, U_1, D_1, R_1, L_2$	v_{10}, v_7
$c_{0,2}$	U_1, D_1, R_1, L_2	v_{10}, v_{11}

3. Personal-cell Search on SGI

어떤 셀들은 쿼리 사용자의 어떤 친구도 포함하지 않음에도 불구하고, FCS는 셀들 간의 친구 관계로 인하여 이러한 특정 셀들을 결코 가지치지(prune) 않는다. 예컨대, $k\ell$ -NCF를 처리할 때, FCS는 불필요한 셀들($c_{2,3}, c_{3,2}, c_{2,4}$ 등)을 체크한다(도 10 참조).

이와 같은 문제점을 해결하기 위하여, $k\ell$ -NCF 쿼리를 처리할 수 있는 진보된 방법으로서 PCS(Personal-cell Search on SGI)를 제안한다.

Definition 6 (personal-cell, 개인 셀). $k\ell$ -NCF 쿼리 $q = (k, \ell, u, pq)$ 가 주어졌을 때, 적어도 하나의 데이터 오브젝트 v (in c)와 u 가 친구이면, 셀 c 는 쿼리 사용자 u 의 개인 셀(personal-cell)이다. 다시 말해, u 의 ℓ -홉 친구가 포함된 셀들의 리스트를 ℓ -홉 개인 셀 리스트(P_u^ℓ)라 할 수 있다.

Definition 7 (ℓ -hop personal-cell list, ℓ -홉 개인 셀 리스트). $k\ell$ -NCF 쿼리 $q = (k, \ell, u, pq)$ 가 주어졌을 때, 쿼리 사용자 u 는 몇몇의 친구들을 갖고 각 친구 v 또한 자신의 친구들을 갖는다고 가정하자. 그러면, u 의 ℓ -홉 개인 셀 리스트 P_u^ℓ 는 다음과 같이 정의된다.

$$P_u^\ell = \begin{cases} \left\{ c \mid \exists v \text{ in } c \in C \wedge \exists e(u, v) \in E \right\} & (\ell = 1), \\ P_u^{\ell-1} \cup \left\{ c \mid \exists v \in V_u^\ell \wedge \exists v' \text{ in } c \in C \wedge \exists e(v, v') \in E \right\} & (\ell > 1). \end{cases}$$

Example 9. 도 11은 3-홉 개인 셀 리스트의 예를 도시한다. 각 정사각형(square)은 그리드 셀(grid cell)을 나타내고 각 포인트는 데이터 오브젝트를 나타낸다. Definition 7에 따라, u 의 1-홉 개인 셀 리스트는 $(c_1, c_5,$

c_6)이다. 마찬가지로, v_4 와 v_5 의 친구 관계로 인하여 2-홉 개인 셀 리스트에는 c_7 이 추가된다. 또한, 3-홉 개인 셀 리스트에는 c_3 이 추가된다.

ℓ -홉 개인 셀 리스트는 미리 PCS가 불필요한 검색(search)을 생략하도록 하기 때문에, PCS는 ℓ -홉 친구 조사의 수행 횟수를 감소시킬 수 있다. 우선, PCS는 u 의 ℓ -홉 친구 리스트와 ℓ -홉 개인 셀 리스트를 생성한다. 셀 체크(cell check)를 위한 기준(criteria)이 ℓ -홉 개인 셀 리스트인 것을 제외하고, 다음의 각 단계들이 FCS로써 수행된다.

1) 쿼리 사용자 u 의 ℓ -홉 친구 리스트 V_u^ℓ 과 ℓ -홉 개인 셀 리스트 P_u^ℓ 을 생성

2) 쿼리 셀 c_q 와 c_q 를 둘러싸는 4 개의 직사각형들을 비어있는 민-힙(empty min-heap) $\mathcal{H}$ 에 삽입

3) $\mathcal{H}$ 의 첫번째 엔트리 e 를 추출하고(또는 디-힙하고), 아래 사항을 체크

· e 가 P_u^ℓ 에 속하는 u 의 개인 셀(personal-cell)이면, V_u^ℓ 를 이용하여 e 내의 각 데이터 오브젝트 v 가 u 의 ℓ -홉 친구인지 체크

· e 가 직사각형이면, e 내의 모든 친구 셀들과 다음 레벨의 직사각형을 $\mathcal{H}$ 에 삽입

4) 쿼리 결과가 도출되거나 $\mathcal{H}$ 의 엔트리가 없어질 때까지 단계 3)을 반복 수행

물론, 단계 4)에서 체크될 다음 엔트리의 *mindist*가 *kNN_dist*보다 큰 경우, 검색은 중단된다.

SGI 기반 PCS의 수도 코드(pseudo code)는 도 12에 도시되어 있다.

Example 10. 도 13은 PCS의 예를 도시한다. 여기서, 옅은 회색의(light gray) 직사각형들은 쿼리 사용자 u 의 개인 셀들을 나타낸다. 쿼리 사용자 u , $k = 2$, $\ell = 2$, 그리고 쿼리 포인트 p_q 가 주어졌을 때, 쿼리 결과는 앞선 예와 같이 (v_{10} , v_{11})이다. 힙 컨텐츠(heap contents)와 각 단계 수행의 결과는 표 4에 도시되어 있다.

표 4

Access (iteration)	Heap contents (ordered by mindist)	Result (data object)
L_0	$c_{1,2}$, U_0 , $c_{1,3}$, D_0 , R_0 , L_1	Φ
U_0	$c_{1,3}$, D_0 , R_0 , $c_{3,3}$, L_1 , U_1	Φ
D_0	$c_{2,1}$, $c_{1,1}$, R_0 , $c_{3,3}$, L_1 , U_1 , D_1	Φ
$c_{1,1}$	R_0 , $c_{3,3}$, L_1 , U_1 , D_1	v_{10}
R_0	$c_{3,3}$, $c_{3,1}$, L_1 , U_1 , D_1 , R_1	v_{10}
$c_{3,1}$	L_1 , U_1 , D_1 , R_1	v_{10} , v_7
L_1	$c_{0,2}$, U_1 , D_1 , R_1 , L_2	v_{10} , v_7
$c_{0,2}$	U_1 , D_1 , R_1 , L_2	v_{10} , v_{11}

표 4에서, 다음에 체크될 엔트리 U_1 의 *mindist*가 *kNN_dist*, 즉 v_{11} 의 *mindist*보다 크기 때문에 Heap의 엔트리가 존재함에도 불구하고 검색 과정을 중단할 수 있다.

[0132] **Update of SGI(Social Grid Index)**

[0133] 이하에서는, SGI의 갱신(update)에 대하여 서술한다. 구체적으로, 기본적인 업데이트(basic update)와 효율적인 인덱스 업데이트 기법(efficient index update method)을 위한 친구 인지 매트릭스(FAM, Friend-Aware Matrix)를 순차적으로 기술한다.

[0135] **1. Basic update**

[0136] 집이나 일터(home or work place)와 같은 각 사용자의 위치(location)와 친구들이 고정된다면, SGI는 변화하지 않을 것이다. 그러나, 실생활에서 데이터 오브젝트들은 그들의 위치 또는 친구 관계를 바꿀 수 있기 때문에 SGI는 적절하게 수정되어야 한다. SGI 업데이트(SGI update)는 SU(Social-Update)와 GU(Geo-Update)를 포함하는 2가지 유형의 동작들로 구성된다. SU는 SGI 상의 셀들의 친구 셀 리스트를 업데이트하는 것이고, GU는 이동하는 데이터 오브젝트들(moving data objects)의 위치에 따라 각 셀의 구성 요소(elements)를 수정하는 것이다.

[0138] 도 14는 기본 업데이트의 세부 내용이 포함된 알고리즘을 도시한다. 우선, 데이터 오브젝트 v 의 이전 위치(old(previous) position) p_v^{old} 와 현재 위치(new(current) position) p_v^{new} 를 초기화한다(initializes). 이전 셀(old cell) c_v^{old} 과 현재 셀(new cell) c_v^{new} 역시 초기화된다(lines 2-3). p_v^{old} 와 p_v^{new} 가 동일하면, 이러한 업데이트 udt_v 는 v 의 친구 업데이트(friend update)이다(line 5). SU가 v 의 새로운 친구(new friend)를 추가하는 것이라면, 새로운 친구 f 는 단지 v 의 1-홉 친구 리스트 V_v^1 에 삽입되고, f 의 셀 c_f 역시 c_v 의 1-홉 친구 셀 리스트 C_v^{1new} 에 삽입된다(line 6-9). 기존의 친구(existing friend) f' 를 삭제하는 경우, 알고리즘은 f' 을 V_v^1 에서 삭제한다(line 11). f' 가 제외된다 하더라도, 2개의 데이터 오브젝트들이 $c_{f'}$ 와 c_v^{new} 에 남아 있으면 $c_{f'}$ 는 C_v^{1new} 로부터 제외될 수 없다. 이를 위하여, C_v^{old} 에 남아있는 데이터 오브젝트들의 친구들을 탐색(traverse)할 필요가 있다(lines 12-20).

[0139] 반면에, p_v^{old} 와 p_v^{new} 가 상이하다면, 이러한 업데이트 udt_v 는 위치 업데이트(location update)이다. c_v^{old} 와 c_v^{new} 가 특히 동일하다면, SGI의 어떠한 업데이트도 불필요하다. 그렇지 않다면, 즉 c_v^{old} 와 c_v^{new} 가 상이하다면, 두 셀들의 업데이트가 시작된다(lines 21-22). v 가 다른 셀 c_v^{new} 로 이동하였기 때문에, 알고리즘은 c_v^{old} 로부터 v 를 삭제하고 새로운 C_v^{1new} 를 생성한다. 그러면, C_v^{1new} 역시 V_v^1 을 탐색(traversing)함으로써 갱신된다(lines 23-32).

[0141] **Example 11.** 도 15는 SGI 업데이트의 5가지 예를 도시한다. 도 15의 (a) 내지 (c)는 SU의 예(Case 1-3)이고, 도 15의 (d)는 GU의 예(Case 4)이고, 도 15의 (e)는 SU와 GU가 결합된 GSU(Geo-Social-Update)의 예(Case 5)이다.

[0142] Case 1에서, 2개의 데이터 오브젝트들인 v_0 와 v_1 이 친구가 되기 때문에(즉, $e(v_0, v_1)$), 셀 c_0 는 c_1 의 (1-홉) 친구 셀 리스트에 추가되고, c_1 역시 c_0 의 (1-홉) 친구 셀 리스트에 추가된다. 반면에, Case 2에서, v_0 와 v_1 는 더 이상 친구가 아니기 때문에 c_0 는 c_1 의 친구 셀 리스트에서 제거된다. Case 3에서 $e(v_0, v_1)$ 가 제거되었음에도 불구하고, v_2 가 여전히 c_0 에 포함되는 v_0 와 친구이기 때문에, SGI의 어떠한 수정도 불필요하다.

그러나, Case 3의 경우, C_0 가 C_1 의 친구 셀 리스트에서 제거되었는지 여부를 체크하기 위하여, C_0 또는 C_1 내의 모든 데이터 오브젝트의 친구 리스트들을 탐색(traverse)할 필요가 있다. Case 4의 경우 또한, v_0 가 동일한 셀 C_0 내에서 이동하였기 때문에(그 결과 C_0 는 여전히 이동 후의 v_0 를 커버함), SGI를 변경할 필요가 없다.

[0143] Case 5에서, C_2 의 친구 셀들은 C_1 과 C_3 이고, v_0 는 다른 셀(즉, C_2 에서 C_0 로)로 이동한다. 다른 케이스들은 상당히 간단하고 비용(costs)이 높지 않은 반면에, Case 5의 경우 SGI가 C_2 의 친구 셀 리스트를 변경할지 여부를 결정해야 하기 때문에 복잡하다. 이를 위해, SGI는 C_2 내의 모든 데이터 오브젝트들의 친구들을 체크한다. 결과적으로, C_3 가 C_2 의 친구 셀 리스트에 잔존하지만 C_1 은 배제된다.

[0144] 5 개의 핵심 예들을 용이하게 적용함으로써, SGI는 모든 케이스들을 처리할 수 있다. 특히, 실생활 예들은 사용자의 위치와 친구를 동시에 변경하는 기능을 갖지 못한다. 다시 말해, 이를 위해 GU(또는 GSU)와 SU가 순차적으로 동작하여야 한다.

[0146] 2. Update with FAM

[0147] 매우 많은 데이터 오브젝트들이 빈번하게 이동하는 때에 업데이트 비용을 줄이는 것이 중요하다. 데이터 오브젝트들이 다른 셀들로 이동할 때마다, 친구 셀 리스트들 매우 빈번하게 탐색(traverse)하여야 하고, 그 비용은 매우 높다. 이러한 이유로 실제 활용에 이용하기에는 적절하지 않다. 이를 해결하기 위하여, 정점들(vertices) 사이의 엣지들(edge)의 개수를 기록하는 매트릭스(matrix, 행렬)인 FAM을 제안한다.

[0148] **Definition 8** (friend-aware matrix, 친구 인지 매트릭스). $\sqrt{|C|} \times \sqrt{|C|}$ SGI에 대하여, 비어 있지 않은 (non-empty) 각 셀 c 는 $\sqrt{|C|} \times \sqrt{|C|}$ 매트릭스 M_c 를 갖는다. 여기서, M_c 의 성분(element) $m_c^{c'}$ 는 셀 c 와 다른 셀 c' 내의 정점들(vertices)의 엣지들의 개수를 나타낸다. M_c 가 아래 수학적 식 5에 의해 정의되는 FAM(friend-aware matrix, 친구 인지 매트릭스)이다.

수학적 식 5

$$m_c^{c'} = \left| \left\{ e \mid \exists v \text{inc} \in C \wedge \exists v' \text{inc}' \in C \wedge \exists e(v, v') \in E \right\} \right|$$

[0150] FAM은 각 셀들이 업데이트 되어야 하는지 여부를 효율적으로 검토할 수 있도록 도와준다. 특히, 친구 리스트들(예, lines 25-28 in Algorithm 6)의 반복적인 탐색(traversal)을 효율적으로 줄일 수 있다. 게다가, FAM과 관련된 동작들(FAM-related operations)은 항상 오리지널 SGI 업데이트(original SGI update) 내에서 수행되기 때문에, FAM을 유지하는 것은 용이하다. 요컨대, FAM을 이용한 SGI 업데이트는 다음 조사(examination)를 이용하여 $k\ell$ -NCF 쿼리 처리를 보조한다.

[0151] · FAM 내의 2 개의 요소들(elements)이 '0' 이라면, 대응하는 셀들 내에 친구로서의 데이터 오브젝트는 없다.

[0152] · FAM 내의 2 개의 요소들이 '0' 보다 크다면, 대응하는 셀들 내에 친구로서의 데이터 오브젝트들이 적어도 2 개 존재한다.

[0154] 도 16a는 FAM을 이용한 SGI 업데이트의 상세 알고리즘을 도시하고, 도 16b는 도 16a에 도시된 CheckFAM() 함수의 상세 알고리즘을 도시한다. 대부분의 알고리즘이 기본 기법과 유사하지만, FAM이 비용면에서 효율적인 결과를 만든다. 엣지(edge)를 추가할 때, 이 알고리즘은 FAM의 대응하는 각 요소에 '1'을 더한다(즉, $1 + m_c^{c'}$ (lines 10-11, 22-23)). 엣지를 삭제할 때, 친구의 존재를 효율적으로 체크하기 위하여 함수 CheckFAM이 수행된

다(lines 14-15, 24-25). CheckFAM에서, FAM의 대응하는 각 요소로부터 1이 감소된다(즉, $-1 + m_c^{c'}$).

- [0156] **Example 12.** 도 17은 FAM을 이용한 업데이트의 일 예를 도시한다. SGI가 존재하고, 정의 8(Definition 8)에 따라 비어 있지 않은 각 셀(non-empty cell)은 자신의 FAM(2×2 매트릭스)을 갖는다. v_0 가 c_3 로부터 c_0 로 이동할 때, c_3 의 1-홉 친구 셀 리스트에 c_2 가 친구 셀로 유지되는지 여부를 체크하여야 한다. 이를 위해, 기본 업데이트 기법(basic update method)은 c_3 내의 모든 데이터 오브젝트의 친구 리스트들을 탐색(traverse)하여야 한다. 불행히도, SGI 업데이트의 비효율을 야기한다. FAM_{c_3} 의 하나의 요소, 즉 $m_{c_3}^{c_2}$ 가 '0'이 아님을 체크하는 것만으로도 쉽게 알 수 있기 때문에, 제안하는 기법은 상술한 기본 기법 보다 효율적이다.
- [0158] 도 18은 본 발명의 일 실시 예에 의한 친구 검색 장치의 기능 블록도를 도시한다. 도 18을 참조하면, 친구 검색 서버 등으로 명명될 수도 있는 친구 검색 장치(100)는 쿼리 수신부(110), 리스트 생성부(130), 쿼리 처리부(150), 및 저장부(170)를 포함한다. 실시 예에 따라, 친구 검색 장치는 업데이트부(190)를 더 포함할 수도 있다.
- [0159] 친구 검색 장치(100)는 사용자 단말(미도시)로부터 $k\ell$ -NCF 쿼리, 즉 사용자의 ℓ -홉 친구들 중에서 가장 근접한 k 명의 친구에 대한 검색 요청(또는 사용자와 가장 근접하게 위치한 k 명의 ℓ -홉 친구들에 대한 검색 요청)을 수신하고, 이에 대한 응답인 k 명의 친구 리스트를 출력한다. 출력된 k 명의 친구 리스트는 상기 사용자 단말로 송신될 수 있다. 이러한 실시예는 상기 사용자가 속한 SNS(Social Network Service)를 제공하는 SNS 서버가 친구 검색 장치(100)일 경우이다.
- [0160] 실시 예에 따라, 친구 검색 장치(100)는 상기 사용자의 $k\ell$ -NCF 쿼리를 SNS 서버를 경유하여 수신할 수도 있다. 친구 검색 장치(100)는 k 명의 친구 리스트를 출력하고, 출력된 k 명의 친구 리스트는 상기 SNS 서버를 경유하여 다시 상기 사용자 단말로 송신될 수 있다. 즉, 이 경우는 SNS 서버와 친구 검색 서비스를 제공하는 서버가 분리되어 있는 경우이다.
- [0161] 상술한 바와 같이, 친구 검색 장치(100)는 SNS 서버로 구현되거나 SNS 서버와는 구별되는 별도의 서버로 구현될 수 있으나, 앞서 상술한 친구 검색 알고리즘들(NCS, FCS, 및 PCS) 중 적어도 하나를 이용하여 쿼리에 대한 응답을 생성할 수 있다.
- [0163] 쿼리 수신부(110)는 사용자 단말 또는 별도의 서버(예컨대, SNS 서버)로부터 사용자 u 의 $k\ell$ -NCF 쿼리를 수신할 수 있다. 쿼리 수신부(110)에 의해 수신된 $k\ell$ -NCF 쿼리는 저장부(170)에 저장될 수 있다.
- [0164] 리스트 생성부(130)는 사용자 u 의 ℓ -홉 친구 리스트 V_u^ℓ 와 사용자 u 의 ℓ -홉 개인 셀 리스트 P_u^ℓ 를 생성한다. 생성된 V_u^ℓ 와 P_u^ℓ 는 저장부(170)에 저장될 수 있다.
- [0165] 쿼리 처리부(150)는 NCS, FCS, 및 PCS 중 적어도 하나의 친구 검색 알고리즘을 수행하여 사용자 u 의 $k\ell$ -NCF 쿼리에 대한 응답을 생성하고, 생성된 응답을 사용자 u 의 단말 또는 별도의 서버(예컨대, SNS 서버)로 송신할 수 있다. 상기 응답이 별도의 서버로 송신된 경우, 상기 별도의 서버는 상기 응답을 상기 사용자 단말로 송신할 수 있다.
- [0166] 쿼리 처리부(150)가 PCS 알고리즘을 이용하여 친구 검색을 수행하는 경우에 대하여 구체적으로 설명하면 다음과 같다.
- [0167] 우선, 쿼리 처리부(150)는 쿼리 사용자, 질의자 u 를 포함하고 있는 셀(쿼리 셀) c_q 와 c_q 를 둘러싸는 4 개의

직사각형들을 힙 $\mathcal{H}$ 에 삽입한다. 오브젝트들을 삽입하기 전에 $\mathcal{H}$ 는 초기화될 수도 있으며, $\mathcal{H}$ 에 삽입된 데이터들(오브젝트들)은 항상 해당 오브젝트와 질의 지점 P_q 사이의 최소 거리 $mindist$ 의 오름 차순으로 정렬될 수 있다.

[0168] 이후, 쿼리 처리부(150)는 $\mathcal{H}$ 의 엔트리들 중 $mindist$ 가 가장 작은 엔트리를 추출(또는 디-힙)하고, 아래 사항을 체크한다. 실시예에 따라, $\mathcal{H}$ 의 엔트리들이 $mindist$ 의 오름 차순으로 정렬되어 있는 경우, 쿼리 처리부(150)는 $\mathcal{H}$ 의 첫번째 엔트리를 추출(또는 디-힙)할 수 있다.

[0169] 즉, 엔트리 e 가 P_u^ℓ 에 속하는 u 의 개인 셀(personal-cell)이면, V_u^ℓ 를 이용하여 e 내의 각 데이터 오브젝트 v 가 u 의 ℓ -홉 친구인지 여부를 결정한다(v 가 u 의 ℓ -홉 친구라면, 쿼리 처리부(150)는 해당 오브젝트 v 를 결과 집합 $\mathcal{R}$ 에 삽입한다).

[0170] 이와는 다르게, 엔트리 e 가 직사각형이면, e 내의 모든 친구 셀들과 다음 레벨의 직사각형을 $\mathcal{H}$ 에 삽입한다.

[0171] 쿼리 처리부(150)는 위 과정을 반복적으로 수행하되, $\mathcal{H}$ 의 엔트리가 없거나 다음에 체크될 엔트리(예컨대, $\mathcal{H}$ 의 첫번째 엔트리)의 $mindist$ 가 kNN_dist 보다 크다고 판단되는 경우, 질의 처리(즉, 친구 검색)를 종료한다.

[0173] 저장부(170)에는 $k\ell$ -NCF 쿼리를 수행하는 과정 중에 이용되는 데이터, 생성되는 데이터 등이 저장될 수 있다. 구체적으로, 저장부(170)에는 $k\ell$ -NCF 쿼리, 각 사용자의 위치 정보, 사용자들 간의 친구 관계 정보, 질의자 u 의 ℓ -홉 친구 리스트, 질의자 u 의 ℓ -홉 개인 셀 리스트, 질의자 u 의 ℓ -홉 친구 셀 리스트, 각 단계에서의 kNN_dist 등이 저장될 수 있다.

[0175] 업데이트부(190)는 SGI(Social Grid Index)를 업데이트할 수 있다. 사용자 단말 또는 별도의 서버(예컨대, SNS 서버)로부터 수신되는 각 사용자의 위치와 친구 관계에 대한 정보는 저장부(170)에 저장될 수 있다. 이때, 새로운 위치 정보 또는 새로운 친구 관계에 대한 정보가 수신되는 경우, SGI의 업데이트(갱신) 동작이 필요하다.

[0176] 업데이트부(190)는 상술한 Basic update와 Update with FAM 알고리즘 중 적어도 하나를 이용하여 SGI를 업데이트할 수 있다. 이하에서는, Update with FAM을 이용한 업데이트 과정에 대하여 상술한다.

[0177] 우선, 업데이트부(190)는 SGI의 각 셀에 대응하는 매트릭스(예컨대, 상술한 FAM) M_c 를 생성할 수 있다. 이때, 오브젝트를 포함하고 있는 셀(non-empty cell)들 각각에 대응하는 매트릭스만 생성될 수도 있다. 또한, 생성된 매트릭스는 SGI의 크기와 동일한 크기를 가질 수 있다. 예컨대, $\sqrt{|C|} \times \sqrt{|C|}$ 의 SGI가 주어진다면, 생성된 매트릭스의 크기 역시 $\sqrt{|C|} \times \sqrt{|C|}$ 이다. M_c 의 성분 $m_c^{c'}$ 는 셀 c 와 셀 c' 내의 정점들(vertices)의 엣지들의 개수를 나타내며, 이에 대한 구체적인 정의는 수학식 5와 같다.

[0178] 구체적으로, 업데이트부(190)는 사용자 단말(또는 SNS 서버)로부터 받은 위치 정보 및/또는 친구 관계 정보에 따라 각 매트릭스의 구성(또는 성분)을 수정함으로써 업데이트를 진행할 수 있다. 업데이트부(190)에 의해 생성되거나 갱신된 매트릭스들은 저장부(170)에 저장될 수 있다. 매트릭스들이 저장부(170)에 저장되면, 리스트 생성부(130)는 저장된 매트릭스들을 이용하여 친구 리스트와 개인 셀 리스트를 생성할 수 있다.

[0180] 본 발명은 상술한 친구 검색 장치(100) 뿐만 아니라, 친구 검색 장치(100)에 의해 수행되는 친구 검색 방법을 포함할 수 있다. 친구 검색 방법에 관한 설명은 상술한 친구 검색 장치(100)의 동작과 동일하기 때문에 이에 대한 상세한 설명은 생략하기로 한다.

[0182] 이상에서 설명된 친구 검색 장치(100)는 하드웨어 구성 요소, 소프트웨어 구성 요소, 및/또는 하드웨어 구성 요소 및 소프트웨어 구성 요소의 집합으로 구현될 수 있다. 예를 들어, 실시 예들에서 설명된 장치 및 구성 요소는, 예를 들어, 프로세서, 컨트롤러, ALU(Arithmetic Logic Unit), 디지털 신호 프로세서(Digital Signal Processor), 마이크로컴퓨터, FPA(Field Programmable array), PLU(Programmable Logic Unit), 마이크로프로세서, 또는 명령(instruction)을 실행하고 응답할 수 있는 다른 어떠한 장치와 같이, 하나 이상의 범용 컴퓨터 또는 특수 목적 컴퓨터를 이용하여 구현될 수 있다. 처리 장치는 운영 체제(Operation System, OS) 및 상기 운영 체제 상에서 수행되는 하나 이상의 소프트웨어 애플리케이션을 수행할 수 있다. 또한, 처리 장치는 소프트웨어의 실행에 응답하여, 데이터를 접근, 저장, 조작, 처리 및 생성할 수도 있다. 이해의 편의를 위하여, 처리 장치는 하나가 사용되는 것으로 설명된 경우도 있지만, 해당 기술 분야에서 통상의 지식을 가진 자는, 처리 장치가 복수 개의 처리 요소(Processing Element) 및/또는 복수 유형의 처리 요소를 포함할 수 있음을 알 수 있다. 예를 들어, 처리 장치는 복수 개의 프로세서 또는 하나의 프로세서 및 하나의 컨트롤러를 포함할 수 있다. 또한, 병렬 프로세서(Parallel Processor)와 같은, 다른 처리 구성(Processing Configuration)도 가능하다.

[0183] 소프트웨어는 컴퓨터 프로그램(Computer Program), 코드(Code), 명령(Instruction), 또는 이들 중 하나 이상의 조합을 포함할 수 있으며, 원하는 대로 동작하도록 처리 장치를 구성하거나 독립적으로 또는 결합적으로(Collectively) 처리 장치를 명령할 수 있다. 소프트웨어 및/또는 데이터는, 처리 장치에 의하여 해석되거나 처리 장치에 명령 또는 데이터를 제공하기 위하여, 어떤 유형의 기계, 구성 요소(Component), 물리적 장치, 가상 장치(Virtual Equipment), 컴퓨터 저장 매체 또는 장치, 또는 전송되는 신호 파(Signal Wave)에 영구적으로, 또는 일시적으로 구체화(Embody)될 수 있다. 소프트웨어는 네트워크로 연결된 컴퓨터 시스템 상에 분산되어서, 분산된 방법으로 저장되거나 실행될 수도 있다. 소프트웨어 및 데이터는 하나 이상의 컴퓨터 판독 가능 기록 매체에 저장될 수 있다.

[0184] 실시 예에 따른 방법은 다양한 컴퓨터 수단을 통하여 수행될 수 있는 프로그램 명령 형태로 구현되어 컴퓨터 판독 가능 매체에 기록될 수 있다. 상기 컴퓨터 판독 가능 매체는 프로그램 명령, 데이터 파일, 데이터 구조 등을 단독으로 또는 조합하여 포함할 수 있다. 상기 매체에 기록되는 프로그램 명령은 실시 예를 위하여 특별히 설계되고 구성된 것들이거나 컴퓨터 소프트웨어 당업자에게 공지되어 사용 가능한 것일 수도 있다. 컴퓨터 판독 가능 기록 매체의 예에는 하드 디스크, 플로피 디스크 및 자기 테이프와 같은 자기 매체(Magnetic Media), CD-ROM, DVD와 같은 광기록 매체(Optical Media), 플롭티컬 디스크(Floptical Disk)와 같은 자기-광 매체(Magneto-optical Media), 롬(ROM), 램(RAM), 플래시 메모리 등과 같은 프로그램 명령을 저장하고 수행하도록 특별히 구성된 하드웨어 장치가 포함된다. 프로그램 명령의 예에는 컴파일러에 의해 만들어지는 것과 같은 기계어 코드뿐만 아니라 인터프리터 등을 사용해서 컴퓨터에 의해서 실행될 수 있는 고급 언어 코드를 포함한다. 상기된 하드웨어 장치는 실시 예의 동작을 수행하기 위해 하나 이상의 소프트웨어 모듈로서 작동하도록 구성될 수 있으며, 그 역도 마찬가지이다.

[0186] 본 발명은 도면에 도시된 실시 예를 참고로 설명되었으나 이는 예시적인 것에 불과하며, 본 기술 분야의 통상의 지식을 가진 자라면 이로부터 다양한 변형 및 균등한 타 실시 예가 가능하다는 점을 이해할 것이다. 예를 들어, 설명된 기술들이 설명된 방법과 다른 순서로 수행되거나, 및/또는 설명된 시스템, 구조, 장치, 회로 등의 구성 요소들이 설명된 방법과 다른 형태로 결합 또는 조합되거나, 다른 구성 요소 또는 균등물에 의하여 대치되거나 치환되더라도 적절한 결과가 달성될 수 있다. 따라서, 본 발명의 진정한 기술적 보호 범위는 첨부된 등록청구범위의 기술적 사상에 의해 정해져야 할 것이다.

부호의 설명

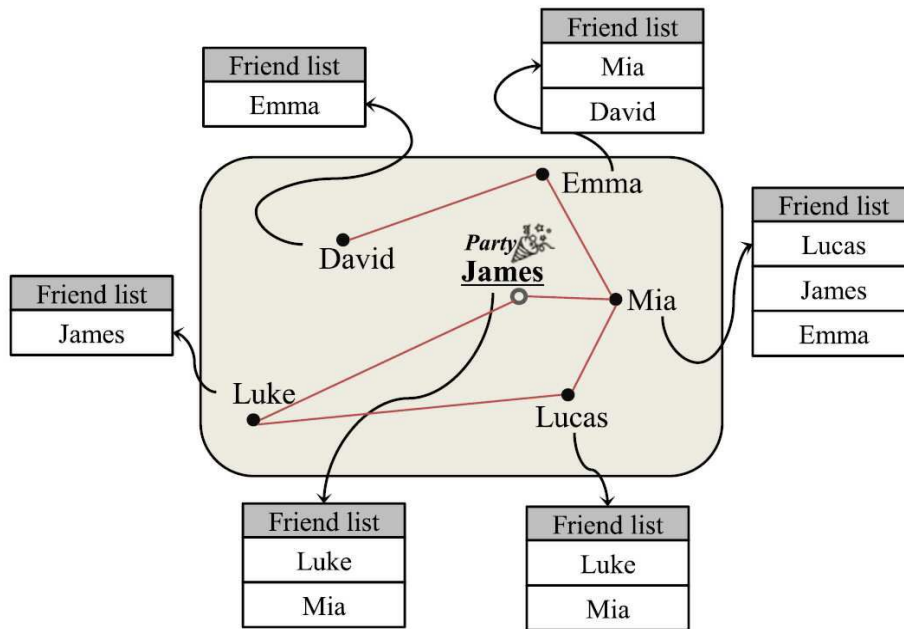
[0188] 100 : 친구 검색 장치
110 : 쿼리 수신부
130 : 리스트 생성부
150 : 쿼리 처리부

170 : 저장부

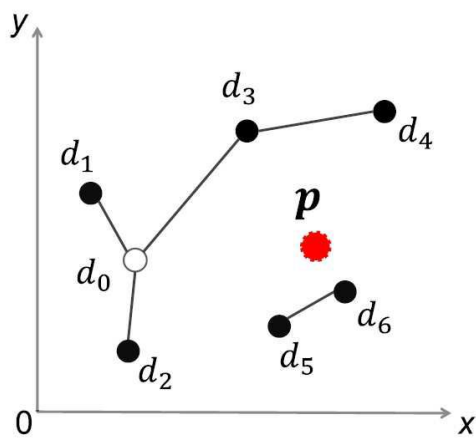
190 : 업데이트부

도면

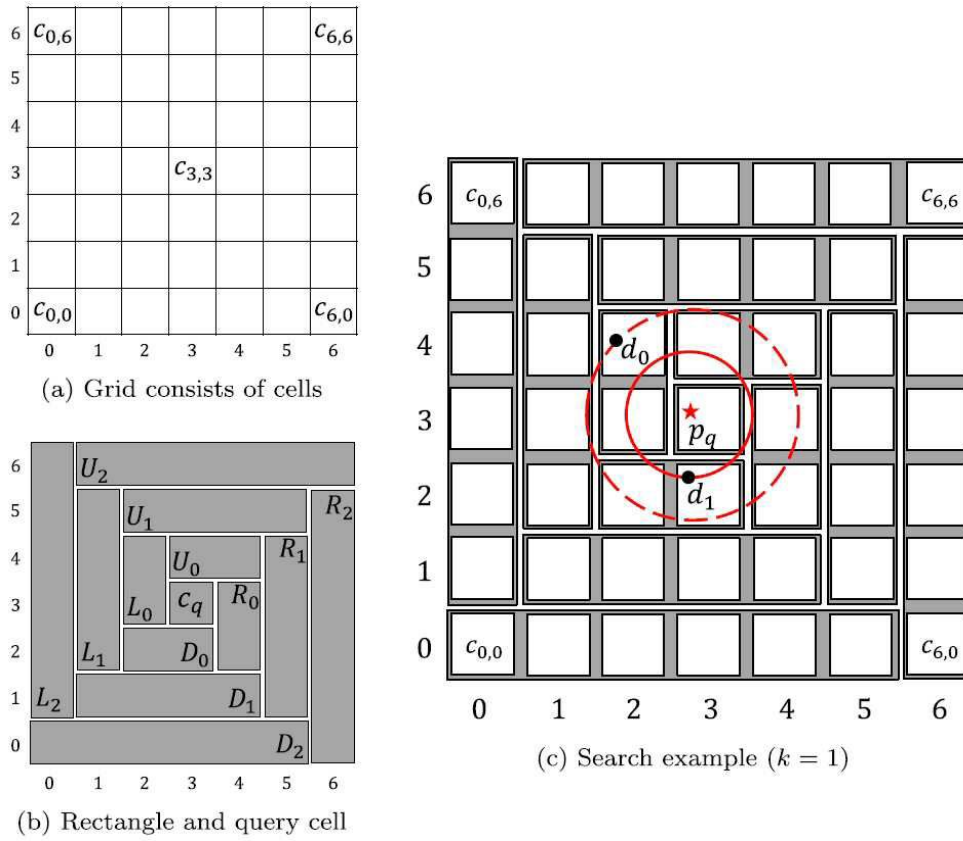
도면1



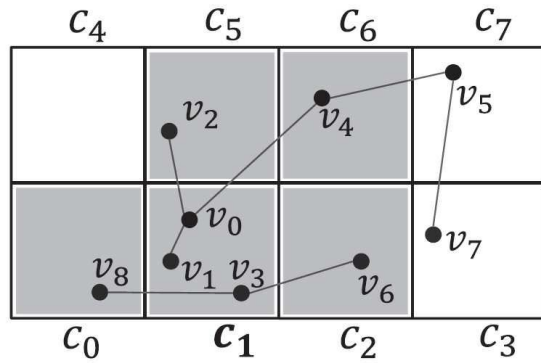
도면2



도면3

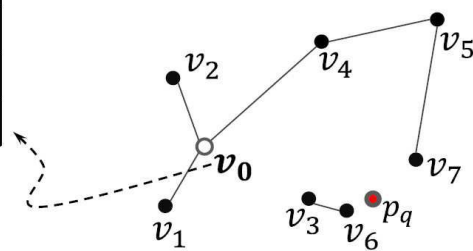


도면4



도면5

ℓ -hop friend lists of v_0	
1-hop friend list	(v_1, v_2, v_4)
2-hop friend list	(v_1, v_2, v_4, v_5)
3-hop friend list	$(v_1, v_2, v_4, v_5, v_7)$



도면6a

Algorithm 1: Neighboring Cell Search.

Input : $k\ell$ -NCF query $q = (u, k, \ell, p_q)$
Output : result list $\mathcal{RL}$

Procedure:

```

1 Set result list  $\mathcal{RL} = \emptyset$ , mindist-ordered candidate list  $\mathcal{CL} = \emptyset$ , mindist-ordered min-heap  $\mathcal{H} = \emptyset$ ;
2 Set  $current\_hop = 1$ ,  $kNN\_dist = \infty$ ;
3 while  $current\_hop \leq \ell$  do
4   for each data object  $v \in V_u^\ell$  do
5     for each friend  $f$  of  $v$  do
6       Insert  $f$  of  $v$  into  $V_u^\ell$ ;
7    $current\_hop = 1 + current\_hop$ ;
8 Insert query cell  $c_q$  and its surrounding four rectangles into  $\mathcal{H}$ ;
9 while  $\mathcal{H} \neq \emptyset$  or  $kNN\_dist \leq \text{mindist of the first entry of } \mathcal{H}$  do
10  Entry  $e = \text{pop } \mathcal{H}$ ;
11  if  $e$  is cell  $c$  then
12    FindCellData( $e$ );
13  else //  $e$  = a rectangle
14    for each  $c$  in  $e$  do
15      Insert  $c$  with mindist into  $\mathcal{H}$ ;
16      Insert the next rectangle in the same direction into  $\mathcal{H}$ ;
17 Insert top- $k$  elements from  $\mathcal{CL}$  to  $\mathcal{RL}$ ;
18 return  $\mathcal{RL}$ ;

```

도면6b

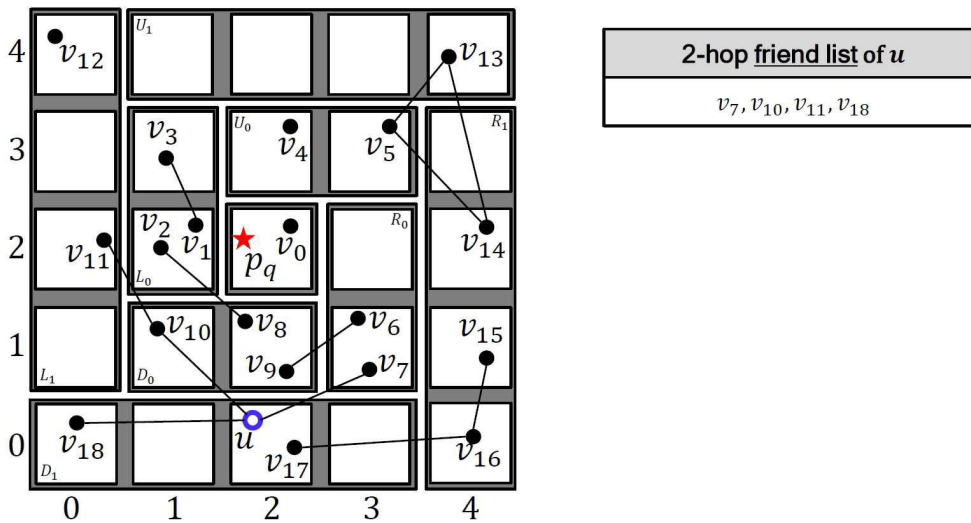
Algorithm 2: Function FindCellData(cell c).

```

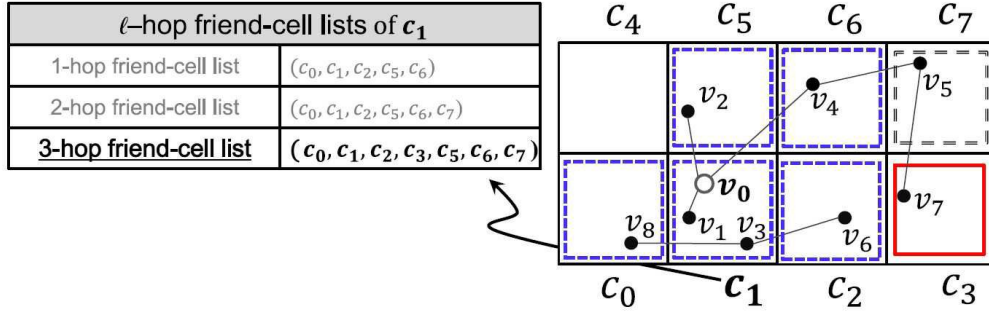
1 Function FindCellData(cell  $c$ ):
2   for each data object  $v$  in  $c$  do
3     if  $v \in V_u^\ell$  then
4       if  $kNN\_dist > \text{mindist of } v$  then
5         Insert  $v$  with mindist into  $\mathcal{CL}$ ;
6       if  $|\mathcal{CL}| \geq k$  then
7         Set  $kNN\_dist = \text{mindist of the } k\text{th element of } \mathcal{CL}$ ;

```

도면7



도면8



도면9a

Algorithm 3: Friend-Cell Search.

Input : $k\ell$ -NCF query $q = (u, k, \ell, p_q)$, SGI

Output : result list $\mathcal{RL}$

Procedure:

```

1 Set result list  $\mathcal{RL} = \emptyset$ , mindist-ordered candidate list  $\mathcal{CL} = \emptyset$ , mindist-ordered min-heap  $\mathcal{H} = \emptyset$ ;
2 Set  $current\_hop = 1$ ,  $kNN\_dist = \infty$ ;
3 while  $current\_hop \leq \ell$  do
4   for each data object  $v \in V_u^\ell$  do
5     for each friend  $f$  of  $v$  do
6       Insert  $f$  into  $V_u^\ell$ ;
7    $current\_hop = 1 + current\_hop$ ;
8 MakeFriendCell(user cell  $c_u$ ,  $\ell$ );
9 if query cell  $c_q \in C_u^\ell$  then
10   Insert  $c_q$  into  $\mathcal{H}$ ;
11 Insert its surrounding four rectangles into  $\mathcal{H}$ ;
12 while  $\mathcal{H} \neq \emptyset$  or  $kNN\_dist \leq$  mindist of the first entry of  $\mathcal{H}$  do
13   Entry  $e = \text{pop } \mathcal{H}$ ;
14   if  $e$  is  $fc$  then
15     FindsCellData( $e$ );
16   else //  $e$  is a rectangle
17     for each  $fc$  in  $e$  do
18       Insert  $fc$  with mindist into  $\mathcal{H}$ ;
19       Insert the next rectangle in the same direction into  $\mathcal{H}$ ;
20 Insert top- $k$  elements from  $\mathcal{CL}$  to  $\mathcal{RL}$ ;
21 return  $\mathcal{RL}$ ;

```

도면9b

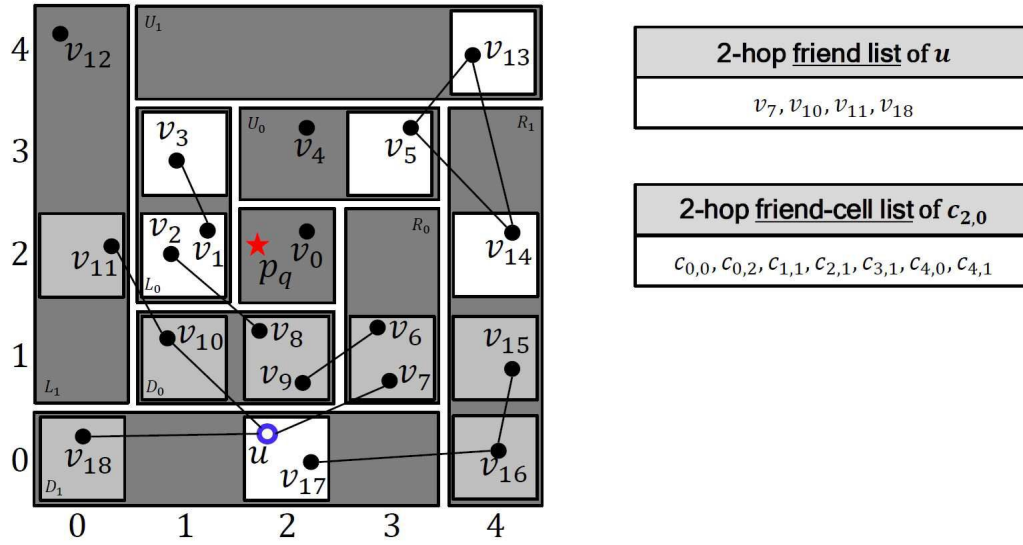
Algorithm 4: Function MakeFriendCell(cell c , friendship degree ℓ).

```

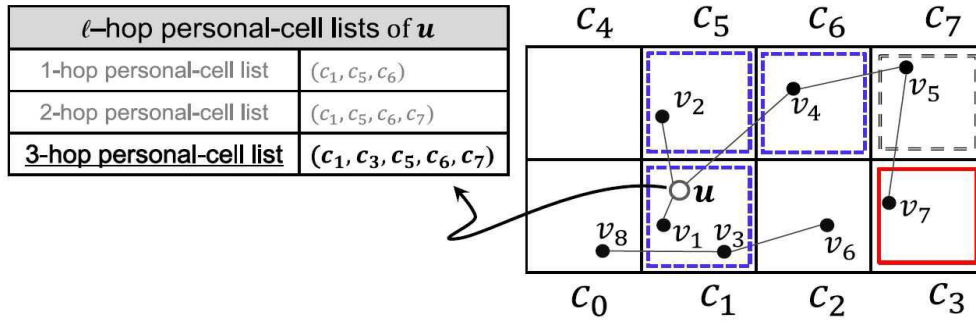
1 Function MakeFriendCell(cell  $c_u$ , friendship degree  $\ell$ ):
2   Set  $cell\_hop = 1$ ;
3   while  $cell\_hop \leq \ell$  do
4     for each cell  $c \in C_u^\ell$  do
5       for each friend-cell  $fc$  of  $c$  do
6         Insert  $fc$  into  $C_u^\ell$ ;
7      $cell\_hop = 1 + cell\_hop$ ;

```

도면10



도면11



도면12

Algorithm 5: Personal-Cell Search.

Input : $k\ell$ -NCF query $q = (u, k, \ell, p_q)$, SGI
Output : result list $\mathcal{RL}$

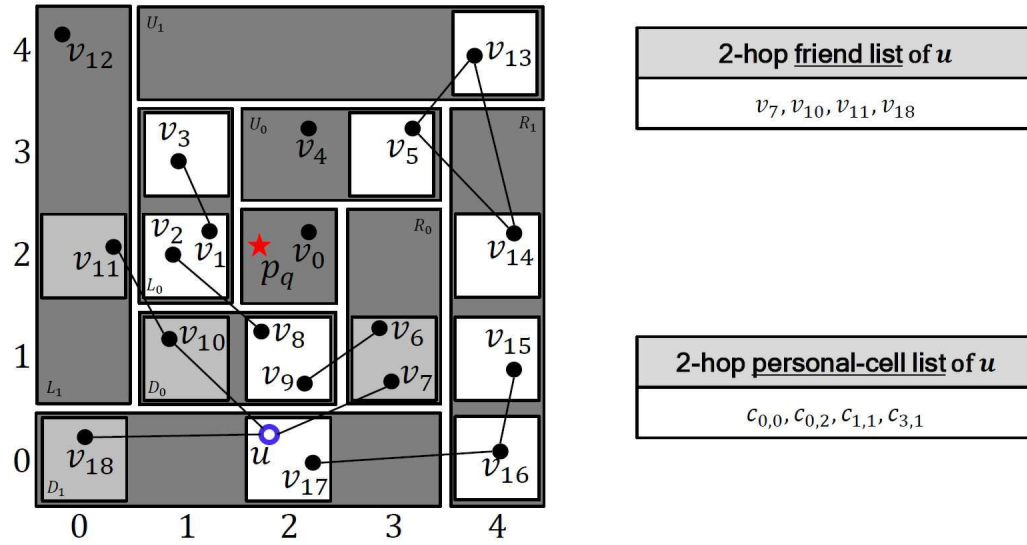
Procedure:

```

1 Set result list  $\mathcal{RL} = \emptyset$ , mindist-ordered candidate list  $\mathcal{CL} = \emptyset$ , mindist-ordered min-heap  $\mathcal{H} = \emptyset$ ;
2 Set  $\text{current\_hop} = 1$ ,  $kNN\_dist = \infty$ ;
3 while  $\text{current\_hop} \leq \ell$  do
4   for each data object  $v \in V_u^\ell$  do
5     for each friend  $f$  of  $v$  do
6       Insert  $f$  into  $V_u^\ell$ ;
7       Insert a personal-cell  $pc$  covering  $f$  into  $P_u^\ell$ ;
8    $\text{current\_hop} = 1 + \text{current\_hop}$ ;
9 if query cell  $c_q \in P_u^\ell$  then
10  Insert  $c_q$  into  $\mathcal{H}$ ;
11 Insert its surrounding four rectangles into  $\mathcal{H}$ ;
12 while  $\mathcal{H} \neq \emptyset$  or  $kNN\_dist \leq \text{mindist of the first entry of } \mathcal{H}$  do
13  Entry  $e = \text{pop } \mathcal{H}$ ;
14  if  $e$  is  $pc$  then
15    FindsCellData( $e$ );
16  else //  $e$  = a rectangle
17    for each  $pc$  in  $e$  do
18      Insert  $pc$  with mindist into  $\mathcal{H}$ ;
19      Insert the next level rectangle in the same direction into  $\mathcal{H}$ ;
20 Insert top- $k$  elements from  $\mathcal{CL}$  to  $\mathcal{RL}$ ;
21 return  $\mathcal{RL}$ ;

```

도면13



도면14

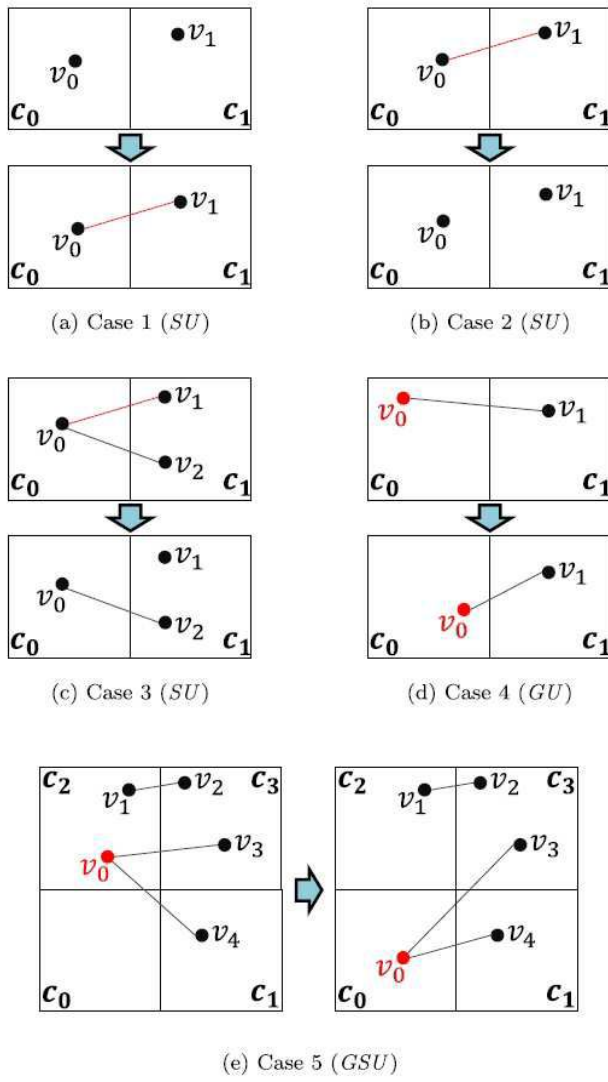
Algorithm 6: Function BasicUpdate(data object v 's update udt_v).

```

1 Function BasicUpdate(data object  $v$ 's update  $udt_v$ ):
2   Set  $p_v^{old}$  = old position of  $v$ ,  $p_v^{new}$  = new position of  $v$ ;
3   Set  $c_v^{old}$  = a cell covering  $v$  at  $p_v^{old}$ ,  $c_v^{new}$  = a cell covering  $v$  at  $p_v^{new}$ ;
4   Set isFriendCell = false;
5   if  $udt_v$  is friend update then                                     //  $p_v^{old} = p_v^{new}$ ,  $c_v^{old} = c_v^{new}$ 
6     if  $udt_v$  contains new friend  $f$  then
7       Insert  $f$  into  $v$ 's 1-hop friend list  $V_v^1$ ;
8       if  $c_f \notin C_{c_v^{new}}^1$  then                                     //  $C_{c_v^{new}}^1$  = 1-hop friend-cell list of  $c_v^{new}$ 
9         Insert  $c_f$  into  $C_{c_v^{new}}^1$ ;                                     // SU
10    else                                                             //  $udt_v$  contains existing friend  $f'$ 
11      Delete  $f'$  from  $V_v^1$ ;
12      for each  $v'$  in  $c_v^{new}$  except  $v$  do
13        for each friend  $v''$  of  $v'$  do
14          if  $c_{f'} = c_{v''}$  then
15            isFriendCell = true;
16            break;
17          if isFriendCell = true then
18            break;
19          if isFriendCell = false then
20            Delete  $c_{f'}$  from  $C_{c_v^{new}}^1$ ;                                     // SU
21    else                                                             //  $udt_v$  is location update,  $p_v^{old} \neq p_v^{new}$ 
22      if  $c_v^{old} \neq c_v^{new}$  then
23        Set  $C_{c_v^{old}}^1 = \phi$ ;
24        Delete  $v$  from  $c_v^{old}$ ;                                     // GU
25        for each  $v'$  in  $c_v^{old}$  do
26          for each friend  $v''$  of  $v'$  do
27            if  $c_{v''} \notin C_{c_v^{old}}^1$  then
28              Insert  $c_{v''}$  into  $C_{c_v^{old}}^1$ ;                                     // SU
29        Insert  $v$  into  $c_v^{new}$ ;                                     // GU
30        for each friend  $v''$  of  $v$  do
31          if  $c_{v''} \notin C_{c_v^{new}}^1$  then
32            Insert  $c_{v''}$  into  $C_{c_v^{new}}^1$ ;                                     // SU

```

도면15



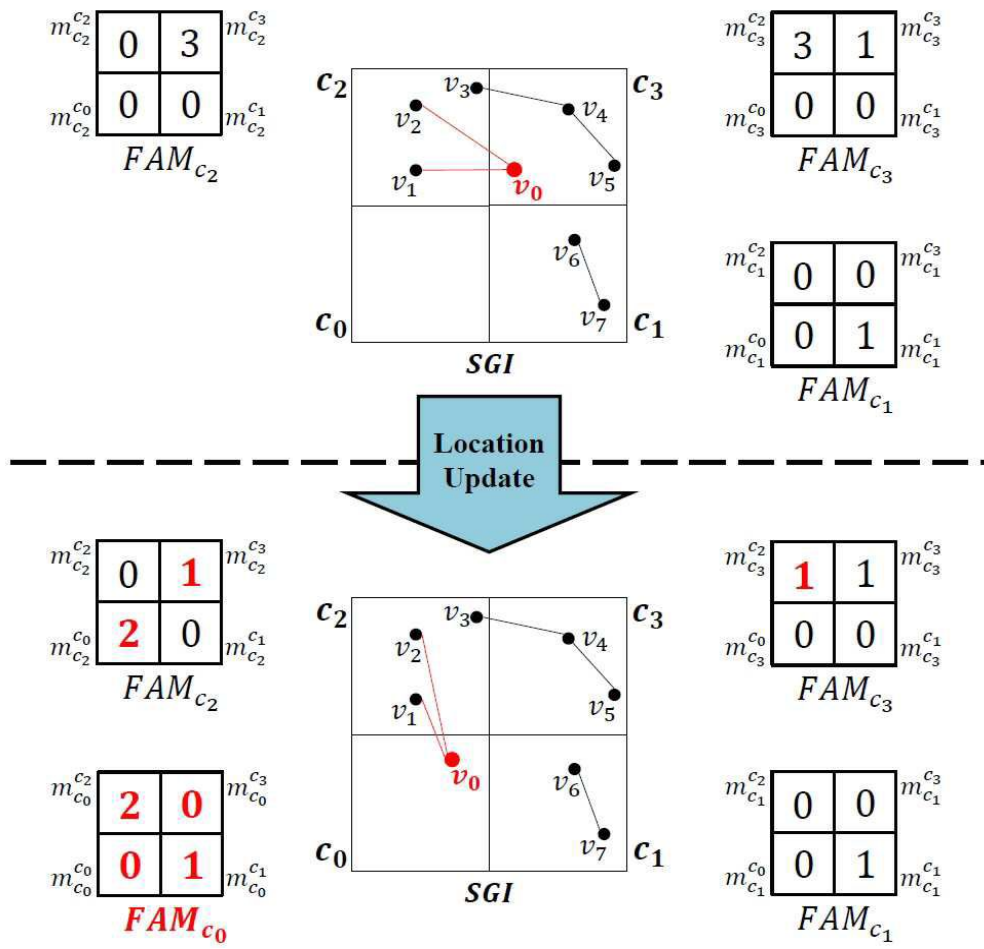
도면16a

Algorithm 7: Function UpdateWithFAM(data object v 's update udt_v).	
1	Function UpdateWithFAM(data object v 's update udt_v):
2	Set p_v^{old} = old position of v , p_v^{new} = new position of v ;
3	Set c_v^{old} = a cell covering v at p_v^{old} , c_v^{new} = a cell covering v at p_v^{new} ;
4	Set $M_{c_v^{old}}$ = FAM of c_v^{old} , $M_{c_v^{new}}$ = FAM of c_v^{new} ;
5	if udt_v is friend update then // $p_v^{old} = p_v^{new}$, $c_v^{old} = c_v^{new}$
6	if udt_v contains new friend f then
7	Insert f into v 's 1-hop friend list V_v^1 ;
8	if $c_f \notin C_{c_v^{new}}^1$ then // $C_{c_v^{new}}^1$ = 1-hop friend-cell list of c_v^{new}
9	Insert c_f into $C_{c_v^{new}}^1$; // SU
10	$M_{c_v^{new}}$'s element $m_{c_f}^{c_v^{new}} = 1 + m_{c_f}^{c_v^{new}}$;
11	$m_{c_v^{new}}^{c_f} = 1 + m_{c_v^{new}}^{c_f}$;
12	else // udt_v contains existing friend f'
13	Delete f' from V_v^1 ;
14	CheckFAM (c_v^{new} , $c_{f'}$);
15	CheckFAM ($c_{f'}$, c_v^{new});
16	else // udt_v is location update, $p_v^{old} \neq p_v^{new}$
17	if $c_v^{old} \neq c_v^{new}$ then
18	Insert v into c_v^{new} ; // GU
19	for each friend v' of v do
20	if $c_{v'} \notin C_{c_v^{new}}^1$ then
21	Insert $c_{v'}$ into $C_{c_v^{new}}^1$; // SU
22	$m_{c_{v'}}^{c_v^{new}} = 1 + m_{c_{v'}}^{c_v^{new}}$;
23	$m_{c_v^{new}}^{c_{v'}} = 1 + m_{c_v^{new}}^{c_{v'}};$
24	CheckFAM (c_v^{old} , $c_{v'}$);
25	CheckFAM ($c_{v'}$, c_v^{old});
26	Delete v from c_v^{old} ; // GU

도면16b

Algorithm 8: Function CheckFAM(cell c_v , cell $c_{v'}$).	
1	Function CheckFAM(cell c_v , cell $c_{v'}$):
2	$m_{c_v}^{c_{v'}} = -1 + m_{c_v}^{c_{v'}};$
3	if $m_{c_v}^{c_{v'}} = 0$ then
4	Delete v' from $C_{c_v}^1$; // SU

도면17



도면18

100

