Density-Based Clustering of Places Using Geo-Social Network Data

BY

GAURI V NAIR, ROLL NO: 171CO116

ARJUN S, ROLL NO: 171CO209

DEPT. OF COMPUTER SCIENCE AND ENGINEERING,

NITK SURATHKAL

Spatial Clustering

- Unsupervised grouping of places into clusters
- Important applications in Urban Planning and Marketing
- Disadvantage - Disregard important information about:
  - Who are related to the clustered places (People)
  - When are they related to the clustered places (Time)
- Example: K-means
- This disadvantage can be overcome by density-based clustering

Density-Based Clustering

- Divides a large collection of points into densely populated regions
- Most appropriate clustering paradigm for:
  - Spatial Data with low dimensionality
- Density-Based Clusters
  - have arbitrary shapes and sizes
  - exclude objects in outliers
- Example: DBSCAN (Density-based spatial clustering of applications with noise)

Geo-Social Network Applications

Gowalla

FOURSQUARE

brightkite

Source: Wikipedia

- ❑ Allows users to:
  - capture their geographic locations
  - Share them in the social network

  by an operation called checkin (<uid, pid, time>)

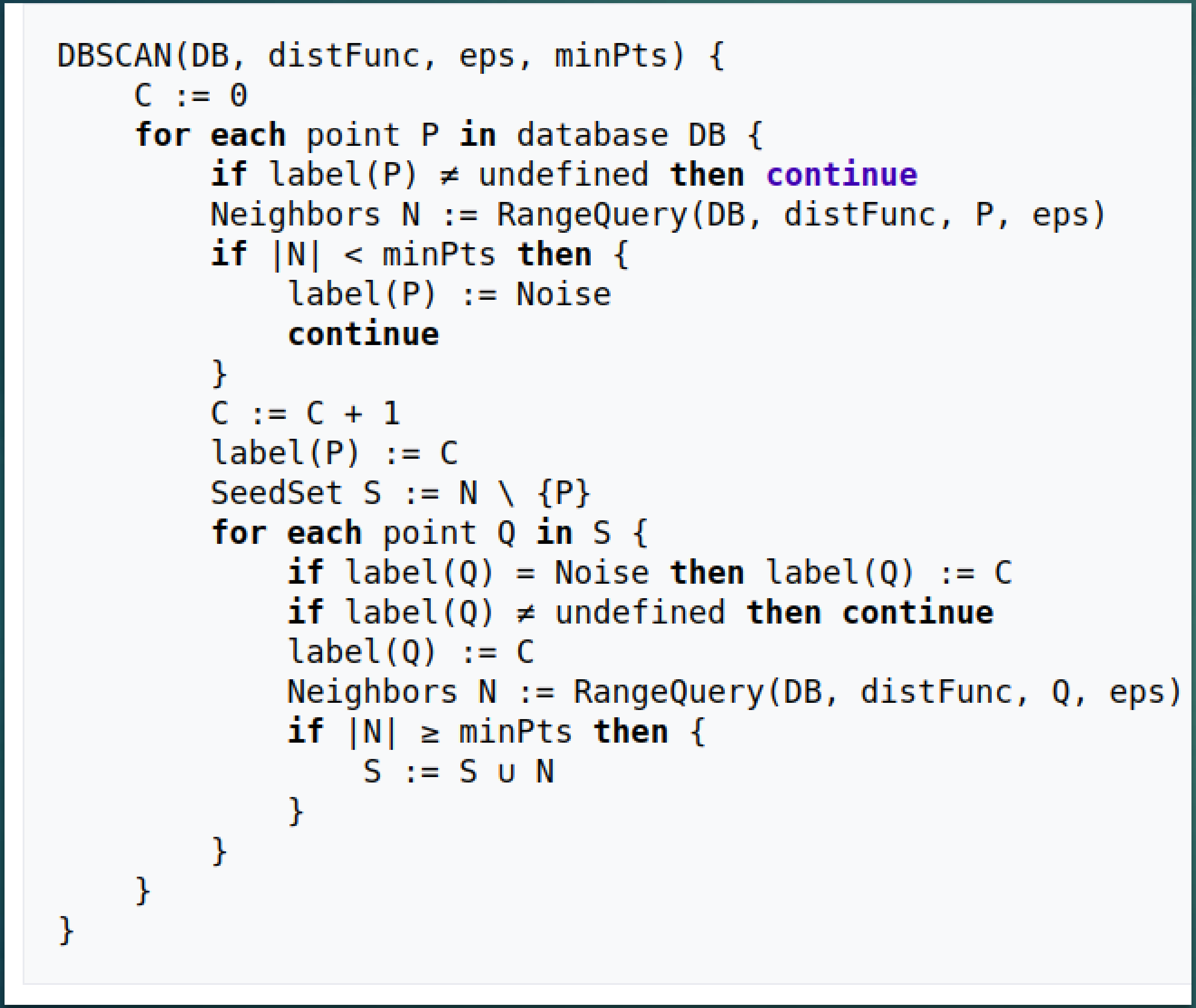
Clustering of Places in a GeoSN Network - Applications

- Generalization and Characterization of Places
  - Geographic data analysis
    - Urban planning - identifying regions which have uniform demographic statistics
- Data cleaning
  - cleaning of semantics, which are given to places being in the same cluster
- Marketing
  - Geo-Social-temporal Clusters
    - understanding the shopping habits of various social groups

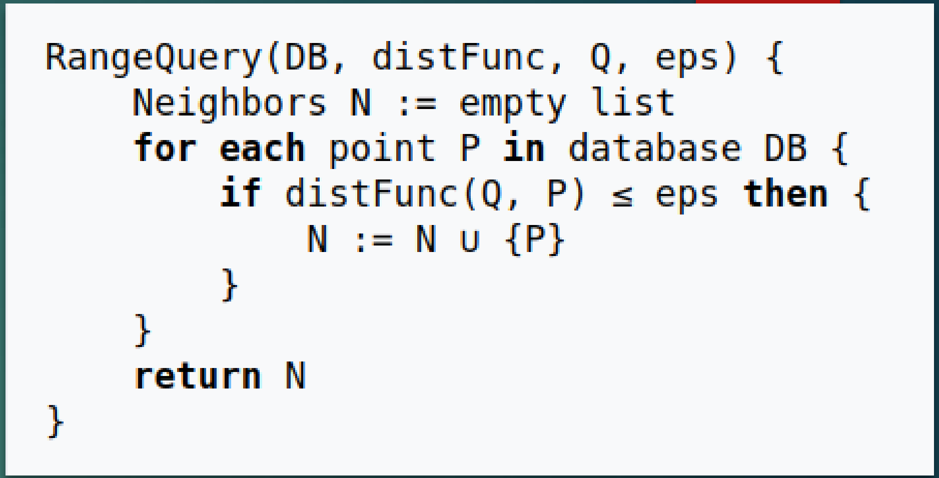
DBSCAN

- DBSCAN is one of the most common data clustering algorithms – proposed in 1996.
- Finds the spatial *eps-neighborhood* of each point p in the dataset
- If the number of places in *eps-neighborhood* is no less than *MinPts* – *p* is called a *core point* -> it will form a cluster or will be a part of cluster.
- Dense *eps-neighborhoods* are put into the same cluster if they contain the cores of each other.

DBSCAN Algorithm



```
DBSCAN(DB, distFunc, eps, minPts) {
    C := 0
    for each point P in database DB {
        if label(P) ≠ undefined then continue
        Neighbors N := RangeQuery(DB, distFunc, P, eps)
        if |N| < minPts then {
            label(P) := Noise
            continue
        }
        C := C + 1
        label(P) := C
        SeedSet S := N \ {P}
        for each point Q in S {
            if label(Q) = Noise then label(Q) := C
            if label(Q) ≠ undefined then continue
            label(Q) := C
            Neighbors N := RangeQuery(DB, distFunc, Q, eps)
            if |N| ≥ minPts then {
                S := S ∪ N
            }
        }
    }
}
```

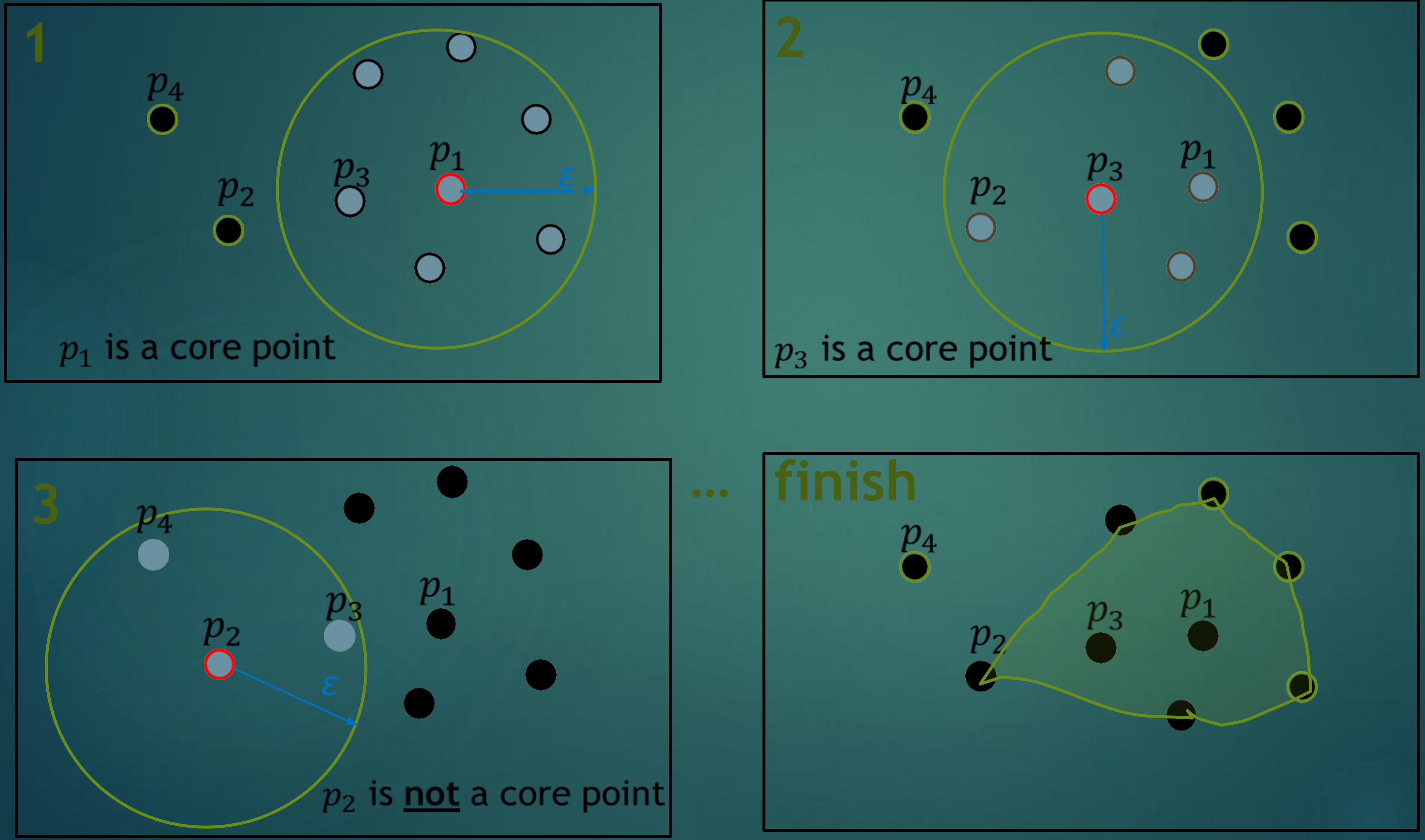


```
RangeQuery(DB, distFunc, Q, eps) {
    Neighbors N := empty list
    for each point P in database DB {
        if distFunc(Q, P) ≤ eps then {
            N := N ∪ {P}
        }
    }
    return N
}
```

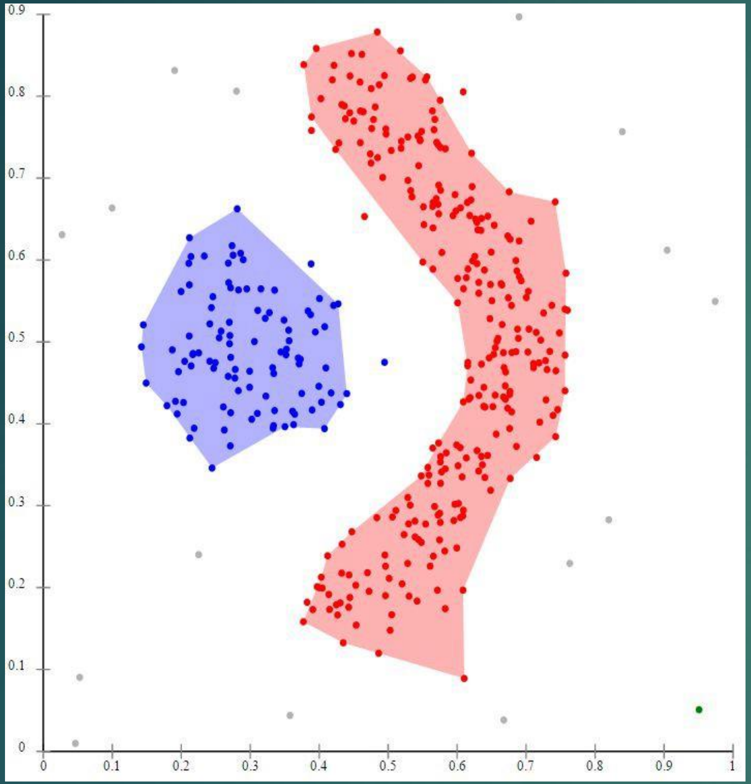
- *Source: Wikipedia*

Example

MinPts = 4

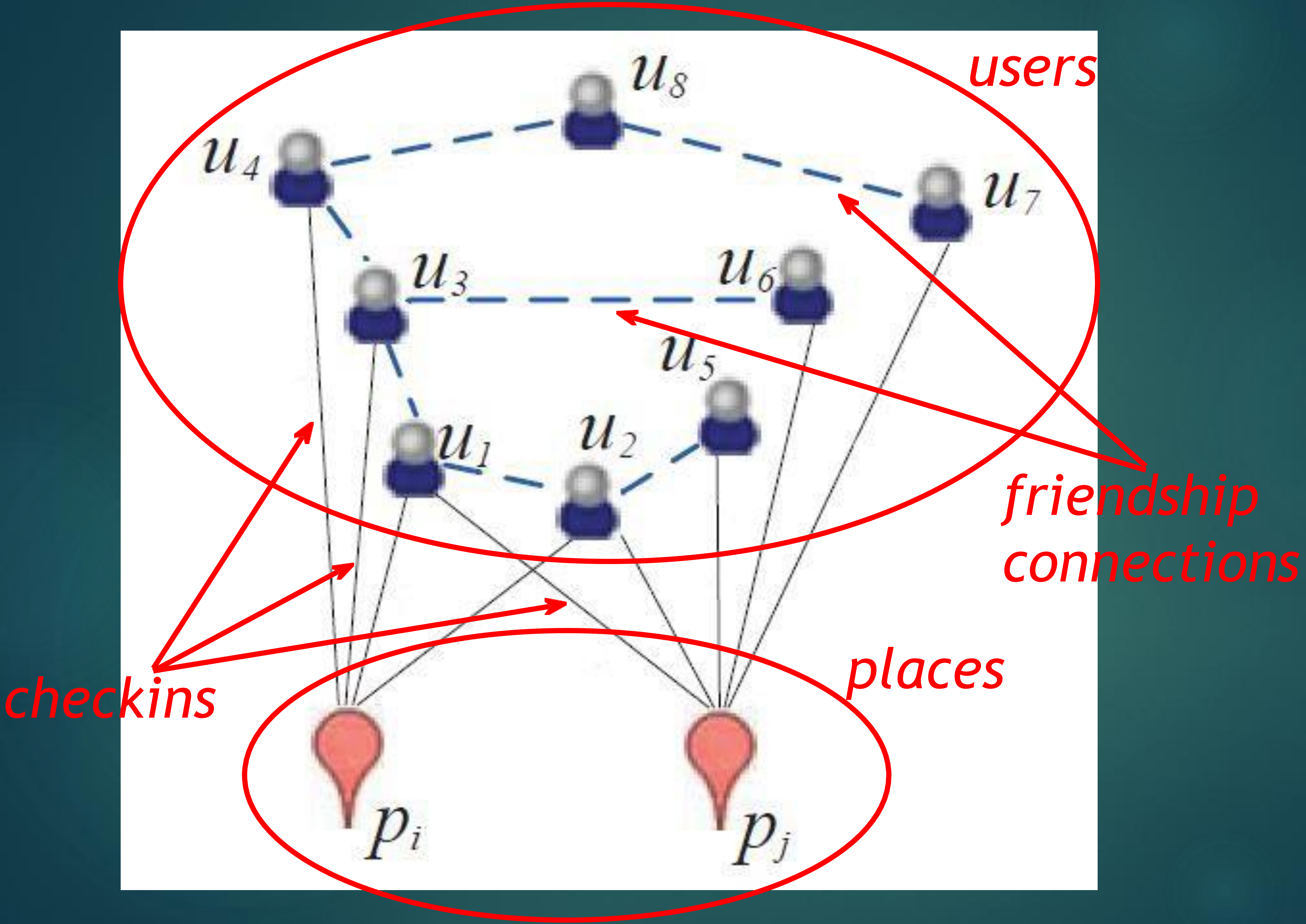


DBSCAN Example

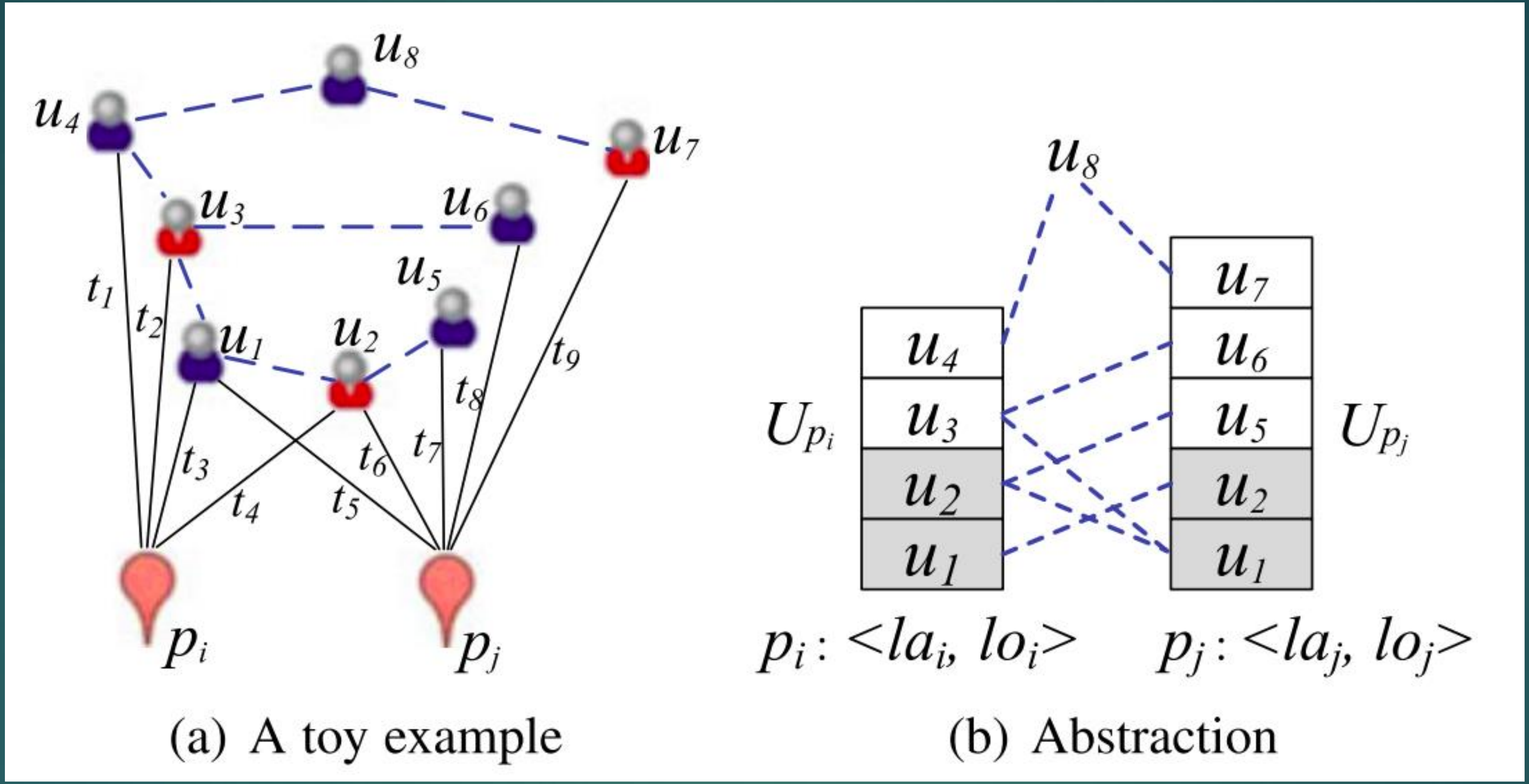


Source: Wikipedia

DBSCAN can find non-linearly separable clusters. This dataset cannot be adequately clustered with k-means or Gaussian Mixture EM clustering.


users
u_8
u_4
u_7
u_3
u_6
u_5
u_1
u_2
friendship connections
checkins
places
p_i
p_j

Example and Storage Structure of GeoSNs



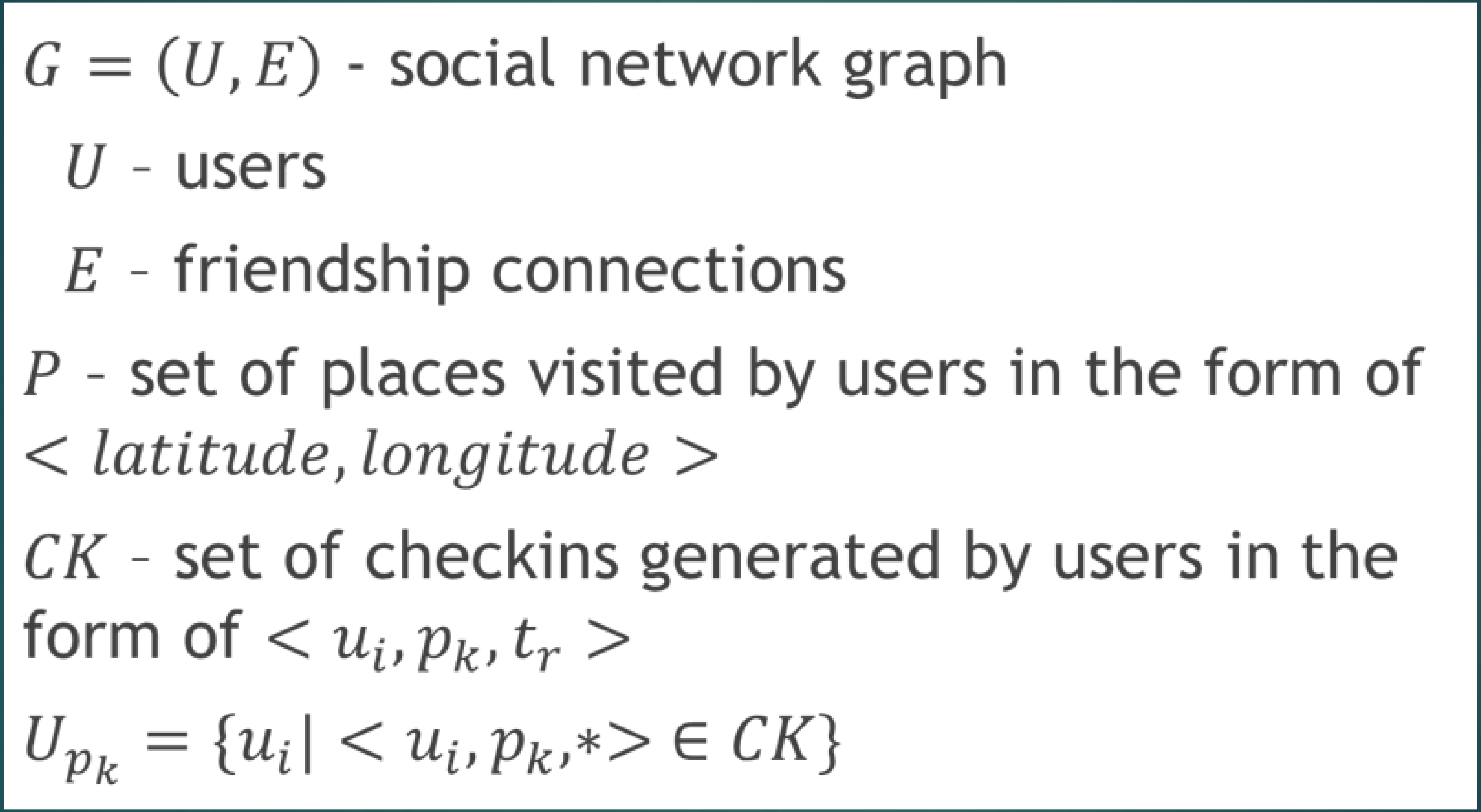
(a) A toy example (b) Abstraction

Source: Dingming Wu, Jieming Shi, and Nikos Mamoulis, *Density-Based Place Clustering Using Geo-Social Network Data - Vol.30*, IEEE Transactions on Knowledge and Data Engineering, May 2018.

Density-Based Clustering Places in Geo-Social Networks (DCPGS)

- Extends DBSCAN to consider both the spatial and social distance between places
- Define U_{pi} the set of users who checked in *pi*
- Social Distance between places *pi* and *pj*, we use:
  - Set of common users in U_{pi} and U_{pj}
  - Users in one place's record who are friends with visitors of the other place.

Input



$G = (U, E)$ - social network graph

U - users

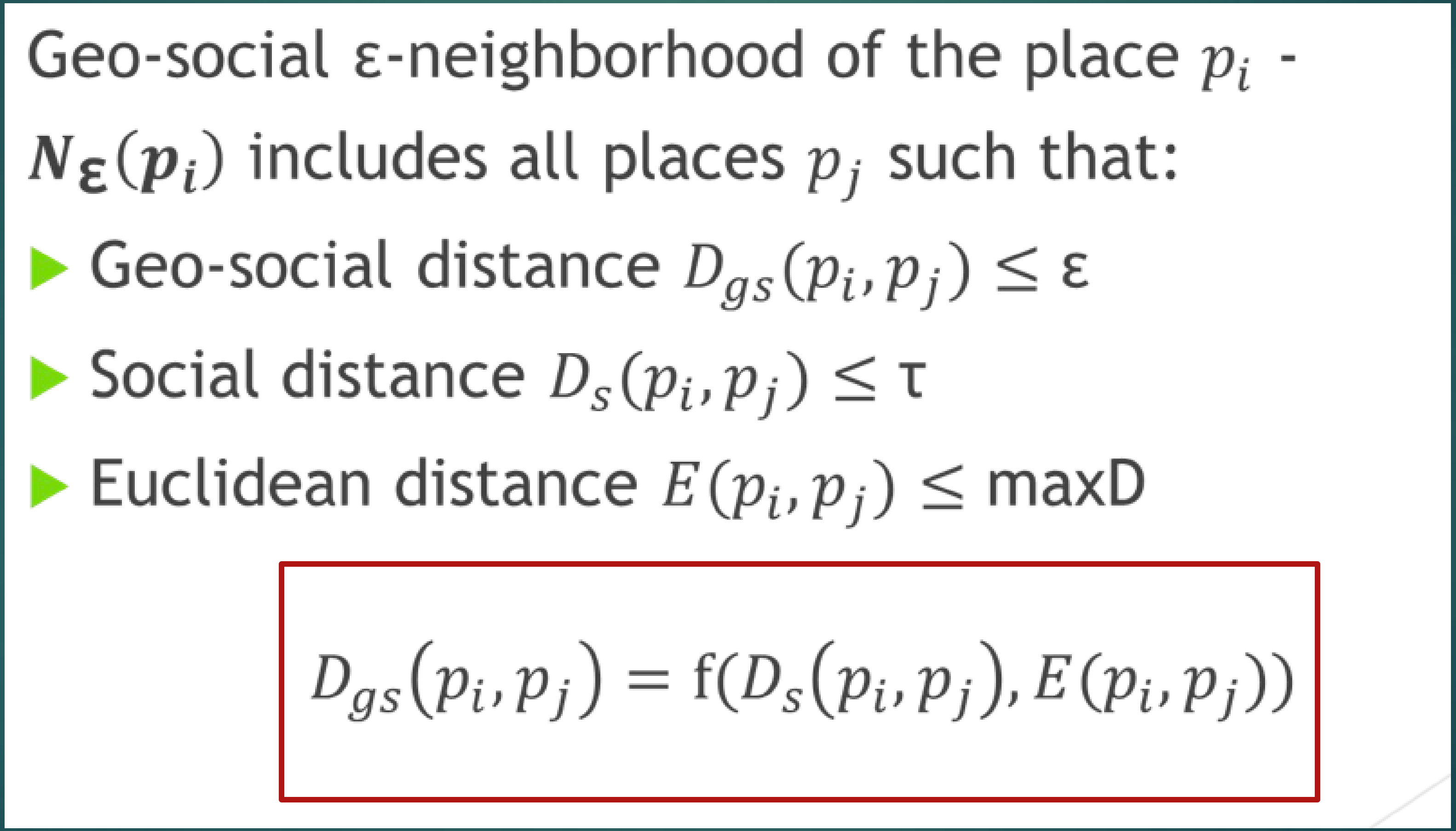
E - friendship connections

P - set of places visited by users in the form of $< latitude, longitude >$

CK - set of checkins generated by users in the form of $< u_i, p_k, t_r >$

$U_{p_k} = \{u_i | < u_i, p_k, * > \in CK\}$

DCPGS Geo-Social ε-Neighborhood Definition



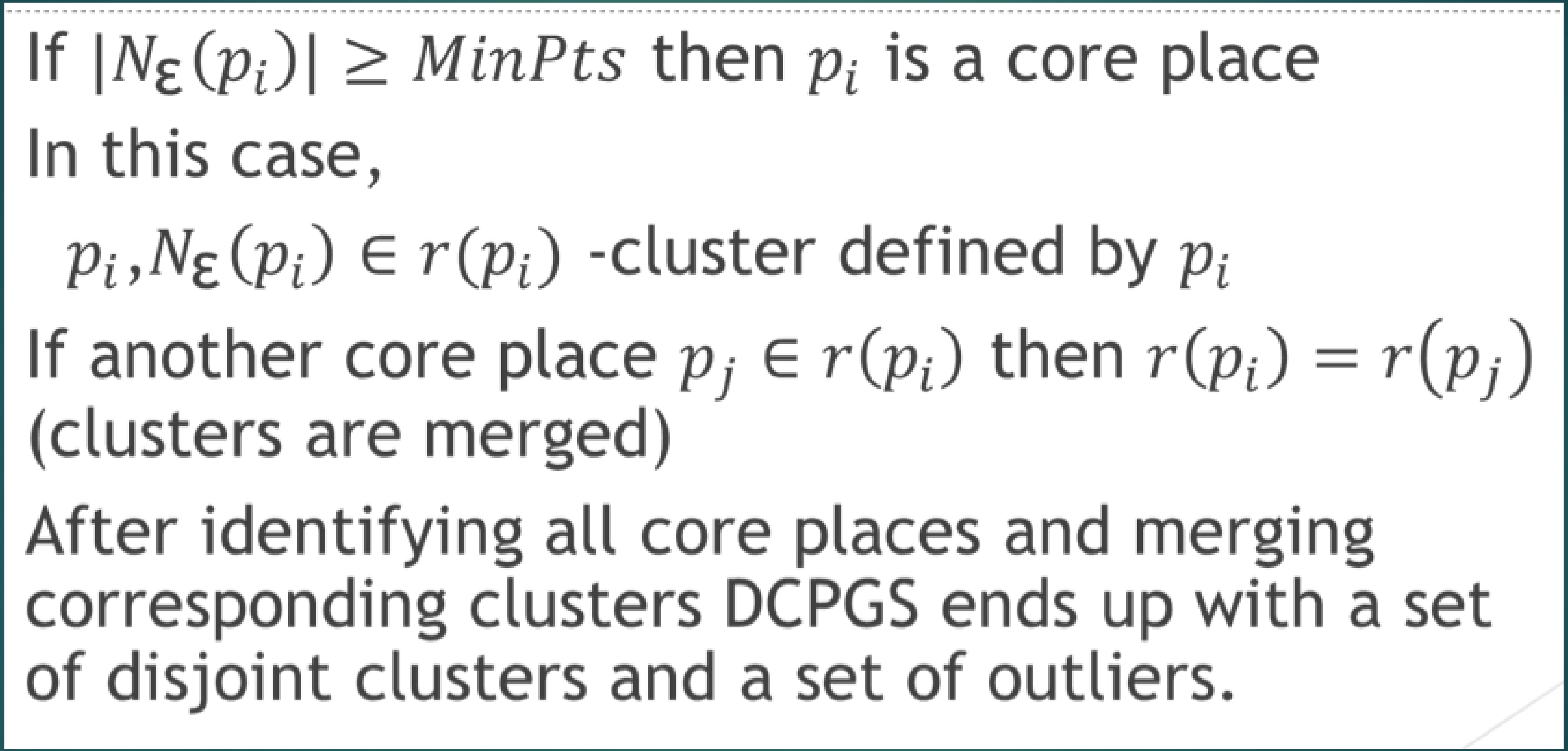
Geo-social ε-neighborhood of the place p_i -

$\boldsymbol{N_{\varepsilon}(p_i)}$ includes all places p_j such that:

- Geo-social distance $D_{gs}(p_i, p_j) \leq \varepsilon$
- Social distance $D_s(p_i, p_j) \leq \tau$
- Euclidean distance $E(p_i, p_j) \leq \mathrm{maxD}$

$$\boxed{D_{gs}(p_i, p_j) = \mathrm{f}(D_s(p_i, p_j), E(p_i, p_j))}$$

DCPGS Algorithm idea in a nutshell



If $|N_{\varepsilon}(p_i)| \geq MinPts$ then p_i is a core place

In this case,

$p_i, N_{\varepsilon}(p_i) \in r(p_i)$ -cluster defined by p_i

If another core place $p_j \in r(p_i)$ then $r(p_i) = r(p_j)$ (clusters are merged)

After identifying all core places and merging corresponding clusters DCPGS ends up with a set of disjoint clusters and a set of outliers.

Distance Functions

Normalized Euclidean distance $D_p(p_i, p_j) = \frac{E(p_i,p_j)}{maxD}$

$p_i \in r(p_j)$ then $0 \leq D_p(p_i, p_j) \leq 1$

$$D_{gs}(p_i, p_j) = \omega \cdot D_p(p_i, p_j) + (1 - \omega) \cdot D_s(p_i, p_j)$$

$\omega \in [0,1]$

Social Distance

$$CU_{ij} = \left\{u_a \in U_{p_i} \,\middle|\, u_a \in U_{p_j} \ or \ \exists u_b \in U_{p_j}, (u_a, u_b) \in E\right\} \cup \left\{u_a \in U_{p_j} \,\middle|\, u_a \in U_{p_i} \ or \ \exists u_b \in U_{p_i}, (u_a, u_b) \in E\right\}$$

$$\boxed{D_s(p_i, p_j) = 1 - \frac{|CU_{ij}|}{|U_{p_i} \cup U_{p_j}|}}$$

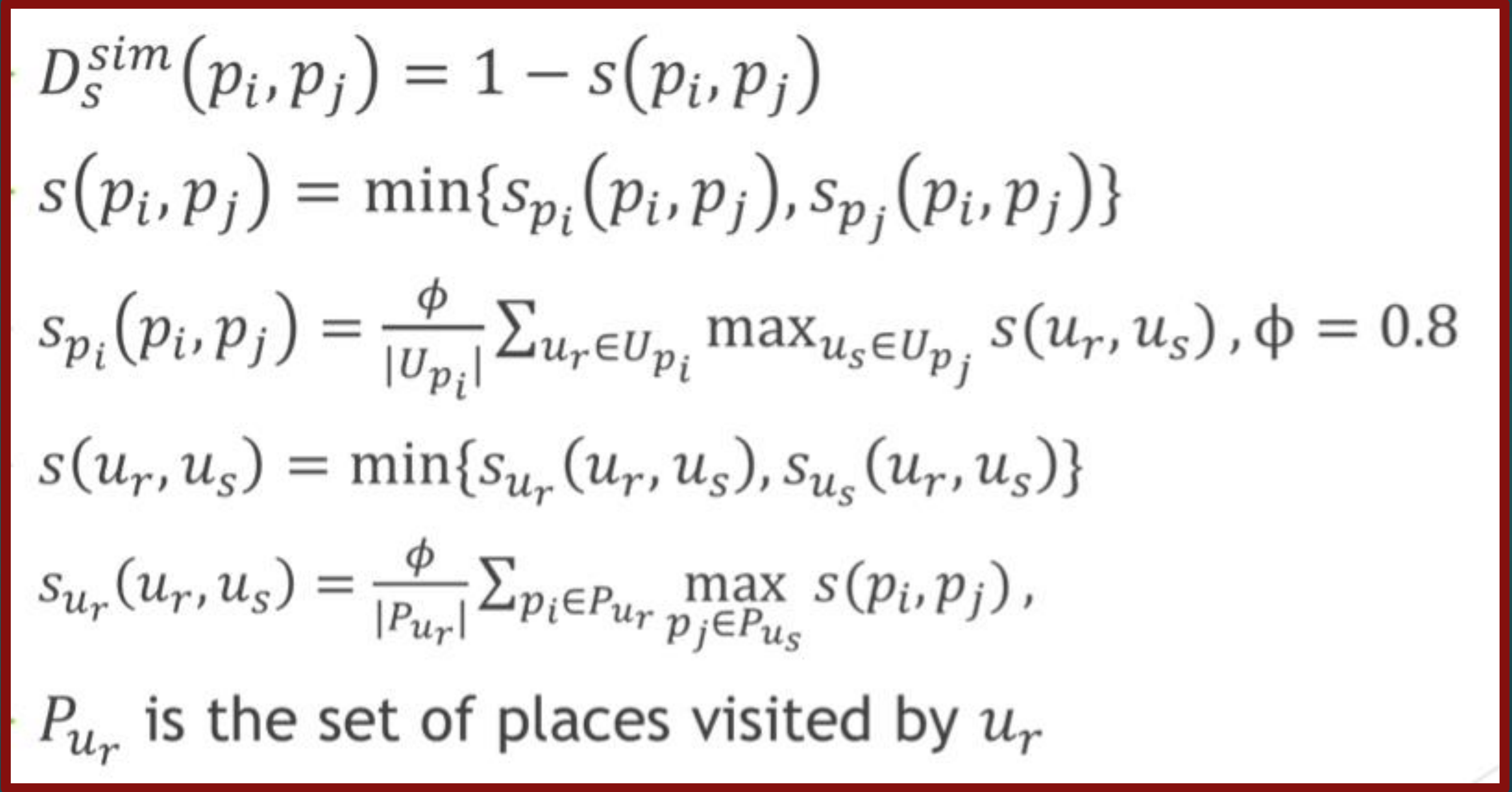
Alternatives to D$_s$
(1) Jaccard

$$J(p_i, p_j) = \frac{|U_{p_i} \cap U_{p_j}|}{|U_{p_i} \cup U_{p_j}|}$$

$$\boxed{D_s^{Jac}(p_i, p_j) = 1 - J(p_i, p_j)}$$

This method disregards the social network.

Alternatives to D_s
(2) SimRank



$$D_S^{sim}(p_i, p_j) = 1 - s(p_i, p_j)$$

$$s(p_i, p_j) = \min\{s_{p_i}(p_i, p_j), s_{p_j}(p_i, p_j)\}$$

$$s_{p_i}(p_i, p_j) = \frac{\phi}{|U_{p_i}|} \sum_{u_r \in U_{p_i}} \max_{u_s \in U_{p_j}} s(u_r, u_s)\, , \phi = 0.8$$

$$s(u_r, u_s) = \min\{s_{u_r}(u_r, u_s), s_{u_s}(u_r, u_s)\}$$

$$s_{u_r}(u_r, u_s) = \frac{\phi}{|P_{u_r}|} \sum_{p_i \in P_{u_r}} \max_{p_j \in P_{u_s}} s(p_i, p_j)\, ,$$

P_{u_r} is the set of places visited by u_r

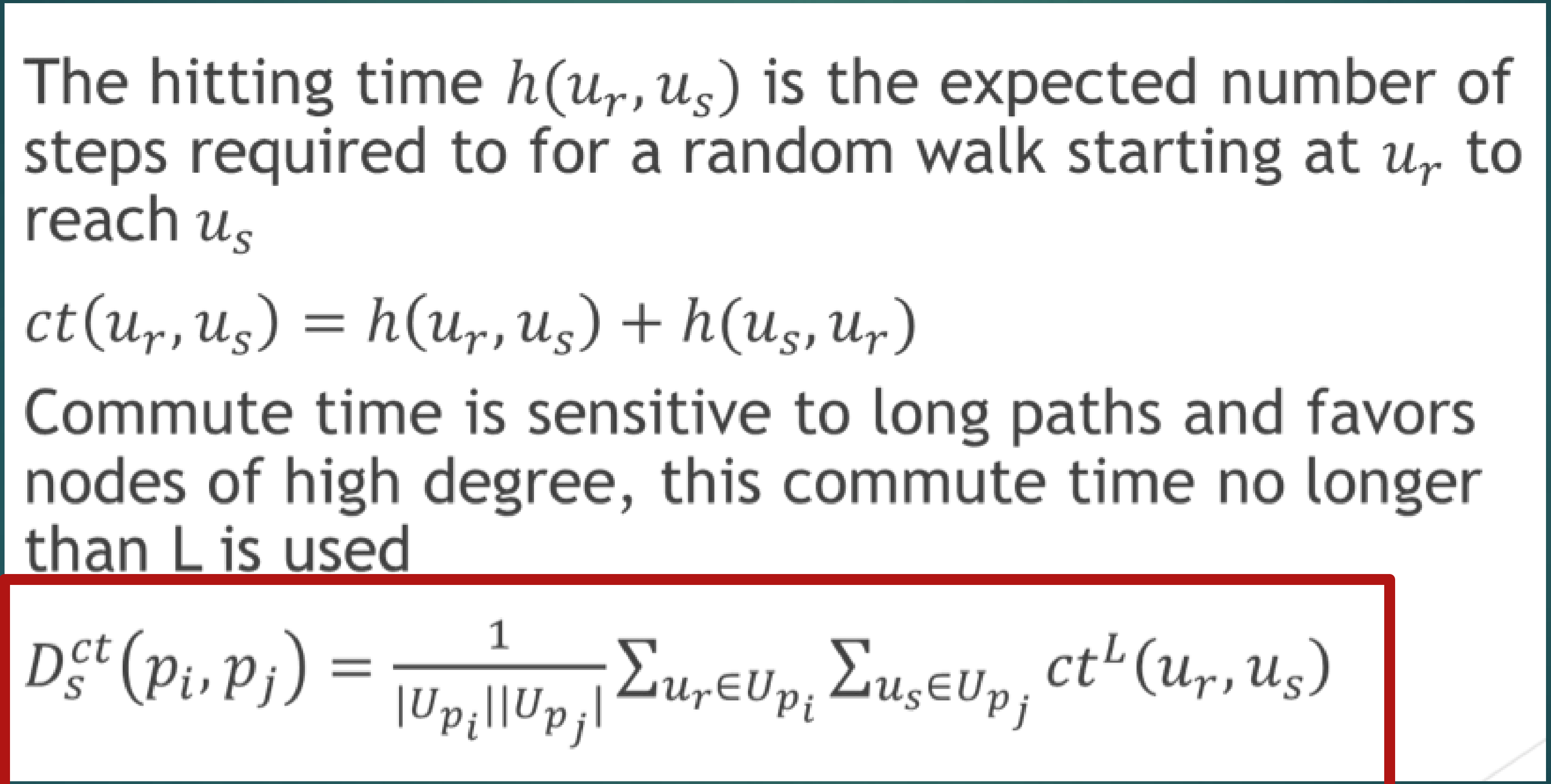
Alternatives to D$_s$
(3) Katz

$$\mathcal{K}_a(u_r, u_s) = \sum_{l=1}^{L} \beta^l |paths_{u_r,u_s}^l|$$

$paths_{u_r,u_s}^l$ - the set of all length-l paths from u_r to u_s

$$\boxed{D_S^{Katz}(p_i, p_j) = 1 - \frac{1}{|U_{p_i}||U_{p_j}|} \sum_{u_r \in U_{p_i}} \sum_{u_s \in U_{p_j}} \mathcal{K}_a(u_r, u_s)}$$

Alternatives to D$_s$ (4) Commute Time



The hitting time $h(u_r, u_s)$ is the expected number of steps required to for a random walk starting at u_r to reach u_s

$$ct(u_r, u_s) = h(u_r, u_s) + h(u_s, u_r)$$

Commute time is sensitive to long paths and favors nodes of high degree, this commute time no longer than L is used

$$\boxed{D_S^{ct}(p_i, p_j) = \frac{1}{|U_{p_i}||U_{p_j}|} \sum_{u_r \in U_{p_i}} \sum_{u_s \in U_{p_j}} ct^L(u_r, u_s)}$$

Incorporating Temporal Information - Methods

- History-Frame Geo-Social Clustering
- Damping Window

$$D_{TS}(p_i, p_j) = 1 - \left(\sum_{u \in CU_{ij} \cap U_{p_i} \cap U_{p_j}} \max\{u^w(p_i), u^w(p_j)\} + \sum_{u \in CU_{ij} \cap (U_{p_i} \setminus U_{p_j})} u^w(p_i) + \sum_{u \in CU_{ij} \cap (U_{p_j} \setminus U_{p_i})} u^w(p_j) \right) / |U_{p_i} \cup U_{p_j}|$$

- Temporally Contributing Users

$$D_{TS}(p_i, p_j) = 1 - |TCU_{ij}| / |U_{p_i} \cup U_{p_j}|$$

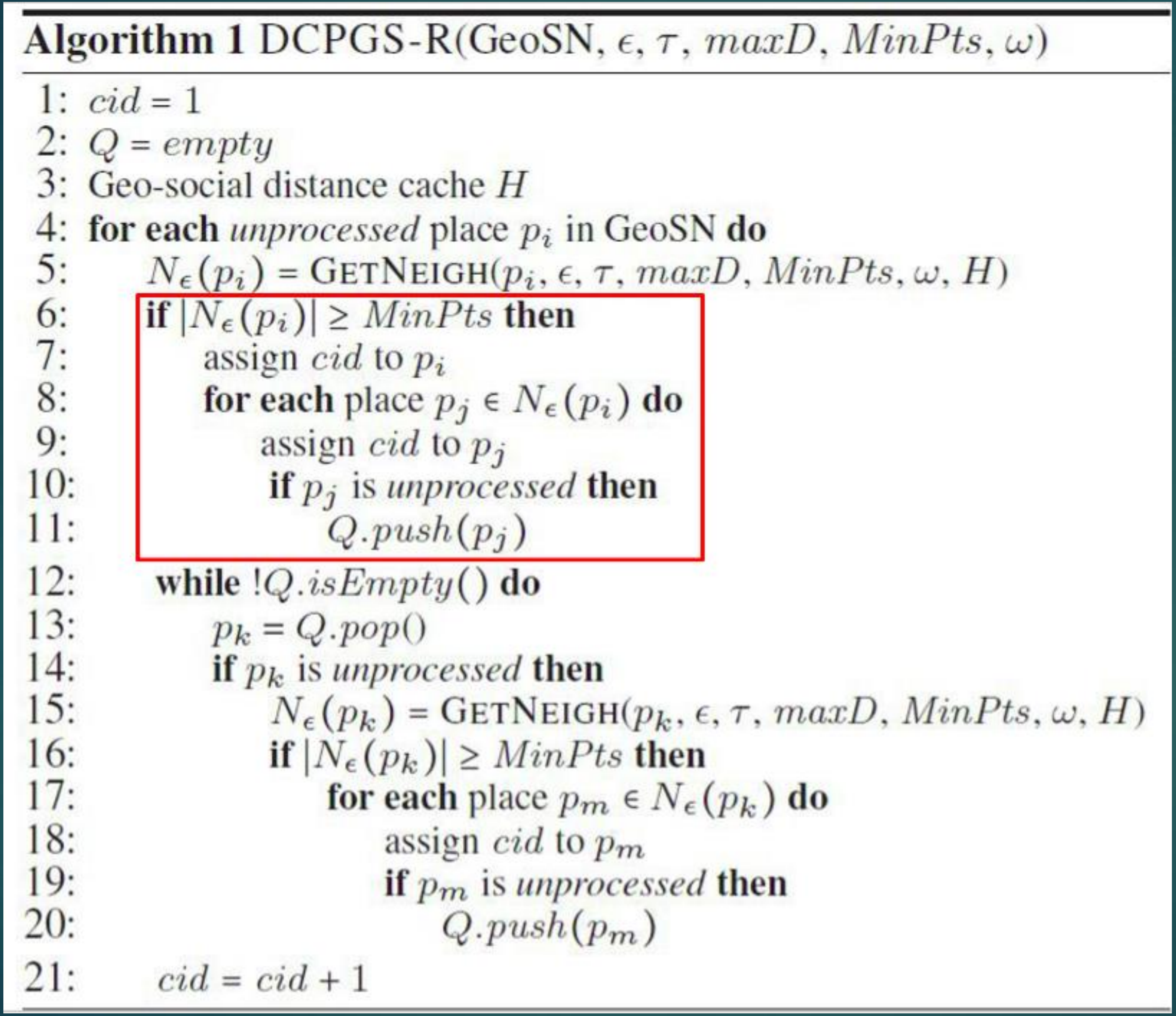
Algorithms

DCPGS-R AND DCPGS-G

DCPGS-R: R-tree based

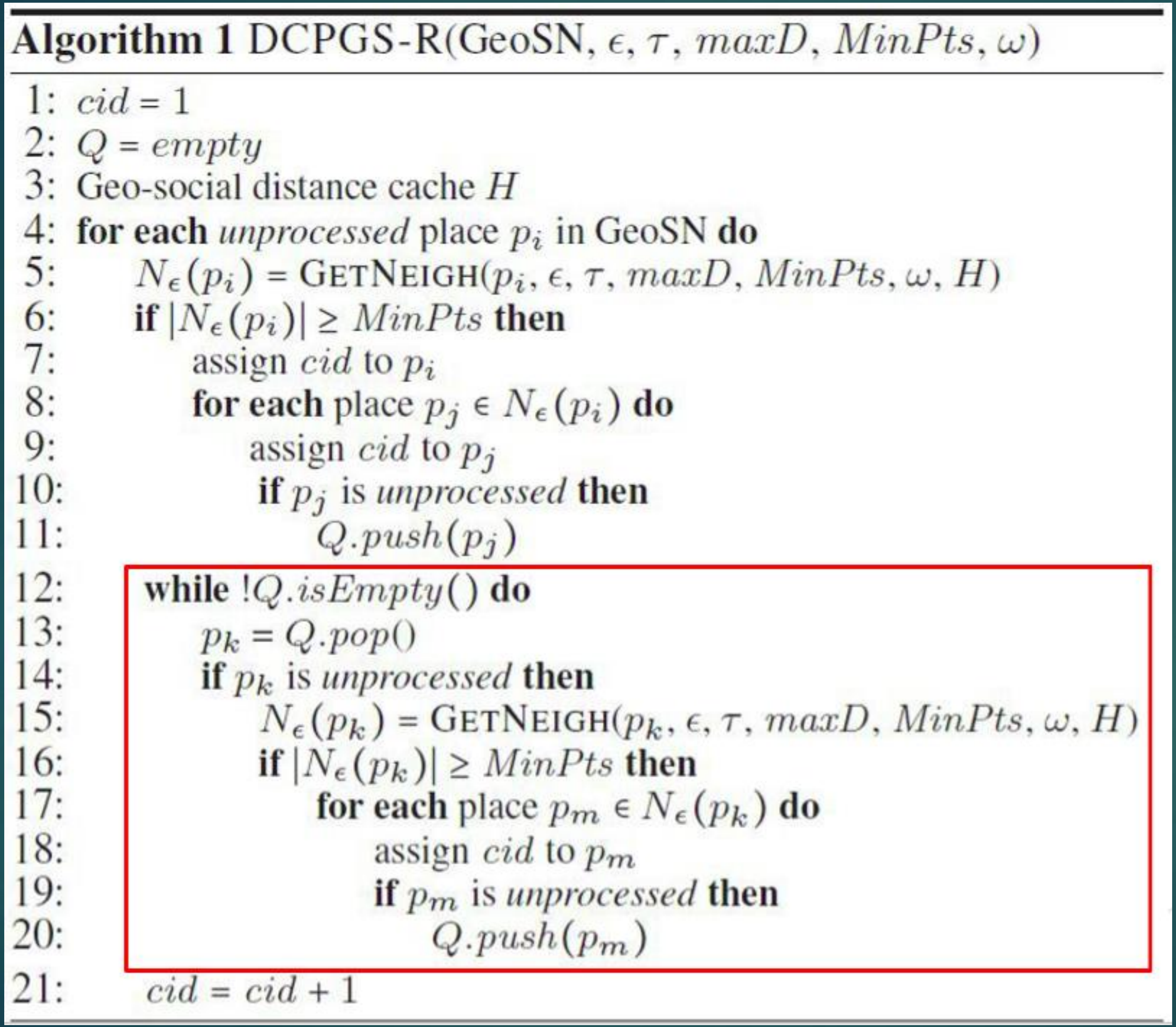
- The algorithm uses R-Tree to facilitate the search of geo-social ε-neighborhood for a given place

- For the sake of efficiency the social network is stored in a hash table – each pair of friends as an entry



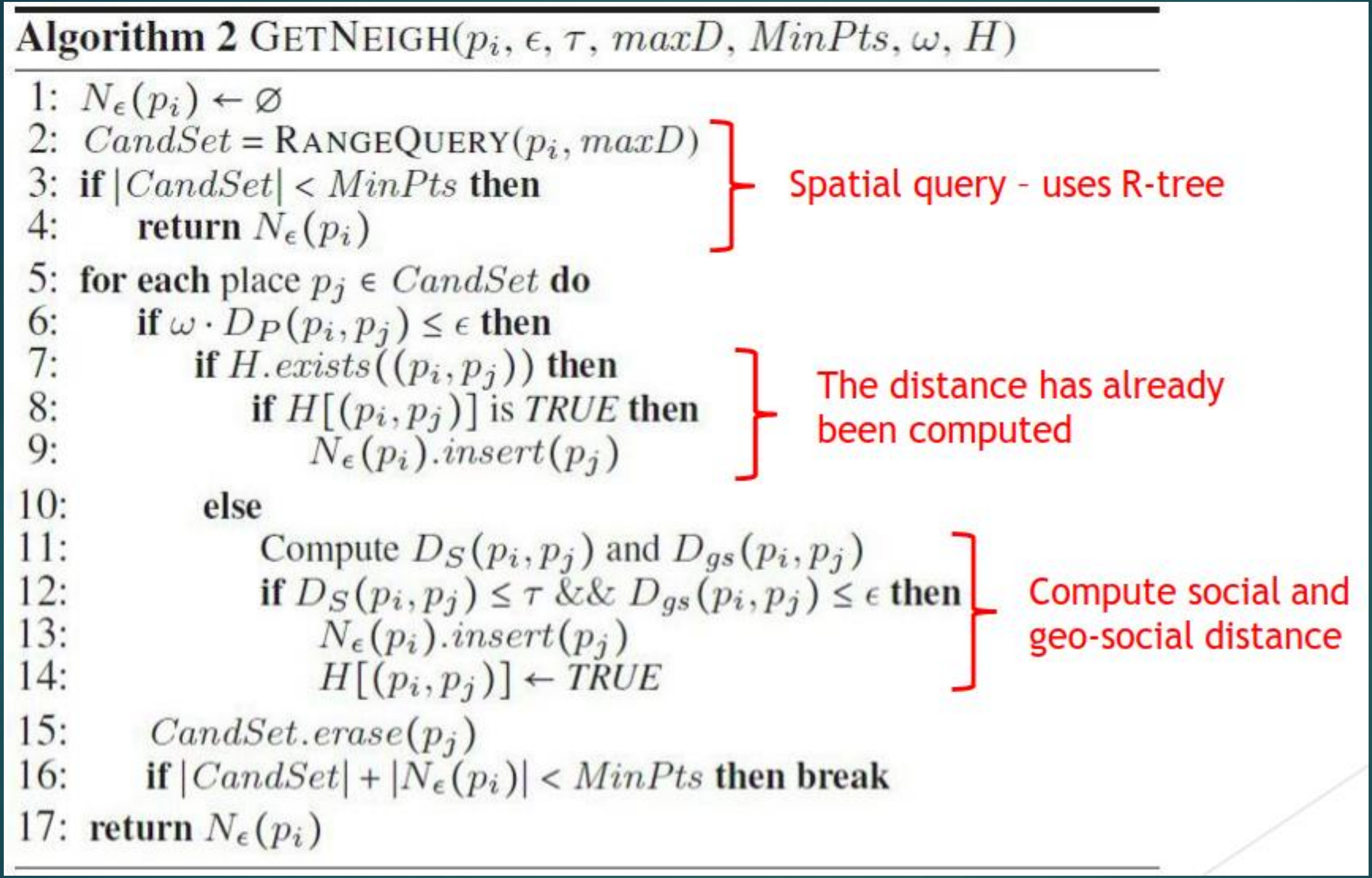
Algorithm 1 DCPGS-R(GeoSN, ϵ, τ, $maxD$, $MinPts$, ω)

```
1:  $cid = 1$
2:  $Q = empty$
3:  Geo-social distance cache $H$
4:  for each unprocessed place $p_i$ in GeoSN do
5:      $N_\epsilon(p_i) = \text{GetNeigh}(p_i, \epsilon, \tau, maxD, MinPts, \omega, H)$
6:      if $|N_\epsilon(p_i)| \geq MinPts$ then
7:          assign $cid$ to $p_i$
8:          for each place $p_j \in N_\epsilon(p_i)$ do
9:              assign $cid$ to $p_j$
10:             if $p_j$ is unprocessed then
11:                 $Q.push(p_j)$
12:     while $!Q.isEmpty()$ do
13:         $p_k = Q.pop()$
14:         if $p_k$ is unprocessed then
15:             $N_\epsilon(p_k) = \text{GetNeigh}(p_k, \epsilon, \tau, maxD, MinPts, \omega, H)$
16:             if $|N_\epsilon(p_k)| \geq MinPts$ then
17:                 for each place $p_m \in N_\epsilon(p_k)$ do
18:                     assign $cid$ to $p_m$
19:                     if $p_m$ is unprocessed then
20:                         $Q.push(p_m)$
21:     $cid = cid + 1$
```



Algorithm 1 DCPGS-R(GeoSN, ϵ, τ, $maxD$, $MinPts$, ω)

```
1:  $cid = 1$
2:  $Q = empty$
3:  Geo-social distance cache $H$
4:  for each unprocessed place $p_i$ in GeoSN do
5:      $N_\epsilon(p_i)$ = GetNeigh($p_i$, $\epsilon$, $\tau$, $maxD$, $MinPts$, $\omega$, $H$)
6:      if $|N_\epsilon(p_i)| \geq MinPts$ then
7:          assign $cid$ to $p_i$
8:          for each place $p_j \in N_\epsilon(p_i)$ do
9:              assign $cid$ to $p_j$
10:             if $p_j$ is unprocessed then
11:                 $Q.push(p_j)$
12:     while $!Q.isEmpty()$ do
13:         $p_k = Q.pop()$
14:         if $p_k$ is unprocessed then
15:             $N_\epsilon(p_k)$ = GetNeigh($p_k$, $\epsilon$, $\tau$, $maxD$, $MinPts$, $\omega$, $H$)
16:             if $|N_\epsilon(p_k)| \geq MinPts$ then
17:                 for each place $p_m \in N_\epsilon(p_k)$ do
18:                     assign $cid$ to $p_m$
19:                     if $p_m$ is unprocessed then
20:                         $Q.push(p_m)$
21:     $cid = cid + 1$
```



Algorithm 2 GETNEIGH(p_i, ϵ, τ, $maxD$, $MinPts$, ω, H)

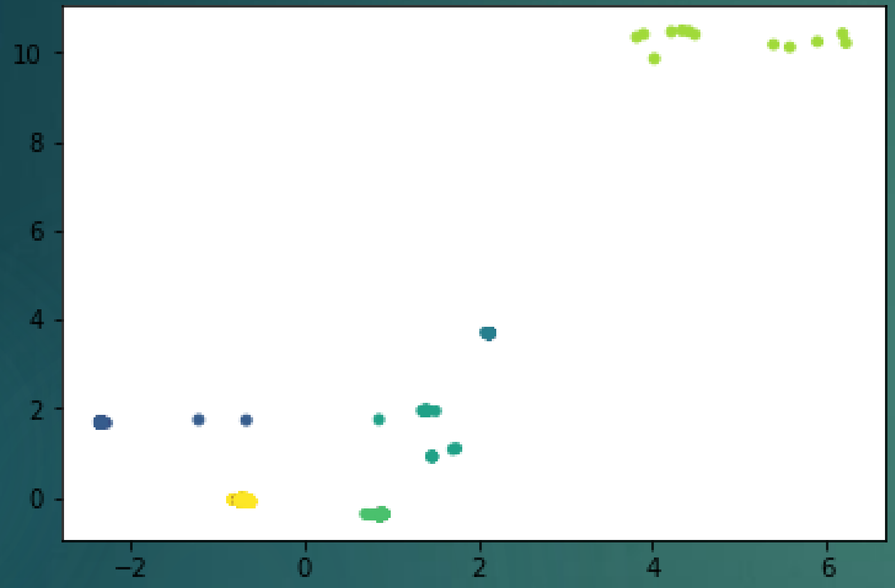
```
 1: $N_\epsilon(p_i) \leftarrow \varnothing$
 2: $CandSet$ = RANGEQUERY($p_i, maxD$)
 3: if $|CandSet| < MinPts$ then
 4:     return $N_\epsilon(p_i)$
 5: for each place $p_j \in CandSet$ do
 6:     if $\omega \cdot D_P(p_i, p_j) \leq \epsilon$ then
 7:         if $H.exists((p_i, p_j))$ then
 8:             if $H[(p_i, p_j)]$ is $TRUE$ then
 9:                 $N_\epsilon(p_i).insert(p_j)$
10:         else
11:             Compute $D_S(p_i, p_j)$ and $D_{gs}(p_i, p_j)$
12:             if $D_S(p_i, p_j) \leq \tau$ && $D_{gs}(p_i, p_j) \leq \epsilon$ then
13:                 $N_\epsilon(p_i).insert(p_j)$
14:                 $H[(p_i, p_j)] \leftarrow TRUE$
15:     $CandSet.erase(p_j)$
16:     if $|CandSet| + |N_\epsilon(p_i)| < MinPts$ then break
17: return $N_\epsilon(p_i)$
```

Spatial query – uses R-tree

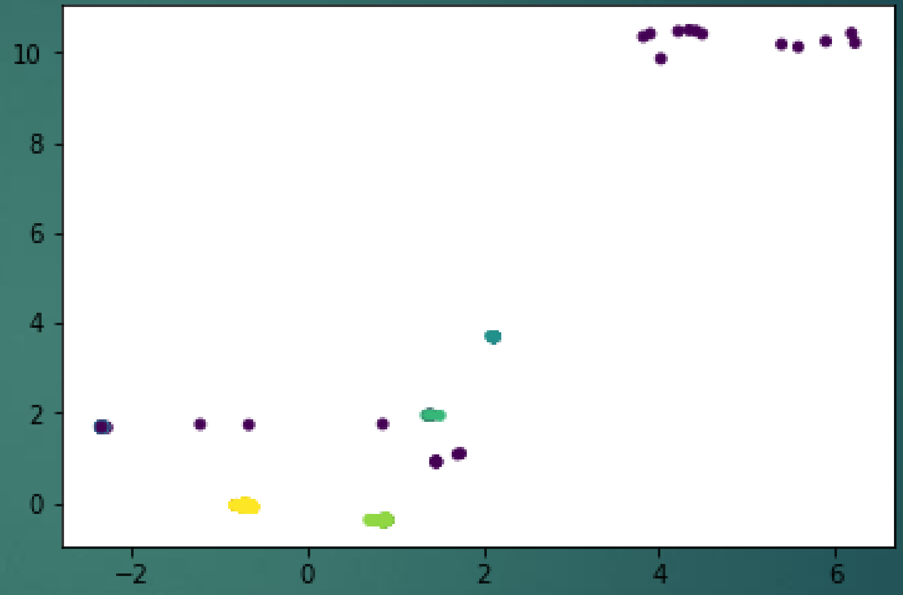
The distance has already been computed

Compute social and geo-social distance

Results

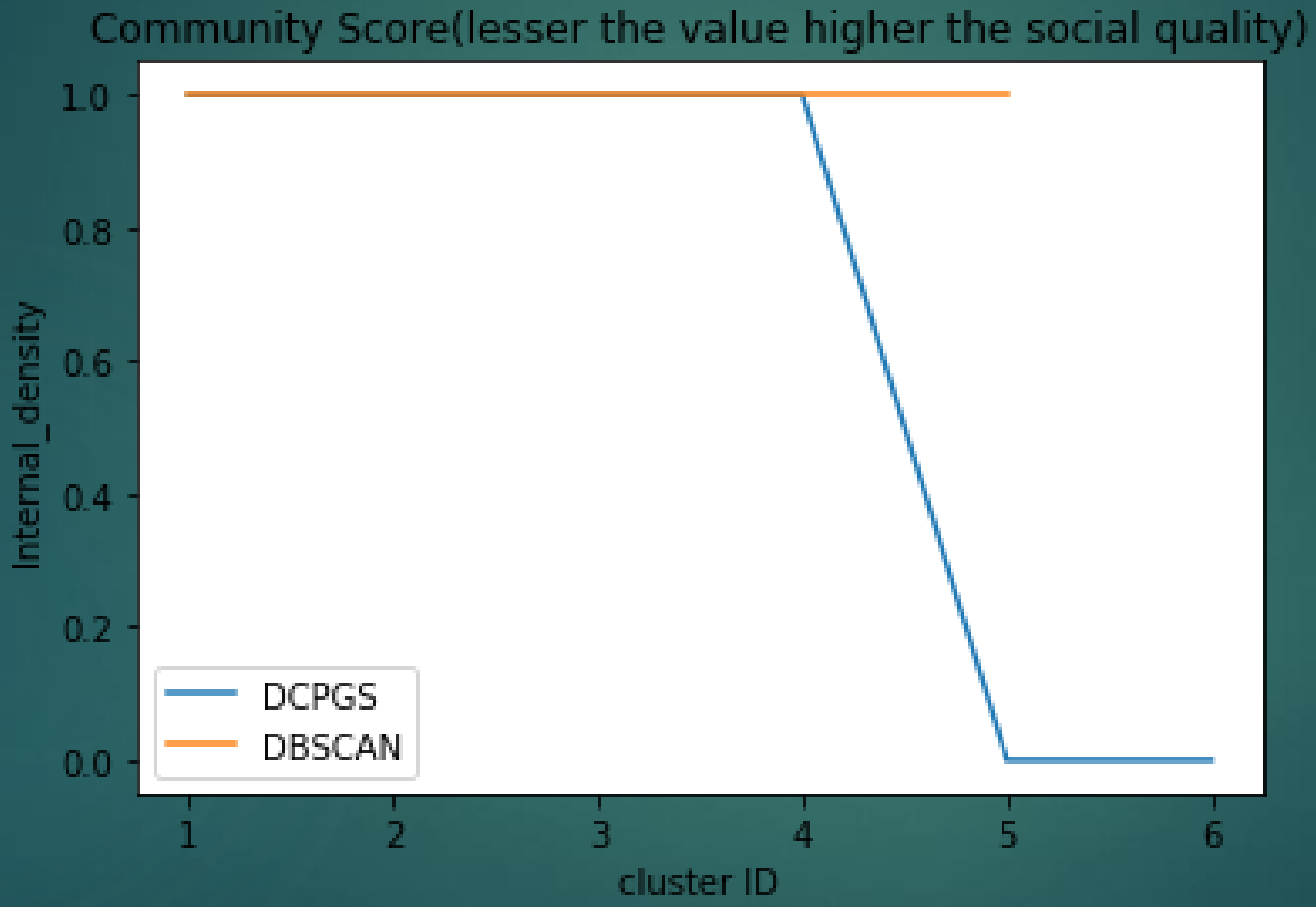


DBSCAN



DCPGS

Community Score



References

- **Gowalla Dataset:** https://snap.stanford.edu/data/loc-gowalla.html
- Dingming Wu, Jieming Shi, and Nikos Mamoulis, *Density-Based Place Clustering Using Geo-Social Network Data - Vol.30*, IEEE Transactions on Knowledge and Data Engineering, May 2018.
- Dingming Wu, Jieming Shi, and Nikos Mamoulis, *Clustering in Geo-Social Networks - Vol.30*, IEEE Data Eng. Bul., 2015.