A Geo-Social Network (GeoSN) couples social network with location-based services (LBS).

- Social network (social graph) $G=(E,V)$
- Location: 2-D Map location, GPS coordinates, altitude...
Geo-Social Apps

- Find friends within a range of a Point of Interest (POI).
  - Foursquare, Facebook, ...
- Find friends near me.
  - Facebook, twitter, WeChat, ...
- Notify me when a friend enters a certain range
  - Geoloqi
- Nearby event notification\recommendation
  - Now, Eventbrite, ...
- Search nearby users
  - WeChat, Tinder, ...
Industry Landscape

- **Foursquare**
  - 30M users, millions of check-ins per day.

- More and more users are using mobile devices to access social networks
  - Facebook, Twitter, Weibo, ...

- Messenger apps are also introducing location based services to users
  - WhatsApp, WeChat, Line, ...
How are the queries processed?

- **Challenges:**
  - Online task: User location is dynamic.
  - Social Network is complex and large.

- **No industry white papers documenting the processing of queries.**

- **In Academia:**
  - GeoSN query: Nearby friends with common interests. [Huang and Liu, Geoinformatics ’09]
  - **Proximity detection among friends. [Yiu et al., PVLDB ‘10] , [Amir et al., Pervasive and Mobile Computing ‘07]**
  - $k$-Geo-Social Circle of Friend Query ($k$-gCoFQ $(u, k)$) [Liu et al., DASFAA ‘12 ]
  - Socio-Spatial Group Query ($SSGQ(n, k, q)$) [Yang et al., SIGKDD ‘12 ]
Efficient proximity detection among mobile users via self-tuning policies

M. L. Yiu, L. H. U, S. Saltenis, and K. Tzoumas. (PVLDB ’10)

- **Proximity detection**
  - find each pair of friends such that the Euclidean distance between them is within the given distance threshold.
  - A straightforward solution forces each user to report his location to the server periodically.
    - Frequent updates → Communication overhead/bandwidth → scalability problems

- **Problem Definition:** Given a set U of mobile users, the social network G=(E,V) among them, and a spatial distance threshold $\varepsilon(i,j)$ per friend pair; find out each pair $(u_i,u_j)$ that satisfies two conditions:
  - the users $(u_i,u_j)$ are adjacent in G, i.e $(u_i,u_j) \in E$
  - the euclidean distance $\text{dist}(u_i,u_j) \leq \varepsilon(i,j)$

- **Proposed Idea:**
  - Tracks a mobile region instead of tracking user’s position
  - User sends update to the server when he is out of his mobile region
  - Search process can result in false positives
    - to be further filtered by probing a few users for their exact locations.
Proximity Detection Methods

Fixed Radius Mobile Detection (FMD) : continuous proximity detection among the users (a client-server algorithm).

At $t=0$:

- $\text{mindist}(R(u_3), R(u_1)) > \varepsilon \rightarrow (u_3, u_1)$ cannot be within proximity.
- $\text{mindist}(R(u_1), R(u_2)) < \varepsilon \rightarrow (u_1, u_2)$ to be within proximity.
- the server probes $u_1$ and $u_2$, then detects their actual proximity,
- sends them a proximity message

At $t = \Delta T$ (the arrows represents the velocity vector):

- The users are within their mobile regions
- No updates are sent to the server
Proximity Detection Methods, Cont.

- Self Tuning Mobile Region Algorithms: Improving FMD algorithm.
  - Parameters are not always good ⇒ Self-tuning
  - Instead of fixing the value of $\lambda =$ (radius of a mobile region), now associate each user $u$ with an individual $\lambda$ and a tuning parameter $\alpha$

- Expansion and contraction of mobile regions:
  - Contraction: $R(u) = R(u) \times 1/\alpha$ ← Radius too large (probing probability is too high)
  - Expansion: $R(u) = R(u) \times \alpha$ ← Radius too small (update probability is too high)

User moves out of region ⇒ Update
Two regions getting close ⇒ Probe
Motivation: Many works overlook crucial data management issues:

- Data storage methods
  - [Yiu et al., PVLDB ‘10] did not specify storage method of social graph
  - [Liu et al., DASFAA‘12] used adjacency matrix, which may incur significant storage overhead
  - [Yang et al., SIGKDD’12] make use of hybrid indices (with both social and spatial data), which may suffer from scalability problem w.r.t high check-in rates

- Social and geographical data may be administered by different entities.
  - In UK and Japan, Facebook Places cooperates with Factual, which provides infrastructure for location-based services.
  - Glancee, a location-based service app, uses Facebook’s social graph to connect nearby users.
  - A user who has both a Twitter or Facebook and a Foursquare account can post his Foursquare check-in at Twitter or Facebook.
SM and GM can be administrated by different entities.
  ○ Implement GeoSN queries without owning geo-social data e.g., Agora app.

Fully dynamic geographical dataset vs. relatively static social structures
  ○ Foursquare’s system downtime.

Easy integration of new, more efficient data structures without modifications.

Novel GeoSN query types = either a different combination of existing primitives or new ones.
Primitives

- Social Primitives
  - GetFriends(u)
  - AreFriends(u,v)
- Geo Primitives
  - GetUserLocation(u)
  - RangeUsers(q,r)
  - NearestUsers(q,k)

Primitives are atomic actions: No states

Efficiency: Depends on storage scheme:

- AreFriends - Adjacency matrix
- GetFriends - Adjacency Lists
- GetUserLocation – Hash Table
- RangeUsers & NearestUsers – Spatial Indices
Query Processing: RangeFriends(u, q, r)

Social Primitives
- GetFriends(u)
- AreFriends(u, u_i)

Geographical Primitives
- GetUserLocation(u)
- RangeUsers(q, r)
- NearestUsers(q, k)

Algorithm 1: $RF_1(u, r)$
1. $F = \text{GetFriends}(u)$
2. For each user $u_i \in F$
3. $\text{GetUserLocation}(u_i)$
4. If $|q, u_i| \leq r$
5. add $u_i$ into $R$
6. Return $R$

No Spatial Index
Adjacency list
Independent of check-ins
Dense social network

Algorithm 2: $RF_2(u, r)$
1. $R_1 = \text{GetFriends}(u)$
2. $R_2 = \text{RangeUsers}(q, r)$
3. $R = R_1 \cap R_2$
4. Return $R$

Spatial Index
Adjacency list
Sparse check-ins
# primitives

Algorithm 3: $RF_3(u, r)$
1. $U = \text{RangeUsers}(q, r)$
2. For each user $u_i \in U$
3. If $\text{AreFriends}(u, u_i)$
4. add $u_i$ into $R$
5. Return $R$

Spatial Index
Adjacency matrix
Sparse check-ins
# primitives

Friends of user $u$ within range $r$. 
Query Processing: NearestFriends(u,q,k)

**Social Primitives**
- GetFriends(u)
- AreFriends(u, u_i)

**Geographical Primitives**
- GetUserLocation(u)
- RangeUsers(q, r)
- NearestUsers(q, k)

**Algorithm 1: NF_1(u, q, k)**
1. F = GetFriends(u)
2. For each user u_i ∈ F
3. GetUserLocation(u_i)
4. Sort F (asc. ||q, u_i||)
5. R = top-k of F
6. Return R

**Algorithm 2: NF_2(u, q, k)**
1. F = GetFriends(u)
2. While |R| < k
3. u_i = NextNearestUser(q)
4. If u_i ∈ F, add u_i into R
5. Return R

**Algorithm 3: NF_3(u, q, k)**
1. While |R| < k
2. u_i = NextNearestUser(q)
3. If AreFriends(u, u_i)
4. add u_i into R
5. Return R

k nearest friends of user u to location q.
Query Processing: Nearest Star Group (NSG)

Query result: k nearest groups of m users to q, such that the users in every group are connected through a common friend.

- Social connected by a common friend
- Closest to the location q (minimum sum of distances)

Example: the next group of 3 people who come to the restaurant will receive 20% discount!

\[ k = 1, \ m = 3 \]

Result: \{ u_5, u_1, u_3 \}
Query Processing: Nearest Star Group, Cont.

**Skeleton for NSG algorithms**

**Input**: Location q, positive integers m, k  
**Output**: Result set R

1. Initialize R, $b_s$, $b_{un}$  
2. While $b_{un} < b_s$  
3. Get the next nearest user to q  
4. Construct Groups  
5. Update result R and $b_s$, $b_{un}$  
6. Refine R  
7. Return R

**Notations:**

- $b_s$ : the current best aggregate distance achieved by the already examined users (seen).
- $b_{un}$ : the lower aggregate distance that can been achieved by non-retrieved users (unseen).

$k = 1, m = 3$

Result: \{ $u_5$, $u_1$, $u_3$ \}
Lemma 1. It holds that $NSG_{u,q,m} = \{u\} \cup NF(u, q, m-1)$.

Example ($k = 1$, $m = 3$)

$b_{un}: m-1$ closest users to $q$ + last retrieved user

**Iteration 1**

$u_1, u_5, u_7$

$q$

$u_4$

$b_{un} = 15 < b_s = 31.3$

**Iteration 2**

$u_1, u_5, u_7$

$q$

$u_4$

$b_{un} = 17.8 < b_s = 29$

**Iteration 3**

$u_1, u_5, u_7$

$q$

$u_4$

$b_{un} = 21.4 < b_s = 29$

**Iteration 4**

$u_1, u_5, u_7$

$q$

$u_4$

$b_{un} = 21.8 < b_s = 26.8$

**Iteration 5**

$u_3, u_5, u_7$

$q$

$u_4$

$b_{un} = 22.7 < b_s = 26.3$

**Iteration 6**

$u_3, u_5, u_7$

$q$

$u_4$

$b_{un} = 23.6 < b_s = 26.3$

**Iteration 7**

$u_3, u_5, u_7$

$q$

$u_4$

$b_{un} = 26.4 > b_s = 26.3$

$\bigcirc = \text{construction of NSG} \quad \bigcirc = \text{Aggr. distance of NSG} \quad \bigtriangleup = \text{current best} \quad \bigstar = \text{belongs to current best group}$
Example \((k = 1, m = 3)\)

\(b_{un} : m - 1 \) closest users to \(q\) + last retrieved user

<table>
<thead>
<tr>
<th>(u_1)</th>
<th>(u_2)</th>
<th>(u_3)</th>
<th>(u_4)</th>
<th>(u_5)</th>
<th>(u_6)</th>
<th>(u_7)</th>
<th>(u_8)</th>
<th>(u_9)</th>
<th>(u_{10})</th>
</tr>
</thead>
<tbody>
<tr>
<td>(q)</td>
<td>5</td>
<td>6.4</td>
<td>10</td>
<td>10.4</td>
<td>11.3</td>
<td>12.2</td>
<td>15</td>
<td>15.5</td>
<td>15.9</td>
</tr>
</tbody>
</table>

**Iteration 1**

\(b_{un} = 15 < b_s = \infty\)

**Iteration 2**

\(b_{un} = 17.8 < b_s = \infty\)

**Iteration 3**

\(b_{un} = 21.4 < b_s = \infty\)

**Iteration 4**

\(b_{un} = 21.8 < b_s = 26.8\)

**Iteration 5**

\(b_{un} = 22.7 < b_s = 26.3\)

**Iteration 6**

\(b_{un} = 23.6 < b_s = 26.3\)

**Iteration 7**

\(b_{un} = 26.4 > b_s = 26.3\)

- \(\circ\) = construction of group
- \(\bigcirc\) = Aggr. distance of group
- \(\triangledown\) = current best
- \(\bullet\) = belongs to current best group
Query on individual users
Geo-Social Ranking Top-k query

Background:

A restaurant wants to advertise to potential customers in social network.

Advertisement is not free. You can advertise to only k users. You want to make the most of it.

The figure on the right: q (marked by a star) is the location of restaurant. The other dot cycles are users. Links indicate friendships.
Geo-Social Ranking Top-k query

A potential good customer -- Some Features (These features should be available from geo-social network).

1. Be close to the restaurant
2. Has many friends near the restaurant (with hope that the customer would bring friends to the restaurant)
Geo-Social Ranking Top-k query

3. Are the friends of a user friends to each other?

4. Preference of a user (French Food, Chinese Food, Japanese Food, Sweet, Spicy...)

5. ....
Geo-Social Ranking Top-k query

Challenge:

1. What features to include. The more features, the more information we can get. Yet it is more complicated to handle.

2. The features themselves do not have order. But we want top-k results. We need to quantify the features (turn them into numbers).
   a. Ways to quantify each features.
   b. How to combine the scores of each feature to get a total score.
Geo-Social Ranking Top-k query

One Example of Model Construction:

Features:

1. Be close to the restaurant
2. Has many friends near the restaurant (with hope that the customer would bring friends to the restaurant)
Geo-Social Ranking Top-k query

Quantification

1. Closeness to the restaurant for a node $v$ -- $||q, v||$ Euclidean distance from $v$ to $q$.
2. The number of friends that are near $q$ -- $|V_i|$, where $V_i$ is the set of a user $v$’s friends that are near $q$.

We will discuss about how to find $V_i$ latter.
Combining the scores -- linear combination

Weight: $w$

1. $w \times |V_i| + (1 - w) \times (-||q, v||)$?

If only we know $V_i$. Revised it a little bit (to reflect that friends should be near $q$):

2. $f = w \times |V_i| + (1 - w) \times (-\sum_{v \in V_i} ||q, v||)$

We choose $V_i$ such that it maximize the value of $f$. 
3. Revised further. Normalize each part of the score

\[ f = w \times \frac{|V_i|}{F} + (1 - w)(1 - \frac{\sum_{v \in V_i} ||q, v||}{F \times C}) \]

Here F is the maximum degree of some vertex plus one. \( F = \text{degree}_{\text{max}} + 1 \), and, \( \forall i, |V_i| \leq F \)

C is the maximum distance from q to some vertex
Geo-Social Ranking Top-k query

\[ f = w \cdot \frac{|V_i|}{F} + (1 - w) \left( 1 - \frac{\Sigma_{v \in V_i} ||q, v||}{F \cdot C} \right) \]

For a friend \( u \) of \( v \), if we include it into \( V_i \),

Then \( f \) increases by: \( \frac{w}{F} \)

Decreases by: \((1 - w) \cdot ||u, q||/(F \cdot C)\)

By setting \( \frac{w}{F} \geq (1 - w) ||u, q||/(F \cdot C) \)

We get, \( ||u, q|| \leq \frac{w \cdot C}{1 - w} \)

\( V_i = \{v_i\} \cup \{u : u \text{ is a friend of } v_i \text{ and } ||q, u|| \leq \frac{w \cdot C}{1 - w} \} \)
Geo-Social Ranking Top-k query

**Input:** Location $q$, positive integer $k$, weight $w$, radius $C$

**Output:** Result set $R$ // $R$ is sorted on $f_{LC}$ in desc. order

1. $b_s = 0$, $R = \emptyset$
2. $U = RangeUsers(q, \frac{w \cdot C}{1 - w})$
3. For each $v_i \in U$
4.   $V_i = \{v_i\} \cup (GetFriends(v_i) \cap U)$
5. $f_{LC}(q, v_i) = w \cdot \frac{|V_i|}{F} + (1 - w) \cdot (1 - \frac{\sum_{v \in V_i} ||q, v||}{F \cdot C})$
6. If $f_{LC}(q, v_i) > b_s$
7.   add $\{v_i, f_{LC}(q, v_i), V_i\}$ to $R$
8.   $b_s = \text{score of the } k^{th} \text{ tuple in } R$
9. While $|R| < k$
10. $v_i = NextNearestUser(q)$ outside relevant range
11. add $\{v_i, f_{LC}(q, v_i), V_i\}$ to $R$
12. Return $R$
To get a different ranking algorithm:

1. Ways to quantify the features, i.e., how to characterize a feature with number. Not unique. Application dependent. Different quantification method gives different query results.

2. Ways to combine the feature. In the previous example we use linear combination. It is simple and could be extend to higher dimensions.

3. Number of features to include. But if the dimension is too high, some techniques may not be applicable, say, building an index over all dimensions for efficient search.
Geo-Social Ranking Top-k query

Constructing another model -- An Example

Features to be considered:

1. Be close to the restaurant
2. Has many friends near the restaurant (with hope that the customer would bring friends to the restaurant)
3. Preference of a user (French Food, Chinese Food, Japanese Food, Sweet, Spicy...)

Quantification:

1. Be close to the restaurant

For a user $v$, the distance score of $v$ is defined as

$$f_g(v) = 1 - \frac{||v, q||}{C}$$

$C$ is the maximum distance from $q$ to some vertex

A little different from our previous setting.

But still quiet intuitive.
Quantification:

2. Has many friends near the restaurant (with hope that the customer would bring friends to the restaurant)

We name this the social score of a user $v$, which is now defined as:

$$f_s(v) = \frac{deg_v}{\text{degree}_{\text{max}}}$$

Shouldn’t this score reflect that friends should be near $q$?

Explain it latter.
Quantification:

3. Preference of a user (French Food, Chinese Food, Japanese Food, Sweet, Spicy...)

First, we describe the restaurant with a set of terms:

\[ T_q = \{ \text{Chinese Food, Sweet, ...} \} \]

Also, the user v has a set of descriptions:

\[ T_v = \{ \text{Chinese Food, Spicy, ...} \} \]

The preference score \( f_p \) measures the similarity between \( T_q \) and \( T_v \).

There are many measures to achieve this. For example, the number of common terms between these two measures or . We use TS as abbreviation for Term Similarity:

\[ f_p = TS(T_q, T_v) \]
Combining the scores -- linear combination

\[ f_{LC} = \alpha_g \cdot f_g + \alpha_s \cdot f_s + \alpha_p \cdot f_p \]

Such that

\[ \alpha_g + \alpha_s + \alpha_p = 1 \]

For the moment, the weights are set manually.
Geo-Social Ranking Top-k query

Query Processing -- Building Index

We name this index Geo-Social Keyword Index (GSKI).

1. First, divide the geographical space into $g^h \times g^h$ equally sized rectangles (cells). In the example on the right, $g = 2$ and $h = 2$.
2. Each intermediate cell has $g \times g$ child cells. We build this recursively until one cell is left.
3. We labeled the cells by $C_{k_{i,j}}$, where $k$ denotes the height of this cell and $i, j$ are the indexes.
4. The leaf (intermediate) cell $c$ has three attributes.

A. $\|c,q\|_{\text{min}}$ : the minimum distance from some vertex (child cell) in this cell to $q$.

B. $D_C$ : the maximum degree of some vertex (child cell) covered by this cell.

C. $T_c$ : contains all users’ (child cells’) terms of whom are covered by this cell. $T_c = \bigcup_{v \in c} T_v$

So $\alpha_a(1 - \frac{\|c,q\|_{\text{min}}}{\text{max_dist}}) + \alpha_D \frac{D_C}{\text{deg_max}} + \alpha_t \cdot TS(T_c, T_v)$ serves as an upper bound for the scores of users (child cells) covered by this cell.
Geo-Social Ranking Top-k query

**Input:** Social Graph $G = (V, E)$, integer $k$, location $q$, set of terms $T_q$, weights $\alpha_g$, $\alpha_s$, $\alpha_t$  
**Output:** Top-$k$ users according to $F$

1. Define $H$ as an empty heap of $GSKI$ cells sorted according to their scores in decre. order  
2. Add the root cell of $GSKI$ to $H$  
3. While $H$ is not empty  
   4. $e =$ top entity of $H$ // it also removes $e$ from $H$  
   5. If $e$ is an intermediate cell of $GSKI$  
      6. For each child $c$ of $e$  
         7. Add to $H$ cell $c$ with score  $\alpha_g \cdot (1 - \frac{||c,q||_{max, dist}}{max, dist}) + \alpha_s \cdot \frac{D_c}{max, deg} + \alpha_t \cdot TS(T_c, T_q)$  
   8. Else If $e$ is a leaf cell of $GSKI$  
      9. For each user $v \in V_e$  
         10. Add to $H$ user $v$ with score $\alpha_g \cdot (1 - \frac{||v,q||_{max, dist}}{max, dist}) + \alpha_s \cdot \frac{deg(v)}{max, deg} + \alpha_t \cdot TS(T_v, T_q)$  
   11. Else // $e$ is a user  
      12. Add $e$ to $R$  
   13. If $|R| = k$ then stop the execution  
14. Return $R$
Some questions:

1. Is there a way to quantitatively compare the query results of different models? For example, we have constructed two models, one with two features and one with three. Is the model with three features significantly better than the one with two features?

2. Is there a model that is flexible on the number of features while it maintains efficient to generate query results. Can we include more features available from social network?