A Moving Object Index for Efficient Query Processing with Peer-Wise Location Privacy

Dan Lin (Missouri University of Science and Technology)
Christian S. Jensen (Aarhus University)
Rui Zhang (The University of Melbourne)
Lu Xiao (Missouri University of Science and Technology)
Jiaheng Lu (Renmin University of China)

Outline

- Introduction
- Related Work
- PEB-tree
- Performance Study
- Conclusion

Introduction

- Proliferation of location-based services
  - GPS Navigator, Family Locator, Thing Finder
- Location privacy issues
  - *Provider-wise privacy*
    - Service providers disclose locations to malicious parties
  - *Peer-wise privacy*
    - Locations are seen by unauthorized users

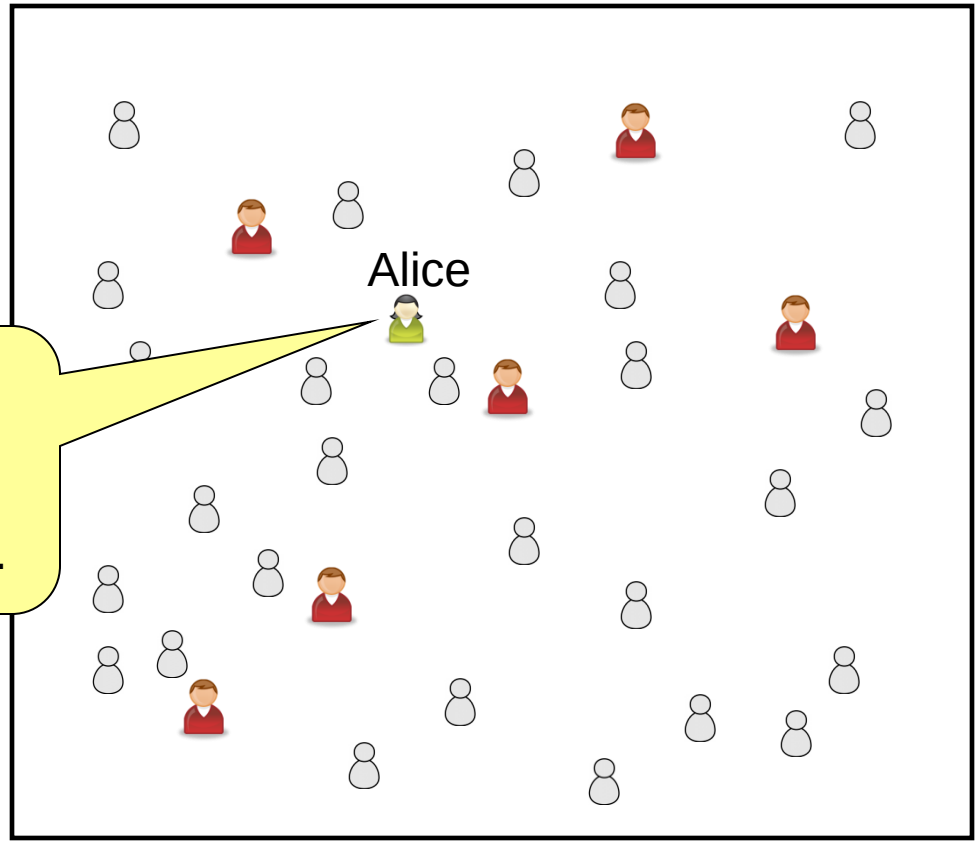
Spatial cloaking
Location distortion
K-anonymization
Encryption

… …?

Problem Definition

- Users specify their location privacy preferences
  - E.g., my colleagues can see my locations only during work hours
- Privacy-aware Queries
  - Retrieve users who allow the query issuer to view their locations and who satisfy the location constraints specified by the query issuer.
  - *Privacy-aware Range Query*
  - *Privacy-aware kNN Query*

Problem Statement



I want to go out for lunch with a friend.

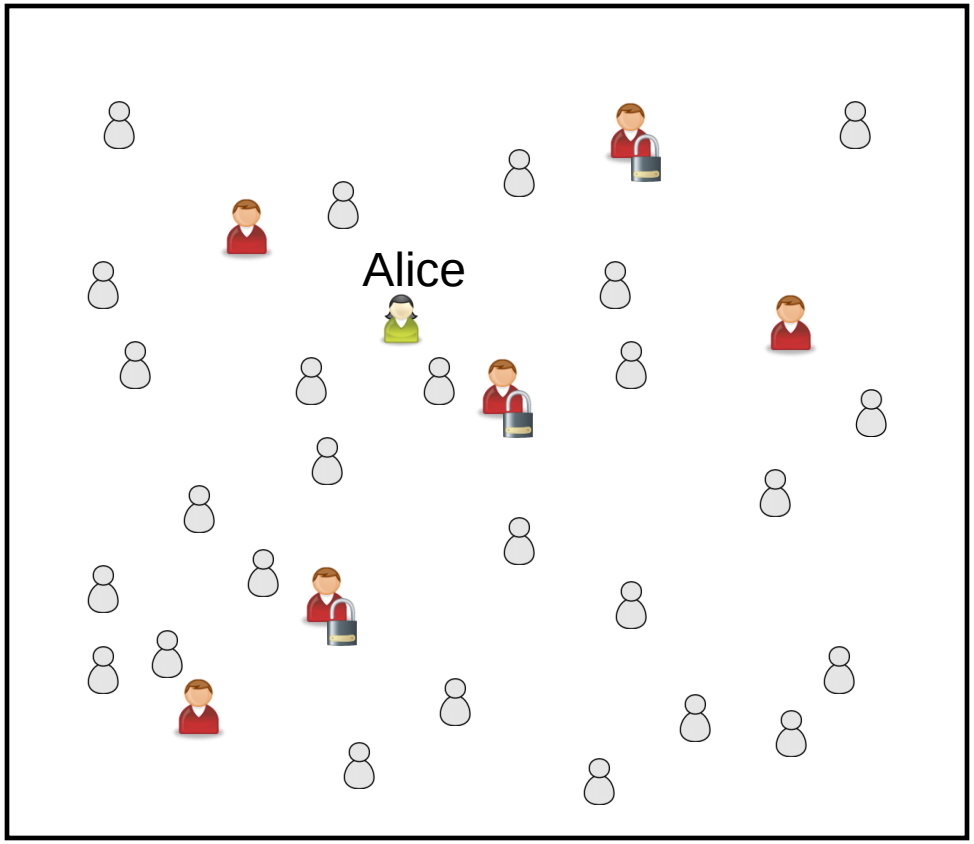
Find my nearest friend.

Alice's friend

Other users

Problem Statement

Some of Alice friends do not want to disclose locations at this moment.

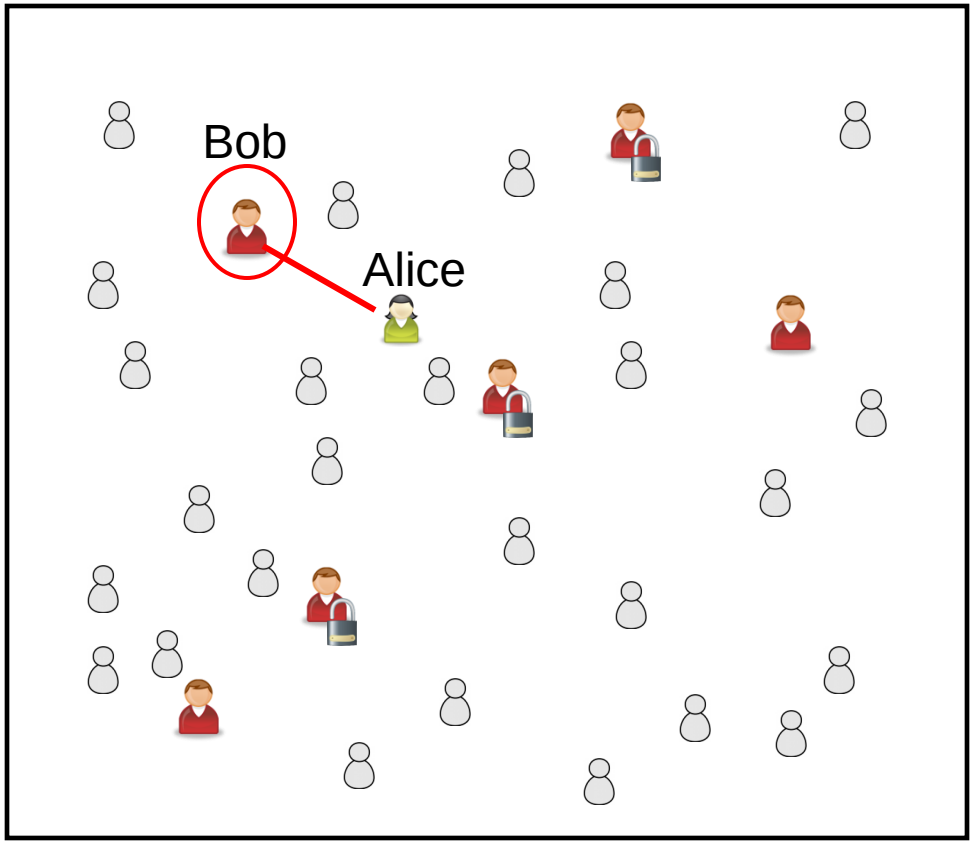


Alice's friend

Other users

Problem Statement

Bob is the nearest and allows Alice to see his location.



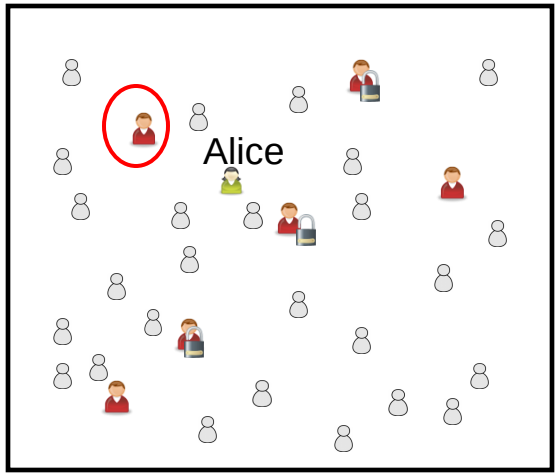
Alice's friend

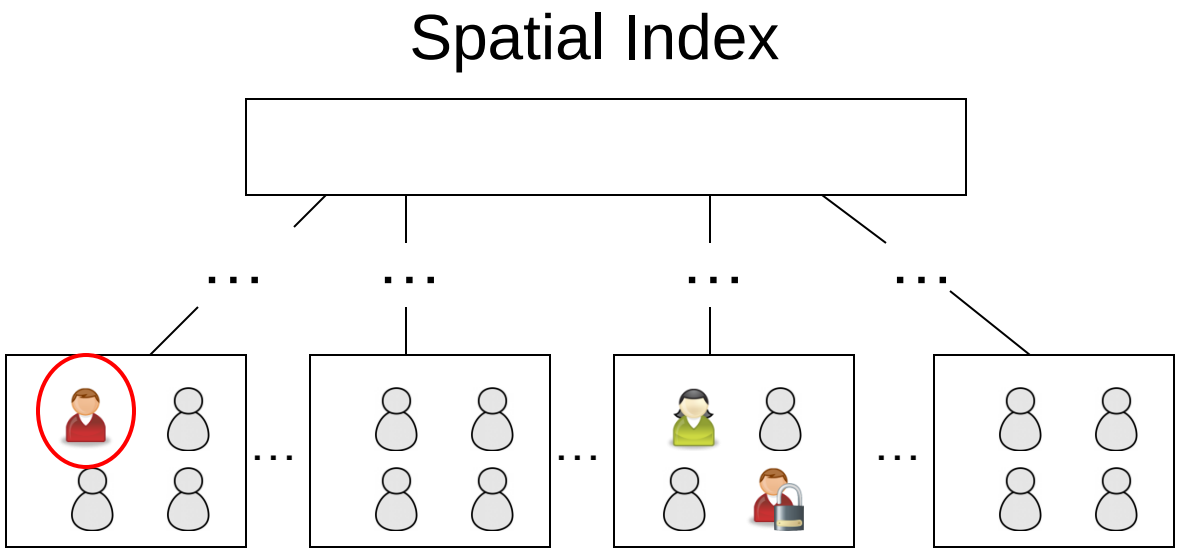
Other users

Related Works

- Spatial Indexing Approach

  Index users only based on their location proximity



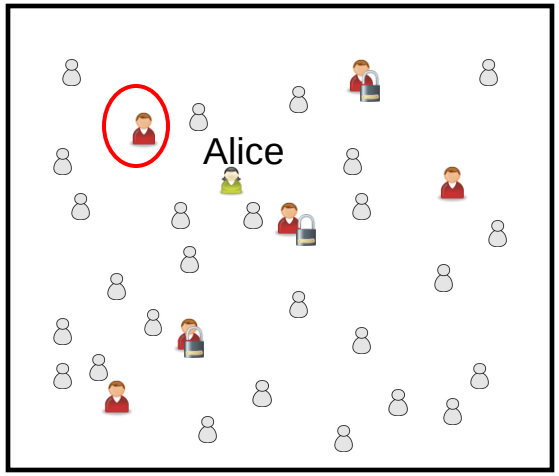


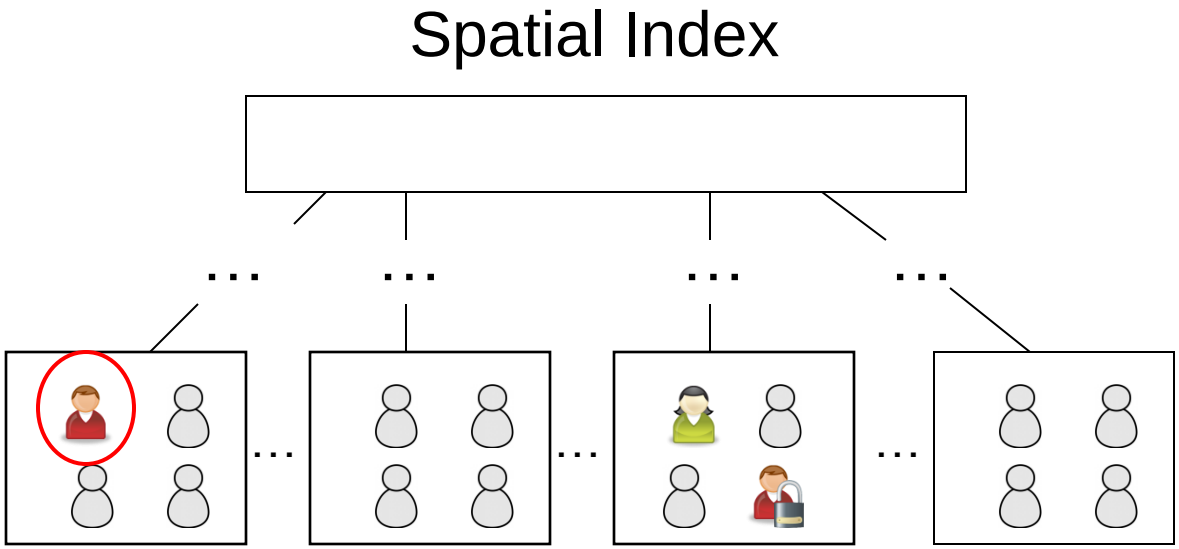
Related Works

- Spatial Indexing Approach

  Use an iterative algorithm to find the answer.

  Find the user near Alice. Check its privacy policy.
  If not satisfied, find the next nearby user.



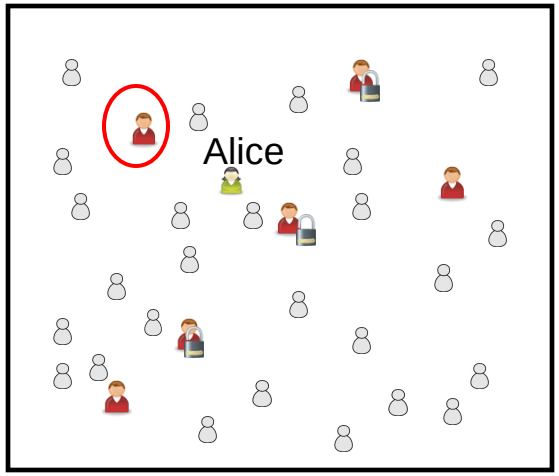


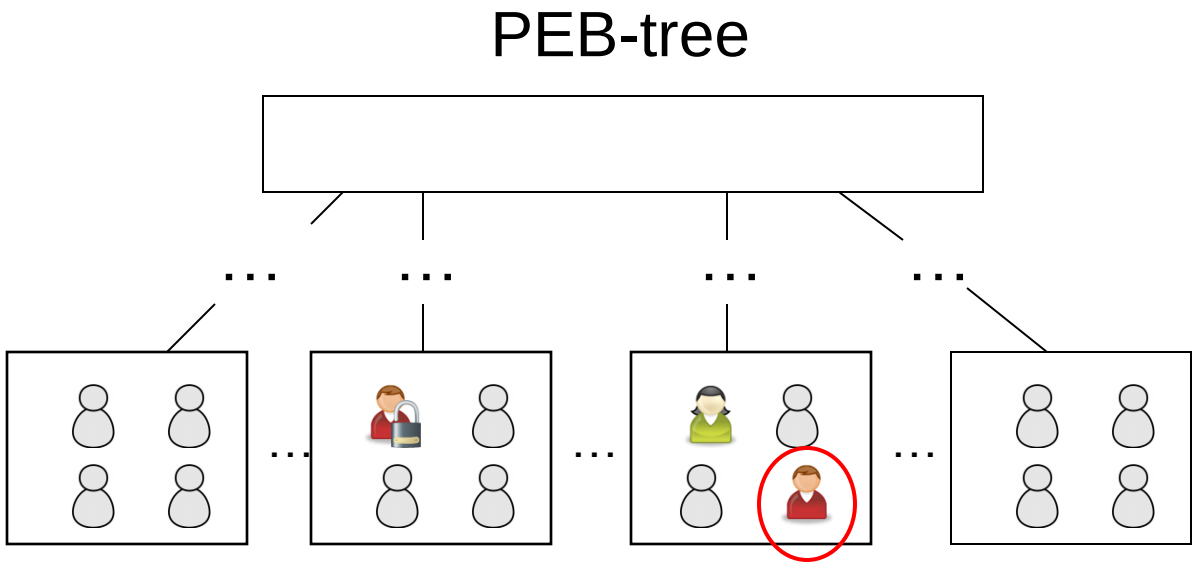
Outline

- Introduction
- Related Work
- PEB-tree
- Performance Study
- Conclusion

Policy Embedded B^x-tree (PEB-Tree)

- **Goal:** Organize users based on both spatial proximity and privacy policy compatibility to enhance query performance.





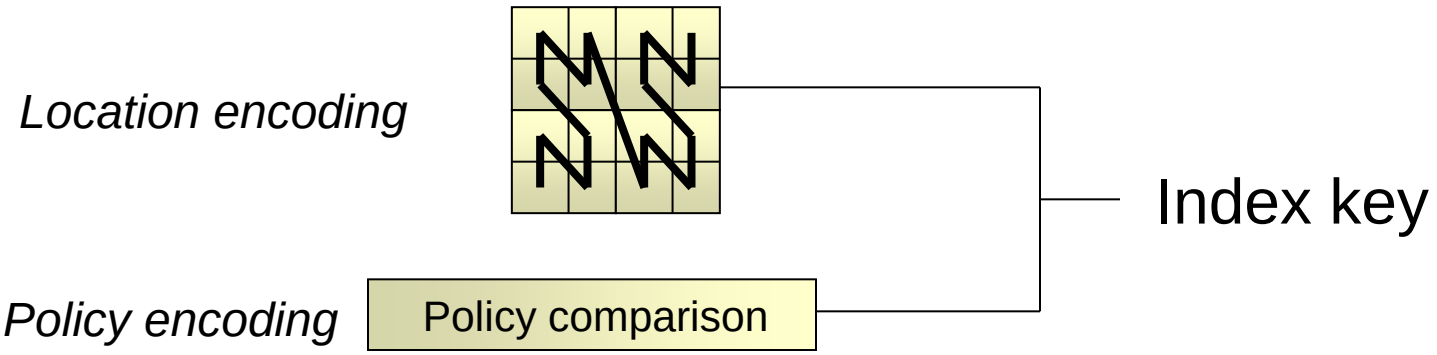
PEB-TREE

- PEB-tree Construction
- Query Algorithms

PEB-TREE Construction

Index key generation:

- Location encoding
  - B^x-tree as the base structure
  - Handle continuously changing locations of users
- Policy encoding
  - Capture compatibilities among location privacy policies belonging to different users.
- Integrate location encoding and policy encoding to form the index key of the PEB-tree.



PEB-TREE Construction

Policy encoding

- Step 1: Compare each pair of location privacy policies
  - Policy similarity score: $\alpha \in [0, 1]$
    - determined by the size of the region and the duration of the time interval specified in the policy.
- Step 2: Compute the degree of policy compatibility between each pair of users

$$C(u_1, u_2) = \begin{cases} \frac{1}{2}(1+\alpha) & P_{1\to 2} \leftrightarrow P_{2\to 1} \\ \alpha & P_{1\to 2} \nleftrightarrow P_{2\to 1} \\ 0 & \alpha = 0 \end{cases}$$

u_1 and u_2 will disclose locations

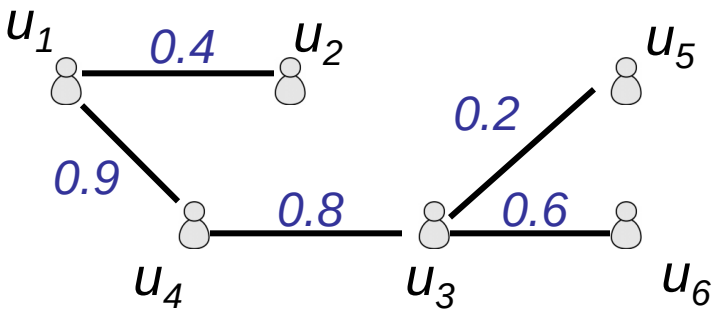
u_1 and u_2 will NOT disclose locations to each other simultaneously at any time.

u_1 and u_2 are not related.

PEB-TREE Construction

Policy encoding

- Step 3: Assign sequence values to users.
  - Users with non-zero compatibility scores are connected by lines
  - Sort users in a descending order of the number of connections
  - $SV(u_j) = SV(u_i)+(1-C(u_j,u_i))$

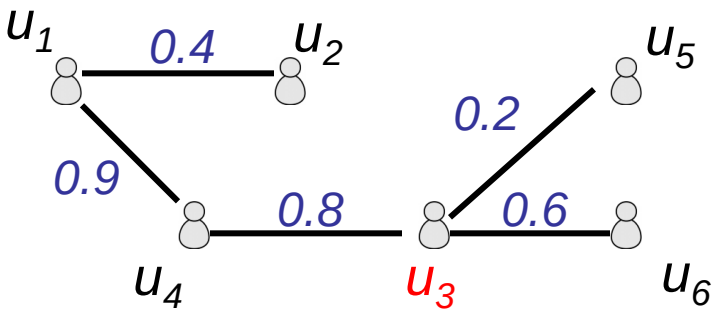


User ID	Sequence Value (SV)
U3	
U1	
U4	
U2	
U5	
U6	

PEB-TREE Construction

Policy encoding

- Step 3: Assign sequence values to users.
  - Users with non-zero compatibility scores are connected by lines
  - Sort users in a descending order of the number of connections
  - $SV(u_j) = SV(u_i)+(1-C(u_j,u_i))$
  - Starting value=2

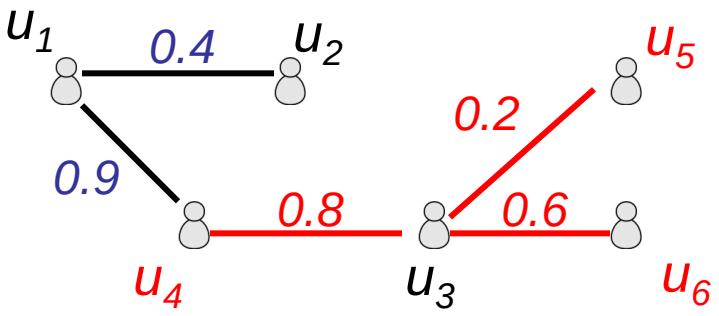


User ID	Sequence Value (SV)
U3	2
U1	
U4	
U2	
U5	
U6	

PEB-TREE Construction

Policy encoding

- Step 3: Assign sequence values to users.
  - Users with non-zero compatibility scores are connected by lines
  - Sort users in a descending order of the number of connections
  - $SV(u_j) = SV(u_i)+(1-C(u_j,u_i))$
  - Starting value=2

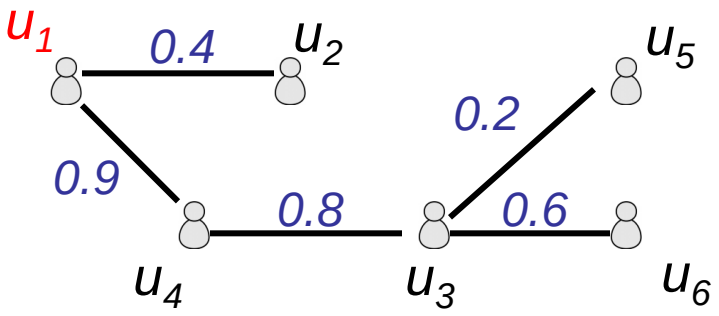


User ID	Sequence Value (SV)
U3	2
U1	
U4	2.2=2+(1-0.8)
U2	
U5	2.8=2+(1-0.2)
U6	2.4=2+(1-0.6)

PEB-TREE Construction

Policy encoding

- Step 3: Assign sequence values to users.
  - Users with non-zero compatibility scores are connected by lines
  - Sort users in a descending order of the number of connections
  - $SV(u_j) = SV(u_i)+(1-C(u_j,u_i))$
  - Starting value=2, group interval =2

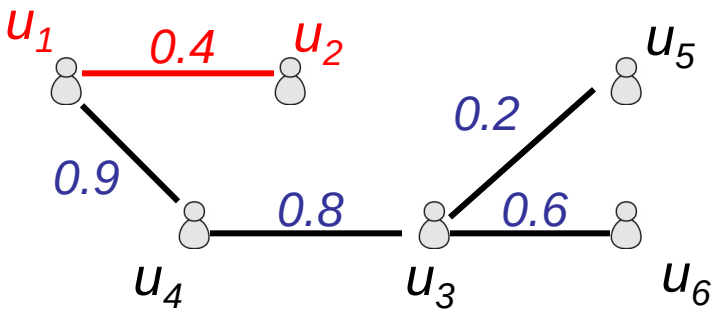


User ID	Sequence Value (SV)
U3	2
U1	4=2+2
U4	2.2=2+(1-0.8)
U2	
U5	2.8=2+(1-0.2)
U6	2.4=2+(1-0.6)

PEB-TREE Construction

Policy encoding

- Step 3: Assign sequence values to users.
  - Users with non-zero compatibility scores are connected by lines
  - Sort users in a descending order of the number of connections
  - $SV(u_j) = SV(u_i)+(1-C(u_j,u_i))$
  - Starting value=2, group interval =2



User ID	Sequence Value (SV)
U3	2
U1	4=2+2
U4	2.2=2+(1-0.8)
U2	4.6=4+(1-0.4)
U5	2.8=2+(1-0.2)
U6	2.4=2+(1-0.6)

PEB-TREE Construction

- A PEB-Tree key consists of three components: *TID, ZV, SV*
  - TID and ZV are used to model dynamic locations of users (similar to the B^x-tree)
    - TID: timestamp
    - ZV: space-filling curve value
  - SV: sequence value obtained from policy encoding

$$PEB\_key = [TID]_2 \oplus [SV]_2 \oplus [ZV]_2$$

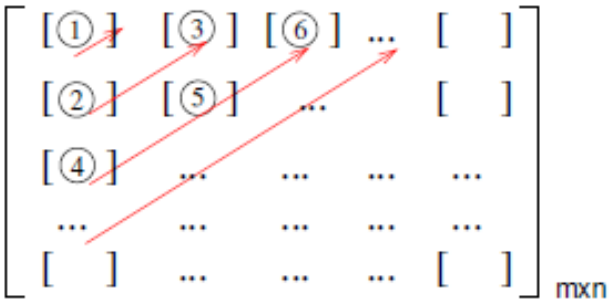
- Insertion and deletion of objects in the PEB-tree are similar to those for the B+-tree.

Privacy-aware KNN Queries

- Perform range queries Iteratively with incrementally enlarged search regions
- Consider the search ranges of both ZV and SV values for each time partition.
  - SV(u1), …, SV(um): sequence values of users related to the query issuer
  - ZV_{si}, ZV_{ei} : location query ranges

$$\begin{bmatrix} [SV(u_1)\oplus[ZV_{s1};ZV_{e1}] & SV(u_1)\oplus[ZV_{s2};ZV_{e2}] & \cdots & SV(u_1)\oplus[ZV_{sn};ZV_{en}] \\ [SV(u_2)\oplus[ZV_{s1};ZV_{e1}] & SV(u_2)\oplus[ZV_{s2};ZV_{e2}] & \cdots & SV(u_2)\oplus[ZV_{sn};ZV_{en}] \\ \cdots & \cdots & \cdots & \cdots \\ [SV(u_m)\oplus[ZV_{s1};ZV_{e1}] & SV(u_m)\oplus[ZV_{s2};ZV_{e2}] & \cdots & SV(u_m)\oplus[ZV_{sn};ZV_{en}] \end{bmatrix}_{mxn}$$

Search Matrix



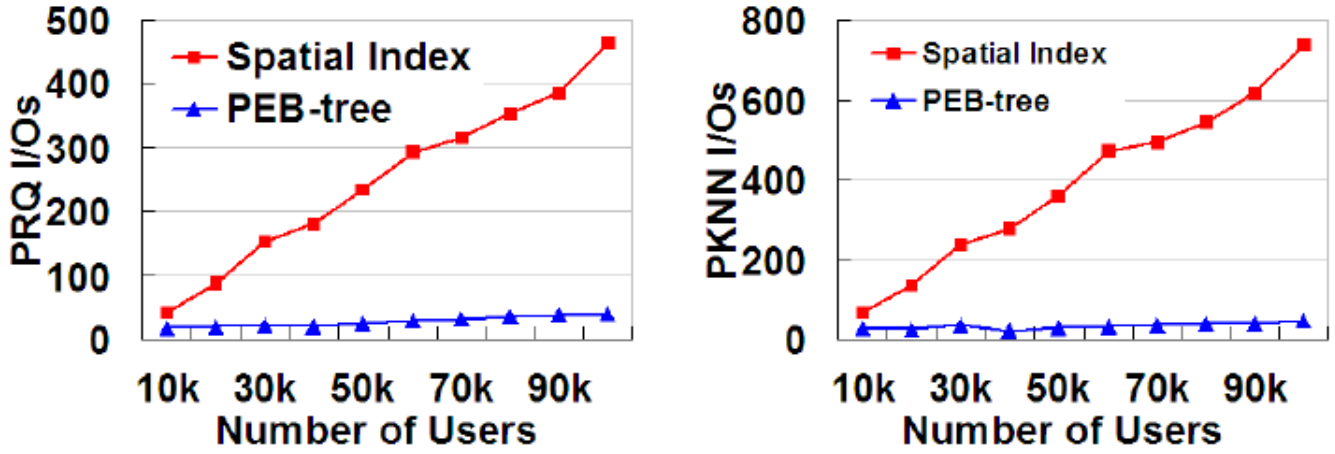
Triangular Search Order

Performance Study

- Test Datasets
  - Two types of location distribution
    - Uniformly distributed user positions
    - Positions distributed in a spatial network
  - Randomly generated policies
    - With varying spatial ranges and time intervals
  - Number of users: 10,000 to 100,000
  - Number of policies per user: 10, …, 50, …, 100
  - Grouping factor: Model the overlapping of user friends
- The PEB-tree is compared with the B^x-tree.

Performance Study

- Varying Number of Users

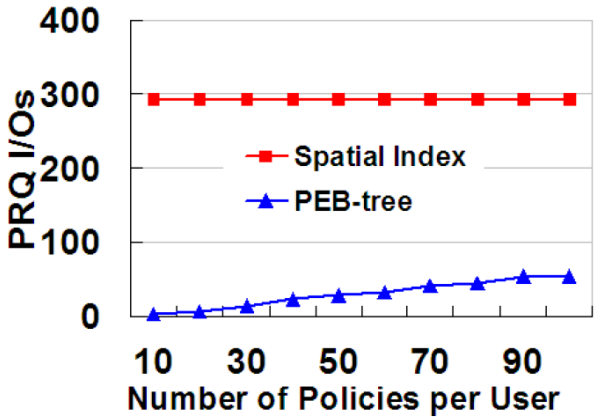


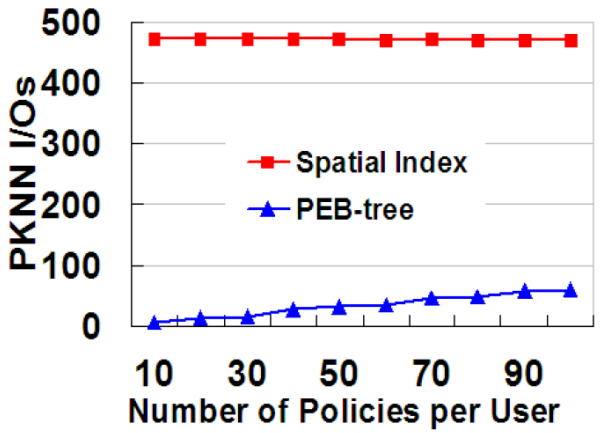
The spatial index needs to retrieve all users inside the query range, regardless of whether or not they are allowed to be seen by the query issuer.

The PEB-tree stores users based on both location and policy proximity, and search is narrowed by using both location and policy constraints.

Performance Study

- Varying Number of Policies per user





The spatial index does not consider policies, so is not affected.

The PEB-tree needs to search more users related to the query issuer due to the increase of policies.

Conclusion

- We consider the problem of efficiently supporting range and kNN queries in a setting that affords moving users of location-based services peer-wise location privacy.
- We present a new indexing technique, called the PEB-tree, along with efficient querying algorithms.

Thank You!

Questions?