**LETTER**  
Special Section on Data Engineering and Information Management

**ℓ-Close Range Friends Query on Social Grid Index**

Changbeom SHIM\(^1\), Wooli KIM\(^1\), Wan HEO\(^1\), Sungmin YI\(^1\), **Nonmembers**, and Yon Dohn CHUNG\(^{1(\ast)}\), **Member**

**SUMMARY** The development of smart devices has led to the growth of Location-Based Social Networking Services (LBSNSs). In this paper, we introduce an ℓ-Close Range Friends query that finds all ℓ-hop friends of a user within a specified range. We also propose a query processing method on **Social Grid Index** (SGI). Using real datasets, the performance of our method is evaluated.

**key words**: spatial database, location-based service, geo-social networks, close range friends query

1. **Introduction**

The proliferation of location-aware smart devices allows people to post their position to Location-Based Social Networking Services (LBSNSs) such as Foursquare, Facebook, Meetup, and Instagram. Although plenty of studies for the services have progressed, some additional studies are needed to cover several specific situations. For example (See Fig. 1), a user Jack holds a party at a point p and wants to invite his friends or the friends of his friends within a reasonable range (circle). Among four persons in the range, Mia and Lucas can be invited because they are connected with Jack by friendship lines. In this situation, if we use existing methods (e.g., [1]), it is difficult to know whether they are friends of the user or acquaintances of the user’s friends. In turn, a new type of geo-social query processing method, which considers spatial and social network data at the same time, is needed. In this paper, we define an ℓ-Close Range Friends (ℓ-CRF) query, and propose an efficient ℓ-CRF query processing method.

2. **Problem Statement**

Let \( G=\{V,E\} \) be an undirected graph, where each vertex \( v \in V \) corresponds to data object and each edge \( e \in E \) represents a friendship of the data objects, i.e., if there is an edge \( e(v_i,v_j) \), it signifies that \( v_i \) and \( v_j \) are friends. All data objects have their own position \( p_v \) in Euclidean space, and a spatial distance \( d(p_v,p_w) \) indicates a distance between two positions. A social distance \( s(v_i,v_j) \) of two data objects \( v_i \) and \( v_j \) represents the minimum number of edges (hop) between two data objects in the graph.

**Definition 1** (ℓ-CRF query). Given a query user \( v \), a spatial radius \( r \), a friendship degree \( ℓ \) and a point \( p_q \), ℓ-CRF query finds a result set \( R \), which consists of all ℓ-hop friends of \( v \) within \( r \) from \( p_q \). \( R \) is defined as follows:

\[
R = \{ v' | v' \neq v \land s(v,v') \leq ℓ \land d(p_q, p_v') \leq r \}
\]

3. **The Proposed Method**

3.1 Naïve Search Algorithm

We present two kinds of naïve approaches for processing ℓ-CRF query where each method consists of two major steps:

- Naïve1: Finding all data objects in a query range. \( \rightarrow \) Checking whether the data objects are ℓ-hop friends of the query user or not.
- Naïve2: Finding all ℓ-hop friends of a query user. \( \rightarrow \) Checking whether the ℓ-hop friends are within the query range or not.

We adopt a grid-index structure [2] for processing the range query, and the shortest path (distance) algorithm in [3] for checking the ℓ-hop friends of the query user.

Although two naïve algorithms can find the result of ℓ-CRF query, these methods cause some inefficiencies - that is, when there are a great number of data objects in the query range, or a large number of ℓ-hop friends of the query user. To tackle the drawbacks, an efficient method is necessary.

3.2 The Social Grid Index

**Social Grid Index.** For an efficient ℓ-CRF query processing, we suggest a **Social Grid Index** (SGI), which is a grid-index structure including friend cells of each cell.
is given, the procedure of our method is as follows:

1. Finding \( \ell \)-hop friend cells of a query cell \( c \) (includes a query user \( v \)) in the given range, and \( \ell \)-hop friends of \( v \). The \( \ell \)-hop friend cell is decided based on the Definition 2. For example (See Fig. 2), 2-hop friend cells of a \( c_1 \) are \( c_1, c_5, c_6, c_7 \).

2. Checking whether each data object in the cells (found at the first step) is the \( \ell \)-hop friend of \( v \) or not, and the distance between the position of each data object and query point \( p_q \) is less than \( r \), at once.

All identified data objects are returned as the result of the query. The details are presented in Algorithm 1.

### Algorithm 1: \( \ell \)-CRF Query Processing

**Procedure:** \( \ell \)-CRF query processing on \( SGI(v, r, \ell, p_q) \)

1. initialize a set \( R = \emptyset \), a heap \( H = \emptyset \), two lists \( FC = \emptyset, T = \emptyset \);
2. \( c_i \leftarrow \) a cell that includes a query user \( v \);
3. Add all \( \ell \)-hop friend cells of \( c_i \);
4. for each cell \( c \in FC \) do
5. if \( \text{mindist}(c, p_q) \leq r \) then
6. \( c' \) ← pop the next entry of \( H \);
7. \( T \leftarrow \) all \( \ell \)-hop friends of \( v \);
8. while \( H \) is not empty do
9. \( c' \leftarrow \) pop the next entry of \( H \);
10. for each data object \( v' \) in \( c' \) do
11. if \( v' \in T \land d(p_{v'}, p_q) \leq r \) then
12. insert \( v' \) into \( R \);
13. return \( R \);

**Definition 2 (Friend cell).** Let \( C \) be a set of grid cells and \( c \in C \) be a cell. Suppose that \( v_i \) and \( v_j \) are friends, and \( v_i \) and \( v_j \) in \( c_i \) and \( c_j \), respectively. Then \( c_i \) and \( c_j \) are friend cells.

For example, in Fig. 2, a cell \( c_1 \) has three friend cells \( c_1, c_5 \) and \( c_6 \), because \( v_0 \) in \( c_1 \) has three friends \( v_1 \) in \( c_1, v_2 \) in \( c_5 \) and \( v_3 \) in \( c_6 \).

**Query Processing on SGI.** The main idea of our method is to minimize the number of visited cells. When \( \ell \)-CRF query is given, the procedure of our method is as follows:

1. Finding \( \ell \)-hop friend cells of a query cell \( c \) (includes a query user \( v \)) in the given range, and \( \ell \)-hop friends of \( v \). The \( \ell \)-hop friend cell is decided based on the Definition 2. For example (See Fig. 2), 2-hop friend cells of a \( c_1 \) are \( c_1, c_5, c_6, c_7 \).

2. Checking whether each data object in the cells (found at the first step) is the \( \ell \)-hop friend of \( v \) or not, and the distance between the position of each data object and query point \( p_q \) is less than \( r \), at once.

All identified data objects are returned as the result of the query. The details are presented in Algorithm 1.

### 4. Performance Evaluation

**Setup and Dataset.** All experiments were implemented by a Java on an Intel Core i3 with 3.4GHz CPU and 16GB RAM.

1. Finding \( \ell \)-hop friend cells of a query cell \( c \) (includes a query user \( v \)) in the given range, and \( \ell \)-hop friends of \( v \). The \( \ell \)-hop friend cell is decided based on the Definition 2. For example (See Fig. 2), 2-hop friend cells of a \( c_1 \) are \( c_1, c_5, c_6, c_7 \).

2. Checking whether each data object in the cells (found at the first step) is the \( \ell \)-hop friend of \( v \) or not, and the distance between the position of each data object and query point \( p_q \) is less than \( r \), at once.

All identified data objects are returned as the result of the query. The details are presented in Algorithm 1.

**References**


