2017 IEEE 33rd International Conference on Data Engineering

Answering Location-Aware Graph Reachability Queries on GeoSocial Data

Mohamed Sarwat
School of Computing, Informatics,
& Decision Systems Engineering
Tempe, Arizona 85281
Email: msarwat@asu.edu

Yuhan Sun
School of Computing, Informatics,
& Decision Systems Engineering
Tempe, Arizona 85281
Email: YuhanSun@asu.edu

***Abstract*—Thanks to the wide spread use of mobile and wearable devices, popular social networks, e.g., Facebook, prompts users to add spatial attributes to social entities, e.g., check-ins, traveling posts, and geotagged photos, leading to what is known as, *The GeoSocial Graph*. In such graph, users may issue a *Reachability Query with Spatial Range Predicate* (*abbr.* RangeReach). RangeReach finds whether an input vertex can reach any spatial vertex that lies within an input spatial range. The paper proposes GEOREACH, an approach that adds spatial data awareness to a graph database management system. GEOREACH allows efficient execution of RangeReach queries, yet without compromising a lot on the overall system scalability. Experiments based on system implementation inside Neo4j prove that GEOREACH exhibits up to two orders of magnitude better query performance and up to four times less storage than the state-of-the-art spatial and reachability indexing approaches.**

I. INTRODUCTION

In the past decade, social media managed to unprecedentedly connect hundreds of millions of people all over the world. Facebook for example has around 1.59 billion active users monthly. Thanks to the wide spread use of mobile and wearable devices, popular social networks, e.g., Facebook, prompts users to add spatial attributes to social entities, e.g., check-ins, traveling posts, and geotagged photos. Such spatial attributes are already being tied to social networking data to form what is commonly known as *The GeoSocial Graph* [4], [1]. The GeoSocial graph brings both the physical space and social relationships into effect together. In consequence, many interesting GeoSocial applications arose. Examples of such applications include but are not limited to the following:

- **Application 1 – Point-of-Interest Recommendation:** Assume a user is visiting San Diego to attend ICDE and looking for a restaurant to have dinner. The user may ask *Yelp!* to find restaurants in San Diego such that these restaurants are already visited (and/or like, highly rated) by her friends (and perhaps friends of friends).
- **Application 2 – Advertisement:** A charity organization is holding a local fund raising party taking place in San Diego. To advertise the party, the organizing committee creates an event on Facebook and then keeps track of whether the ad for such Facebook event reaches people in the social graph who live in San Diego.
- **Application 3 – CyberSecurity:** Assume an FBI agent is investigating the spread of terrorist groups' ideologies worldwide via social media. The agent may track the twitter account of a terrorist group to analyze whether the posted tweets can reach (via the social graph) individuals living in New York City and perhaps other vital areas nationwide. This information may help the government better assess the threat posed by these ideologies.

The aforementioned applications take as input a GeoSocial graph. For instance, the GeoSocial graph in Application 1 consists of two types of vertices `Person` and `Restaurant`. Each restaurant has a geospatial location and hence we call it a spatial vertex. An edge between two `Person` vertices represents a friendship and an edge between a `Person` vertex and a `Restaurant` vertex semantically means that this person liked (or highly rated) this restaurant. Application 1 needs to first filter the restaurants based on their location to find restaurants that lie within the geographical area of San Diego. Then, the application returns those restaurants that are reachable via a GeoSocial graph path (i.e., liked by her social network) to the querying user. In this paper, we call such query a *Reachability Query with Spatial Range Predicate* (*abbr.* RangeReach). In technical terms, a RangeReach query takes as input a graph vertex v and a spatial range R and returns true only if v can reach (via a graph path) any spatial vertex (that possesses a spatial attribute) which lies within the extent of R (formal definition is given in Section II). The state-of-the-art approach to execute a RangeReach query uses a spatial index [2], [3] to locate all spatial vertices V_R that lie within the spatial range R and then uses a reachability index [5], [6] to find out whether v can reach any vertex in V_R. We call this approach *Spatial-Reachability Indexing (abbr. SpaReach)*. SpaReach needs to probe the reachability index for spatial vertices that may never be reached from v. Moreover, SpaReach has to store and maintain two index structures which may preclude scalability.

In this paper, we propose GEOREACH, a scalable and time-efficient approach that answers graph reachability queries with spatial range predicates (RangeReach). GEOREACH is equipped with a light-weight data structure, namely SPA-Graph, that augments the underlying graph data with spatial indexing directories. When a RangeReach query is issued, the system employs a pruned-graph traversal algorithm. As

This work is supported by the National Science Foundation under Grant 1654861.

2375-026X/17 $31.00 © 2017 IEEE
DOI 10.1109/ICDE.2017.76

opposed to the SpaReach approach, GEOREACH leverages the Spa-Graph' s auxiliary spatial indexing information to early prune those graph paths that are guaranteed not to reach any spatial vertices inside the query range. GEOREACH decides the amount of spatial indexing entries (attached to the graph) that strikes a balance between query processing efficiency on one hand and scalability (in terms of storage overhead) on the other hand. In summary, the main contributions of this paper are as follows: (1) To the best of the authors' knowledge, the paper is the first that formally motivates and defines RangeReach, a graph query that enriches classic graph reachability query with spatial range predicate; (2) The paper proposes GEOREACH a simple, yet effective and generic, approach that efficiently executes of RangeReach queries issued on a GeoSocial graph. (3) The paper experimentally evaluates GEOREACH using real graph datasets based on a system implementation on top of Neo4j. The experiments show that GEOREACH exhibits up to two orders of magnitude better query response time and occupies up to four times less storage than the state-of-the-art spatial and reachability indexing approaches.

The rest of the paper is organized as follows: Section II lays out the preliminary background and related work. Details of GEOREACH are explained in Sections III. Section V experimentally evaluates the performance of GEOREACH. Finally, Section VI concludes the paper.

II. PRELIMINARIES

GeoSocial Graph. GeoSocial Graph is a directed property graph $G = (V, E, V_S)$ where (1) V is a set of vertices such that each vertex has a set of properties (attributes); (2) $E \subseteq V \times V$ where (u, v) denotes an edge from u to v and (3) $V_S \subseteq V$ where $\forall v \in V_S$, $v.spatial$ exists. $v.spatial$ may be a geometrical point, rectangle, or a polygon. For ease of presentation, we assume it is a point (longitude and latitude). Figure 1 depicts a toy example of a GeoSocial graph whose V_S are drawn with black.

RangeReach(v,R). RangeReach is a graph reachability query with spatial range predicate. RangeReach finds whether a graph vertex v can reach a specific spatial region (range) R. Given a vertex $v \in V$ in a Graph G and a spatial range R, RangeReach can be mathematically described as follows:

$$\textit{RangeReach(v, R)} = \begin{cases} true & \text{if } \exists\ v' \text{ such that} \\ & (1)\ v' \in V_S \\ & (2)\ v' \sqsubset R \\ & (3)\ v \rightsquigarrow v' \\ false & \text{Otherwise.} \end{cases} \quad (1)$$

As given in Equation 1, if any spatial vertex $v' \in V_S$ that lies within the extent of the spatial range R is reachable from the input vertex v ($v \rightsquigarrow v'$), then RangeReach(v, R) returns true (i.e., $v \rightsquigarrow R$). For example, in Figure 1, RangeReach(a, Q) = *true* since a can reach at least one spatial vertex f in R.

III. OUR APPROACH: GEOREACH

A. Data Structure

GEOREACH augments a graph structure with spatial indexing entries to form what we call SPatially-Augmented Graph (SPA-Graph). Vertices in original GeoSocial Graph are augmented with new attributes (spatial reachability information) and converted to three types of vertices. A SPA-Graph has three different types of vertices, described as follows:

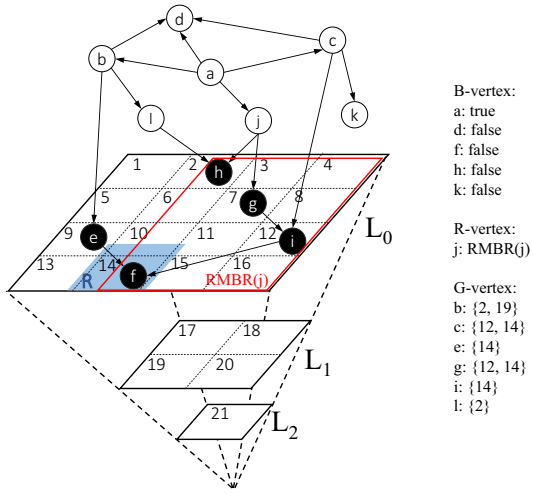


Fig. 1: SPA-Graph Overview

- **B-Vertex:** a B-Vertex v ($v \in V$) stores an extra bit (i.e., boolean), called Spatial Reachability Bit (abbr. GeoB) that determines whether v can reach any spatial vertex in the graph.
- **R-Vertex:** an R-Vertex v ($v \in V$) stores a Reachability Minimum Bounding Rectangle (abbr. RMBR(v)). RMBR(v) represents the minimum bounding rectangle MBR(S) (represented by a top-left and a lower-right corner point) that encloses all spatial vertices that are reachable from vertex v.
- **G-Vertex:** a G-Vertex v stores a list of spatial grid cells, called the Reachability Grid (abbr. ReachGrid(v)). Each grid cell G in ReachGrid(v) belongs to a hierarchical grid data structure that splits the total physical space into spatial grid cells. Each spatial vertex will be located in a grid cell. ReachGrid(v) contains all grid cells where reachable spatial vertices of v are located in.

Example. Figure 1 gives an example of a SPA-Graph. There are three layers of grids, denoted as L_0, L_1, L_2 from top to bottom. The uppermost layer L_0 is split into 4×4 grid cells, denoted as G_1 to G_{16}. The middle layer gird L_1 is split into four cells G_{17} to G_{20}. Each cell in L_{i+1} covers four cells in L_i ($i \geq 0$). One feasible augmented structure is listed on the right side of the figure.

B. Query Processing

This section explains the query processing algorithm. The main idea behind the GEOREACH is to leverage GeoB, RMBR and ReachGrid stored in a B-Vertex, R-Vertex or a G-Vertex to reduce graph traversal space. The algorithm traverses the graph from v. For each visited vertex, three cases might happen:

Case I (B-vertex): GeoB is false, it cannot reach any spatial vertex and hence the algorithm stops traversing all

graph paths after this vertex. Otherwise, further traversal from current B-vertex is required when GeoB value is true.

Case II (R-vertex): For a visited R-vertex u, there are three conditions that may happen. Case II.A: RMBR(u) lies within R. In such case, the algorithm terminates and returns **true**. Case II.B: RMBR(u) does not overlap with R. All reachable spatial vertices of u is outside R. So, traversal after u can be pruned. Case III.C: RMBR(u) is partially covered by R. In this case, the algorithm keeps traversing neighbors of u.

Case III (G-vertex): For a G-vertex u, three cases may happen. Case III.A: R fully contains any grid cell in ReachGrid(u). The algorithms terminates and returns *true*. Case III.B: R has no overlap with any grid in ReachGrid(u). Since v cannot reach any grid overlapped with R, the branch from v is pruned. Case III.C: R only partially overlaps with some reachable grids, further traversal is performed.

Use SPA-Graph in Figure 1 to show how to solve RangeReach(a, R). The traversal starts at a. It is a B-vertex and its GeoB value is true (Case I), the algorithm recursively traverses out-edge neighbors of a ($\{b, c, d, j\}$) and perform recursive checking. ReachGrid(b) = $\{2, 19\}$. Both G_2 and G_{19} are partially covered by Q (Case III.C), hence it is possible for b to reach Q. Further traversal from b is required. For another neighbor c, ReachGrid(c) is $\{12, 14\}$. G_{14} lies in Q (Case III.A). So the algorithm terminates the graph traversal and returns *true* as the query answer.

IV. Initialization & Maintenance

The GeoReach initialization algorithm first follows the topological order to accumulate spatial reachability information for each vertex from its neighbors. This step will determine the type of each vertex. Then adjacent grid cells in ReachGrid are merged according to `MERGE_COUNT`. To determine the type of each vertex, the initialization algorithm takes into account the following system parameters:

`MAX_RMBR`: This parameter represents the maximum space area of each RMBR. If area of RMBR(v) is larger than `MAX_RMBR` v will be degraded from a R-vertex to B-vertex.

`MAX_REACH_GRIDS`: This parameter sets up the maximum number of grid cells in each ReachGrid. If the number of grid cells in ReachGrid(v) exceed `MAX_REACH_GRIDS` v will be degraded from a G-vertex to R-vertex.

The algorithm first determines initial type of v by checking exact types of out-neighbors of v. If there is any B-vertex v' with GeoB(v') = *true*, v is directly initialized to a B-vertex with GeoB(v) = *true*. Otherwise, if there is any R-vertex, the function will return an R-vertex type, which means that v is initialized to R-vertex. If either of the above happens, the function returns G-vertex type. Based on the initial vertex type, the algorithm may encounter the following three cases:

Case I (B-vertex): The algorithm directly sets v as a B-vertex and GeoB(v) = *true* because there must exist one out-edge neighbor v' of v such that GeoB(v') = *true*.

Case II (G-vertex): For each vertex $v' \in V_v^{out}$, ReachGrid(v) is updated by adding in ReachGrid(v') and Grid(v'). Each time the algorithm compares |ReachGrid| with `MAX_REACH_GRIDS` to decide whether v should be degraded from G-vertex to R-vertex.

Case III (R-vertex): For each $v' \in V_v^{out}$, the algorithm updates RMBR(v) with RMBR(v') and $v.spatial$. Each time, the algorithm checks area of RMBR(v). If it is larger than `MAX_RMBR`, v is degraded to a B-vertex.

After all vertices are initialized, reachable grid cells in each G-vertex will be merged referring to `MERGE_COUNT`. The merge starts from the highest layer L_0 to the lowest layer. For each quad-cells region in each layer, if number of reachable grid cells exceeds `MERGE_COUNT`, these cells will be merged to the corresponding cell in the higher level layer.

Example. Figure 1 depicts a SPA-Graph with `MAX_RMBR` = 0.8A and `MAX_REACH_GRIDS` = 3, where A is area of the whole space. One correct topological order is $[a, j, b, c, e, g, l, i, k, h, f, d]$. By following the reverse order, vertices d, f, h, k are processed first. They cannot reach any spatial vertex, so their ReachGrids are empty. Their types should be B-vertex and are assigned to *false* values. Then ReachGrid(i) will be calculated as ReachGrid(i) = ReachGrid(f) $\cup$ Grid(f) = $\emptyset \cup \{14\} = \{14\}$. Similarly, ReachGrid of l, g, e, c and b can be calculated and their values are listed in the figure. Next ReachGrid(j) = {2, 7, 12, 14}. Since |ReachGrid(j)| > `MAX_REACH_GRIDS`, j will be degraded to an R-vertex and stored with RMBR(j) which is drawn with red rectangle in the figure. It is not degraded to B-vertex because area of RMBR(j) is smaller than `MAX_RMBR`. For a, it will be initialized with R-vertex type because one of its out-neighbor j an R-vertex. So the algorithm will directly calculate RMBR(a) without considering ReachGrid(a). Area of RMBR(a) is larger than `MAX_RMBR`, so a is degraded to a B-vertex with *true* value.

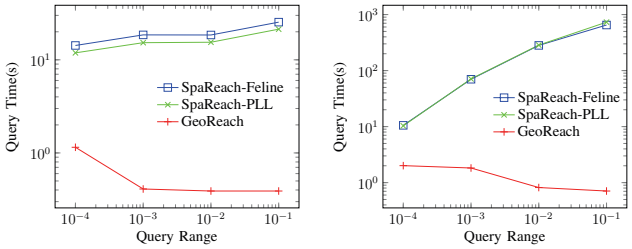
V. Experimental Evaluation

In this section, we present a comprehensive experimental evaluation of GeoReach 's effectiveness and performance. We evaluate the performance of the three following approaches: (1) GeoReach: The proposed GeoReach approach with the following default parameters: (a) `MERGE_COUNT` set to 0 which means that adjacent grid cells are not merged, and (b) `MAX_RMBR` set to 1 which indicates that RMBRs are not degraded to GeoB and there is no B-Vertex. (d) The total geographical space is split into 128×128 grid cells in the highest grid layer L_0. (2) SpaReach-PLL: A SpaReach approach that harnesses an R-Tree spatial index and reachability index proposed in [6]. (3) SpaReach-Feline: SpaReach-Feline is implemented with R-Tree index and Feline state-of-the-art reachability index proposed in [5].

TABLE I: GeoSocial Graph Datasets (K = 10^3)

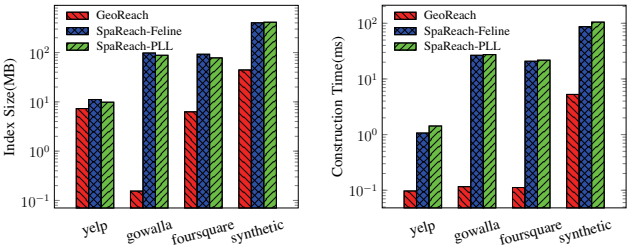
Dataset	$\|V\|$	$\|V_S\|$	$\|E\|$	d_{avg}
Yelp	629K	77K	6033K	9.59
Gowalla	1477K	1280K	5881K	3.98
Foursquare	3296K	1143K	18723K	5.68
Synthetic	6540K	3924K	15011K	2.3

Experimental Environment. Graph data, reachability indices of both SpaReach and index of GeoReach are stored as



(a) Yelp (b) Synthetic

Fig. 2: Query response time



(a) Index size (b) Construction time

Fig. 3: Indexing Overhead

attributes of each graph vertex in Neo4j graph database. R-Tree spatial indices in two SpaReach approaches are implemented by using Gist index in Postgres. The source code for evaluating query response time is implemented in Java and compiled with java-7-openjdk-amd64. Source code of index construction is implemented in c++ and complied using g++ 4.8.4. All experiments are run on a desktop with an 3.60 GHz CPU, 32GB RAM running Ubuntu 14.04 Linux OS.

Datasets. Both real and synthetic datasets are used. Detailed information of these datasets are listed in Table I where d_{avg} is the average degree of the graph. There are three real GeoSocial datasets, including Yelp[1], Gowalla[2], and Foursquare[3]. For brevity, only results of Yelp are illustrated in the paper. Synthetic dataset is generated by assigning spatial locations to vertices in citeseerx graph, which is widely used for evaluating graph reachability queries [6]. 60% of all the vertices are randomly elected as spatial and they are randomly distributed in a square space.

A. Query Response Time

In this section, SpaReach-PLL and SpaReach-Feline are compared with GEOREACH. Spatial selectivity of the input query rectangle varies from 0.0001 to 0.1. For each selectivity, we run 500 random queries. Figure 2 depicts the query time of three approaches. As opposed to the other two approaches, GEOREACH takes much less query time. When query range size increases, GEOREACH performs better because GEOREACH can efficiently prune many vertices. Its performance decreases when selectivity is near 0.0001 because the space is divided into 128×128 grid cells and size of each grid cell is around 0.0001. Both SpaReach methods perform worse when size of query rectangle increases. When the query rectangle is larger, more spatial vertices will be located inside the range, which incurs a high overload in reachability checking step.

B. Indexing Overhead

Figure 3a depicts the storage overhead incurred by all three approaches. GEOREACH requires less storage overhead than the other two methods. It proves that GEOREACH can be well-scalable with proper setting of parameters. SpaReach-PLL and SpaReach-Feline requires high storage overhead due to spatial and reachability indices. As a result, both of them are not scalable when the graph has complex structure and large number of spatial vertices.

Figure 3b depicts the initialization time incurred by all the methods. It clearly shows that GEOREACH takes less time to initialize than its competitors in all cases. Initialization time of both SpaReach-PLL and SpaReach-Feline are dominant by spatial index. It inclines that the graph of larger size (more spatial vertices) will incur more construction time.

[1]https://www.yelp.com/dataset_challenge

[2]https://snap.stanford.edu/data/loc-gowalla.html

[3]https://archive.org/details/201309_foursquare_dataset_umn

VI. Conclusion and Future Work

The paper motivates location-aware reachability queries – a family of queries that process graph reachability predicates and spatial range predicates side-by-side over a GeoSocial graph. A novel approach GEOREACH is proposed to efficiently answers the query. Extensive experiments show that GEOREACH can be scalable and query-efficient than existing spatial and reachability indexing approaches in relatively sparse graphs. Even in dense graphs, GEOREACH can outperform existing approaches in terms of storage overhead and initialization time. In the future, we plan to study the extensibility of GEOREACH to support different spatial predicates (e.g., spatial join). We also plan to study the applicability of GEOREACH to various application domains including: Spatial Influence Maximization, Location and Social-Aware Recommendation and Location-Aware Citation Network Analysis.

References

[1] N. Armenatzoglou, S. Papadopoulos, and D. Papadias. A general framework for geo-social query processing. *PVLDB*, 6(10):913–924, 2013.

[2] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. In *SIGMOD*, pages 322–331, May 1990.

[3] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1990.

[4] J. Shi, N. Mamoulis, D. Wu, and D. W. Cheung. Density-based place clustering in geo-social networks. In *SIGMOD*, pages 99–110. ACM, 2014.

[5] R. R. Veloso, L. Cerf, W. Meira Jr, and M. J. Zaki. Reachability queries in very large graphs: A fast refined online search approach. In *EDBT*, pages 511–522. Citeseer, 2014.

[6] Y. Yano, T. Akiba, Y. Iwata, and Y. Yoshida. Fast and scalable reachability queries on graphs by pruned labeling with landmarks and paths. In *CIKM*, pages 1601–1606. ACM, 2013.