Fast Geosocial Reachability Queries

Panagiotis Bouros
Johannes Gutenberg University Mainz
Mainz, Germany
bouros@uni-mainz.de

Theodoros Chondrogiannis
University of Kostanz
Kostanz, Germany
theodoros.chondrogiannis@uni.kn

Daniel Kowalski
Johannes Gutenberg University Mainz
Mainz, Germany
dkowalsk@students.uni-mainz.de

ABSTRACT

The proliferation of location-based services and social networks has given rise to *geosocial networks*, which model not only the social interactions between users but also their spatial activities. We study the efficient computation of a recently proposed query for geosocial networks called *Geosocial Reachability* query (RangeReach), which comes as a hybrid of the traditional spatial selection (range) query and the graph reachability problem. Intuitively, given a geosocial network G, a vertex v, and a spatial region R, RangeReach(G, v, R) determines whether v can reach any vertex in G with spatial activity inside R. We consider an interval-based labeling scheme proposed in the past for graph reachability to devise two novel solutions for RangeReach. The first takes a social-first approach, prioritizing the graph reachability predicate. The second treats both predicates at the same time by transforming the problem of answering RangeReach queries into queries over a three-dimensional space that models the spatial and interval-based reachability information in the geosocial network. Our experimental analysis compares our proposed solutions against a baseline spatial-first approach powered by spatial indexing and a graph reachability technique, as well as the state-of-the-art method for RangeReach queries.

1 INTRODUCTION

The ubiquity of mobile location-aware devices (smartphones and watches, tablets, etc.) and the proliferation of social networks have given rise to *geosocial networks*, where users not only form social connections to each other but also perform geo-referenced actions, e.g., posts and check-ins. Examples of such networks include traditional social networks extended with geospatial information, such as X (formally known as Twitter) and Facebook, and networks natively offering geosocial services, such as Yelp and Foursquare. Previous research in geosocial networks has focused on query processing [3, 23, 24, 42, 50, 51, 53] and indexing [54, 55], on the collision with recommender systems [37, 46] and on analysis tasks such as influence maximization [8, 39] and community search [11, 26, 27].

In this paper, we study the efficient computation of the *Geosocial Reachability* query or simply RangeReach, proposed by Sarwat and Sun in [47]. A RangeReach(G, v, R) query comes as a hybrid of a traditional *spatial range* or *selection*, which identifies all points or *spatial vertices* u inside the query region R, and a *graph reachability* query, which determines whether the directed graph G contains a path from query vertex v to any of these spatial vertices u. Such a query finds application in multiple scenarios. In *Points-of-Interest* recommendation, users can query for restaurants in a particular area of the city that their friends or friends of their friends have visited in the past. In *Geoadvertising* [7], RangeReach can help determine the best location to open a shop or how to advertise an event [8] based on users that have direct or indirect (via friendship relationships) previous activity in particular parts of a city. Another application area is the study of infectious diseases [29, 31], e.g., COVID-19, where RangeReach can assist on monitoring and understanding how they spread in specific areas through human interaction.

A straightforward approach for RangeReach(G, v, R) queries is to first identify every spatial vertex u inside query region R, i.e., evaluate the spatial range query on R, and then evaluate the graph reachability query from v to u. We call this approach *spatial-first* or SpaReach. To boost SpaReach, traditional spatial indexing such as the R-tree [30] can be combined with graph reachability techniques [52, 67, 69]. SpaReach prioritizes the spatial predicate of RangeReach, and thus, it is sensitive to the selectivity of the spatial selection. Especially when the answer to the RangeReach query is negative, we must test the connectivity of all spatial vertices contained inside R to query vertex v. To address such weaknesses, Sarwat and Sun [47] proposed GeoReach, which does not prioritize one of the RangeReach predicates. The method constructs an auxiliary structure termed the *SPA-Graph* to partially materialize the social (graph) and the spatial reachability information of the input geosocial network G. Nevertheless, GeoReach still needs to traverse part of the geosocial network, as the method does not take advantage of any graph reachability indexing technique.

Given these shortcomings, we consider interval-based labeling (specifically the scheme proposed by Agrawal et al. in [1]) as the basis for two novel RangeReach methods. First, we devise a *social-first* approach termed SocReach, which, contrary to SpaReach, prioritizes the graph reachability predicate of the query. The labeling scheme is used to compute all reachable vertices from the query vertex v, which are then spatially verified to check if at least one lies inside region R. Second, we propose the 3DReach method, which, similar to GeoReach, evaluates both query predicates at the same time. The main idea of 3DReach is to model the input geosocial network and its interval-based labeling inside a three-dimensional space. With this transformation, a RangeReach query is rewritten as a set of three-dimensional range queries.

The key contributions of this paper are summarized as follows:

- We investigate the application of the interval-based labeling on geosocial networks. To our knowledge, this is the first time that interval-based labeling is used in such context. As geosocial networks do not exhibit the characteristics of hierarchies or knowledge bases [1, 18], we present a specialized construction algorithm in Section 3.
- In Section 4, we propose two novel evaluation methods for RangeReach queries, which capitalize on the interval-based labeling. The SocReach method operates under a social-first principle, an approach never considered before

while 3DReach builds upon an entirely novel transformation of both the network and its labeling scheme in a three-dimensional space. Note that, our methods can be easily incorporated in existing systems for geosocial networks as no custom data structures are required; only typical spatial or multi-dimensional indexing for 3DReach.

- Graph reachability methods assume that the input is a directed acyclic graph (DAG); hence, the strongly connected components are collapsed into single vertices. In Section 5, we discuss arbitrary graphs and how to model the spatial extent of their strongly connected components (SCC).
- Section 6 compares our SocReach and 3DReach methods against the state-of-the-art GeoReach and two versions of SpaReach, using four real-world geosocial networks. To delve into the performance of the methods, we select the datasets to represent two different cases of geosocial networks either dominated by a large SCC or containing multiple ones. In the first case, the evaluation of RangeReach queries is dominated by their spatial range predicate, while in the second, the overall cost is divided between the graph reachability and the spatial range. Our analysis shows that 3DReach is the best evaluation method, outperforming GeoReach and SpaReach in all tests by a wide margin, for both cases of geosocial networks.

The rest of the paper is organized as follows. Section 2 formally defines the RangeReach query and revisits the SpaReach and GeoReach approaches. Section 7 discusses the related work on graph reachability and spatial selection (range) queries. Last, Section 8 concludes our paper.

2 PRELIMINARIES

In this section, we introduce necessary notation and background for the problem of geocial reachability.

2.1 Notation and Problem Definition

We model a social network as a *directed* graph $G = (V, E)$ where every vertex $v \in V$ represents an entity of the network (e.g., users or venues). Every edge $(u, v) \in E \subseteq V \times V$ indicates a friend relationship between the users corresponding to vertices u and v, e.g., the u *FOLLOWS* v users relationship on Twitter, or between user u and venue v, e.g., the u *CHECKS-IN* v relationship on Foursquare. A *geosocial* network is a social network where a vertex v can be associated with the coordinates of a point in the two-dimensional space, denoted by $v.point$; for simplicity, we call such v, a *spatial vertex*. For the rest of the text, we denote a geosocial network as $G = (V, E, P)$ where set P contains the points from all spatial vertices in the network.[1]

We next revisit the definition of two well-established problems/queries which act as the building blocks for the problem at hand.

Definition 2.1 (Graph Reachability). Given a directed graph $G = (V, E)$ and two vertices $v, u \in V$, the *graph reachability* query GReach(G, v, u) determines that v can reach u iff G contains a path from vertex v to u.

Definition 2.2 (Spatial Range Query). Given a collection of points P and a spatial region R in the same two-dimensional space, the *spatial range* query SRange(P, R) returns all points from P located inside region R.

[1]In this work, we assume that the spatial vertices are represented as points in the two-dimensional space. However, our analysis and the proposed solutions can be easily extended to arbitrary geometries and the three-dimensional space.

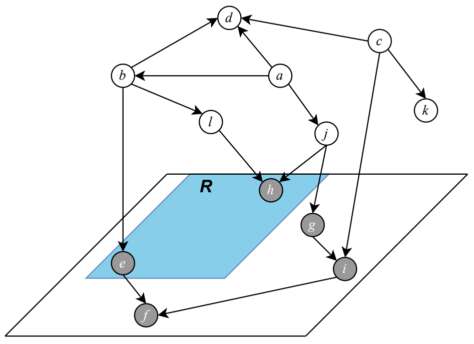

Figure 1: Running example; spatial vertices in gray.

We now formally define the problem of *geosocial reachability*, previously introduced in [47].

Problem 1 (GeoSocial Reachability). *Given a geosocial network $G = (V, E, P)$, a vertex $v \in V$ and a spatial region R, the* geosocial reachability *query* RangeReach(G, v, R) *determines that vertex v can* geosocially reach *region R iff G contains a path from v to a vertex u with $u.point$ located inside region R. Formally:*

$$\text{RangeReach}(G, v, R) = \begin{cases} \textit{TRUE}, & \textbf{if } \exists\, u \in V \text{ such that} \\ & \quad \text{GReach}(G, v, u) = \textit{TRUE} \textbf{ and} \\ & \quad u.point \in \text{SRange}(P, R) \\ \textit{FALSE}, & \textbf{otherwise} \end{cases}$$

Example 2.3. Consider the geosocial network G in Figure 1. The query RangeReach(G, a, R) returns *TRUE* as there exists a path in G from the query vertex a to either of the spatial vertices e and h, located inside the query region R; on the other hand, the query RangeReach(G, c, R) returns *FALSE*.

2.2 Existing Solutions

We next revisit the two existing solutions for RangeReach queries.

2.2.1 *The Spatial-first Approach.*

Intuitively, the RangeReach query comes as a hybrid that involves a graph (or social) predicate and a spatial predicate. Under this premise, a straightforward approach for processing RangeReach$(G = (V, E, P), v, R)$ queries is first to evaluate the SRange(P, R) spatial range query identifying every $u.point \in P$ located inside the query region R, and then execute a series of graph reachability GReach(G, v, u) queries, i.e., one for each vertex $u \in V$ with $u.point \in$ SRange(P, R). This spatial-first approach is captured by the SpaReach algorithm discussed in [47]. The algorithm terminates either when a GReach(G, v, u) query provides a positive answer, in which case, the SpaReach returns *TRUE* for the RangeReach query, or after all graph reachability queries provide a negative answer, in which case, SpaReach returns *FALSE*.

Example 2.4. Consider again the network in Figure 1 and query RangeReach(G, a, R). SpaReach issues a spatial range query to determine the spatial vertices inside R, i.e., vertices e and h. Then, it returns *TRUE* as GReach(G, a, e) = *TRUE*. In contrast, the algorithm returns *FALSE* for RangeReach(G, c, R) as both GReach(G, c, e) = *FALSE* and GReach(G, c, h) = *FALSE*.

To enhance SpaReach, spatial indexing is used for the spatial range query and a graph labeling scheme for every consecutive graph reachability query (see Section 7). Sarwat and Sun [47]

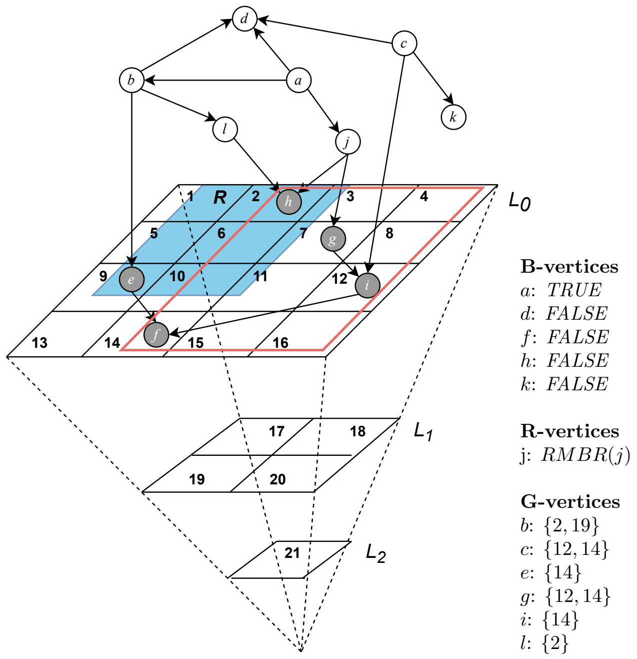

Figure 2: SPA-Graph for the geosocial network in Figure 1; *RMBR*(j) highlighted in red.

considered two versions of the spatial-first approach, which both rely on an R-tree [30] to index the points in P and then use either the *PLL* reachability index [64] (Method SpaReach-PLL), or the *Feline* reachability index [59] (Method SpaReach-Feline).

2.2.2 The GeoReach *Method* [47]. The key idea of GeoReach is to augment the vertices of the geosocial network $G = (V, E, P)$ with precomputed spatial and graph reachability information. This new, augmented network is called the *SPA-Graph*. To construct SPA-Graph, the authors first partition the two-dimensional space using a hierarchical grid. Every spatial vertex in G is contained inside one cell of this grid. Then, each vertex $v \in V$ is classified into one of the following three classes:

(1) **B-vertices**: the vertex carries the *Spatial Reachability Bit* or *GeoB*(v), which is set to *TRUE* if v can reach any spatial vertex in the network;
(2) **R-vertices**: the vertex stores the *Reachability Minimum Bounding Rectangle* or *RMBR*(v), which represents the minimum bounding rectangle in the two-dimensional space that encloses all spatial vertices, reachable from v;
(3) **G-vertices**: the vertex stores the *Reachability Grid* set or *ReachGrid*(v), which contains the grid cells (potentially from different levels) wherein all spatial vertices reachable from v are located.

The SPA-Graph of a geosocial network is constructed offline based on three system parameters. `MAX_RMBR` captures the maximum allowed extent of *RMBR*(v); if the extent of the *MBR* exceeds this limit then vertex v is downgraded to a B-vertex. `MAX_REACH_GRIDS` specifies the maximum allowed cardinality of *ReachGrid*(v); if the size of the set exceeds this limit, then v is downgraded to an R-vertex, and its *RMBR*(v) is stored instead of the *ReachGrid*(v) set. Finally, `MERGE_COUNT` determines whether adjacent quad-cells inside the *ReachGrid*(v) of a G-vertex v should be merged; If the number of adjacent cells exceeds this limit, then the cells are merged and represented by a cell in the next level; the merge process starts from level L_0, i.e., the most detailed partitioning of the space.

Example 2.5. Figure 2 illustrates the SPA-Graph for the geosocial network G in Figure 1, with the construction parameters set as follows: `MAX_RMBR` = $0.8 \cdot$ *SPACE*, `MAX_REACH_GRIDS` = 3 and `MERGE_COUNT` = 1, where *SPACE* is the extent of the entire two-dimensional space covered by G. Vertex a is classified as a B-vertex because its *RMBR* covers an area larger than `MAX_RMBR`. In contrast, j is an R-vertex and *RMBR*(j) is drawn in red. The *ReachGrid*(b) for G-vertex b would initially contain cells 2, 9 and 14 but because of `MERGE_COUNT` = 1, adjacent quad-cells 9 and 14 are merged and replaced by cell 19 in L_1.

Given a RangeReach(G, v, R) query, the GeoReach method traverses the SPA-Graph of the geosocial network G starting from the query vertex v (e.g., in a breadth-first fashion) and leverages the above auxiliary information to prune the search space. Specifically, let u be the current vertex. GeoReach stops its expansion if u is a B-vertex and *GeoB*(u) = *FALSE* as v will not be able to reach a spatial vertex via u. Similarly, the method prunes an R-vertex u if *RMBR*(u) has no overlap to the query region R, as no reachable spatial vertex of u is contained inside R. In contrast, if *RMBR*(u) intersects R, GeoReach must consider the outgoing edges of u, while if *RMBR*(u) is in fact *fully* contained inside R, GeoReach can safely terminate the search and return *TRUE* as the final answer to the RangeReach query. Last, if v is a G-vertex, GeoReach uses *ReachGrid*(u) in a similar fashion. Specifically, the method prunes vertex u if query region R does not intersect a cell in *ReachGrid*(u), expands u if R intersects a grid cell, and terminates the search returning *TRUE* when a cell is fully contained inside R.

Example 2.6. Consider again SPA-Graph in Figure 2 and query RangeReach(G, a, R). GeoReach starts off with the query vertex a, which is a B-vertex. As *GeoB*(a) = *TRUE*, the method expands the search using the (a, b), (a, d), (a, j) edges. Vertex b is a G-vertex with *ReachGrid*$(b) = \{2, 19\}$. With cell 2 fully contained inside the query region R, GeoReach terminates the search and returns *TRUE*. Now, consider RangeReach(G, c, R). After examining the query G-vertex c, GeoReach returns *FALSE*, as neither cell 12 or 14 in *ReachGrid*(c) overlap with the query region R.

2.2.3 Discussion. The solutions discussed above suffer from the following shortcomings. First, both methods may perform poorly for RangeReach queries with a negative answer. In this case, SpaReach needs to evaluate all possible graph reachability queries based on the spatial vertices located inside the query region R, while GeoReach may need to traverse a large part of the SPA-Graph. In addition, SpaReach is sensitive to the selectivity of the RangeReach spatial predicate. A region R that contains a large number of spatial vertices results in a potentially large number of reachability queries to be evaluated, especially in the case of RangeReach queries with a negative answer. On the other hand, GeoReach does not prioritize either of the RangeReach predicates but, at the same time, does not take advantage of any graph labeling or indexing scheme proposed for the graph reachability problem. To tackle these shortcomings and accelerate the evaluation of RangeReach queries, we present two novel methods that capitalize on interval-based labeling for graph reachability queries in the following sections.

3 INTERVAL-BASED LABELING

To better understand the proposed methods, Section 3.1 briefly presents the necessary background on interval-based labeling. Then, Section 3.2 describes how to construct the scheme for a geosocial network. Similar to graph reachability literature (see Section 7.1), we assume that the input network is a directed-acyclic graph (DAG); in Section 5 we discuss how to handle arbitrary graphs. Lastly, Section 3.3 elaborates on the complexity and space requirements of interval labeling.

3.1 Background

The original interval-based labeling scheme [21, 22] focuses only on tree inputs; the scheme assigns to every vertex v a $[pre(v), post(v)]$ label, where $pre(v)$ is the pre-order traversal number of v and $post(v)$, its post-order. As an ancestor vertex v appears before (after) a descendant vertex u in the pre(post)-order traversal of the tree, a vertex u is reachable from v *iff* the following test is successful, $pre(v) < pre(u)$ and $post(v) < post(u)$.

In this work, we consider a variant of interval-based labeling, which operates on graphs, originally proposed by Agrawal et al. in [1]. The key idea behind this scheme is the introduction of a spanning tree (termed the *tree-cover*) to distinguish between tree and non-tree edges in the input graph G. Specifically, every vertex in the spanning tree T of the graph is assigned an $[index(v), post(v)]$ label, where $post(v)$ is again the post-order traversal number of v and $index(v)$ is the lowest post-order number of v's descendants; note that $index(v) \leq post(v)$ holds for every non-leaf vertex and $index(v) = post(v)$, for each leaf vertex. Furthermore, every vertex v may receive additional labels based on G's edges excluded from the spanning tree T. Consider such a non-spanning tree edge (v, u). To capture the reachability information stemming from the existence of such an edge, the scheme propagates the labels of vertex u to v and to all v's ancestors. After examining all non-spanning tree edges in the graph, every vertex is associated with a set of interval-based labels denoted by $\mathcal{L}(v)$. The size of such an interval-based scheme, i.e., the number of generated labels, clearly depends on the size of the spanning tree T. Agrawal et al. [1] studied how to construct the optimal spanning tree based on the number of ancestors per vertex. Besides, it is also possible to use compression to reduce $|\mathcal{L}(v)|$ of a vertex v, either by absorbing subsumed intervals, e.g., label [3,5] absorbs [4,5], or by merging adjacent intervals, e.g., labels [1,4] and [4,5] are merged and replaced by $[1, 5]$.

Given the interval-based labeling scheme for a graph G, we can determine whether a vertex v can reach another vertex u, i.e., answer the GReach(v, u) graph reachability query, by checking the inclusion of $post(u)$ in the labels of v. More specifically:

Lemma 3.1. *Let v, u be two vertices in a graph G. Vertex u is reachable from v in G* iff *there exists a label in $\mathcal{L}(v)$ that contains the post-order number of u. Formally:*

$$\text{GReach}(G, v, u) = \begin{cases} \mathit{TRUE}, & \textbf{if } \exists\ [\ell, h] \in \mathcal{L}(v) \text{ such that} \\ & \quad \ell \leq post(u) \leq h \\ \mathit{FALSE}, & \textbf{otherwise} \end{cases}$$

3.2 Labeling (Geo)Social Networks

The labeling scheme described in the previous section was proposed for input graphs in the shape of hierarchies, mainly to determine transitive relationships in knowledge bases [18]. These graphs typically contain a particular vertex with only outgoing edges, which becomes the root of their spanning tree. However, a geosocial network G may contain multiple vertices with only outgoing edges, each acting as the root for a separate spanning tree. Under this premise, we define a spanning *forest*, i.e., a collection of spanning trees, for a geosocial network.

ALGORITHM 1: Constructing interval-based labeling

```
Inputs    : geosocial network $G = (V, E, P)$
Variables : spanning forest $F = (V, E_F)$, set of
            non-spanning edges $E_{NF} = E \setminus E_F$, priority
            queue $Q$, post-order number $post(v)$, $\forall v \in V$
Output    : interval labels $\mathcal{L}(v)$, $\forall v \in V$

1  compute spanning forest $F$ of $G$;
2  while all vertices in $F$ not visited do
3      select a spanning tree $T$ in $F$;
4      traverse $T$ in post-order ;            ▷ Compute $post(\cdot)$'s
5  foreach vertex $v \in F$ do
6      $\mathcal{L}(v) = \{[post(v), post(v)]\}$;     ▷ Initialize $\mathcal{L}(v)$
7  foreach root vertex $v \in F$ do        ▷ Initialize $Q$ with
     roots
8      determine priority of $v$;                  ▷ Based on $G$
9      $Q$.push($v$);
10 while $Q$ is not empty do
11     $v \leftarrow Q$.pop();                    ▷ Get current vertex
12     foreach outgoing edge $(v, u) \in E_F$ do
13         $\mathcal{L}(v) \bigcup = \mathcal{L}(u)$;
14         foreach each ancestor $w$ of $v$ do ▷ Using labels
15             $\mathcal{L}(w) \bigcup = \mathcal{L}(u)$;
16         calculate priority of $u$;              ▷ Based on $G$
17         if $u \notin Q$ then
18             $Q$.push($u$);
19 $E_{NF} \leftarrow E \setminus E_F$;        ▷ Determine non-spanning edges
20 sort $E_{NF}$ by post-order number of their source vertex;
21 foreach edge $(v, u) \in E_{NF}$ do
22     $\mathcal{L}(u) \bigcup = \mathcal{L}(v)$;
23     foreach each ancestor $w$ of $v$ do           ▷ Using labels
24         $\mathcal{L}(w) \bigcup = \mathcal{L}(u)$;
25 foreach vertex $v \in T$ do
26     compress $\mathcal{L}(v)$;           ▷ absorb and merge labels
27 return $\mathcal{L}(v)$, $\forall v \in V$;
```

To construct the interval-based labeling for a geosocial network G, we present Algorithm 1. Similar to [1], the construction can be divided into three key steps:

(1) Compute the spanning forest F of the input G and assign post-order numbers to its vertices (Lines 1–4),
(2) Use each spanning tree T in forest F to construct the initial set of $\mathcal{L}(v)$ labels for every vertex v (Lines 5–18), and
(3) Examine all non-spanning edges to extend and construct the final $\mathcal{L}(v)$ label sets (Lines 19–24)

The differences to [1] are in the first two steps. Specifically for the first step, after computing the spanning forest F in Line 1, we assign the post-order number $post(v)$ to all network vertices G. For this purpose, in Lines 3-4, Algorithm 1 iteratively selects and traverses a spanning tree T from F in post-order fashion.

This process repeats until all vertices are visited. In the second step, after initializing $\mathcal{L}(v) = \{[post(v), post(v)]\}$ (Lines 5–6), we use the priority queue Q to examine the graph vertices and determine their labels based on the spanning forest F. The queue is initialized in Lines 7–9 with every possible root vertex in F, i.e., the vertices with only outgoing edges. The priority of a vertex is determined based on its number of incoming edges (in increasing order); ties are solved by the post-order number. With this priority policy in place, the vertices of zero incoming edges (i.e., the roots) are guaranteed to be examined first by the construction procedure. In each iteration of Lines 10–18, we remove the top element of Q as the current vertex v (Line 11). Next, we use its outgoing edges in the spanning forest F to assign labels, similar to how Agrawal et al. examines the non-spanning edges in [1]. Specifically, consider such an edge (v, u); we propagate the labels of u to $\mathcal{L}(v)$ (Line 13) and to each ancestor vertex w of v in F (Line 14–15). As vertex v is visited before u, we can identify its ancestors using the current version of the labeling scheme. This is reminiscent of a stabbing query on $post(v)$, which can be accelerated by traditional interval indexing such as the interval tree [25] or recent highly efficient in-memory structures [16, 17].

Note that we push vertex u to the queue to expand the construction process (Lines 16–18). Finally, the algorithm examines the non-spanning edges (v, u) in Lines 19–24, similarly to [1] copying the labels $\mathcal{L}(u)$ of each target vertex u to $\mathcal{L}(v)$ and v's ancestors. Algorithm 1 completes the construction process by compressing the generated labels in Lines 25–26.

Example 3.2. Consider again the geosocial network in Figure 1; Table 1 details the construction of its interval-based labeling. As the first step, Algorithm 1 computes the spanning forest F shown in Figure 3. The algorithm traverses the spanning trees in F, i.e., the one rooted by vertex a and the second, by c, in post-order fashion. After initializing label sets $\mathcal{L}(\cdot)$ with $[post(\cdot), post(\cdot)]$, each vertex with zero incoming edges is added to the priority queue, i.e., $Q = \{a, c\}$. Next, the algorithm uses Q to traverse the spanning forest F, to determine the labels in the second column of Table 1. Algorithm 1 first visits vertex a and examines its outgoing edges (a, b), (a, d) and (a, j). This results in copying the contents of $\mathcal{L}(b)$, $\mathcal{L}(d)$ and $\mathcal{L}(j)$ to $\mathcal{L}(a)$, i.e., $\mathcal{L}(a) = \{[9,9]$ $[4,4]$ $[5,5]$ $[8,8]\}$. Last, vertices b, d, and j are added to Q to expand the search, i.e., $Q = \{c, b, j, d\}$; note that c exhibits the highest priority as it has zero incoming edges, followed by b and j with one incoming edge and d with two. The algorithm then processes c similarly to a, resulting in $\mathcal{L}(c) = \{[5,5]$ $[10,10]$ $[11,11]\}$ and $Q = \{b, i, j, k, d\}$. The next vertex to visit is b. Using its outgoing edges (b, e) and (b, l), the labels of vertex e and l are copied to $\mathcal{L}(b)$ and to $\mathcal{L}(a)$ of its ancestor a. As a result, we have $\mathcal{L}(b) = \{[4,4]$ $[2,2]$ $[3,3]\}$ and $\mathcal{L}(a) = \{[9,9]$ $[4,4]$ $[5,5]$ $[8,8]$ $[2,2]$ $[3,3]\}$. Last, the priority queue is also updated to $Q = \{e, i, j, k, l, d\}$. Algorithm 1 proceeds in a similar manner until Q is depleted.

The next step is to examine the non-spanning edges of the network sorted by the post-order number of their source, i.e., (l, h), (b, d), (g, i), (i, f) and (c, d) Using (l, h), the labels of h are copied to $\mathcal{L}(l)$ and its ancestors b and a. Hence, we have $\mathcal{L}(l) \bigcup = \{[7,7]\}$, $\mathcal{L}(b) \bigcup = \{[7,7]\}$ and $\mathcal{L}(a) \bigcup = \{[7,7]\}$. The algorithm proceeds similarly to construct the labels in the third column of Table 1. Finally, Algorithm 1 compresses the labels for each vertex v to produce the final version of $\mathcal{L}(v)$, which corresponds to the last column in Table 1.

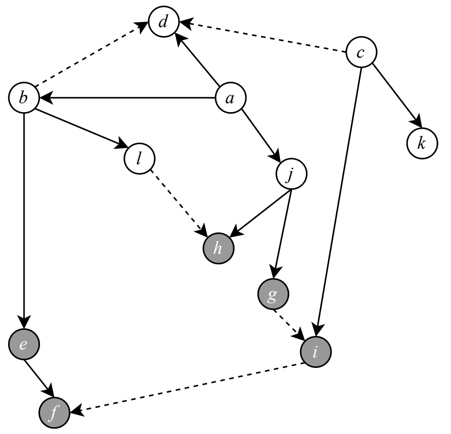

Figure 3: Spanning forest for the geosocial network in Figure 1; non-spanning edges are drawn with dashed lines. The forest contains two spanning trees; rooted by vertex *a* and *c*.

3.3 Analysis

We start with the time complexity of constructing the interval-based labeling. The cost of Algorithm 1 is linear to the sum of the number of vertices and edges in the input network G, i.e., $O(|V| + |E|)$. Essentially, the algorithm visits every vertex and edge in G three times in total. The first two times are when the spanning forest F is constructed and then traversed to determine the post-order number for each vertex, in Lines 1–4. Then, in Lines 10–18, Algorithm 1 visits each network vertex and every spanning edge for the third time. To determine the ancestors of current vertex v when propagating labels in Lines 14-15, we do not traverse the spanning forest backwards; instead, we utilize the current (incomplete) labeling scheme. Finally, the non-spanning edges are visited for the last time in Lines 19–24; note that the label propagation in Lines 23-24 is again handled using the existing scheme.

Finally, the space complexity of the interval-based labeling is $O(|V|^2)$, similar to all labeling schemes for graph reachability, which essentially compress the transitive closure of the graph (see Section 7.1). However in practice, the scheme occupies significantly less space as previous studies have shown. Additionally, note that the label compression in Lines 25–26 further reduces the space requirements. In Section 6, Table 6 highlights the merit of compression, reporting the number of labels contained in both the uncompressed and compressed schemes for our test datasets.

4 EVALUATING RangeReach QUERIES

To answer RangeReach queries, we could straightforwardly employ the interval-based labeling from Section 3 under the spatial-first approach in Section 2.2.1. Essentially, we use the interval labels for the reachability queries defined on the spatial vertices in the query region. Despite its simplicity, we do not expect this idea to accelerate SpaReach as the most recent graph reachability methods (see Section 7.1), outperform interval-based labeling.[2] Instead, we design two novel methods that use interval-based labeling in different fashions.

[2]In Section 6, we confirm this intuition experimentally.

Table 1: Interval-based labeling for the geosocial network in Figure 1.

vertex v ($post(v)$)	$\mathcal{L}(v)$		
	spanning forest	non-spanning edges	final
a (9)	[9,9] [4,4] [5,5] [8,8] [2,2] [3,3] [1,1] [7,7] [6,6]	[7,7] [5,5] [1,1] [10,10]	[1,10]
b (4)	[4,4] [2,2] [3,3] [1,1]	[7,7] [5,5]	[1,5] [7,7]
c (12)	[12,12] [10,10] [11,11]	[1,1] [5,5]	[1,1] [5,5] [10,12]
d (5)	[5,5]		[5,5]
e (2)	[2,2] [1,1]		[1,2]
f (1)	[1,1]		[1,1]
g (6)	[6,6]	[1,1] [10,10]	[1,1] [6,6] [10,10]
h (7)	[7,7]		[7,7]
i (10)	[10,10]	[1,1]	[1,1] [10,10]
j (8)	[8,8] [7,7] [6,6]	[1,1] [10,10]	[1,1] [6,8] [10,10]
k (11)	[11,11]		[11,11]
l (3)	[3,3]	[7,7]	[3,3] [7,7]

4.1 A Social-first Approach

We begin with a *social-first* approach termed SocReach, which prioritizes the graph (social) predicate of a RangeReach query. Intuitively, queries are evaluated in two steps, similar to the spatial-first approach. We first determine the descendants $\mathcal{D}(v)$ of the query vertex v in the geosocial network, and then we test if there exists a vertex $u \in \mathcal{D}(v)$, with $u.point$ located inside R. If such a vertex exists, SocReach returns *TRUE* as the final answer to the RangeReach(G, v, R) query. Otherwise, if all spatial containment tests fail, the method returns *FALSE*.

As the set of descendant vertices $\mathcal{D}(v)$ is computed on-the-fly, the spatial containment tests cannot be truly accelerated by any spatial indexing. Moreover, on average, not all spatial tests will be conducted in practice for queries with a positive answer. Therefore, we focus on the first step of SocReach, to investigate how we can compute the $\mathcal{D}(v)$ set fast. The majority of the solutions discussed in Section 7.1 exclusively target the graph reachability problem (Definition 2.1), i.e., answering GReach(v, u) queries. Under this premise, the proposed schemes cannot be employed to determine the descendants of the query vertex v. However, as discussed in [18], interval-based labeling can be used for this purpose; the idea is captured by the formula below, which follows from Lemma 3.1:

$$\mathcal{D}(v) = \{u \in G : \exists\ [\ell, h] \in \mathcal{L}(v), \ell \le post(u) \le h\} \qquad (1)$$

Essentially, every $[\ell, h]$ interval inside the $\mathcal{L}(v)$ set of the query vertex v defines a typical (relational) range query over the post-order numbers of the network vertices. Depending on the size of the geosocial network and whether gaps in the post-order numbers are used to accommodate updates (vertex insertions), these range queries can be evaluated using, for instance, a traditional B$^+$-tree which indexes $post(\cdot)$ or even as simple for loops on the array storing the network vertices in main memory.

Example 4.1. Consider again the network G in Figure 1, its interval-based labeling scheme in Table 1. For RangeReach(G, a, R) query, set $\mathcal{D}(a)$ contains vertices $\{f, e, l, b, d, g, h, j, a, i\}$ whose $post(\cdot) \in$ [1,10]. For vertex e, $e.point$ is contained inside region R and so, SocReach returns *TRUE*. For RangeReach(G, c, R), set $\mathcal{D}(c)$ contains vertices with post-order numbers inside ranges [1,1], [5,5] and [10,12], i.e., $\mathcal{D}(c) = \{f, d, i, k, c\}$. SocReach returns *FALSE* for this query as vertices d, c and k are not spatial and $f.point$ and $i.point$ are not contained in R.

4.2 The 3DReach Method

Despite offering an alternative to SpaReach, the SocReach approach exhibits a similar weakness. As the social (or graph) predicate of RangeReach is prioritised (instead of the spatial), a larger number of spatial containment tests may need to be evaluated, especially for query vertices with a high out-degree and/or for queries with a negative answer. In view of this, we devise a novel solution termed 3DReach, which not only capitalizes on interval-based labeling but also considers both RangeReach predicates at the same time.

The key idea of 3DReach is to model the spatial vertices of the input geosocial network $G = (V, E, P)$ and its interval-based labeling $\mathcal{L}$ inside a three-dimensional space. The first two dimensions model the original two-dimensional space wherein every $v.point \in P$ is located, while the third dimension models the domain of the post-order numbers in $\mathcal{L}$. Under this transformation, every spatial vertex v in the network is modelled as a three-dimensional point $(v.point, post(v))$. Consequently, a RangeReach(G, v, R) query is rewritten as a set of three-dimensional range queries or rectangular cuboids, one for each label $[\ell, h] \in \mathcal{L}(v)$. The base of every cuboid corresponds to the query region R, thus capturing the original spatial range predicate of the query. Also, the cuboid is positioned in-between values ℓ and h in the third dimension, capturing the computation of $\mathcal{D}(v)$ and, therefore, the graph predicate of the query. As such, to answer a RangeReach(G, v, R) query, it suffices to check whether the 3D point of a spatial vertex u is located inside one of these cuboids, since the following would hold: (1) $u.point \in R$ and (2) $\ell \le post(u) \le h$, i.e., GReach(v, u) = *TRUE*.

Example 4.2. Figures 4(a) and (c) illustrate the transformation employed by 3DReach, and how queries RangeReach(G, a, R) and RangeReach(G, c, R) are evaluated, respectively. As $\mathcal{L}(a)$ contains one label, i.e., [1,10], RangeReach(G, a, R) is captured by a single 3D range query or simply the light blue cuboid in Figure 4(a). The cuboid contains the 3D point that models spatial vertex e and so, 3DReach returns *TRUE*. On the other hand, RangeReach(G, c, R) is represented by the three cuboids (more precisely, one cuboid and two planes) in Figure 4(c) corresponding to the $\mathcal{L}(c) = \{$[1,1] [5,5] [10,12]$\}$ labels. Since no cuboid contains a spatial vertex, 3DReach returns *FALSE* as the answer to the query.

3DReach computes both the social and the spatial predicate in one step but, executing multiple three-dimensional range queries

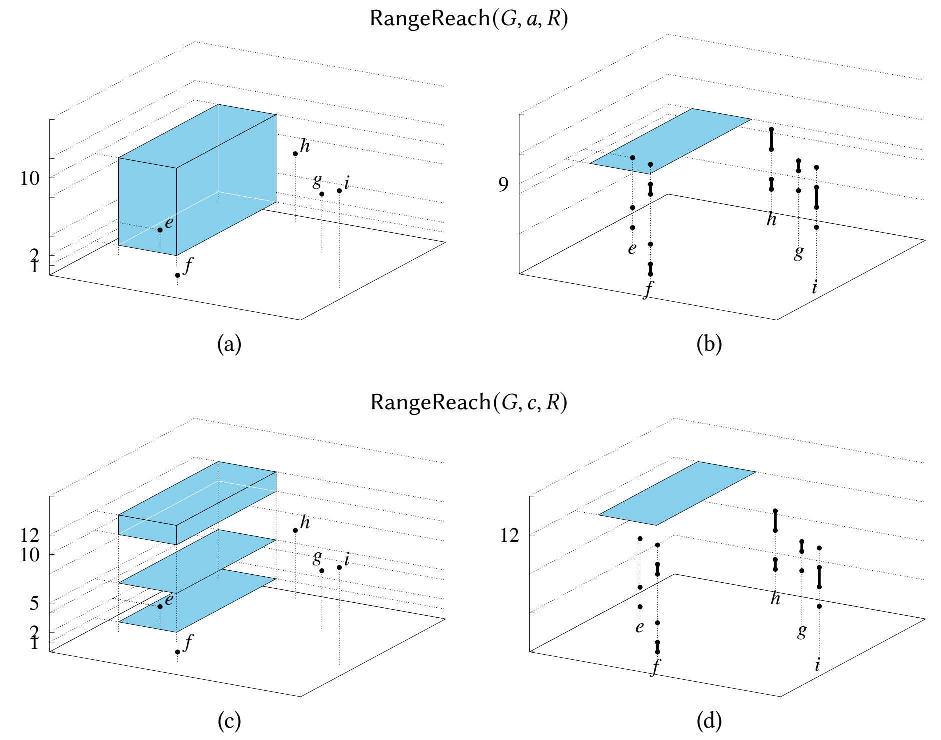

Figure 4: 3DReach variants on Figure's 1 geosocial network; query areas highlighted in light blue.

might slow down the evaluation of RangeReach, especially in case of a negative answer, when all the three-dimensional range queries must be evaluated. In view of this, we devise a 3DReach variant under which a RangeReach query is rewritten as a *single* three-dimensional range query. In Section 6, we extensively compare the two variants. The key idea is to construct the *reversed* interval-based labeling scheme for the geosocial network G. Every label $[\ell, h] \in \mathcal{L}(v)$ of this scheme captures a range of post-order numbers for the ancestors of vertex v. We can construct the *reversed* interval-based labeling for a network by first reversing its edges and then employing again Algorithm 1. Table 2 illustrates the *reversed* interval-based labeling for the network in Figure 1. With the *reversed* labeling scheme, every spatial vertex v is now modelled as a set of vertical line segments, one for each $[\ell, h]$ label in $\mathcal{L}(v)$. Finally, under this new three-dimensional transformation, a RangeReach(G, v, R) query is represented by a single plane, parallel to the first two dimensions of the space which captures the coordinates of the query range R, but positioned at $post(v)$ in the third dimension. This *line-based* variant of 3DReach returns *TRUE* if the query plane 'cuts' at least one vertical line segment, which means that there exists at least one spatial vertex in the geosocial network whose *point* is inside R, with v being its ancestor vertex.

Example 4.3. Consider again our running example. Figures 4(b) and (d) illustrate the *line*-based variant of 3DReach. Observe how every spatial vertex is now modelled as a set of vertical lines compared to (a) and (c), and the *point*-based 3DReach. For query RangeReach(G, a, R), the 3D point for spatial vertex e is on the query plane which results to a positive answer. On the other hand, the RangeReach(G, c, R) query is now represented by only one plane compared to a cuboid and two planes in Figure 4(c), allowing the method to issue a single, unsuccessful three-dimensional range query and immediately return *FALSE*.

5 DEALING WITH ARBITRARY GRAPHS

Typically, the vast majority of the graph reachability methods (see Section 7.1) assume that the input graph G is a DAG. Otherwise, a simple solution is to first identify all strongly connected components in G using, e.g., Tarjan's algorithm [57], and then replace each component with a (super-)vertex. By definition, every pair of vertices inside a strongly connected component can reach each other. We adopt the same approach to construct our interval-based labeling scheme with a necessary extension. Due to spatial vertices, we need to define the spatial information of every strongly connected component in the input geosocial network that contains spatial vertices. Let v_c be a super-vertex that represents a component C, we consider the following two alternatives:

(1) Replace v_c by the spatial vertices contained inside C, replicating its reachability information. More specifically, every such vertex u inherits its labels from super-vertex v_c, i.e., $\mathcal{L}(u) = \mathcal{L}(v_c)$.
(2) Define as spatial information for v_c the minimum bounding rectangle (*MBR*) that includes the points of all spatial vertices contained in the component C; under this premise, $v_c.point$ is no longer a single point but the above *MBR*.

Section 6 includes extra tests to compare these two approaches. In addition, we consider two distinct cases of arbitrary graphs with respect to the number of their SCCs

6 EXPERIMENTAL ANALYSIS

We finally present our experimental analysis on RangeReach queries. The tests ran on a machine with 32 Intel Core i9 CPUs clocked at 5.8GHz with 64 GBs of RAM, running Ubuntu Linux. All data (i.e., networks and index structures) resided in main memory.

Table 2: Reversed interval-based labeling for the geosocial network in Figure 1.

vertex v ($post(v)$)	$\mathcal{L}(v)$		
	spanning forest	non-spanning edges	final
a (9)	[9,9]		[9,9]
b (4)	[4,4] [9,9]		[4,4] [9,9]
c (12)	[12,12]		[12,12]
d (5)	[5,5] [9,9]	[4,4] [9,9] [12,12]	[4,5] [9,9] [12,12]
e (2)	[2,2] [4,4] [9,9]		[2,2] [4,4] [9,9]
f (1)	[1,1] [2,2] [4,4] [9,9]	[6,6] [8,8] [10,10] [12,12]	[1,2] [4,4] [6,6] [8,10] [12,12]
g (6)	[6,6] [8,8] [9,9]		[6,6] [8,9]
h (7)	[7,7] [8,8] [9,9]	[3,3] [4,4] [9,9]	[3,4] [7,9]
i (10)	[10,10] [12,12]	[6,6] [8,8] [9,9]	[6,6] [8,10] [12,12]
j (8)	[8,8] [9,9]		[8,9]
k (11)	[11,11] [12,12]		[11,12]
l (3)	[3,3] [4,4] [9,9]		[3,4] [9,9]

Table 3: Dataset characteristics

dataset	# users	# venues	# checkins/ratings	$\|V\|$	$\|E\|$	$\|P\|$	# SCCs	# vertices in largest SCC
Foursquare	2,119,987	1,132,617	4,801,576	3,252,604	19,685,786	1,132,617	1,400,154	1,852,251
Gowalla	407,533	2,723,102	35,676,249	3,130,635	23,778,362	2,723,102	2,723,103	407,533
WeePlaces	16,022	971,309	7,658,368	987,331	2,758,946	971,309	971,311	16,021
Yelp	1,987,693	150,310	6,990,247	2,138,003	21,357,271	150,310	1,238,535	892,152

6.1 Setup

We experimented with four real geosocial networks; Foursquare[3] [38, 46], Gowalla[4] [41], WeePlaces[4] and Yelp[5], which contain users as social (non-spatial) vertices and venues as spatial. Table 3 summarizes their characteristics. Each venue is associated with a pair of geo-coordinates. The edges of the networks model friend relationships between users and check-ins or ratings from users to venues. The inputs represent two distinct cases of geosocial networks. The social vertices of Gowalla and WeePlaces are fully connected to each other, forming a large SCC; notice how the number of users equals the number of vertices in the largest SCC. In this case, the cost of RangeReach queries is dominated by the cost of its spatial range predicate. In contrast, the social vertices of Foursquare and Yelp form several SCCs distributing the cost between graph reachability and spatial range.

For our analysis, the following RangeReach evaluation methods were implemented in C++, compiled using gcc (v11.4.0) with the -O3 flag activated.[6] For the methods employing an R-tree, we used the implementation included in the Boost library.[7]

- SpaReach-BFL adopts the spatial-first approach in Section 2.2.1. An R-tree indexes the spatial vertices in the network and the BFL reachability scheme [52] answers GReach queries.
- SpaReach-INT also adopts the spatial-first approach, but employs the interval-based labeling [1] for the GReach queries.
- GeoReach is the state-of-the-art evaluation method from [47]; We used the source code provided by the authors[8]. We set the construction parameters as suggested by the authors.
- SocReach is our social-first method presented in Section 4.1.
- 3DReach from the first part of Section 4.2, uses the interval-based labeling to represent every spatial vertex as a three-dimensional point. The three-dimensional space is indexed by a 3D R-tree.
- 3DReach-Rev, the line-based variant of 3DReach from Section 4.2, utilizes the *reversed* interval-based labeling to index spatial vertices as sets of vertical line segments. The three-dimensional space is again indexed by a 3D R-tree.

To assess the performance of the methods, we measured their average runtime over 1000 queries, while varying the extent of the query region R inside the $\{1\%, 2\%, \mathbf{5\%}, 10\%, 20\%\}$ range, as a percentage of the entire space covered by the network, and the degree of the query vertex v inside the $\{[1–49], [50–99], \mathbf{[100\text{-}149]}, [150-199], [200-\ldots]\}$ intervals, based on the number of *outgoing* edges.[9] In each test, we vary the value of one of the above parameters while setting the other to its default value, highlighted in bold. To better understand how the spatial predicate of a RangeReach query affects the performance, we also vary its spatial selectivity inside the $\{0.001\%, 0.01\%, 0.1\%, 1\%\}$ value range, as the percentage of the spatial vertices found inside the query range R over the total number of network vertices. This test was also conducted in [47].

6.2 Handling Arbitrary Graphs

All four tested geosocial networks are not acyclic; following the typical practice, we converted them into DAGs, replacing every strongly connected component by a super-vertex. In this context,

[3]https://academictorrents.com/details/b24c73949308b3f6bdd8fea1a485534392eef338
[4]https://www.yongliu.org/datasets.html
[5]https://www.yelp.com/dataset
[6]Source code and test datasets are available in https://github.com/pbour/rangereach
[7]https://www.boost.org
[8]https://github.com/DataSystemsLab/GeoGraphDB–Neo4j

[9]We ignore the incoming edges in the degree computation, as we are only interested in the spatial vertices reachable from the query vertex v but not vice versa.

Table 4: Index size [MBs]: space occupied by the SPA-graph for GeoReach, the BFL scheme and the 2D R-tree for SpaReach-BFL, the interval-based labeling scheme and the 2D R-tree for SpaReach-INT, the interval-based labeling scheme and the 3D R-tree for 3DReach, the 3D R-tree for 3DReach-Rev; in parenthesis, the MBR-based variant (if exists)

dataset	SpaReach-BFL	SpaReach-INT	GeoReach	SocReach	3DReach	3DReach-Rev
Foursquare	78.8 (87.4)	28.6 (37.2)	17.1	13.9	33.5 (46.4)	69.4 (69.4)
Gowalla	160 (181)	56.1 (76.9)	8.56	20.8	67.9 (99.0)	157 (157)
Weeplaces	57.1 (64.5)	20.0 (27.4)	1.30	7.41	24.2 (35.3)	55.8 (55.8)
Yelp	58.6 (59.8)	29.3 (30.4)	240	27.3	29.9 (31.7)	44.8 (44.8)

Table 5: Indexing time [secs]: building the SPA-graph for GeoReach, the BFL scheme and the 2D R-tree for SpaReach-BFL, the interval-based labeling scheme and the 2D R-tree for SpaReach-INT, the interval-based labeling scheme and the 3D R-tree for 3DReach and 3DReach-Rev; in parenthesis, the MBR-based variant (if exists)

dataset	SpaReach-BFL	SpaReach-INT	GeoReach	SocReach	3DReach	3DReach-Rev
Foursquare	7.80 (7.81)	9.49 (9.50)	1636	9.35	9.50 (9.51)	10.6 (10.8)
Gowalla	9.19 (9.22)	12.1 (12.1)	294	11.76	12.1 (12.2)	13.6 (13.7)
Weeplaces	1.37 (1.38)	2.09 (2.09)	1.20	12.47	2.14 (2.25)	2.73 (2.77)
Yelp	6.74 (6.74)	9.15 (9.15)	3745	9.13	9.15 (9.17)	9.94 (9.96)

Table 6: Interval-based labeling stats

dataset			reversed	
	uncompressed	compressed	uncompressed	compressed
Foursquare	2,954,273	1,820,038	2,954,273	2,681,946
Gowalla	5,446,205	2,723,103	5,446,205	5,446,204
Weeplaces	1,942,621	971,311	1,942,621	1,942,620
Yelp	3,737,986	3,584,013	3,737,986	2,645,720

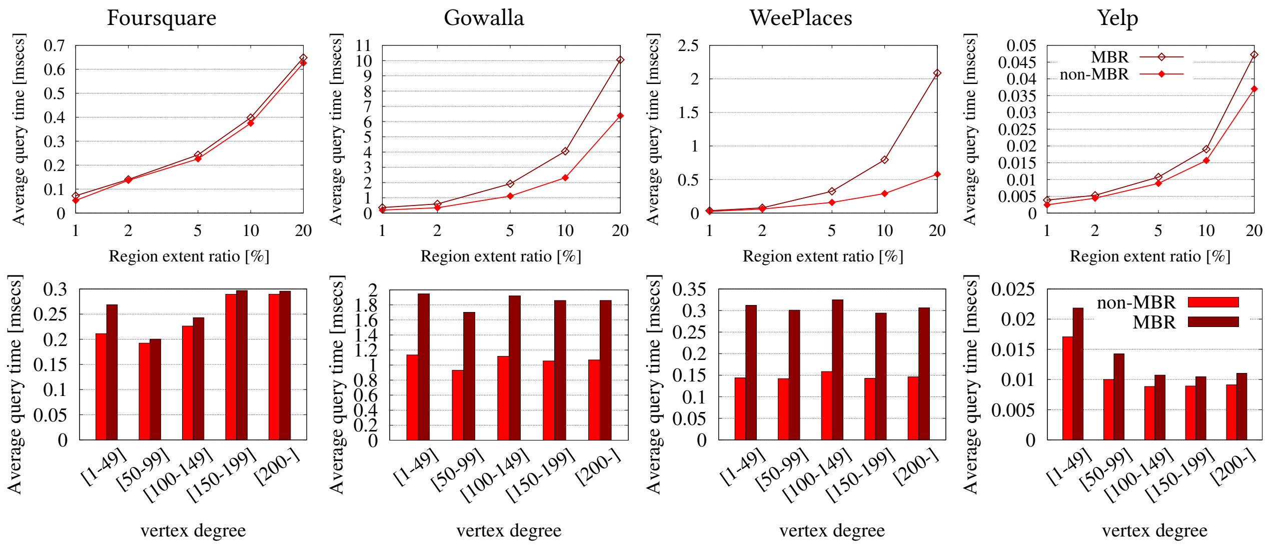

Figure 5: Handling spatial strongly connected components

our first set of experiments studies the best practice to deal with strongly connected components that contain spatial vertices.

We implemented two variants for the spatial-first, SpaReach-BFL and SpaReach-INT, and the 3DReach methods, following the discussion in Section 5. The first variant replicates the reachability information of each super-vertex (strongly connected component) to all contained vertices and completely discards the super-vertices when answering RangeReach queries. The second variant (termed MBR-based) defines the spatial information of a super-vertex as the MBR which encloses the points of all its spatial vertices. Note that we exclude from this discussion our SocReach which does not involve any spatial indexing, and GeoReach which always operates under a non-MBR principle, by design.

Table 4 and 5 report on the indexing costs of each variant; the numbers for the MBR-based are given in parentheses. For all networks, we observe that the MBR-based variant insignificantly affects the indexing time, but does notably increase the space requirements by 10% for SpaReach-BFL, 26% for SpaReach-INT and 33% for 3DReach, on average; for 3DReach-Rev, we observe no increase in the space requirements because Boost's implementation of the R-tree stores segments and boxes in a similar manner. The main reason for the increase space is that spatial indexing becomes more expensive; the 2D R-tree for the MBR-based variant of the spatial-first methods no longer indexes points but rectangles, and the 3D R-tree for 3DReach and 3DReach-Rev indexes boxes, instead of points and lines, respectively.

With regards to the querying performance, Figure 5 compares the two SpaReach-INT variants while varying the extent of the query range and the degree of the query vertex. For simplicity, we omit the results for the variants of the other methods, where similar observations were made. The plots clearly show that the non-MBR based variant always outperforms the MBR one. As already explained in the previous paragraph, the key reason is the higher cost of the spatial range query on the 2D R-tree and the 3D R-trees, which no longer index points or segments but rectangles or boxes. Under this premise, we only consider the non-MBR variant of the evaluation methods, for the rest of our analysis.

6.3 Determining the Best SpaReach

In the second set of experiments, we seek the best spatial-first method, comparing SpaReach-BFL to SpaReach-INT. Tables 4 and 5 (first and second column) report on their indexing cost while Figure 6 reports their average query time. The results confirm our intuition from the first paragraph in Section 4. In almost all cases, SpaReach-BFL outperforms SpaReach-INT in answering RangeReach queries. Since both methods utilize a 2D R-tree to determine the spatial vertices inside the query region R, the advantage of SpaReach-BFL stems from the advantage of the BFL labeling scheme over the inteval-based labeling schemes in answering graph reachability queries (i.e., GReach). Under this, the results fully align with the previous studies on graph reachability (see Section 7.1). The performance gap is more pronounced on the larger geosocial networks of Foursquare and Gowalla, compared to Weeplaces and Yelp; those networks contain more spatial vertices which increases the average number of reachability queries employed when answering a RangeReach query.

Considering also the indexing cost of each spatial-first method, we essentially observe a typical space-time tradeoff. SpaReach-BFL has higher space requirements (2-3 times) in the expense of faster reachability queries, compared to SpaReach-INT. Nevertheless, our main focus is on fast query processing and therefore, we only consider SpaReach-BFL for the rest of our analysis.

6.4 Comparing Evaluation Methods

Finally, we compare our proposed SocReach and the two 3DReach methods against the GeoReach state-of-the-art and the best spatial-first method, SpaReach-BFL. Again, Tables 4 and 5 report on the indexing cost, while Figure 7 reports on the query performance.

There are two key takeways from our tests. On the one hand, SocReach is not competitive to the rest of the methods, with the exception to GeoReach for the smaller networks of WeePlaces and Yelp. This result was expected as SocReach prioritizes the social predicate of RangeReach and does not employ any spatial indexing to accelerate the necessary spatial containment tests. On the other hand (second takeaway), we also observe that the 3DReach methods are overall the fastest evaluation methods; they typically outperform state-of-the-art GeoReach, SpaReach-BFL and SocReach by multiple orders of magnitude, when varying both the query region extent and the degree of the query vertex. There are two reasons for this advantage. First, neither 3DReach or 3DReach-Rev need to traverse the geosocial network as GeoReach does, which naturally slows down the computation. Second, with the 3DReach methods, query evaluation is a single step process where the social (graph reachability) and spatial (range query) predicate of a RangeReach query are handled at the same time, while SpaReach-BFL prioritizes the spatial range predicate and SocReach, the social one. Between the two 3DReach methods, their times are very similar, but the original 3DReach which indexes 3D points instead of line segments is usually faster. 3DReach-Rev does evaluate a single 3D range query every time but this query takes longer in the 3D R-tree due to more complex data being indexed.

Regarding the tested parameters, the performance of SpaReach-BFL is negatively affected by the increase of both the query region extent and the number of contained spatial vertices; for the former, the cost of the spatial range query increases, while for the latter, the average number of the necessary graph reachability queries goes up. SocReach and GeoReach are affected in a similar fashion by all three parameters, albeit for different reasons. The performance of SocReach improves when increasing the query region extent and the contained spatial vertices as the method has to perform fewer spatial containment tests in average, but when increasing the query vertex degree, the number of descendants to be considered also rises, slowing down the queries. For GeoReach, the pruning with $GeoB(\cdot)$, $RMBR(\cdot)$ and $ReachGrid(\cdot)$ becomes more effective as we increase the query region extent and the number of contained points accelerating the query computation, but when we increase its out-degree, more paths need to be traversed from the query vertex, slowing down the queries. Last, the 3DReach methods are positively affected by the increase in the out-degree of the query vertex; this effect is more obvious for Foursquare and Yelp because these networks have the largest social cores, i.e., contain the highest number of users. In contrast, the spatial parameters have mixed effect on the 3DReach methods. The extent of the query region tends to affect very little the performance; although the 3D range queries have a larger extent as well, their computing cost is balanced by the fact that a contained point or an intersecting segment is easily found. Increasing the number of spatial vertices in the query range typically slows down the methods as it takes more time to compute the 3D range queries.

Last, regarding the indexing, the 3DReach methods offer competitive times and space requirements, especially the original 3DReach, whose 3D R-tree stores points instead of line segments. Observe how close are the index size numbers in Table 4 for 3DReach to GeoReach and SocReach, which usually exhibit the lower space requirements, and its indexing times in Table 5 to SpaReach-BFL, whose index is the faster to build. To elaborate on space requirements and highlight the merit of label compression, Table 6 reports the number of labels for interval-based labeling, both compressed and uncompressed. We observe that compression reduces the number of labels by 36% on average, for the original interval scheme used by 3DReach, while there is no significant benefit for the reverse scheme of 3DReach-Rev, which also explains why 3DReach-Rev exhibits higher indexing cost compared to 3DReach- in Tables 4 and 5.

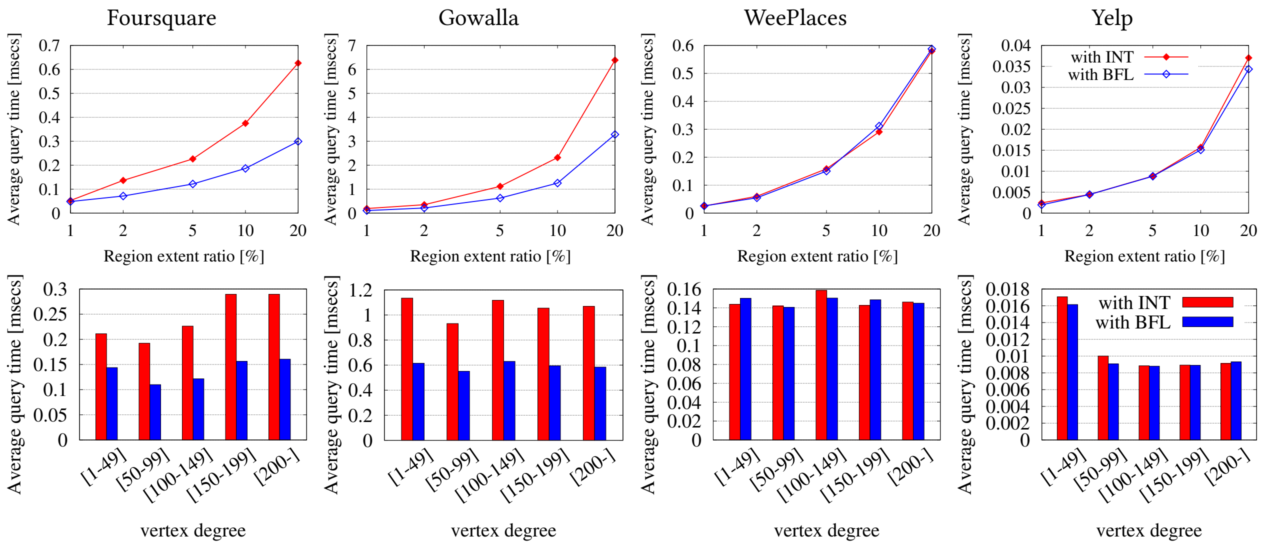

Figure 6: Determining the best spatial-first method

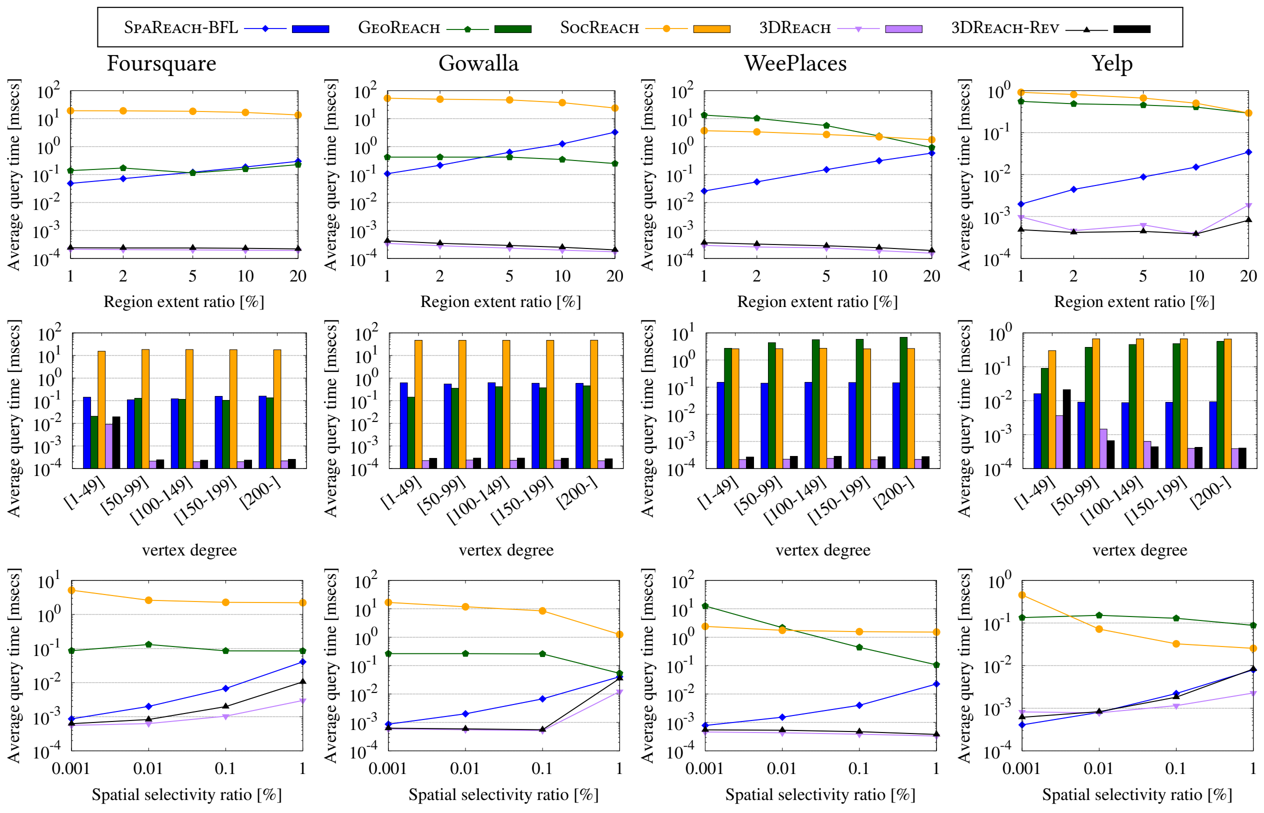

Figure 7: Comparing evaluation methods

7 RELATED WORK

We discuss additional related work besides the existing evaluation methods in Section 2.2 and the interval-based labeling in Section 3.

7.1 Graph Reachability

A straightforward solution to the reachability problem is to use the transitive closure (*TC*) of the graph. Then, GReach queries are answered in constant $O(1)$ time, but the $O(n^2)$ offline cost of maintaining *TC* is prohibitively high. In contrast, a GReach query can be answered with no offline cost by traversing the graph in $O(|V| + |E|)$ time. However, both solutions are impractical, especially for large graphs.

In view of the above, a plethora of indexing or labeling schemes have been proposed, trying to balance the online query cost, the offline construction cost, and the storage requirements for the index. Typically, these solutions adopt one of the following approaches. The goal of the methods adopting the *Label-Only* approach is to compress the graph *TC* for fast online querying; only the labels constructed offline are used during query evaluation. The interval-based labeling by Agrawal et al. [1] (Section 3) adopts this idea. Other works compress *TC* by a minimal number of pair-wise disjoint vertex chains [12, 32, 33]. Similar to [1], a spanning tree is used as the basis for indexing *TC* in *dual-labeling* [61], but the scheme is suitable only for sparse graphs. The 2-hop scheme [14, 15, 19, 48] constructs for each vertex v a list with part of the vertices that can reach v ($L_{in}[v]$) and part of those reachable from v ($L_{out}[v]$). Then, a GReach(v, u) is answered by checking whether $L_{out}[v]$ and $L_{in}[u]$ share a vertex or not. The 3-HOP scheme [35] improves 2-hop labeling by capitalizing on the Chain-Cover idea [32], while Path-hop [9] improves 3-HOP by replacing the chain decomposition with a tree structure. Some works also targeted how to reduce the construction cost of 2-hop, e.g., TF-Label [13], TL and DL [34], BLL [64], and TOL [70]. Last, Tang et al. [56] employ graph partitioning to accelerate GReach; the original graph is divided into two subgraphs, one topologically labeled and one labeled using 2-hop.

On the other hand, the methods that follow the *Label+G* approach use both the computed labels and potentially graph traversal to answer GReach; essentially, the focus here is on reducing the offline construction cost incurred by compressing the graph *TC*. Tree+SSPI [10] and GRIPP [58] employ interval-based labeling on the spanning tree of the input graph but also traverse this graph in a depth-first fashion, if needed. GRAIL [65, 66] uses a number of spanning trees to generate vertex labels, but, if this ensemble of labels is not enough to decide on the reachability, GRAIL uses depth-first search. FERRARI [49] also uses multiple labels per vertex. Feline [59] uses two topological orders to cover as many unreachable vertices as possible. IP [62, 63] is based on k-min-wise independent permutation to define multiple types of vertex labels. BFL [52] incorporates bloom filtering in its labeling scheme. Last, the multi-dimensional labeling scheme MG-Tag [69] relies on graph partitioning. Different to previous *Label+G* methods, when a GReach query cannot be answered using the labels, MG-Tag traverses the smaller in size, partitioned graph instead of the original one. Recently, directed acyclic graph reduction [67, 68] was further considered to accelerate reachability queries. The idea is to reduce the size of the input graph by computing its transitive reduction followed by the equivalence reduction.

SpaReach can use any of the above schemes to evaluate the involved GReach queries. For our tests, we consider BFL [52] due to its promising results. In contrast, as discussed in Section 4.1, SocReach cannot benefit from the majority of the works above, as their focus is on accelerating GReach and not on other graph problems such as computing the descendants of a vertex. Finally, our 3DReach explicitly builds upon the interval-based labeling.

7.2 Spatial Range Queries

Spatial *range* queries are typically evaluated with spatial indexing. In principle, these structures employ partitioning to prune the search space. A wide range of indices are proposed in the literature for both point and non-point data; our focus is on the former.

Depending on the nature of the partitioning, spatial indices can be classified into two classes. On the one hand, indices based on *space-oriented partitioning* (SOP) divide the space into disjoint partitions. A grid [6], which divides the space into cells (partitions) using axis-parallel lines, is the simplest SOP index. Hierarchical indices that fall in this category are the kd-tree [5] and the quad-tree [28]. On the other hand, indices based on *data-oriented* partitioning (DOP) allow the extent of the partitions to overlap but ensure that their contents are disjoint (i.e., each object is assigned to precisely one partition). The R-tree [30] and its variants (e.g., the R*-tree [4]) are the most popular methods in this class. The R-tree is a height-balanced tree, which generalizes the B$^+$-tree in the multi-dimensional space and hierarchically groups object MBRs to blocks. BLOCK [44] is a recently proposed main-memory DOP index, which uses a hierarchy of grids.

Lately, following the trend for relational data, learned indices have been proposed also for spatial data, where the main idea is to learn the spatial distribution of the objects and then define a lightweight index. Intuitively, the query evaluation is guided by models instead of a sparse index. Wang et al. [60] first map the data to a one-dimensional space, using their Z-order, and then construct a multi-staged learned index for one-dimensional data, similar to [36]. ML-Index [20] also uses a transformation to the one-dimensional space, but relative to the distance to the closest reference point. In LISA [40], the data are organized using a grid; the one-dimensional order of the cells and the data distribution determines the grouping of cells and the corresponding learned models. RSMI [45] suggests a rank space-based ordering, which becomes scalable by a recursive partitioning and learning strategy. Flood index [43] automatically adapts itself to a particular dataset and query workload by jointly optimizing the index structure and data storage layout. Al-Mamun et al. summarize learning indices for spatial (and multi-dimensional) data in [2].

Without loss of generality, our implementation of SpaReach uses an R-tree to index the spatial vertices similar to [47], as it is the most dominant structure for spatial data. Last, the R-tree employed by our 3DReach can be replaced by another structure as long as it is able to index the three-dimensional space defined for the spatial vertices of the geosocial network.

8 CONCLUSIONS AND FUTURE WORK

We studied the computation of RangeReach(G, v, R) queries in geosocial networks [47]. We proposed two novel methods which build on the interval-based labeling from [1]. SocReach employs the labeling to determine the descendants of the query vertex v and then compares their location to the query region R. The 3DReach approach first maps the network vertices into a three-dimensional space, which expands the original 2D space with an extra dimension from the interval-based labeling. Under this transformation, RangeReach is rewritten as a set of 3D range queries. Our experimental analysis on real geosocial networks showed that 3DReach is the fastest method for RangeReach queries, outperforming both the GeoReach [47] state-of-the-art method and the methods that prioritize either the spatial (SpaReach) or the social predicate of the query, such as our SocReach.

In the future, we plan to investigate how our approach can efficiently handle updates in the network and the role of optimal (e.g., shallow) spanning forests in the construction of the interval-based labeling, similar to [1]. Furthermore, we plan to examine the incorporation of our methods in existing systems for geosocial networks and consider the computation of other types of geosocial queries.

ACKNOWLEDGMENTS

The work builds upon Daniel Kowalski's Bachelor thesis at JGU Mainz. Theodoros Chondrogiannis was supported by Grant No. CH 2464/1-1 of the Deutsche Forschungsgemeinschaft (DFG).

REFERENCES

[1] Rakesh Agrawal, Alexander Borgida, and H. V. Jagadish. 1989. Efficient Management of Transitive Relationships in Large Data and Knowledge Bases. In *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data, Portland, Oregon, USA, May 31 - June 2, 1989.* ACM Press, 253–262. https://doi.org/10.1145/67544.66950

[2] Abdullah Al-Mamun, Hao Wu, and Walid G. Aref. 2020. A Tutorial on Learned Multi-dimensional Indexes. In *SIGSPATIAL '20: 28th International Conference on Advances in Geographic Information Systems, Seattle, WA, USA, November 3-6, 2020.* ACM, 1–4. https://doi.org/10.1145/3397536.3426358

[3] Nikos Armenatzoglou, Stavros Papadopoulos, and Dimitris Papadias. 2013. A General Framework for Geo-Social Query Processing. *Proc. VLDB Endow.* 6, 10 (2013), 913–924. https://doi.org/10.14778/2536206.2536218

[4] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. 1990. The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, Atlantic City, NJ, USA, May 23-25, 1990.* ACM Press, 322–331. https://doi.org/10.1145/93597.98741

[5] Jon Louis Bentley. 1975. Multidimensional Binary Search Trees Used for Associative Searching. *Commun. ACM* 18, 9 (1975), 509–517. https://doi.org/10.1145/361002.361007

[6] Jon Louis Bentley and Jerome H. Friedman. 1979. Data Structures for Range Searching. *ACM Comput. Surv.* 11, 4 (1979), 397–409. https://doi.org/10.1145/356789.356797

[7] Panagiotis Bouros, Tamraparni Dasu, Yaron Kanza, Matthias Renz, and Dimitris Sacharidis (Eds.). 2021. *LocalRec '21: Proceedings of the 5th ACM SIGSPATIAL International Workshop on Location-based Recommendations, Geosocial Networks and Geoadvertising, Virtual Event / Beijing, China, 2 November 2021.* ACM. https://doi.org/10.1145/3486183

[8] Panagiotis Bouros, Dimitris Sacharidis, and Nikos Bikakis. 2014. Regionally influential users in location-aware social networks. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, Dallas/Fort Worth, TX, USA, November 4-7, 2014.* ACM, 501–504. https://doi.org/10.1145/2666310.2666489

[9] Jing Cai and Chung Keung Poon. 2010. Path-hop: efficiently indexing large graphs for reachability queries. In *Proceedings of the 19th ACM Conference on Information and Knowledge Management, CIKM 2010, Toronto, Ontario, Canada, October 26-30, 2010.* ACM, 119–128. https://doi.org/10.1145/1871437.1871457

[10] Li Chen, Amarnath Gupta, and M. Erdem Kurul. 2005. Stack-based Algorithms for Pattern Matching on DAGs. In *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005.* ACM, 493–504. http://www.vldb.org/archives/website/2005/program/paper/wed/p493-chen.pdf

[11] Lu Chen, Chengfei Liu, Rui Zhou, Jianxin Li, Xiaochun Yang, and Bin Wang. 2018. Maximum Co-located Community Search in Large Scale Social Networks. *Proc. VLDB Endow.* 11, 10 (2018), 1233–1246. https://doi.org/10.14778/3231751.3231755

[12] Yangjun Chen and Yibin Chen. 2008. An Efficient Algorithm for Answering Graph Reachability Queries. In *Proceedings of the 24th International Conference on Data Engineering, ICDE 2008, April 7-12, 2008, Cancún, Mexico.* IEEE Computer Society, 893–902. https://doi.org/10.1109/ICDE.2008.4497498

[13] James Cheng, Silu Huang, Huanhuan Wu, and Ada Wai-Chee Fu. 2013. TF-Label: a topological-folding labeling scheme for reachability querying in a large graph. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013.* ACM, 193–204. https://doi.org/10.1145/2463676.2465286

[14] Jiefeng Cheng, Jeffrey Xu Yu, Xuemin Lin, Haixun Wang, and Philip S. Yu. 2006. Fast Computation of Reachability Labeling for Large Graphs. In *Advances in Database Technology - EDBT 2006, 10th International Conference on Extending Database Technology, Munich, Germany, March 26-31, 2006, Proceedings (Lecture Notes in Computer Science, Vol. 3896).* Springer, 961–979. https://doi.org/10.1007/11687238_56

[15] Jiefeng Cheng, Jeffrey Xu Yu, Xuemin Lin, Haixun Wang, and Philip S. Yu. 2008. Fast computing reachability labelings for large graphs with high compression rate. In *EDBT 2008, 11th International Conference on Extending Database Technology, Nantes, France, March 25-29, 2008, Proceedings (ACM International Conference Proceeding Series, Vol. 261).* ACM, 193–204. https://doi.org/10.1145/1353343.1353370

[16] George Christodoulou, Panagiotis Bouros, and Nikos Mamoulis. 2022. HINT: A Hierarchical Index for Intervals in Main Memory. In *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022.* ACM, 1257–1270. https://doi.org/10.1145/3514221.3517873

[17] George Christodoulou, Panagiotis Bouros, and Nikos Mamoulis. 2024. HINT: a hierarchical interval index for Allen relationships. *VLDB J.* 33, 1 (2024), 73–100. https://doi.org/10.1007/S00778-023-00798-W

[18] Vassilis Christophides, Dimitris Plexousakis, Michel Scholl, and Sotirios Tourtounis. 2003. On labeling schemes for the semantic web. In *Proceedings of the Twelfth International World Wide Web Conference, WWW 2003, Budapest, Hungary, May 20-24, 2003.* ACM, 544–555. https://doi.org/10.1145/775152.775230

[19] Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. 2002. Reachability and distance queries via 2-hop labels. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 6-8, 2002, San Francisco, CA, USA.* ACM/SIAM, 937–946. http://dl.acm.org/citation.cfm?id=545381.545503

[20] Angjela Davitkova, Evica Milchevski, and Sebastian Michel. 2020. The ML-Index: A Multidimensional, Learned Index for Point, Range, and Nearest-Neighbor Queries. In *Proceedings of the 23rd International Conference on Extending Database Technology, EDBT 2020, Copenhagen, Denmark, March 30 - April 02, 2020.* OpenProceedings.org, 407–410. https://doi.org/10.5441/002/EDBT.2020.44

[21] Paul F. Dietz. 1982. Maintaining Order in a Linked List. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA.* ACM, 122–127. https://doi.org/10.1145/800070.802184

[22] Paul F. Dietz and Daniel Dominic Sleator. 1987. Two Algorithms for Maintaining Order in a List. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA.* ACM, 365–372. https://doi.org/10.1145/28395.28434

[23] Yerach Doytsher, Ben Galon, and Yaron Kanza. 2010. Querying geo-social data by bridging spatial networks and social networks. In *Proceedings of the 2010 International Workshop on Location Based Social Networks, LBSN 2010, November 2, 2010, San Jose, CA, USA, Proceedings.* ACM, 39–46. https://doi.org/10.1145/1867699.1867707

[24] Yerach Doytsher, Ben Galon, and Yaron Kanza. 2012. Querying socio-spatial networks on the world-wide web. In *Proceedings of the 21st World Wide Web Conference, WWW 2012, Lyon, France, April 16-20, 2012 (Companion Volume).* ACM, 329–332. https://doi.org/10.1145/2187980.2188041

[25] Herbert Edelsbrunner. 1980. *Dynamic Rectangle Intersection Searching.* Technical Report 47. Institute for Information Processing, Technical University of Graz, Austria.

[26] Yixiang Fang, Reynold Cheng, Xiaodong Li, Siqiang Luo, and Jiafeng Hu. 2017. Effective Community Search over Large Spatial Graphs. *Proc. VLDB Endow.* 10, 6 (2017), 709–720. https://doi.org/10.14778/3055330.3055337

[27] Yixiang Fang, Zheng Wang, Reynold Cheng, Xiaodong Li, Siqiang Luo, Jiafeng Hu, and Xiaojun Chen. 2019. On Spatial-Aware Community Search. *IEEE Trans. Knowl. Data Eng.* 31, 4 (2019), 783–798. https://doi.org/10.1109/TKDE.2018.2845414

[28] Raphael A. Finkel and Jon Louis Bentley. 1974. Quad Trees: A Data Structure for Retrieval on Composite Keys. *Acta Informatica* 4 (1974), 1–9. https://doi.org/10.1007/BF00288933

[29] Samuel R. Friedman and Sevgi Aral. 2001. Social networks, risk-potential networks, health, and disease. *Journal of urban health: bulletin of the New York Academy of Medicine* 78, 3 (2001), 411—-418. https://doi.org/10.1093/jurban/78.3.411

[30] Antonin Guttman. 1984. R-Trees: A Dynamic Index Structure for Spatial Searching. In *SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, USA, June 18-21, 1984.* ACM Press, 47–57. https://doi.org/10.1145/602259.602266

[31] Alison L. Hill, David G. Rand, Martin A. Nowak, and Nicholas A. Christakis. 2010. Infectious Disease Modeling of Social Contagion in Networks. *PLoS Comput. Biol.* 6, 11 (2010). https://doi.org/10.1371/JOURNAL.PCBI.1000968

[32] H. V. Jagadish. 1990. A Compression Technique to Materialize Transitive Closure. *ACM Trans. Database Syst.* 15, 4 (1990), 558–598. https://doi.org/10.1145/99935.99944

[33] Ruoming Jin, Ning Ruan, Yang Xiang, and Haixun Wang. 2011. Path-tree: An efficient reachability indexing scheme for large directed graphs. *ACM Trans. Database Syst.* 36, 1 (2011), 7:1–7:44. https://doi.org/10.1145/1929934.1929941

[34] Ruoming Jin and Guan Wang. 2013. Simple, Fast, and Scalable Reachability Oracle. *Proc. VLDB Endow.* 6, 14 (2013), 1978–1989. https://doi.org/10.14778/2556549.2556578

[35] Ruoming Jin, Yang Xiang, Ning Ruan, and David Fuhry. 2009. 3-HOP: a high-compression indexing scheme for reachability query. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2009, Providence, Rhode Island, USA, June 29 - July 2, 2009.* ACM, 813–826. https://doi.org/10.1145/1559845.1559930

[36] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018.* ACM, 489–504. https://doi.org/10.1145/3183713.3196909

[37] Justin J. Levandoski, Mohamed Sarwat, Ahmed Eldawy, and Mohamed F. Mokbel. 2012. LARS: A Location-Aware Recommender System. In *IEEE 28th International Conference on Data Engineering (ICDE 2012), Washington, DC, USA (Arlington, Virginia), 1-5 April, 2012.* IEEE Computer Society, 450–461. https://doi.org/10.1109/ICDE.2012.54

[38] Justin J Levandoski, Mohamed Sarwat, Ahmed Eldawy, and Mohamed F Mokbel. 2012. Lars: A location-aware recommender system. In *2012 IEEE 28th international conference on data engineering.* IEEE, 450–461.

[39] Guoliang Li, Shuo Chen, Jianhua Feng, Kian-Lee Tan, and Wen-Syan Li. 2014. Efficient location-aware influence maximization. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014.* ACM, 87–98. https://doi.org/10.1145/2588555.2588561

[40] Pengfei Li, Hua Lu, Qian Zheng, Long Yang, and Gang Pan. 2020. LISA: A Learned Index Structure for Spatial Data. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020.* ACM, 2119–2133. https://doi.org/10.1145/3318464.3389703

[41] Yong Liu, Wei Wei, Aixin Sun, and Chunyan Miao. 2014. Exploiting Geographical Neighborhood Characteristics for Location Recommendation. In

Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, CIKM 2014, Shanghai, China, November 3-7, 2014. ACM, 739–748. https://doi.org/10.1145/2661829.2662002

[42] Kyriakos Mouratidis, Jing Li, Yu Tang, and Nikos Mamoulis. 2015. Joint Search by Social and Spatial Proximity. *IEEE Trans. Knowl. Data Eng.* 27, 3 (2015), 781–793. https://doi.org/10.1109/TKDE.2014.2339838

[43] Vikram Nathan, Jialin Ding, Mohammad Alizadeh, and Tim Kraska. 2020. Learning Multi-Dimensional Indexes. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*. ACM, 985–1000. https://doi.org/10.1145/3318464.3380579

[44] Matthaios Olma, Farhan Tauheed, Thomas Heinis, and Anastasia Ailamaki. 2017. BLOCK: Efficient Execution of Spatial Range Queries in Main-Memory. In *Proceedings of the 29th International Conference on Scientific and Statistical Database Management, Chicago, IL, USA, June 27-29, 2017*. ACM, 15:1–15:12. https://doi.org/10.1145/3085504.3085519

[45] Jianzhong Qi, Guanli Liu, Christian S. Jensen, and Lars Kulik. 2020. Effectively Learning Spatial Indices. *Proc. VLDB Endow.* 13, 11 (2020), 2341–2354. http://www.vldb.org/pvldb/vol13/p2341-qi.pdf

[46] Mohamed Sarwat, Justin J. Levandoski, Ahmed Eldawy, and Mohamed F. Mokbel. 2014. LARS*: An Efficient and Scalable Location-Aware Recommender System. *IEEE Trans. Knowl. Data Eng.* 26, 6 (2014), 1384–1399. https://doi.org/10.1109/TKDE.2013.29

[47] Mohamed Sarwat and Yuhan Sun. 2017. Answering Location-Aware Graph Reachability Queries on GeoSocial Data. In *33rd IEEE International Conference on Data Engineering, ICDE 2017, San Diego, CA, USA, April 19-22, 2017*. IEEE Computer Society, 207–210. https://doi.org/10.1109/ICDE.2017.76

[48] Ralf Schenkel, Anja Theobald, and Gerhard Weikum. 2004. HOPI: An Efficient Connection Index for Complex XML Document Collections. In *Advances in Database Technology - EDBT 2004, 9th International Conference on Extending Database Technology, Heraklion, Crete, Greece, March 14-18, 2004, Proceedings (Lecture Notes in Computer Science, Vol. 2992)*. Springer, 237–255. https://doi.org/10.1007/978-3-540-24741-8_15

[49] Stephan Seufert, Avishek Anand, Srikanta J. Bedathur, and Gerhard Weikum. 2013. FERRARI: Flexible and efficient reachability range assignment for graph indexing. In *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*. IEEE Computer Society, 1009–1020. https://doi.org/10.1109/ICDE.2013.6544893

[50] Jieming Shi, Nikos Mamoulis, Dingming Wu, and David W. Cheung. 2014. Density-based place clustering in geo-social networks. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*. ACM, 99–110. https://doi.org/10.1145/2588555.2610497

[51] Ammar Sohail, Arif Hidayat, Muhammad Aamir Cheema, and David Taniar. 2018. Location-Aware Group Preference Queries in Social-Networks. In *Databases Theory and Applications - 29th Australasian Database Conference, ADC 2018, Gold Coast, QLD, Australia, May 24-27, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 10837)*. Springer, 53–67. https://doi.org/10.1007/978-3-319-92013-9_5

[52] Jiao Su, Qing Zhu, Hao Wei, and Jeffrey Xu Yu. 2017. Reachability Querying: Can It Be Even Faster? *IEEE Trans. Knowl. Data Eng.* 29, 3 (2017), 683–697. https://doi.org/10.1109/TKDE.2016.2631160

[53] Yuhan Sun, Nitin Pasumarthy, and Mohamed Sarwat. 2017. On Evaluating Social Proximity-Aware Spatial Range Queries. In *18th IEEE International Conference on Mobile Data Management, MDM 2017, Daejeon, South Korea, May 29 - June 1, 2017*. IEEE Computer Society, 72–81. https://doi.org/10.1109/MDM.2017.20

[54] Yuhan Sun and Mohamed Sarwat. 2018. A generic database indexing framework for large-scale geographic knowledge graphs. In *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPATIAL 2018, Seattle, WA, USA, November 06-09, 2018*. ACM, 289–298. https://doi.org/10.1145/3274895.3274966

[55] Yuhan Sun and Mohamed Sarwat. 2021. Riso-Tree: An Efficient and Scalable Index for Spatial Entities in Graph Database Management Systems. *ACM Trans. Spatial Algorithms Syst.* 7, 3 (2021), 12:1–12:39. https://doi.org/10.1145/3450945

[56] X. Tang, Z. Chen, H. Zhang, X. Liu, Y. Shi, and A. Shahzadi. 2018. An optimized labeling scheme for reachability queries. *Computers, Materials & Continua* 55, 2 (January 2018), 267–283. http://www.techscience.com/cmc/v55n2/22897

[57] Robert Endre Tarjan. 1972. Depth-First Search and Linear Graph Algorithms. *SIAM J. Comput.* 1, 2 (1972), 146–160. https://doi.org/10.1137/0201010

[58] Silke Trißl and Ulf Leser. 2007. Fast and practical indexing and querying of very large graphs. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China, June 12-14, 2007*. ACM, 845–856. https://doi.org/10.1145/1247480.1247573

[59] Renê Rodrigues Veloso, Loïc Cerf, Wagner Meira Jr., and Mohammed J. Zaki. 2014. Reachability Queries in Very Large Graphs: A Fast Refined Online Search Approach. In *Proceedings of the 17th International Conference on Extending Database Technology, EDBT 2014, Athens, Greece, March 24-28, 2014*. OpenProceedings.org, 511–522. https://doi.org/10.5441/002/EDBT.2014.46

[60] Haixin Wang, Xiaoyi Fu, Jianliang Xu, and Hua Lu. 2019. Learned Index for Spatial Queries. In *20th IEEE International Conference on Mobile Data Management, MDM 2019, Hong Kong, SAR, China, June 10-13, 2019*. IEEE, 569–574. https://doi.org/10.1109/MDM.2019.00121

[61] Haixun Wang, Hao He, Jun Yang, Philip S. Yu, and Jeffrey Xu Yu. 2006. Dual Labeling: Answering Graph Reachability Queries in Constant Time. In *Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, 3-8 April 2006, Atlanta, GA, USA*. IEEE Computer Society, 75. https://doi.org/10.1109/ICDE.2006.53

[62] Hao Wei, Jeffrey Xu Yu, Can Lu, and Ruoming Jin. 2014. Reachability Querying: An Independent Permutation Labeling Approach. *Proc. VLDB Endow.* 7, 12 (2014), 1191–1202. https://doi.org/10.14778/2732977.2732992

[63] Hao Wei, Jeffrey Xu Yu, Can Lu, and Ruoming Jin. 2018. Reachability querying: an independent permutation labeling approach. *VLDB J.* 27, 1 (2018), 1–26. https://doi.org/10.1007/S00778-017-0468-3

[64] Yosuke Yano, Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. 2013. Fast and scalable reachability queries on graphs by pruned labeling with landmarks and paths. In *22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, CA, USA, October 27 - November 1, 2013*. ACM, 1601–1606. https://doi.org/10.1145/2505515.2505724

[65] Hilmi Yildirim, Vineet Chaoji, and Mohammed Javeed Zaki. 2010. GRAIL: Scalable Reachability Index for Large Graphs. *Proc. VLDB Endow.* 3, 1 (2010), 276–284. https://doi.org/10.14778/1920841.1920879

[66] Hilmi Yildirim, Vineet Chaoji, and Mohammed J. Zaki. 2012. GRAIL: a scalable index for reachability queries in very large graphs. *VLDB J.* 21, 4 (2012), 509–534. https://doi.org/10.1007/S00778-011-0256-4

[67] Junfeng Zhou, Jeffrey Xu Yu, Na Li, Hao Wei, Ziyang Chen, and Xian Tang. 2018. Accelerating reachability query processing based on DAG reduction. *VLDB J.* 27, 2 (2018), 271–296. https://doi.org/10.1007/S00778-018-0495-8

[68] Junfeng Zhou, Shijie Zhou, Jeffrey Xu Yu, Hao Wei, Ziyang Chen, and Xian Tang. 2017. DAG Reduction: Fast Answering Reachability Queries. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*. ACM, 375–390. https://doi.org/10.1145/3035918.3035927

[69] Shuang Zhou, Pingpeng Yuan, Ling Liu, and Hai Jin. 2018. MGTag: a Multi-Dimensional Graph Labeling Scheme for Fast Reachability Queries. In *34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16-19, 2018*. IEEE Computer Society, 1372–1375. https://doi.org/10.1109/ICDE.2018.00153

[70] Andy Diwen Zhu, Wenqing Lin, Sibo Wang, and Xiaokui Xiao. 2014. Reachability queries on large dynamic graphs: a total order approach. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*. ACM, 1323–1334. https://doi.org/10.1145/2588555.2612181