HKU CS Tech Report TR-2016-03

Fast Matching of Detour Routes and Service Areas

Siqiang Luo[†] Reynold Cheng[†] Xiaokui Xiao[‡] Ben Kao[†] Shuigeng Zhou[§] Jiafeng Hu[†]

[†]The University of Hong Kong, Hong Kong [‡]Nanyang Technological University, Singapore [§]Fudan University, China

***Abstract*—Novel road-network services, such as ride sharing, spatial crowdsourcing and keyword-aware routing often need to find out whether there exists a route that can go through (or *match*) a service area. A ride-sharing system, for instance, may need to find for a moving taxi a route that passes through a region in which a passenger is located. A fundamental question is: given two locations s and t, and an area $\mathcal{A}$ on the road network, is there a "reasonably short" route from s to t that also crosses $\mathcal{A}$? This *Route and Area Matching* (ROAM) query is important as 1) its efficient evaluation allows the application involved to recommend appropriate services to objects in an online manner; 2) its algorithm can be incorporated into some sophisticated systems to optimize their performance. Despite its importance, the ROAM query has not been well studied especially when "service area" instead of "service point" is concerned. We examine efficient ROAM query algorithms. Particularly, we develop solutions for testing the existence of a ρ-*route*, which is an s-t path that passes through $\mathcal{A}$, with a length of at most $(1+\rho)$ times the shortest distance between s and t. We propose index-free and index-based algorithms. Our index structures, with a size linear to the number of road network nodes, enable significant query improvement over baseline approaches. We have performed extensive experiments on several large road network datasets to validate the efficiency and scalability of our approaches. In addition, we incorporate our efficient algorithm to an existing keyword-aware routing system and one dynamic ride-sharing system. Our experiments show that an efficient ROAM query algorithm helps improve the performance of these systems.**

I. Introduction

Location-based services, such as ride-sharing, spatial crowdsourcing and keyword-aware routing, have attracted tremendous interest in recent years. A taxi ride-sharing application (e.g., RYDE [1], uberPool[1]) enables passengers to share a taxi ride; a spatial crowdsourcing system [2]–[4] invites mobile Internet users (called workers) to visit a given place to conduct a task (e.g., shooting photos of attractions [5]); a keyword-aware routing system [6], [7] allows a mobile user to describe objects of interest (e.g., Italian restaurant) by keywords, and recommends to her interesting ones. These applications, which take users' locations into account, provide convenience and business opportunities.

The above applications share a common characteristic: they consider an *object* m (e.g., a taxi or a mobile user) moving from an origin s to a destination t, and recommend to m a service (e.g., a restaurant) or service request (e.g., a shared ride request). Then, m traverses to the service area, conducts some activities (e.g., picking up a passenger), and continues her journey to t. A service is recommended to m, only if m's detour from s to the service area (and then go to t) does not seriously affect her original travel plan from s to t. Fig. 1(a) shows an example in ride-sharing: a taxi, located at s, is carrying her current passenger to t (solid line). The ride-sharing system receives requests of two passengers, who also want to go to t, at locations x and y. The enclosed circles indicate the areas the passengers are willing to walk to be picked up. The system accepts the request from x as its detour cost is small enough. The taxi then takes a detour (dashed line) to pick up x, after which she continues to go to t. Fig. 1(b) illustrates a spatial crowdsourcing system for collecting recent photos of attractions. Each task is an attraction enclosed with a suitable photo-shooting area [5]. A worker who is walking to home (t) from her office (s), is recommended the tasks that are close to her path (e.g., task 1). As another example, in Fig. 1(c), a user who has just viewed a movie (in s) would like to have a meal before going home (in t). She wants to go to certain food mall and picks a restaurant there. She can input "*food mall*" in a keyword-aware routing system [6], and the system recommends to her a food mall which costs her a little extra traveling time.

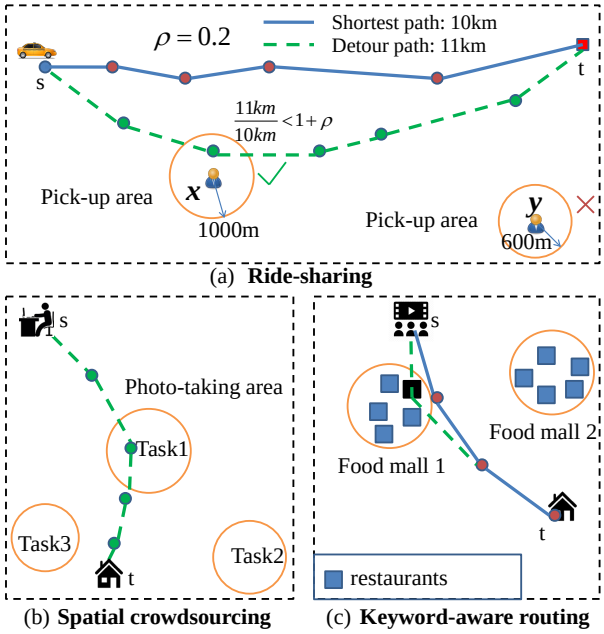


Fig. 1: Illustrating ROAM queries.

To recommend appropriate services, a system should consider the amount of effort needed for an object to approach the suggested service area. Suppose that a service (request) e is initiated in a region $\mathcal{A}$ (e.g., a pick-up area). Then, an object m on a route from s to t would be interested in handling e, only if its detour from s to $\mathcal{A}$ does not seriously affect her original

[1]https://www.uber.com/

travel plan from s to t. Therefore, before recommending e to m, the system should check whether there is a "reasonably short" path from s to t that passes through $\mathcal{A}$. For example, the length of the path can be required to be at most (1+ ρ) times the shortest distance from s to t, with $\rho \geq 0$ a parameter decided by the system or the object involved. We refer to the decision problem of checking whether such a path (called ρ-route) exists as the *ROute and Area Matching (ROAM)* query. In Fig. 1(a), for instance, $\rho = 0.2$. The dashed line is a 0.2-route between s and t, because its length is less than $(1 + \rho = 1.2)$ times the shortest distance between s and t. Moreover, this 0.2-route passes through one pick-up point within the pick-up area around x, and so the ROAM query on (s, t, x) returns "true" to indicate that there is a "match" of x and the given (s, t) pair. However, the pick-up area around y is far from the shortest path between s and t, and so the ROAM query for y returns "false", revealing that the length of a detour taking the passenger in y is more than 1.2 times that of the shortest distance between s and t.

Drawbacks of Existing Solutions. A simple way of answering a ROAM query is to examine all paths from s to t, and check if there is a ρ-route that intersects $\mathcal{A}$. This solution is prohibitively expensive, due to the enormous number of paths that have to be examined. Two classes of solutions can also be adapted. The first is based on Dijkstra's algorithm [8]. The idea is to perform Dijkstra's algorithm with s and t as source nodes respectively. We examine whether the two searches meet at a node inside area $\mathcal{A}$ and the meeting node is on a ρ-route from s to t (See details in Sec. IV-A). This solution is inefficient when handling queries on large road networks. The other approach uses well-studied point-to-point shortest distance algorithms [8]–[14]. Particularly, we can sample a portion of nodes in $\mathcal{A}$, then we compute the shortest distances between each sample to s and t respectively to examine whether the sample is on a ρ-route from s to t. Once such a sample is found, we can conclude that a ρ-route from s to t passing through $\mathcal{A}$ exists. This approach is inefficient when many samples are involved and we cannot guarantee its correctness as it depends on the samples. A few recent works in spatial crowdsourcing [3], [4] and ride-sharing [15]–[19] addressed related but different issues. Those techniques proposed suffer from at least one of the following deficiencies. First, they gauge the distance between object m and service request e by their Euclidean distance. As pointed out in [11], Euclidean distance is often inaccurate, since an object's movement is constrained by a road network. Second, previous works assume that a service is handled at a precise location; in practice, a service can be served in a region. In ride-sharing, a passenger may walk to a pick-up location nearby; a spatial crowdsourcing task, such as taking photos for a monument, can be done within some distance from it. Last, existing works (e.g. [6], [19]) often overlook the possibility of optimizing the cost of validating a detour constraint, as they tend to involve a more sophisticated queries (e.g., TSP-like queries). Therefore, existing solutions either cannot efficiently support ROAM query or has a orthogonal objective.

Our Contributions. This paper presents a comprehensive study on ROAM queries, which is a fundamental query that supports many road-network applications (e.g., ride-sharing, spatial crowdsourcing and keyword-aware routing). We first develop an index-free approach, known as the *bi-directional search with temporary stop*, which improves query evaluation by considering the relationship between (s, t) pairs and the service area. The approach yields significant improvement over the basic solution. To further improve the efficiency, we then design a data structure called *Sketch*, and based on it a *Sketch Framework*. The Sketch Framework is a general search framework that can incorporate any ROAM query algorithm. We also analyze our approach, and show that the size of the *Sketch* is linear to the number of network nodes.

We have conducted experiments on seven large real datasets, the largest of which contains 0.5 billion edges. On a dataset about New York's taxi trajectories, *Sketch* is up to 30 times faster than the index-free approach. Our solution is scalable to large datasets. In terms of effectiveness, the ROAM query yields up to 10 times more matches between services and objects, compared with traditional methods, which assume that the service request is initiated at a point.

We incorporate our efficient ROAM query algorithms to an existing keyword-aware routing system and a dynamic ride-sharing system. We show that ROAM queries help improve their performance considerably. For example, by simply employing our ROAM query algorithm as a preprocessing to a keyword-aware routing algorithm, the algorithm runs up to seven times faster.

Organization. The rest of the paper is organized as follows. We discuss related works in Sec. II. In section III, we introduce the detailed problem settings. In section IV, we propose two index free methods to handle the ROAM query. We propose our main algorithm in section V and VI. Section VII shows our experimental evaluation. We conclude the paper in Sec. IX.

II. Related Work

We now discuss related works including shortest path queries, ride-sharing and keyword-aware routing systems.

Shortest Path Queries. The shortest path query [8]–[14], [20], [21] is related to the ROAM query. The shortest path algorithms can be generally divided into two types. The first type extends Dijkstra algorithm [8] by considering bi-directional search (e.g., [9], [10]). These algorithms inspire the design of our first two ROAM query algorithms. However, extending these solutions to handle ROAM queries is not trivial, due to the presence of ρ and the service area. The second type of algorithms [12]–[14] is a hierarchical approach. Their common idea is to select some important nodes among the network, and let them be the routing nodes to speed up calculation. We experimentally compare our solution to the shortest distance based solutions. While we borrow the basic concept of "cover edge" from [12] to define the importance of a node, our algorithms are significantly different from [12]. [12] pre-computes pairwise distances among the selected important nodes, which entail quadratic storage. In contrast, we construct a light-weight graph that consumes linear storage on the selected nodes, known as *Sketch*. We present many new ideas which are not in [12]. First, we propose the concepts of sketch edges (Sec. V-C) and bridge edges (Sec. VI-A), which are the most important parts that constitute a *S*ketch (Sec. V). Second, we devise efficient algorithms to construct *S*ketch edges (Sec. V-C) and bridge edges (Sec. VI-A). In addition, we analyze the properties of a sketch, including space complexities

and *distance invariance* (Sec. V-C and V-D). Lastly, based on the *S*ketch, we devise smart algorithms to conduct efficient ROAM queries (Sec. VI).

Spatial Crowdsourcing. Existing spatial crowdsourcing studies [2]–[4], [22] assume an Euclidean space, such that the distance between two locations can be calculated easily. Also, these spatial crowdsourcing systems rarely consider the situations where the volunteer workers have a predefined traveling destination. Therefore, existing studies on spatial crowdsourcing problems are significantly different from the ROAM query. We also note that existing studies for spatial crowdsourcing rarely consider a task location as a service area. They often assume POIs are points. However, as we pointed out, area-based queries are of interest in some applications. Moreover, employing this service area constraints makes the query much more challenging to handle. Directly employing existing point-based methods can be exhaustive when the given area contains a large number of nodes. Therefore, the study of the ROAM query complements to existing point-based queries.

Ride-Sharing.

We compare the methods employed by existing ride-sharing systems [15], [17], [19] with our proposed algorithms for the ROAM query. [19] employs a `P2P` method (detailed explanation can be found in Sec. VII) that can be adapted to address the ROAM query. We experimentally compare this method and show the superiority of our algorithms. The algorithm introduced in [17] does not help much in addressing the ROAM query, as the main technique in [17] relies on the assumption that the Euclidean distance between two nodes is not larger than the corresponding shortest network distance. However, this assumption does not hold when the edge weight represents the traveling time. Moreover, their optimization objective, which is formalized as a skyline query, is significantly different from our ROAM query. As a result, their solution is hard to be adapted for the ROAM query. [15] uses an approximate distance method to estimate the network distance between two nodes. They partition the road network by a grid and use the distance between two centers of grid cells to estimate the distance between two nodes respectively inside the two cells. Since it is a distance approximation method, it cannot address the ROAM query accurately.

Also, some existing ride-sharing systems can benefit from an efficient algorithm for the ROAM query which acts as a *fundamental* operator. In particular, [19] formulates the ride-sharing problem as a form of Traveling Salesman Problem (TSP) (as noted by [18]), while satisfying a detour constraint similar to the ROAM query. As such, a ROAM query can act as a necessary filtering condition when plugged into their solutions and help optimize the performance (See Fig. 7 (h.1) in experiments).

Keyword-Aware Routing. A keyword-aware routing system [6], [7] finds a cost-efficient route on the road network which goes across the POIs collectively marked by the searched keywords. Their algorithm focus is different from the ROAM query, as they focus on how to extract a collection of POIs that are collectively associated with search keywords. Interestingly, our ROAM query can be incorporated to their systems to enhance the performance (See Fig. 7 (h.2) in experiments).

Notation	Description
$\mathcal{G}$	the input road network
$w(u, v)$	the weight of edge (u, v)
$dist(s, t)$	the shortest network distance between node s and node t
$eu\_dist(s, t)$	the Euclidean distance between nodes s and t
$circ(o, r)$	the circular area whose center is o and radius is r
$ball(s, \tau)$	the set of nodes whose network distances from s is at most τ
gd	the grid distance
$lower\_dist(s, t)$	a network distance lower bound between nodes s and t
C_3 (resp. C_5)	a 3×3 (resp. 5×5) sub-grid

TABLE I: Frequently used notations

III. Preliminaries

In this section, we give a formal definition of the ROAM query (Section III-A). We then review Dijkstra's Algorithm in Section III-B, which forms the basis of our discussions.

A. The ROAM Query

A road network is a graph $\mathcal{G}(V, E)$, where each node $u \in V$ represents a road junction and each edge $e \in E$ is a road segment. Each node u is associated with a geographical location (u_x, u_y), while each edge e is associated with a weight $w(e)$, which indicates the cost of traversing e (e.g., the physical length of e or the time required to traverse e). If an edge e has two end nodes u and v, then we use (u, v) to represent e. For any path of $\mathcal{G}$, we define its *length* as the sum of weights of the edges in the path. We let $n = |V|$. The shortest path between two nodes u and v is the path from u to v with the smallest length. We use $(u_1, \ldots, u_h)$ to represent the path composed of nodes $u_1, \ldots, u_h$. Accordingly, the *shortest distance* between u and v, $dist(u, v)$, is the length of the shortest path from node u to node v. We denote the Euclidean distance between u and v as $eu\_dist(u, v)$. Unless otherwise specified, we refer to the shortest distance between two nodes as their distance. We say that a path between s and t is a ρ-route ($\rho \geq 0$), if its length is at most $(1 + \rho) \cdot dist(s, t)$. For ease of presentation, we assume that $\mathcal{G}$ is undirected, that is, $dist(u, v) = dist(v, u)$ for any nodes u and v. We discuss how to extend our solution to directed road networks in Sec. VI-C. The frequently used notations are summarized in Tab I. Let us now define the ROAM query.

definition 1 (ROAM Queries): Given a road network graph $\mathcal{G}$, two nodes s and t, and a circular region $circ(o, r)$ of $\mathcal{G}$ (where o is a node of $\mathcal{G}$ and r is the radius), a ROAM query returns *"true"* if there exists a ρ-route from s to t that intersects the $circ(o, r)$, or *"false"* otherwise.

Here, we assume that the service area has a circular shape. The reason is twofold. First, it makes our solutions easier to present. Second, a circular area is natural in some applications. In ride-sharing, for instance, a user, located at o (e.g., inside a shopping mall), may not want to walk more than a distance r to walk to the taxi. Hence, her service area can be modeled as $circ(o, r)$.

To handle a non-circular service area $\mathcal{A}$, a simple way is to find the largest circle enclosed by it, and use this circle as the approximate service area instead. Then, if a ROAM query returns "true" for this circle, the answer is also true for $\mathcal{A}$. In Sec. VIII, we also discuss a more accurate solution for non-circular area based on the Sketch framework.

We also give another example about how a service area can be defined. Consider a passenger, requesting for a ride, does

not want the ride-sharing system to know about her precise location (e.g., she is in a specialist hospital). To alleviate location privacy concern, *cloaking techniques* can be used, where the user sends to the system a larger area that covers her exact location. This area can be considered as a service area for this passenger.

B. Dijkstra's Algorithm

We briefly review Dijkstra's algorithm [8], which is fundamental to our techniques. Dijkstra's algorithm computes shortest distances from a source node s to all the other nodes. During the search, the nodes are examined in an ascending order of their distances from the source node s. Each node is associated with a state and a distance that indicates the known shortest distance from the node s. The state of a node is one of *unseen*, *labeled* and *scanned*. If a node is *unseen*, it means the distance associated with the node is ∞. In contrast, if a node is *scanned*, it indicates that the distance associated with the node is the exact distance between s and the node. The other nodes with state *labeled* are associated with a finite upper-bound distance of its true distance from s. For ease of presentation, in the following, we say a node is *scanned* (resp. *labeled*, *unseen*), if the state of the node is *scanned* (resp. *labeled*, *unseen*). We call the search based on Dijkstra's algorithm a *Dijkstra search*. We also define the search space of Dijkstra search as *Dijkstra ball*, denoted by $ball(s, \tau)$, to describe the set of nodes whose shortest distances to s are at most τ.

IV. Index-Free Approaches

In this section, we introduce index-free algorithms for the ROAM query. A basic solution is adapted from Dijkstra search (Sec. IV-A). However, such approach is slow. A more clever algorithm makes use of the distance relationship among s, t and o. Based on this idea, we propose `BIS` algorithm (Sec. IV-B). `BIS` considerably outperforms `Basic`.

A. A Basic Approach

We briefly introduce a straightforward solution, called `BASIC`. Our major observation is that every node on a ρ-route has a shortest distance at most $(1+\rho)dist(s,t)$ to s as well as t. Hence, all the ρ-routes are within

$$I = ball\Big(s, (1+\rho)dist(s,t)\Big) \bigcap ball\Big(t, (1+\rho)dist(s,t)\Big).$$

We also observe that, whether there exists a satisfactory ρ-route is equivalent to examining whether there is a node u in the given $circ(o,r)$ such that its sum of distances from s and t is less than $(1+\rho)dist(s,t)$.

Claim 1 (Positive Condition Rule): A ROAM query returns $true$, if and only if there is a node $u \in circ(o,r)$ such that $dist(s,u) + dist(u,t) \leq (1+\rho)dist(s,t)$.

Based on the *Positive Condition Rule*, a straightforward solution of the ROAM query is composed of the following two steps:

1) *Search from s:* We calculate $ball\big(s, (1+\rho)dist(s,t)\big)$; we return *false* if $circ(o,r)$ does not intersect the ball.

2) *Checking from t:* Conduct Dijkstra search from node t, whenever visit a node u satisfying the positive condition rule, we return *true*.

`Basic` contains a large number of fruitless searches. We discuss this in detail in the following sections.

B. First-Cut: Bi-Search with Temporary Stop

One unsatisfactory effect of `Basic` is that the $ball(s, (1+\rho)\ dist(s,t))$ is always fully computed. In quite a few cases, this full search is unnecessary. We propose bi-directional search for ROAM queries, which we refer to as BIS (BI-directional ROAM Search). We exploit the location relationship among s, t and o so as to optimize the search. Our strategy is considerably different from the traditional bi-directional search for the shortest path computation [9], as the traditional bidirectional search overlooks the position of o. Let us first consider the motivating case in Fig. 2 (a) where the given circle is near to the bisector of the line connecting s and t. In this case, an iterative search which takes turns to expand the Dijkstra balls from s and t leads to an early stop of the search (i.e., two balls meeting around the bisector), even when both $ball(s, (1+\rho)dist(s,t))$ and $ball(t, (1+\rho)dist(s,t))$ have not been fully computed.

The second example shown in Fig. 2 (b) motivates us to consider the position of the circle. When the given circle is near to node s, the iterative search still incurs a large search space. The reason is, the search will not stop until the two search balls meet inside the given circle. Therefore, we incorporate into the bidirectional search a more reasonable strategy by considering the position of circle center o. We begin with an iterative expansion from s and t. We will temporarily stop the expansion from one side when the ball touches the central node of the circle and wait for the search from the other side. We call it a *temporary stop* strategy. A simplified procedure is shown in Fig. 2 (c.1) and (c.2). At first, a Dijkstra ball centering at s temporarily stops its expansion when its boundary touches o. Later, the expansion from the other side touches the circle area and finds a node satisfying the positive condition rule (Claim 1). To further improve the efficiency, we next show the following early stop rule, i.e., Lemma 1. With Lemma 1, `BIS` can have a chance to stop when the circle central node o has been scanned by both Dijkstra balls. We summarize the procedures of our bi-directional ROAM search in Alg. 1.

Lemma 1: Denote the nearest node (measured in shortest distance) in $circ(o,r)$ to s (resp. t) is node u_s (resp. u_t). When the circle center o is scanned in both searches, the search can stop if $\min\big\{dist(t,o)+dist(s,u_s), dist(s,o)+dist(t,u_t)\big\} > (1+\rho)dist(s,t)$.

Proof: All the omitted proofs can be found in Appendix. ■

Remarks. There are two cases where we need to resume the temporarily stopped search: 1) the condition of the above Lemma 1 is not fulfilled; 2) when one side of the search is stopped by the radius bound (i.e., $(1+\rho)dist(s,t)$) while the other side is stopped due to *temporary stop* strategy. We call them *resume checking*.

V. The Sketch Framework

In this section, we introduce a general framework to facilitate the index-based ROAM query, based on a novel structure called *Sketch*.

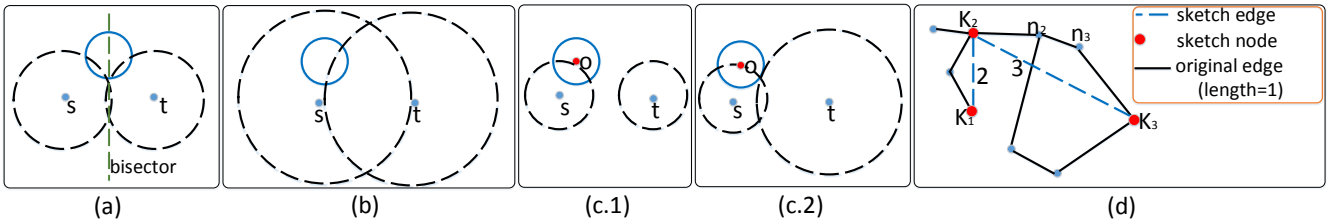


Fig. 2: (a) The cases where bidirectional search is favorite. The search space (black dashed) is small. (b) The cases where traditional bidirectional search is not favorite. The search space (black dashed) is large. (c.1, c.2) Procedures of bidirectional search with temporary stop. (d) Illustration of the Sketch.

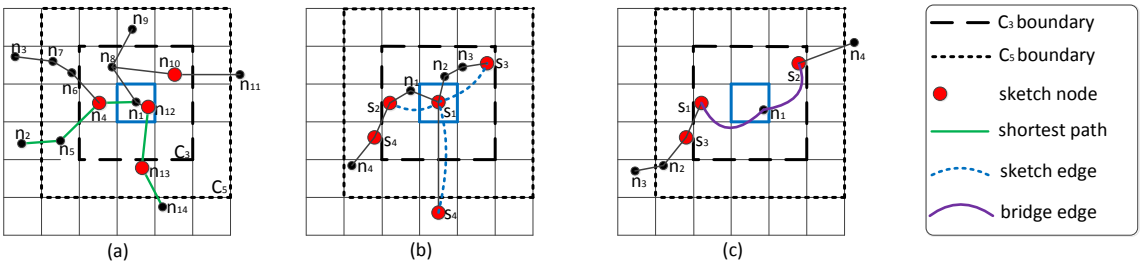


Fig. 3: (a) Examples of *Sketch*, regional graph, *Sketch* node selection. (b) *Sketch* edge examples. (c) Bridge edge examples.

Algorithm 1: BIS($\mathcal{G}, s, t, o, r, \rho$)

```
best_dist ← ∞ ;   /* Current best distance */
temp_stop_s ← false; temp_stop_t ← false;
/* Indicator of temporary stop */
radi_stop_s ← false; radi_stop_t ← false;
/* Indicator of radius bound. */
while expandable from s or t do
    if temp_stop_s = false and radi_stop_s = false then
        u ← expanded node;
        best_dist ← min{best_dist, dist(s,u)+dist(t,u)};
        if u = o then
            temp_stop_s ← true;
        if dist(s,u) > (1+ρ)best_dist then
            radi_stop_s ← true;
        if u ∈ circ(o,r) and
          dist(s,u) + dist(t,u) ≤ (1+ρ)best_dist then
            return true;
        if the condition of Lemma 1 holds then
            return false;
        if u is the central node o then
            temporarily stop the expansion of current side;
        Apply resume checking;
return false;
```

A. Basic Idea of the Sketch

A *Sketch* abstracts the input road network. The high-level idea of employing a *Sketch* resembles our behavior in looking up an online road network: we often first browse a coarser level of the road network to locate the most interesting area, then dive into the specific locations by adjusting the resolution of the network. This daily life behavior inspires us to devise a *road network sketch*, which contains much less nodes and edges. Our aim is to answer a large part of the ROAM query on the *Sketch*. We achieve this target by a careful design of the *Sketch*. In our design, one key feature of the *Sketch* is the *distance invariance property*: the shortest distance between any two nodes on the *Sketch* is the same as that on the original network. We further show that, if there is a ρ-route intersecting the given area, then there must be a ρ-route intersecting a relevant area on the *Sketch*. The result is interesting as it can facilitate a good pruning on the sketch. We also point out the result is *independent* of any specific ROAM query algorithm run on the (Sketch) network. In other words, *any* ROAM query approach $\mathcal{M}$ can be incorporated into the `Sketch Framework` (Alg. 2).

Algorithm 2: Sketch Framework

```
if M(s,t,o,σ+r,ρ) on the Sketch returns false /* σ is
    a constant. */
then
    returns false
return false; return M(s,t,o,r,ρ) on G;
```

In what follows, we will present the *Sketch* construction in Sec. V-B and V-C, and analyze the complexities in Sec. V-D.

B. Grid and Sketch Node Selection

We divide the *Sketch* construction into *Sketch* nodes selection and *Sketch* edges creation. We first describe *Sketch* node selection in this section. The construction is based on an $M \times M$ $(M > 5)$ grid imposed on the road network, with M a parameter. Each cell may contain some network nodes. If a node is in the cell of i-th row and j-th column, we say its grid location is (i, j). Next, we define the grid distance between two nodes u (with grid location (i_1, j_1)) and v (with grid location (i_2, j_2)) as $\max\{|i_1 - i_2|, |j_1 - j_2|\}$, and denote it by $gd(u, v)$. For example, the nodes n_3 and n_4 in Fig. 3 (a) have a grid location $(2, 1)$ (row 2, column 1) and $(3, 3)$ (row 3, column 3) respectively, and their grid distance is $\max\{|2-3|, |1-3|\} = 2$.

Next, we describe our *Sketch* node selection approach. The selection is based on a concept called *shortest path cover* for two sets of nodes $(\mathcal{S}_1, \mathcal{S}_2)$. A shortest path cover for $(\mathcal{S}_1, \mathcal{S}_2)$

is a node set H such that the shortest path between any node $u_1 \in \mathcal{S}_1$ and any node $u_2 \in \mathcal{S}_2$ must pass through at least one node of H. We use *shortest path cover* to describe the *Sketch* node selection, as follows.

Sketch Node Selection. We conduct local Dijkstra searches to select *Sketch* nodes. Specifically, for each cell C, we pick the 5×5 sub-grid C_5 whose central cell is C, and the 3×3 sub-grid C_3 whose central cell is C. Let the nodes inside C as $\mathcal{S}_1$ and the nodes outside of C_5 as $\mathcal{S}_2$. In the example in Fig. 3 (a), C is the 1×1 square in blue solid lines. C_3 is the 3×3 square in dashed lines and C_5 is the 5×5 square in dotted lines. $S_1 = \{n_1\}$ and $S_2 = \{n_2, n_3, n_{11}, n_{14}\}$. Intuitively, a shortest path cover for $(\mathcal{S}_1, \mathcal{S}_2)$ contains structurally important nodes for the region in C_5. We thus compute a shortest path cover with respect to $(\mathcal{S}_1, \mathcal{S}_2)$ and let the nodes in it be *Sketch* nodes. To get a shortest path cover, the idea is, all shortest paths connecting one node in $\mathcal{S}_1$ and one node in $\mathcal{S}_2$ must go across the boundary of the 3×3 sub-grid C_3. We locate the edges, on such a shortest path, that intersect the C_3 boundary while one end node is inside C_3. We call them *cover edges*. For example, the shortest path (n_1, n_4, n_5, n_2) is such a shortest path because n_1 is inside cell C and n_2 is outside of C_5. The cover edge on this path is (n_4, n_5) because it intersects the C_3 boundary (in dashed lines) and n_4 is in C_3. Then we select the *Sketch* nodes based on two conditions: (i) For each cover edge, one of its end node is selected as a *Sketch* node if it is inside C_3; for example, node n_4 of cover edge (n_4, n_5) is selected as a *Sketch* node since n_4 locates inside C_3; (ii) For each cover edge, if one end node is inside the central cell C, then both end nodes are selected as *Sketch* nodes; for example, (n_{12}, n_{13}) is such a cover edge and both n_{12} and n_{13} are selected as *Sketch* nodes.

The above *Sketch* node selection guarantees that, any shortest path between two faraway nodes must pass through one *Sketch* node. We make the following claim.

Lemma 2: The shortest path from node u to node v must pass through a *Sketch* node other than u and v, if two conditions hold: 1) $gd(u, v) \geq 3$; and 2) the path contains at least two edges.

Proof: Denote the cell containing node u as C and the 5×5 sub-grid (resp. 3×3 sub-grid), whose central cell is C, as C_5 (resp. C_3). Then node v must be outside of C_5 since $gd(u, v) \geq 3$. Hence, the shortest path from u to v must contain a cover edge (p, q) and p is in C_3. Then by our *Sketch* node selection strategy, node p is selected as a *Sketch* node. Now, we consider two cases. Case (i): $p \neq u$. In this case, the lemma holds as node p is a *Sketch* node other than u and v. Case (ii): $p = u$. In this case, by our *Sketch* node selection strategy, when one end node p (i.e., u) is inside cell C, the other end node will also be selected. Then node q will be a *Sketch* as well node in this case. We also note that according to the lemma conditions, the shortest path from u to v contains at least two edges, then we have $q \neq v$ (otherwise $(u, v) = (p, q)$, violating the lemma condition). Hence the lemma holds in the second case as node q is a *Sketch* node other than u and v. ■

Computation. We devise efficient algorithms for extracting *Sketch* nodes. Let us focus on a specific 5×5 sub-grid. The key is to locate the cover edges. A straightforward method is to conduct a Dijkstra search from every node inside the central cell. Then, those shortest paths which can reach outside of C_5 are extracted. Finally, the extracted shortest paths are backtracked to locate the cover edges by examining whether it intersects the boundary of C_3 and has a node in C_3. This method, however, is inefficient. It requires as many rounds of Dijkstra searches as the number of nodes in the central cell. A smarter approach is to only conduct Dijkstra searches from the nodes which have an outgoing edge intersecting the boundary of the central cell (n_1 in Fig. 3 (a) is such a node since its outgoing edge (n_1, n_4) intersects the boundary of the cell C). The rationale behind is that any shortest path, which has one end point in the central cell and the other outside of C_5, will intersect the boundary of the central cell. Hence, each such shortest path must contain a sub-path that starts with an edge intersecting the boundary of the central cell C, and ends at an edge intersecting the boundary of the corresponding C_5. For example, in Fig. 3 (a), (n_1, n_4, n_5, n_2) is a such sub-path of the shortest path $(n_{12}, n_1, n_4, n_5, n_2)$. Hence, we only need to enumerate all such sub-paths to locate cover edges, as illustrated by Alg. 3.

Algorithm 3: $SketchNodeSelect(\mathcal{G})$

```
1  V' = ∅
2  for each 5 × 5 sub-grid C_5 do
3      Extract the regional graph, RG, which is the subgraph
         composed of the edges having at least one end node
         inside the spatial region confined by the C_5 sub-grid.
4      for node p in central cell and p has an outgoing edge
         intersecting the boundary of the central cell do
5          Conduct Dijkstra search from p in RG;
6          for leaf node q outside of C_5 do
7              Backtrack along the shortest path from node q to
                 node p and locate the cover edges;
8              for each cover edge (u, v) do
9                  if u (resp. v) is in C_3 then
10                     V' = V' ∪ {u} (resp. V' = V' ∪ {v})
11                 if u (resp. v) is in the central cell then
12                     V' = V' ∪ {v} (resp. V' = V' ∪ {u})
13     return V'
```

Example Procedure. Call back to Fig. 3 (a). The figure shows a regional graph (corresponding to RG in Alg. 3). n_1 is one node which has an outgoing edge intersecting the boundary of cell C. The shortest paths from n_1 are $(n_1, \mathbf{n_4}, n_5, n_2)$, $(n_1, \mathbf{n_4}, n_6, n_7, n_3)$, $(n_1, n_8, \mathbf{n_{10}}, n_{11})$ and $(n_1, \mathbf{n_{12}}, \mathbf{n_{13}}, n_{14})$. The bold nodes n_4, n_{10} and n_{12} are selected as *Sketch* nodes since they have an outgoing edge intersecting the boundary of C_3 and they are inside C_3. Note that, n_{13} is also selected as a *Sketch* node because n_{12} is inside the central cell C and (n_{12}, n_{13}) is a cover edge.

C. Sketch Edge Creation

We introduce how we create the *Sketch* edges. The challenges lie in how to maintain the invariance of the shortest distance between two *Sketch* nodes. Namely, we want the shortest distance between any two *Sketch* nodes u^* and v^* on the *Sketch* the same as their shortest distance on the original network. To this end, for each *Sketch* node u^*, we conduct a Dijkstra search with u^* as the source on the original network. When it scans another *Sketch* node v^*, we create a *Sketch* edge

(u^*, v^*), if the shortest path from u^* to v^* does not contain any *Sketch* node other than u^* and v^*.

The Dijkstra search from u^* only needs to be conducted regionally due to Lemma 2. By Lemma 2, if a shortest path from the *Sketch* node u^* reaches outside of its C_5 sub-grid (i.e., the 5×5 sub-grid with the cell containing u^* as the central cell), the path already passes through another *Sketch* node. This indicates we can safely stop further expansion of the path for creating a *Sketch* edge. Since Lemma 2 only applies when the shortest path from u^* to v^* contains at least two edges, a further discussion is needed if the shortest path itself is an edge, namely (u^*, v^*), of the original network. In this special case, we also need to create a *Sketch* edge (u^*, v^*). This can be conveniently handled regionally as v^* is a neighbor node of u^* in the original network. A pseudo-code is shown in Alg. 4.

Algorithm 4: $SketchEdgeCreate(\mathcal{G})$

```
1 E' = ∅
2 for each 5 × 5 sub-grid C_5 do
3     Extract the regional graph, RG, which is the subgraph
        composed of the edges having at least one end node
        inside the spatial region confined by the C_5 sub-grid.;
4     for each Sketch node u* in the central cell do
5         Conduct Dijkstra search from p in RG;
6         for each leaf node v do
7             Backtrack from node v to node u* along the
                shortest path, and locate the u*'s nearest Sketch
                node v* in the path ;
8             E' = E' ∪ {(u*, v*)};
9 return E'
```

Example Procedure. In Fig. 3 (b), S_1 is a *Sketch* node inside the central cell. Then, we conduct a Dijkstra search from S_1 (Alg. 4 line 5). For each leaf node (n_4, S_3, S_4), backtrack the shortest path to the source node S_1. The backtracked paths are $\{n_4, S_4, \mathbf{S_2}, n_1, S_1\}$, $\{\mathbf{S_3}, n_3, n_2, S_1\}$ and $(\mathbf{S_4}, S_1)$. The bold ones are *Sketch* nodes nearest to S_1 along the paths. Hence, we create three *Sketch* edges from S_1 to S_1's nearest *Sketch* nodes, namely S_2, S_3 and S_4 (Alg. 4 line 8).

Distance Invariance. The shortest distance between two nodes on the *Sketch* is the same as that on the original network. We consider a shortest path $(u_1, \ldots, u_h)$ on the original network, where u_1 and u_h are *Sketch* nodes. Suppose the *Sketch* node closest to u_1 along the shortest path is u^*. By our *Sketch* edge creation strategy, we would create a *Sketch* edge (u_1, u^*). Repeat the similar procedure, we link all the consecutive *Sketch* nodes along the shortest path with *Sketch* edges. Hence, the distance invariance property holds.

Sketch Updates. The *Sketch* can be updated *locally*. The reason is, when an edge weight changes, only a small number, which is up to 50, of 5×5 C_5 sub-grids containing the end nodes of the edge would be affected. We thus can only recompute the *Sketch* edges intersecting these sub-grids for an edge weight update.

D. Sketch Complexities

In this section, we will show that *Sketch* takes $O(\phi^2 M^2)$ space where ϕ is typically a small constant. All the analysis are based on a road network sparsity dimension called *cover dimension*, following the arterial dimension in [13]. For ease of presentation, we will call the 5×5 sub-grid "the C_5 sub-grid of u", if the sub-grid whose central cell contains the node u. Similarly, we define "the C_3 sub-grid of u" as the 3×3 sub-grid whose central cell contains u.

We first introduce *cover dimension*. Cover dimension is defined on the number of cover edges within 5×5 sub-grids. Following [13], we make the following assumption. Our experiments (in []) show that the number of cover edges of a sub-grid is within $[3, 18]$ on average.

Assumption 1 (Cover Dimension): For any square grid on $\mathcal{G}$ and any 5×5 sub-grid C_5, the number of cover edges of C_5 is at most a constant ϕ, referred to as the *cover dimension* of $\mathcal{G}$.

Space Complexity. The number of *Sketch* nodes is closely related to cover dimension ϕ. Since a *Sketch* node must be an end node of a cover edge, the number of *Sketch* nodes is at most 2 times of the number of cover edges. Next, we count the number of cover edges for an imposed $M \times M$ grid. The number of 5×5 sub-grids is at most M^2. Each sub-grid *defines* at most ϕ cover edges. Totally, there are at most $M^2\phi$ cover edges. Therefore, there are at most $2M^2\phi$ *Sketch* nodes. And then, we show for a *Sketch* node u^*, there are at most ϕ *Sketch* edges with u^* as its end node. We consider the C_5 of u^*. For a *Sketch* edge (u^*, v^*), we discuss two cases. First, if the *Sketch* edge reaches outside of the C_5 (like (S_1, S_4) in Fig. 3 (b)), then this edge must be an edge of the original network. Otherwise, the shortest path from u^* to v^* contains at least two edges and they have a grid distance at least 3. Based on Lemma 2, there is a closer *Sketch* node to u^* along the shortest path. This violates our strategy of creating a *Sketch* edge, as an *Sketch* edge (u^*, v^*) is created means there is no other *Sketch* nodes in between along the shortest path from u^* to v^*. Therefore, such *Sketch* edges correspond to edges of the original networks and they are cover edges by definition. It follows that the number of such *Sketch* edges must be at most the number of cover edges, which is at most ϕ. Second, if the *Sketch* edge locates inside C_5, then the number of the *Sketch* edges with u^* as its end node is at most the number of *Sketch* nodes in C_5. Any sketch node u^* in C_5 can be mapped to a 5×5 sub-grid that defines a cover edge associated with u^* and intersects with C_5. Note that, C_5 intersects at most $81 (= 9 \times 9)$ 5×5 sub-grids (including itself). Those intersected sub-grids contain at most 81ϕ cover edges, and thus we have at most $81 \times 2\phi = O(\phi)$ *Sketch* nodes associated with u^* by sketch edges. In total, the number of *Sketch* edges is $2M^2\phi \times O(\phi) = O(\phi^2 M^2)$. We formalize this result as follows.

Lemma 3: The *Sketch* costs $O(\phi^2 M^2)$ space.

Time Complexity. We denote b as the maximal number of edges that intersect a cell, and denote n' as the maximal number of nodes in a cell. We assume the maximal degree of a node is d. In *Sketch* node selection, we conduct at most bM^2 rounds of Dijkstra searches in a 5×5 sub-grid. Each round of Dijkstra search costs $O(d(25n') \log (25n'))$. Totally, it takes $O(bdM^2 n' \log n')$ time to select the *Sketch* nodes. Then, in *Sketch* edge creation, we conduct $O(\phi M^2)$ rounds of Dijkstra searches, since the number of *Sketch* nodes is $O(\phi M^2)$. Each round of Dijkstra search takes $O(d(25n') \log (25n'))$ time.

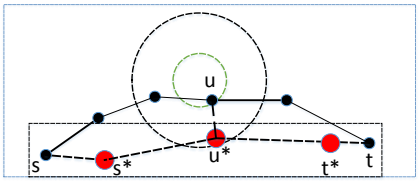


Fig. 4: Key ideas of `Sketch` based pruning.

Totally, it takes $O(d\phi M^2\ n' \log n')$ time. We formalize the results as follows.

Lemma 4: It takes $O((b+\phi)dM^2 n' \log n')$ time to construct a *Sketch*.

VI. The Sketch Solution

In this section, we propose efficient ROAM query algorithms based on the *Sketch*. We first introduce a general idea to use the *Sketch* for faster pruning in Sec. VI-A, namely, the sketch framework. Then, we incorporate the goal-directed search into the sketch framework for ROAM query in Sec. VI-B. Lastly, we describe how to extend the solutions to directed networks in Sec. VI-C.

A. The Sketch framework

Key Idea. We map each node u to its *cover sketch nodes* S_u, if u is not a sketch node (we also call it a non-sketch node). A node u^* is a *cover sketch node* of a non-sketch node u, if u^* satisfies: 1) u^* is an end node of a cover edge defined in the C_5 sub-grid of u; and 2) u^* is inside the C_3 sub-grid of u. The key observation is, when s and t are enough faraway and u is on a ρ-route from s to t, there must be a node in S_u on a ρ-route from s to t defined on the (s,t)-*sketch*. The (s,t)-*sketch* is the *Sketch* augmented with a few created edges that link s and t to their cover sketch nodes. As shown in Fig. 4, there is a ρ-route from s to t (in solid line) passing through certain node u, and we would show under certain circumstance, there must be a ρ-route from s to t (in dashed line) passing through a cover sketch node u^* of u and the route is defined on the (s,t)-sketch (in dashed rectangle). We summarize this result in Lemma 6.

(s,t)-sketch. We introduce how to efficiently create (s,t)-sketch offline. The key is to precompute all edges linking any non-sketch node u to its cover *Sketch* nodes, with the edge weight as the shortest distance between two end nodes. We call them *bridge edges*. An example is shown in Fig. 3 (c). There are two bridge edges (n_1, S_1) and (n_1, S_2), because S_1 and S_2 are inside the C_3 sub-grid of n_1 and they are on the cover edges (S_1, S_3) and (S_2, n_4). Then, when s and t in ROAM query are given in an online manner, we can efficiently extract bridge edges associated with s and t and combine them to the precomputed *Sketch*, forming the (s,t)-sketch. Next, we introduce how we create bridge edges offline. Take node s as an example and neglect the trivial case where node s itself is a *Sketch* node.

To construct bridge edges, a naive approach is to run a Dijkstra search from each non-sketch node until it scans all of its cover sketch nodes. This approach requires n rounds of regional Dijkstra searches, where "regional" means the search range is within a 5×5 sub-grid. A smarter approach is to do it reversely, by running regional Dijkstra search, within a 5×5 sub-grid, from the sketch nodes. When the Dijkstra search, with the sketch node u^* as the source node, scans any non-sketch node p, we create a bridge edge (p, u^*). The bridge edge construction time is $O(n \log n)$ and the space cost is linear to the input network (Lemma 5. Next, we show how to use (s,t)-sketch to conduct efficient ROAM query.

Lemma 5: It takes $O(d\phi n \log n)$ time to construct bridge edges; bridge edges cost $O(\phi n)$ space.

Proof: Since there are at most $O(\phi)$ sketch nodes inside a 5×5 sub-grid, we only need to run at most $O(\phi)$ rounds of Dijkstra search within each sub-grid. Hence, it takes $O(d\phi n \log n)$ time totally. In addition, since each node has at most ϕ bridge edges, it takes $O(\phi n)$ space in total. ■

Lemma 6: For nodes s, t and a non-sketch node u such that $gd(s,u) \geq 4$ and $gd(u,t) \geq 4$, if u is on a ρ-route from s to t on the original network, there exists a cover sketch node u^* of u, such that u^* is on a ρ-route from s to t on the (s,t)-*sketch*.

Use (s,t)-sketch to Prune. We rely on the precomputed sketch and bridge edges, and the key observation (Lemma 6) to conduct an efficient ROAM query. Lemma 6 relates a ρ-route passing through a non-sketch node u and a ρ-route passing through a cover sketch node of u on the (s,t)-sketch. With Lemma 6, to handle ROAM query, the given circle $circ(o,r)$ is enlarged to be $circ(o, r+\sigma)$ with the same center, and we show that the larger circle contains all the cover sketch nodes of the non-sketch nodes in $circ(o,r)$. In Fig. 4. The cover sketch node u^* of node u is covered by a larger circle. We formalize this claim in Claim 2. By combining Lemma 6 and Claim 2, we show our main result in Theorem 1. It states that, under certain conditions, if there is a ρ-route from s to t going across the smaller circle, there must be a ρ-route from s to t going across the larger circle and such a ρ-route is defined on the (s,t)-sketch. As such, we can first examine whether there is a ρ-route from s to t on the (s,t)-sketch crossing $circ(o, r+\sigma)$. If no, we can directly return $false$. Such examination is efficient as the *sketch* is small.

Claim 2: circle $circ(o, r + 2^{1.5}C_d)$ contains all the cover sketch nodes of non-sketch nodes in $circ(o,r)$.

Detailed Query Procedures. The query algorithm is divided into two parts. First, when o is faraway from s and t, we use Theorem 1 to test whether there is a ρ-route from s to t intersecting $circ(o, r+2^{1.5}C_d)$ on the (s,t)-*sketch* with certain ROAM query approach $\mathcal{M}$. If such a ρ-route does not exist, we return $false$, otherwise we resort to the ROAM query on the original network. The procedures are summarized in Alg. 5.

Theorem 1 (Sketch Pruning): If $gd(o,s) \geq 4 + r/C_d$ and $gd(o,t) \geq 4 + r/C_d$, then $ROAM(s,t,o,r,\rho)$ returns $true$ on $\mathcal{G}$ implies $ROAM(s,t,o,r+2^{1.5}C_d,\rho)$ returning $true$ on the (s,t)-sketch, where C_d is the width of a grid cell edge.

Summary. We introduce a general search framework based on *sketch*, which can enhance existing ROAM query algorithms (see experiments in Fig. 7 (g.1) to (g.4)).

B. Optimization with Goal-Directed Search

Previously, we use Dijkstra based algorithm to perform ROAM query. This method can be improved by the goal-

Algorithm 5: $Sketch$-search$(\mathcal{G}, s, t, o, r, \rho)$

```
exist ← true;
get (s,t)-sketch from index;
if gd(o,s) ≥ 4 + r/C_d and gd(o,t) ≥ 4 + r/C_d then
    exist ← M(s,t,o, r + 2^1.5 C_d, ρ) in (s,t)-sketch;
      /* M is a ROAM query algorithm. */
if exist = false then
    return false ;
return M(s,t,o,r,ρ) on G;
```

directed search with *Sketch* node based distance estimation.

Key idea. We propose an extension of A^* search [23] to handle ROAM query. The goal directed search requires shortest distance estimation between two nodes. We use triangle inequality of the shortest distance measure to perform estimation. Since sketch nodes are structurally important, we precompute the shortest distances between a set of selected *Sketch* nodes $(l_1, \ldots, l_k)$ and other nodes. By the triangle inequality, therefore, for any two nodes u and v, we have $dist(p,q) \geq \max_{1 \leq i \leq k}\{|dist(l_i,p) - dist(l_i,q)|\}$.

A^* Extension. We first revisit $A^\star$ search, and then describe how to extend it to handle ROAM query (Lemma 7). In contrast to the classical Dijkstra search, $A^\star$ search notes that the search expansion should be influenced by the location of t. This strategy is called *goal directed search*, where the goal refers to the target node t. Specifically, $A^\star$ search expands the node u with the current smallest value of $dist(s,u) + lower\_dist(u,t)$, where $dis(s,u)$ is the currently found shortest distance between node s and node u, and $lower\_dist(u,t)$ is a lower bound of the shortest distance between node u and the target node t. As such, the search towards the opposite directions (i.e., deviating faraway from t) is cut down considerably. Compared with the edge relaxation of Dijkstra search, $A^\star$ sets the value of $dis(s,v)+lower\_dist(v,t)$ as the priority value for node v.

Following above goal directed search, we show that for ROAM query, the search can stop when it scans a node u such that $dist(s,u)+lower\_dist(u,t) > (1+\rho)dist(s,t)$. The result is shown in Lemma 7. Interested readers are referred to our full technical report [] for a detailed pseudocode.

Lemma 7 (Stop Rule): When the scanned node u has $dist(s,u) + lower\_dist(u,t) > (1+\rho)dist(s,t)$, each node of any ρ-route is scanned.

C. Extension to Directed Road Networks

We discuss how to extend our solutions to directed road networks. In a directed road network, there can be two directed edges (u,v) and (v,u) between nodes u and v. In addition, the weight of (u,v) can be not equal to that of (v,u).

Queries Adaptation. The adaptation relies on two kinds of Dijkstra searches, i.e., *forward* Dijkstra search and *backward* Dijkstra search. Forward Dijkstra search only relaxes each edge (u,v) when node u is scanned, while backward Dijkstra search only relaxes each edge (v,u) when node u is scanned. Based on these two concepts, we conveniently adapt the ROAM query algorithms to directed road networks. For instance, in `Basic` method, we use forward Dijkstra search for source node s and backward Dijkstra search for source node t. Then, for each scanned node u, $dist(s,u) + dist(u,t)$ (instead of $dist(s,u)+dist(t,u)$) is computed. We then check whether this distance is at most $(1+\rho)dist(s,t)$. Following this idea, we further demonstrate the adaptation needed for *sketch construction* and queries.

Sketch Adaptation. When selecting sketch nodes, forward and backward Dijkstra searches are conducted for each node u. Respectively, within each C_5 sub-grid, we have forward (resp. backward) shortest paths from (resp. to) u. In defining cover edges, all those shortest paths are considered and use the same strategy we discussed to select the end nodes of cover edges as sketch nodes. When creating sketch edges, we respectively create forward sketch edge and backward sketch edge. The backward sketch edge (v,u) means the shortest path from u to v does not contain any other sketch nodes. Similarly, we maintain forward bridge edges and backward bridge edges for each node u. The (s,t)-*sketch* is composed of the *sketch* with the forward bridge edges of s and backward bridge edges of t. As such, the sketch pruning (Theorem 1) still holds.

VII. Experiments

In this section, we demonstrate the experimental evaluation for the proposed methods, in terms of efficiency (Sec. VII-B), index cost (Sec. VII-C) and effectiveness (Sec. VII-D).

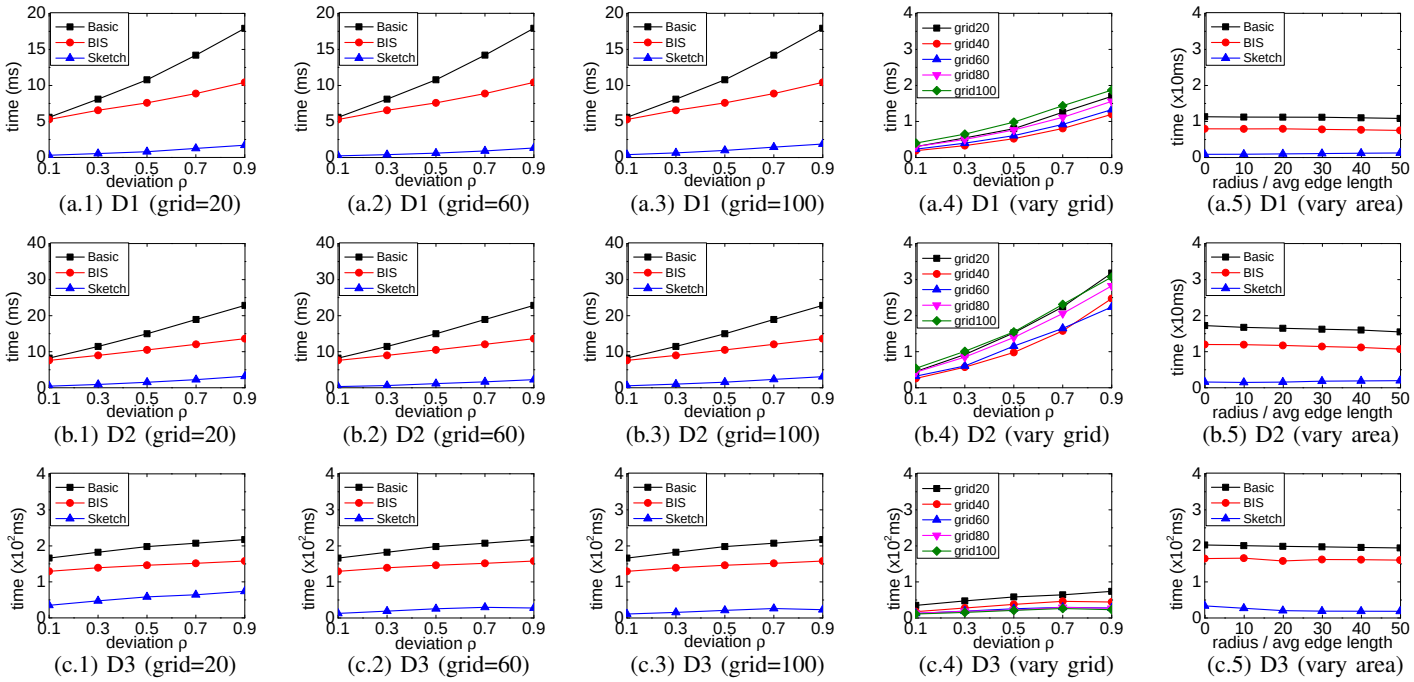
A. Setup

We perform ROAM queries on 7 real datasets, $D1, \ldots, D7$, downloaded from [24]. Tab. II shows the dataset characteristics. The set of ROAM queries on D1 (resp. D2) is generated based on the green (resp. yellow) cab trajectories recorded in Jan 2015 [25]. We set the pick-up/drop-off locations as the (s,t) pairs, and the areas are generated randomly. D3 contains 60 POIs from Florida. The starting location s, destination location t and the location o are randomly selected from the 60 POIs. For D4 to D7, we aim to perform the scalability testing of different large scale datasets. D7 is one of the biggest road network (full USA) published. The evaluation is done on a machine with 16GB memory and a 2.5GHz Intel core i7. We compare the efficiency, index sizes and indexing time. We select the shortest path deviation within a range of $[0.1 - 0.9]$, which means the ρ-route does not stretch up to double of the exact shortest path. Last, we test the usefulness of the ROAM query based on the New York taxi datasets.

Competitors. Besides the comparison among 3 exact methods proposed in this paper, namely `Basic`, `BIS`, and `Sketch`, we also compare `Sketch` method with the solution employed by [19]. Particularly, [19] uses a state-of-the-art index-based shortest distance algorithm (e.g., CH [14]), to compute the shortest distances between s (resp. t) and the pick-up location so as to decide whether the detour cost exceeds $(1+\rho) \times dist(s,t)$. To adapt their methods to the ROAM query which employs a pick-up area instead of a pick-up location, we sample 50% of nodes inside the given area. Once we meet a sample node u and $dist(s,u) + dist(u,t) \leq (1+\rho) \cdot dist(s,t)$, we return $true$ immediately. We name this method P2P as it uses a point to point shortest distance algorithm. For the `Sketch` method, we incorporate it with the goal-directed search introduced in Sec. VI-B unless specified otherwise.

B. Efficiency

Fig. 5 show the efficiency comparisons among the approaches. The running time is the average of the running time

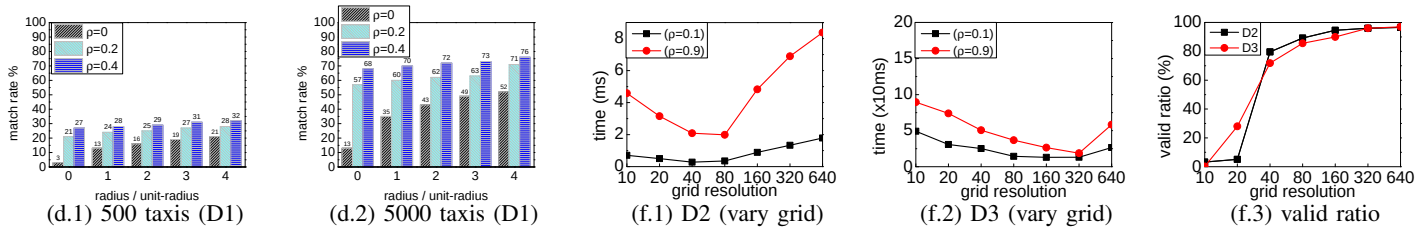


(a.1) D1 (grid=20) (a.2) D1 (grid=60) (a.3) D1 (grid=100) (a.4) D1 (vary grid) (a.5) D1 (vary area)

(b.1) D2 (grid=20) (b.2) D2 (grid=60) (b.3) D2 (grid=100) (b.4) D2 (vary grid) (b.5) D2 (vary area)

(c.1) D3 (grid=20) (c.2) D3 (grid=60) (c.3) D3 (grid=100) (c.4) D3 (vary grid) (c.5) D3 (vary area)

Fig. 5: Efficiency results on practical query datasets D1, D2 and D3.



(d.1) 500 taxis (D1) (d.2) 5000 taxis (D1) (f.1) D2 (vary grid) (f.2) D3 (vary grid) (f.3) valid ratio

Fig. 6: Effectiveness of ROAM query (d.1 and d.2); Resolution study (f.1 to f.3).

ID	Road Network	#Edges	#Nodes	Query set
D1	New York (NY)	733846	264346	G-cab
D2	New York (NY)	733846	264346	Y-cab
D3	Florida (FLA)	2712798	1070376	POI sets
D4	Northeast (NE)	3897636	1524453	Random
D5	Lakes (LK)	6885658	2758119	Random
D6	Western (WU)	15248146	6262104	Random
D7	Full USA (FU)	58333344	23947347	Random

TABLE II: Datasets

spent on 400 queries. The 400 queries consist of 4 groups of 100 queries, where each group of queries is set with area radius of 10 (resp. 20,30,40) times the average edge length respectively. The unit of running time reported is either "ms" (milliseconds) or its scaled version "$\times 10^t$ ms". For example "$\times 10^3$ ms" means the time unit is "1000 ms". As shown in Fig. 5, `BIS` is up to 2 times faster than `Basic` as its search space is relatively smaller. `Sketch` performs the best in efficiency. For dataset D1 and D2, `Sketch` can be up to 30 times faster than the `basic` approach.

Varying ρ. `Sketch` shows a good scalability when the deviation value ρ varies (see Fig. 5). The changes of the deviation value ρ incur a small deviation of the running time. For certain dataset (e.g., Fig. 7 (e.1)), we observe that the running time slightly drops down when ρ increases from 0.7 to 0.9. This is due to our implementation optimization when ROAM query returns $true$. The search can stop once we visit a node u such that $dist(s,u) + lower\_dist(u,t) > (1+\rho)dist(s,t)$. Since there are more queries returning $true$ for $\rho = 0.9$ compared with $\rho = 0.7$. This, sometimes, *can* lead to shorter running time for $\rho = 0.9$.

Grid Resolution Study. We study how to set a reasonable grid resolution. The grid resolution is not monotone to the performance. When the imposed grid is finer, the `Sketch` is larger and thus executing queries on the `Sketch` becomes slower. On the other hand, when the grid is coarser, less queries will satisfy the pruning condition in Theorem 1 (line 3 in Alg. 5), and hence less queries enjoy faster pruning. For ease of presentation, we refer to the percentage of queries satisfying the conditions in line 3 of Alg. 5, among a set of queries, as *valid ratio*. We thus study the relationship among valid ratio, grid resolution and the query performance, as shown in Fig. 6 (f.1) to (f.3). We select two real datasets with significant different distribution of (s,t) distances, namely D2 and D3. We randomly pick 500 test queries for D2 and D3 respectively. The difference of distance distribution of the two datasets is verified in Fig. 6 (f.3), such that the valid ratio has a steady growth for D3 while a drastic valid ratio jump is observed for D2. To gauge the relationship between query time and grid resolution, we pick *another* 2000 queries for testing query running times, as shown in Fig. 6 (f.1) and (f.2). These results show that with

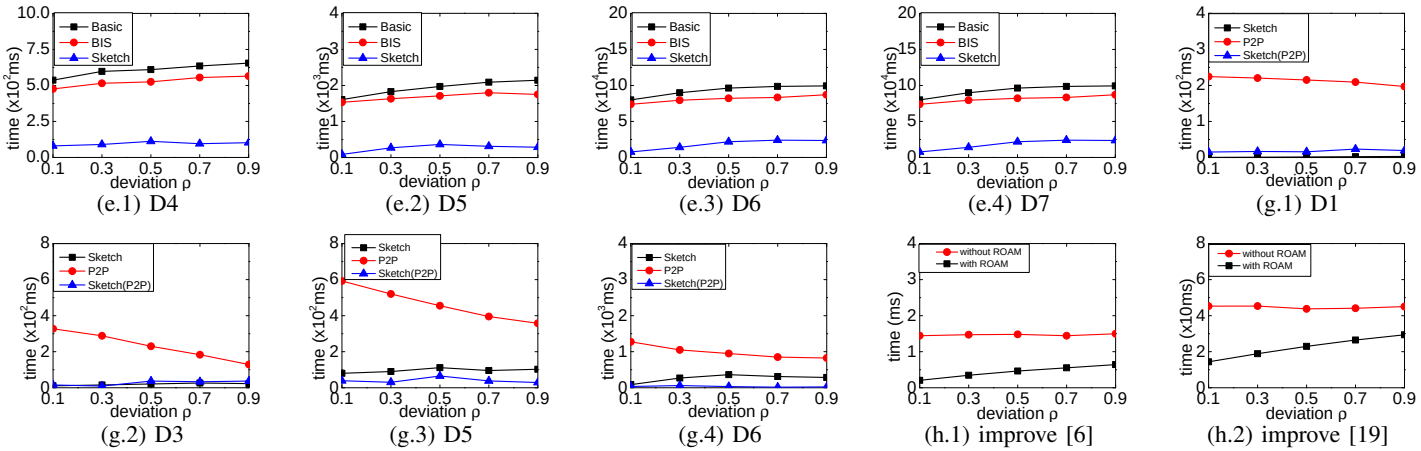


Fig. 7: Scalability on large road networks (e.1-e.4); Comparison to and enhancing shortest distance based algorithms P2P (employed by [19]) (g.1-g.4); Support some existing systems [6], [19] (h.1-h.2).

the increase of grid resolution, the query time first drops down and then goes up. This indicates that when the grid resolution is small, the valid ratio outweighs the sketch size, in affecting the query time, and vise versa when grid resolution is large. In combing these results, we conclude that a good resolution should simultaneously satisfy two criteria: 1) a majority of queries (80% to 90%) can satisfy line 3 in Alg. 5; 2) the grid resolution is not overly large. For example, in this test, 80×80 would be a good choice for D2 and D3.

Meanwhile, we also show that the *sensitivity* of grid resolution within an empirical range is not high. As shown in Fig. 5 (a.4), (b.4) and (c.4), the query time is close when grid resolution varies from 20×20 to 100×100. This means, when a good empirical grid resolution range (e.g., 20×20 to 100×100) is selected by the aforementioned two criteria, it is less important to gauge an optimized detailed resolution.

Varying Area Size. Fig. 5 (a.5), (b.5) and (c.5) show the effect of varying area sizes. We vary the area radius from 0 to 50 times of the average edge length. The grid resolution is set to 100×100 (by default) for the sketch method. ρ is set to 0.5. In `Basic` and `BIS`, the search periphery touches the area earlier if the area is larger. Therefore, the search can stop earlier once it finds a node in the area satisfying the positive condition rule (Claim 1). Such changes have subtle effects on `Sketch`.

Varying Network Size. We confirm the scalability of our algorithms in testing large scale road networks (see Fig. 7 (e.1) to (e.4)). We set 100×100 grid for `Sketch`. The `Sketch` method consistently outperforms the `Basic` approach by around 1 order of magnitude for most settings.

Dataset	Input	Sketch		
		time	sketch	bridge edges
NY	22.1M	1.9mins	4.2M	148.6M
FLA	86.6M	8.1mins	1.3M	270.9M
NE	126.9M	8.3mins	3.5M	1.1G
LKS	232.2M	13.0mins	3.1M	2.2G
WU	534.6M	55.0mins	7.3M	6.6G
USA	2.1G	4.5hours	4.2M	12.8G

TABLE III: Index size and time.

Comparison to P2P. We compare the `sketch` method with `P2P` method in four representative datasets, as shown in Fig. 7 (g.1) to (g.4). We did not show the query performance on the largest dataset D7 because the CH index algorithm for `P2P` method runs out of memory on testing D7. From the results, we conclude that `P2P` performs better when the deviation value ρ increases. The reasons are: 1) the `P2P` method has more early stops when more queries returning $true$. 2) More queries return $true$ when ρ increases. We also observe `P2P` method has a good scalability on large datasets, as the shortest distance algorithm CH can keep very efficient. However, for all the testable datasets, `Sketch` method consistently outperforms `P2P` method. In addition, we show our sketch framework can help enhance them, by incorporating the `P2P` method into our sketch based search. Particularly, before using `P2P` method, we conduct a sketch based pruning first. We use `P2P` method to verify the result only when the sketch based pruning does not work. We refer to this method as `Sketch(P2P)`. `Sketch(P2P)` improves the `P2P` method significantly. This experiment thus shows an evidence that our sketch framework can enhance existing ROAM query algorithms.

Improving Keyword-aware Routing. We show that an efficient evaluation of the ROAM query helps improve a keyword-aware routing algorithm [6]. Keyword-aware routing aims to extract a route from s to t and it goes through several POIs that are marked with input keywords (e.g., *restaurant*), within a time limit (e.g., 2 hours). A satisfactory route can be transformed to a ρ-route of certain ρ and each POI passed by should be on a ρ-route (Interested readers can refer to [**?**]). We implement the fastest algorithm suggested in [6]. We randomly select 5 keywords and 500 POIs marked with each of the keywords on FLA road network. Our Sketch based algorithm can be easily adapted to test whether one POI q is on a ρ-route from s to t, by setting the service area as $circ(q, 0)$. As such, we incorporate our algorithm to their algorithm (marked as "with ROAM"), and a significant efficiency improvement is observed (Fig. 7 (h.1)). The reason is, the most costly computation is the distance computation between POIs. By the ROAM query, some computation is unnecessary as some POIs, which are not on a ρ-route, can be efficiently extracted by a ROAM query algorithm.

Improving Dynamic Ride-sharing. An efficient ROAM query helps improve an important case in dynamic ride-sharing [19]. A dynamic ride-sharing system considers to assign a new

passenger request (v_1, v_2, T) to a taxi, where v_1 (resp. v_2) denotes the passenger's current location (resp. destination) and T is the maximal waiting time of the passenger to be picked up. For each passenger, the service guarantee for her, as suggested by [19], is that her actual trip must be a ρ-route with respect to her origin and destination. An important case is, when a taxi is taking one passenger from s to t, how the coordinate system assigns one of the requests from close passengers (so as to satisfy waiting time) to the taxi so that the service guarantee is still achieved by the detour route (s, v_1, v_2, t)? In other words, (s, v_1, v_2, t) should be a ρ-route from s to t. Hence, an efficient ROAM query can be incorporated to help optimize the passenger request selection. We conduct experiments on the FLA road network. We assume there are 100 queries satisfying the waiting time guarantee are considered to be assigned to the taxi. As shown in Fig. 7 (h.2), with a ROAM query incorporated (named "with ROAM"), the query performance is significantly improved.

C. Index Cost

Tab. III shows the index size and indexing time for `Sketch`. The grid resolution is set to 100×100. For moderate sized datasets (up to millions of edges) such as NY, FLA, NE and LK, the index construction is finished within a few minutes. For larger datasets (up to billions of edges) such as WU and USA, the construction is finished within a few hours. The space costs are moderate compared with the input sizes. Tab. IV shows the index construction time and space cost when using different grid resolutions. The *Sketch* size increases when the grid becomes finer (from 20×20 to 100×100). The results conform to our theoretical analysis that the size of the `Sketch` is linear to the number of grid cells involved. We only show the results in representative datasets. Our tested results in other datasets are similar.

Dataset	grid	time	sketch	bridge edges
FLA	20×20	596s	0.1M	281.5M
	40×40	563s	0.3M	286.3M
	60×60	549s	0.6M	286.6M
	80×80	505s	0.9M	281.1M
	100×100	487s	1.3M	270.9M

TABLE IV: Index size and time for different grid resolutions.

D. Effectiveness

We are interested in the usefulness of ROAM query. We consider a scenario: there are c moving objects (e.g., taxis) and 100 random points or areas (e.g., 100 querying passengers). We say a point or an area matches to the c moving objects if there exist at least one moving object that can match the point or area with ROAM query. In ride-sharing, this indicates at least one taxi can serve the passenger. The moving objects ((s, t) pairs) are randomly drawn from dataset D1. We vary the deviation ρ in [0,0.4]. We assume the 100 moving objects have identical ρ values. If there are p points/areas with a matched moving object, we say the match rate is $p\%$. We compare the match rate of different settings of ROAM to a traditional setting (i.e., radius=0 and $\rho = 0$). In particular, in addition to varying ρ values, we also vary the area radius from 0-4 units of radius, where we set 1 unit radius as 5 times the average edge length. The result is shown in Fig. 6 (d.1) and (d.2). We set $c = 500$ and $c = 5000$. *From the result, we observe that increasing area size and deviation ratio helps increase the match rate.* For example, in Fig. 6 (d.1), the match rates jump from 3% to 32%. Low match rates can be annoying. For example, when the match rate is only 3% in a taxi recommendation, it means 97% passenger queries cannot get a response. ROAM query brings in more possibilities to increase the match rate.

VIII. Further Discussions

We discuss the solutions for non-circular area $\mathcal{A}$. Our `Sketch` based pruning can still be used, by modifying Claim 2 slightly. In particular, we can still find a larger circle that contains all the cover sketch nodes of non-sketch nodes inside $\mathcal{A}$, regardless of whether $\mathcal{A}$ is a circle or not. For instance, we can first extract a circle $\mathcal{C}$, with a radius r and central node o, that covers $\mathcal{A}$. Then Claim 2 still holds, in the sense that $circ(o, r + 2^{1.5} C_d)$ contains all the cover sketch nodes of non-sketch nodes in $\mathcal{C}$ (and of course contains cover sketch nodes of non-sketch nodes in $\mathcal{A}$, because $\mathcal{A}$ is contained in $\mathcal{C}$). Hence, we can still use Theorem 1 for `Sketch` based pruning. Then, we can extract all the nodes inside $\mathcal{A}$ and use the `Sketch(P2P)` method introduced in our experiments to handle ROAM query for a non-circular area.

IX. Conclusions

In this paper, we propose the ROAM query, which supports a broad scope of location based applications. We develop a general pruning framework based on novel data structures for ROAM query, and validate its efficiency and effectiveness on large real road network datasets.

References

[1] "http://www.rydesharing.com/sg/home/."

[2] L. Kazemi and C. Shahabi, "Geocrowd: enabling query answering with spatial crowdsourcing," in *GIS*, 2012, pp. 189–198.

[3] P. Cheng, X. Lian, Z. Chen, R. Fu, L. Chen, J. Han, and J. Zhao, "Reliable diversity-based spatial crowdsourcing by moving workers," *VLDB*, vol. 8, no. 10, pp. 1022–1033, 2015.

[4] H. Yu, C. Miao, Z. Shen, and C. Leung, "Quality and budget aware task allocation for spatial crowdsourcing," in *ICAAMS*, 2015, pp. 1689–1690.

[5] "https://www.clickworker.com/mobile-crowdsourcing/."

[6] X. Cao, L. Chen, G. Cong, and X. Xiao, "Keyword-aware optimal route search," *PVLDB*, vol. 5, no. 11, pp. 1136–1147, 2012.

[7] S. C. Soma, T. Hashem, M. A. Cheema, and S. Samrose, "Trip planning queries with location privacy in spatial databases," *WWW*, pp. 1–32, 2016.

[8] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[9] L. Sint and D. de Champeaux, "An improved bidirectional heuristic search algorithm," *JACM*, vol. 24, no. 2, pp. 177–191, 1977.

[10] R. J. Gutman, "Reach-based routing: A new approach to shortest path algorithms optimized for road networks." *ALENEX/ANALC*, vol. 4, pp. 100–111, 2004.

[11] H. Samet, J. Sankaranarayanan, and H. Alborzi, "Scalable network distance browsing in spatial databases," in *SIGMOD*, 2008, pp. 43–54.

[12] H. Bast, S. Funke, D. Matijevic, P. Sanders, and D. Schultes, "In transit to constant time shortest-path queries in road networks." in *ALENEX*, 2007.

[13] A. D. Zhu, H. Ma, X. Xiao, S. Luo, Y. Tang, and S. Zhou, "Shortest path and distance queries on road networks: towards bridging theory and practice," in *SIGMOD*, 2013, pp. 857–868.

[14] R. Geisberger, P. Sanders, D. Schultes, and D. Delling, "Contraction hierarchies: Faster and simpler hierarchical routing in road networks," in *Experimental Algorithms*. Springer, 2008, pp. 319–333.

[15] S. Ma, Y. Zheng, and O. Wolfson, "Real-time city-scale taxi ridesharing," *TKDE*, vol. 27, no. 7, pp. 1782–1795, 2015.

[16] ——, "T-share: A large-scale dynamic taxi ridesharing service," in *ICDE*, 2013, pp. 410–421.

[17] B. Cao, L. Alarabi, M. F. Mokbel, and A. Basalamah, "Sharek: A scalable dynamic ride sharing system," in *MDM*, vol. 1, 2015, pp. 4–13.

[18] B. Shen, Y. Huang, and Y. Zhao, "Dynamic ridesharing," *SIGSPATIAL Special*, vol. 7, no. 3, pp. 3–10, 2016.

[19] Y. Huang, F. Bastani, R. Jin, and X. S. Wang, "Large scale real-time ridesharing with service guarantee on road networks," *VLDB*, vol. 7, no. 14, pp. 2017–2028, 2014.

[20] B. Ding, J. X. Yu, and L. Qin, "Finding time-dependent shortest paths over large graphs," in *EDBT*, 2008, pp. 205–216.

[21] S. Wang, W. Lin, Y. Yang, X. Xiao, and S. Zhou, "Efficient route planning on public transportation networks: A labelling approach," in *SIGMOD*, 2015, pp. 967–982.

[22] H. To, G. Ghinita, and C. Shahabi, "A framework for protecting worker location privacy in spatial crowdsourcing," *VLDB*, vol. 7, no. 10, pp. 919–930, 2014.

[23] A. V. Goldberg and C. Harrelson, "Computing the shortest path: A search meets graph theory," in *SODA*, 2005, pp. 156–165.

[24] "http://www.dis.uniroma1.it/challenge9/download.shtml."

[25] "http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml."

APPENDIX

Algorithm 6: $GoalDirectedSearch(\mathcal{G}, s, t, o, r, \rho)$

$dist(s,s) \leftarrow 0$, $dist(t,t) \leftarrow 0$;
$stdis = \infty$;
while *expandable from the search from* s **do**
  $u \leftarrow$ expanded node;
  if $u = t$ **then**
    $stdis \leftarrow dist(s,u)$;
  if $dist(s,u) + lower\_dist(u,t) > (1+\rho)stdis$ **then**
    Break;
  for *each edge* (u,v) **do**
    $dist(s,v) = min\{dist(s,v), dist(s,u) + w(u,v)\}$;
    priority of $v \leftarrow dist(s,v) + lower\_dist(v,t)$;
while *expandable from the search from* t **do**
  $u \leftarrow$ expanded node;
  if $dist(s,u) + dist(t,u) \leq (1+\rho)stdis$ **then**
    return $true$;
  if $dist(t,u) + lower\_dist(u,s) > (1+\rho)stdis$ **then**
    return $false$;
  for *each edge* (u,v) **do**
    $dist(t,v) = min\{dist(t,v), dist(t,u) + w(u,v)\}$;
    priority of $v \leftarrow dist(t,v) + lower\_dist(v,t)$;

We experimentally estimate the cover dimension. We impose a grid with resolution $2^{r+2} \times 2^{r+2}$, where r ranges in $[1, 10]$. With each grid imposed, we count the number of cover edges within every 5×5 sub-grid. We figure out the maximal number and the average number of cover edges for all our test datasets. The results are shown in Fig. 8. The x-axis is the resolution parameter r and "MAX" (resp. "AVG") is the maximal (resp. average) number of cover edges within each 5×5 sub-grid. All the results show that in practical settings, the number of cover edges is a small constant.

Proof of Lemma 1.

Proof: By the property of bi-directional Dijkstra search, if there exists one node v such that the its states of the Dijkstra searches from both sides are changed to *scanned*, then the shortest distance is exactly figured out. Therefore, when node

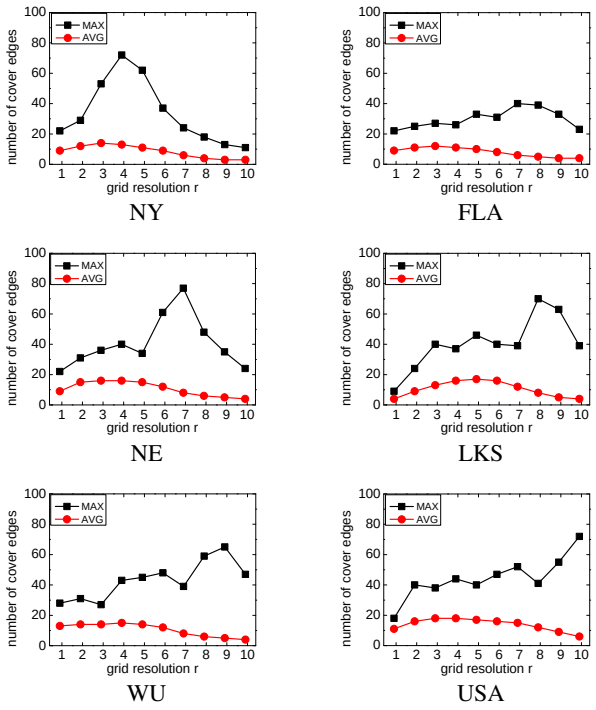


Fig. 8: Cover dimension estimation.

o has been scanned by the searches from both sides, the value $dist(s,t)$ is already known. Next, it suffices to at the moment when o is scanned by both sides and the condition of the lemma holds, there does not have a node $u^* \in circ(o,r)$ which still has at least one side of Dijkstra search do not scan it and that $dist(s,u^*) + dist(t,u^*) \leq (1+\rho)dist(s,t)$. We show this statement in two cases. Case (i): u^* has not been scanned by both sides, then $dist(s,u^*) \geq dist(s,o)$ as o is scanned before u^*. Similarly, $dist(t,u^*) \geq dist(t,o)$. By combining the two inequality we have

$$dist(s,u^*) + dist(t,u^*) \geq dist(s,o) + dist(t,o)$$

On the other hand, since u_s is scanned before o, then $dist(s,o) \geq dist(s,u_s)$. Also note that based on the lemma condition, we have

$$dist(s,u_s) + dist(t,o) > (1+\rho)dist(s,t)$$

Therefore,

$$\begin{aligned} dist(s,u^*) + dist(t,u^*) &\geq dist(s,o) + dist(t,o) \\ &\geq dist(s,u_s) + dist(t,o) \\ &> (1+\rho)dist(s,t) \end{aligned}$$

Case (ii): only one side of the search has scanned u^*, say the search from s. In this case, $dist(s,u^*) + dist(t,u^*) \geq dist(s,u_s) + dist(t,u^*) \geq dist(s,u_s) + dist(t,o) > (1+\rho)dist(s,t)$. ■

Proof of Lemma 6.

Proof: Since $gd(s,u) \geq 4$, node u must be outside of the C_5 sub-grid of s. Hence, the shortest path from node s to node u must pass through a cover edge e. We let the end node of e that locates inside the C_3 sub-grid of s be u^*. By

the definition of the cover sketch node, u^* is one cover sketch node of s. Next, we show that, the shortest path from s to u^* and the shortest path from u^* to t have a total length at most $dist(s,u) + dist(u,t)$. By triangle inequality, we have $dist(u^*,t) \leq dist(u^*,u) + dist(u,t)$. Then, plus $dist(s,u^*)$ to both sides, we have

$$dist(s,u^*) + dist(u^*,t) \leq dist(s,u^*) + dist(u^*,u) + dist(u,t) \\ = dist(s,u) + dist(u,t)$$

Since there is a ρ-route passing u, then $dist(s,u)+dist(u,t) \leq (1+\rho)\cdot dist(s,t)$. Therefore, there is also a ρ-route passing u^* because $dist(s,u^*) + dist(u^*,t) \leq dist(s,u) + dist(u,t) \leq (1+\rho)\cdot dist(s,t)$.

It remains to show $dist(s,u^*)$ and $dist(u^*,t)$ can be computed with the (s,t)-sketch so that the ρ-route from s to u^* and then to t on (s,t)-sketch is also a ρ-route. Since $gd(u,t) \geq 4$ and $gd(u,u^*) \leq 1$, we have $gd(u^*,t) \geq 3$. Similarly, we have $gd(u^*,s) \geq 3$. We first consider when both s and t are not sketch nodes. By Lemma 2, there is another sketch node on the shortest path from s to u^*. Among such intermediate sketch nodes along the path, we find the one that is the cover sketch node of s, denoted as u_0. By our bridge edge creation, (s,u_0) is a bridge edge, which belongs to the (s,t)-sketch. Similarly, we can find a bridge edge (u_1,u^*) along the shortest path from s to u^*. In addition, the shortest distance between u_0 and u_1 can be exactly computed on (s,t)-*sketch*. Therefore, the shortest distance $dist(s,u^*)$ can be exactly computed on the (s,t)-sketch; so does $dist(t,u^*)$. Thus, the route following the shortest path from s to u^* and then to t on the (s,t)-sketch has a same length as its counterpart on the original network. In addition, the shortest distance from s to t on the (s,t)-sketch is at least their shortest on the original network. Given that the route following the shortest path from s to u^* and then to t is a ρ-route on the original network, it then follows that on the (s,t)-sketch, the route following the shortest path from s to u^* and then to t is also a ρ-route. The proof also easily applies to the cases where s or t is a sketch node. ■

Proof of Claim 2.

Proof: For any non-sketch node u, any of its cover sketch nodes must inside the C_3 sub-grid of node u. Next, we show each C_3 sub-grid of a node in $circ(o,r)$ must be totally contained in $circ(o, r+2^{1.5}C_d)$. W.l.o.g, we consider the C_3 sub-grid of node $u \in circ(o,r)$. Then, every point p in the C_3 sub-grid must be within a Euclidean distance $2^{1.5}C_d$ from u. Therefore, the Euclidean distance between p and o, is at most $eu\_dist(p,u) + eu\_dist(u,o) \leq 2^{1.5}C_d + r$. ■

Proof of Theorem 1.

Proof: If $ROAM(s,t,o,r,\rho)$ returns $true$, then there exists one node $o' \in circ(o,r)$ such that $dist(s,o') + dist(t,o') \leq (1+\rho)dist(s,t)$. Given $gd(o,s) \geq 4 + r/C_d$, we have $gd(o',s) \geq 4$. Then, due to sketch node selection, there exists a sketch node p (can be o' itself) along the shortest path from o' to s such that $gd(o',p) \leq 1$. To finish the proof, it suffices to show the following two claims: 1) p is on a ρ-route of the (s,t)-sketch and 2) $p \in circ(o, r+2^{1.5}C_d)$. For 1), due to $gd(o',s) \geq 4$ and $gd(o',p) \leq 1$, we have $gd(p,s) \geq 3$. Also, since $gd(o,t) \geq 4 + r/C_d$, we have $gd(o',t) \geq 4$ and hence $gd(p,t) \geq 3$. Due to the distance invariance of the (s,t)-sketch, therefore, we can exactly compute $dist(s,p)$ and $dist(p,t)$ on the (s,t)-sketch. In addition, by triangle inequality, we have $dist(s,p) + dist(p,t) \leq dist(s,p) + dist(p,o') + dist(o',t) \leq dist(s,o') + dist(o',t) \leq (1+\rho)dist(s,t)$. Note that, the distance computed in (s,t)-sketch is at least $dist(s,t)$, thus p must be on a ρ-route of (s,t)-sketch. For 2), $eu\_dist(o,p) \leq eu\_dist(\ o,o') + eu\_dist(o',p) \leq r + 2^{0.5}C_d \cdot gd(o',p) \leq r + 2^{1.5}C_d$. ■

Proof of Lemma 7.

Proof: We prove by contradition. If a node $u*$ on a ρ-route and $u*$ has not been scanned when the expanded node u has $dist(s,u) + lower\_dist(u,t) > (1+\rho)dist(s,t)$. We consider the shortest path from node s to $u*$,$\{s = u_0, u_1, \ldots, u_k, u_{k+1} = u*\}$. $\forall 0 \leq i \leq k+1$, we have $dist(s,u_i) + dist(u_i,t) \leq (1+\rho)dist(s,t)$ since $u*$ is on a ρ-route. By the goal directed search, this means as long as u_i is in the queue, u_{i+1} must be in the queue, and all u_i must be scanned before u. This leads to a contradiction. ■