ROAM: A Fundamental Routing Query on Road Networks with Efficiency

Siqiang Luo, Reynold Cheng, Ben Kao, Xiaokui Xiao, Shuigeng Zhou, Jiafeng Hu

Abstract—Novel road-network applications often recommend a moving object (e.g., a vehicle) about interesting services or tasks on its way to a destination. A taxi-sharing system, for instance, suggests a new passenger to a taxi while it is serving another one. The traveling cost is then shared among these passengers. A fundamental query is: given two nodes s and t, and an area $\mathcal{A}$ on road network graph $\mathcal{G}$, is there a "good" route (e.g., short enough path) P from s to t that crosses $\mathcal{A}$ in $\mathcal{G}$? In a taxi-sharing system, s and t can be a taxi's current and destined locations, and $\mathcal{A}$ contains all the places to which a person waiting for a taxi is willing to walk. Answering this *Route and Area Matching* (ROAM) Query allows the application involved to recommend appropriate services to users efficiently. In this paper, we examine efficient ROAM query algorithms. Particularly, we develop solutions for finding a ρ-*route*, which is an s-t path that passes $\mathcal{A}$, with a length of at most $(1+\rho)$ times the shortest distance between s and t. The existence of a ρ-route implies that a service or task located at $\mathcal{A}$ can be found for a given moving object m, and that m only deviates slightly from its current route. We present comprehensive studies on index-free and index-based algorithms for answering ROAM queries. Comprehensive experiments show that our algorithm runs up to 30 times faster than baseline algorithms.

Index Terms—Road Networks, Query Performance, Indexing, Graph Algorithms

1 Introduction

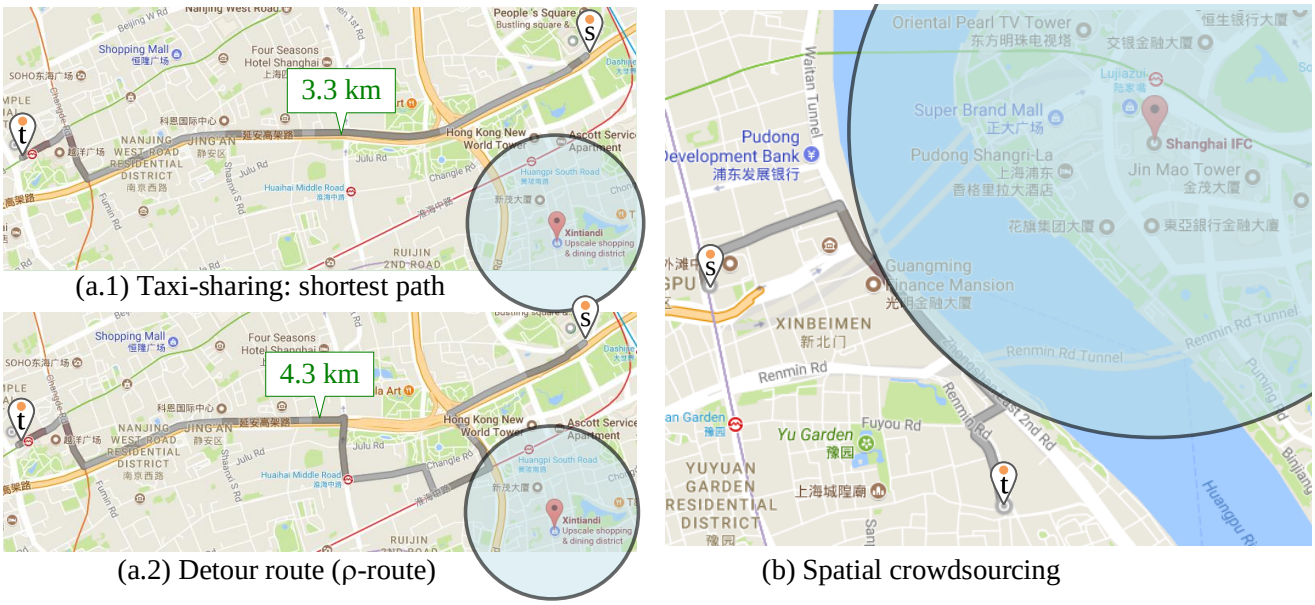
Location-based services, such as taxi-sharing and spatial crowdsourcing, have attracted tremendous interest in recent years. A taxi-sharing application (e.g., RYDE [1], Uber, Didi [2]) enables passengers to share a taxi ride; a spatial crowdsourcing system (e.g., [3], [4], [5], [6]) invites a crowd-sourcing worker to visit a given place to conduct a task (e.g., shooting photos for a place of interest). These applications, which take users' locations into account, provide convenient services and business opportunities.

These applications often share a common characteristic: while an *object* m (e.g., a taxi or a worker) is moving from origin s to destination t, the system recommends to m a "service request" (e.g., pick up a passenger, take a photo). Upon accepting the service request, m traverses to the area associated with that request, conducts the service, and then continues her journey to t. Figure 1(a.1) shows an example in taxi-sharing in Shanghai, a big city of China. A taxi, which picked up a passenger at Shanghai museum (at location s), is traveling along the shortest path (in gray) to Jing'an temple (at location t). The system tells the taxi driver that a new passenger requested for a shared ride at a shopping mall (the center of the circular area), who also plans to go to t. Since the traffic is congested around the shopping mall, the passenger is willing to walk a reasonable distance to increase her chance of being shared ride, and suppose the circular area is where she agrees to be picked up. The driver accepts the request from the passenger because there is a short route, as shown in Figure 1(a.2), that takes a detour from s to the service area to pick up the passenger, after which she continues to t. As another example, Figure 1(b) illustrates a typical spatial crowdsourcing task of taking a photo of a skyscraper in Shanghai (indicated by the location pin enclosed with a large circular area). The large circular area is the area within which shooting a photo of the building is suitable. A volunteer crowd-sourcing worker, who commutes between *the Bund Center* (at location s) and *Shanghai old street* (at location t), can be recommended for this task, as there is a short route (in grey) that starts at *the Bund Center*, going through the area during which a photo of the building can be taken, and finally goes to *Shanghai old street*.

To give a good service recommendation, a system should consider the amount of effort needed for an object to move to the suggested service area. Suppose that a service request e is in an area $\mathcal{A}$ (e.g., a pick-up area, a photo-taking area, a shopping centre). Then, an object m on a route from an origin s to a destination t would be interested to handle e, only if its detour from s to $\mathcal{A}$ does not seriously affect her original travel plan from s to t. Before recommending e to m, therefore, the system should check whether there is a "good" path from s to t that passes through $\mathcal{A}$. For example, the length of the path can be required to be at most $(1+\rho)$ 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 problem of finding such a path (called ρ-route) the *ROute and Area Matching (ROAM)* query. For instance, the route in Figure 1 (a.2) is a ρ-route for $\rho \geq \frac{4.3}{3.3} - 1$, because the length of the detour path is less than $(1+\rho)$ times the shortest distance between s and t. Moreover, this ρ-route passes through the pick-up area $\mathcal{A}$, and so the ρ-route is an answer to the ROAM query on $(s, t, \mathcal{A})$ for any $\rho \geq \frac{4.3}{3.3} - 1$.

Drawbacks of Existing Solutions. A naive way of answering a

- *Siqiang Luo, Reynold Cheng, Ben Kao and Jiafeng Hu are with the Department of Computer Science, the University of Hong Kong. e-mail: {sqluo, ckcheng, kao, jhu}@cs.hku.hk*
- *Xiaokui Xiao is with the School of Computing, National University of Singapore, Singapore. e-mail: xkxiao@nus.edu.sg*
- *Shuigeng Zhou is with Shanghai Key Lab of Intelligent Information Processing, School of Computer Science, Fudan University, and Shanghai Institute of Intelligent Electronics and Systems. e-mail: sgzhou@fudan.edu.cn*



(a.1) Taxi-sharing: shortest path

(a.2) Detour route (ρ-route)

(b) Spatial crowdsourcing

Fig. 1: Illustrating ROAM queries.

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. While a large body of works study the retrieval of the shortest path between two graph nodes [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19]), it is not clear how they can be extended to take into account the detour incurred by handling a service request. A few recent works in spatial crowdsourcing [4], [5], [6] and taxi-sharing [20], [21], [22], [23], [24] start to address this issue, but the 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 [10], Euclidean distance is often inaccurate, since an object's movement is constrained by a road network. Second, previous works assume that e is handled at a precise location; in practice, e can be served in a region. In taxi-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. (One example is in Figure 1 (b), where the photo-taking area crosses a river.) Moreover, when location privacy is a concern, the system may not even have access to the precise location of the service request. For instance, a passenger asking for a shared ride in a sensitive area (e.g., a hospital) may "cloak" her position by submitting a larger region to the system [25]. Therefore, none of the existing techniques can effectively support ROAM queries.

Our Contributions. We conduct an in-depth study on the ROAM query, which is fundamental to a multitude of applications (more discussions can be referred to Section. 8). Our contributions are listed as follows.

- We develop an index-free approach, known as the *bi-directional search with temporary stop*, which significantly reduces the search space and yields significant improvement over the basic solution.
- We propose an index-based approach, dubbed *Sketch*, that has two major advantages over the baseline solutions: 1) faster determination of whether there exists a satisfactory ρ-route, and 2) more efficient extraction of a ρ-route if it exists. We design the *sketch graph* to capture the essence of $\mathcal{G}$. The unique characteristic of the *sketch graph* allows us to efficiently process ROAM queries. There are two technical novelties of our approach. First, the search leverages a projection theorem (Theorem 1), and hence the processing of a ROAM query is transferred to the *sketch graph*, in which the computation overhead is largely reduced. Second, an optimized goal directed traversal is applied to further reduce the search cost, which yields a more efficient extraction of a ρ-route than existing approaches do.
- We analyze the complexities of the *Sketch Graph* based on a notion called *Cover Dimension* (denoted by ϕ), which is shown to be a small constant on real road networks. We further prove that the space cost of the *sketch graph* is $O(\phi|V|)$ (i.e., near optimal) and its construction is also efficient.
- We have conducted experiments on large real road network graphs. On a graph about New York's taxi trajectories, our algorithm is up to 30 times faster than the baseline approach.

Organization. The rest of the paper is organized as follows. Section 2 discusses the problem settings. Section 3 gives the preliminaries. Section 4 presents two index free methods, *Basic* and *BIS*, to handle the ROAM query. Section 5 presents the *sketch graph* and Section 6 gives the ROAM query algorithm based on *sketch graph*. Section 7 shows our experimental evaluation. Section 8 discusses related works. Section 9 concludes the paper. Section 10 gives all the proofs and additional pseudocode and experiments.

2 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 e has two end nodes u and v, we also use notation (u, v) to represent edge e. For any path in $\mathcal{G}$, we define its *length* as the sum of weights of the edges in the path.

The shortest path between two nodes u and v, denoted as $u \rightsquigarrow v$, is the path from u to v with the smallest length. Accordingly, the *shortest distance* between u and v, $dist(u, v)$, is the length of $u \rightsquigarrow v$. We denote the Euclidean distance between u and v as $dist_E(u, v)$. Unless otherwise specified, we refer to the shortest distance between two nodes as their distance. For ease of presentation, we assume that $\mathcal{G}$ is undirected. We will relax this assumption in Section 6.2.

A path between s and t is called a ρ-route ($\rho \geq 0$), if its length is at most $(1 + \rho) \cdot dist(s, t)$. An area, denoted by $circ(o, r)$, is a circular region that is within a Euclidean distance r to a node o. A path is said to intersect $circ(o, r)$ if at least one node in the path is in $circ(o, r)$. We now define the ROAM query.

Definition 1 (The ROAM Query)**.** *Given two nodes s and t and a circular region $circ(o, r)$, a ROAM query returns a ρ-route between s and t that intersects $circ(o, r)$, or false if such ρ-route does not exist.*

In the subsequent discussions, we assume that nodes s and t are both outside $circ(o, r)$, as otherwise the case is trivial: the shortest path $s \rightsquigarrow t$ is a ρ-route. Further, while our solutions are adaptable to handle other service area shapes (see Section 6.2), a circular service area is assumed as it is fairly natural in applications.

3 Preliminaries

We review the Dijkstra's algorithm (a.k.a. Dijkstra search) [7], which is fundamental to the techniques we will present. Dijkstra search computes the 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 shortest known 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 actual distance from s. Initially, the node s is set to state *labeled* and associated with distance 0. After that, an iterative *min-node location* and *relaxation* process is executed:

1) Locate min-node. Locate the node u associated with the smallest distance $dist(s, u)$; Change the state of node u as *scanned*.
2) Relax edge. For edge (u, v) such that the label of v is not *scanned*, update $dist(s, v) \leftarrow min(dist(s, v), dist(s, u) + w(u, v))$; Change the state of node v as *labeled*.

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 also define the search space of Dijkstra search as *Dijkstra ball*, formally defined as follows.

Definition 2 (Dijkstra Ball)**.** *A Dijkstra ball $ball(s, \tau)$ is the set of nodes whose shortest distances to s are at most τ.*

4 Index-Free Approaches

In this section, we introduce index-free ROAM query algorithms. Index-free algorithms avoid storing pre-computed indexes in physical memory, making them suitable for applications that are with stringent memory constraints. For example, when the ROAM query is used in mobile applications, memory-saving algorithms are preferred. However, choices of algorithmic designs are limited if index-free property is required. Among them, the Dijkstra algorithm or its variants are often the first that can be taken into account.

TABLE 1: Notations used in this paper.

Notation	Description
$\mathcal{G}$	road network
$u \rightsquigarrow v$	shortest path from u to v
(u, v)	an edge between u and v
$dist(u, v)$	shortest distance between u and v
$dist_E(u, v)$	Euclidean distance between u and v
$dis[u]$	associated distance of u during search
$circ(o, r)$	a circle with center o and radius r
$ball(u, \tau)$	Dijkstra ball of u with raidus τ
gd	grid distance
C_3 (resp. C_5)	a 3×3 (resp. 5×5) sub-grid
$B(u)$	bridge edges associated with node u
δ	width of a grid cell
Δ	maximal degree of a road network
n	number of nodes in the road network

4.1 Dijkstra Adapted Approach

An algorithm, called *Basic*, can be directly adapted from *Dijkstra* search to answer a ROAM query. To explain, observe that every node u residing in a ρ-route must satisfy $dist(s, u) \leq (1+\rho)dist(s, t)$, based on the definition of the ρ-route. Similarly, $dist(t, u) \leq (1 + \rho)dist(s, t)$. Hence, all the ρ-routes are within

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

Further, given s, t and an area $circ(c, o)$, whether there exists a ρ-route connecting s and t while passing through $circ(o, r)$, is equivalent to examining the existence of a node u residing in $circ(o, r)$ such that its sum of distances from s and t is at most $(1 + \rho)dist(s, t)$. We summarize the observation as follows.

Claim 1 (Positive Condition Rule)**.** *A ROAM query can return a satisfactory ρ-route, if and only if there is a node $u \in circ(o, r)$ such that $dist(s, u) + dist(t, u) \leq (1 + \rho)dist(s, t)$.*

Based on the Positive Condition Rule, a ROAM query can be answered by performing the following two steps:

1) *Forward search:* Explore $ball\big(s, (1 + \rho)dist(s, t)\big)$;
2) *Backward checking:* Conduct Dijkstra search from node t, whenever visit a node u satisfying the positive condition rule, return $(s \rightsquigarrow u) \cup (u \rightsquigarrow t)$.

The above step (1) is due to the Equation 1, while step (2) is conducted based on the positive condition rule. As an index-free approach, *Basic* has incorporated strong prunings such as bounding search radius, and this radius bound is tight. It is therefore interesting to see further improvements under the index-free requirement, as will be discussed shortly.

4.2 Bi-directional Search with Temporary Stop

We now present a better index-free algorithm. Observe that *Basic* always fully explores the $ball(s, (1 + \rho)dist(s, t))$ in the first

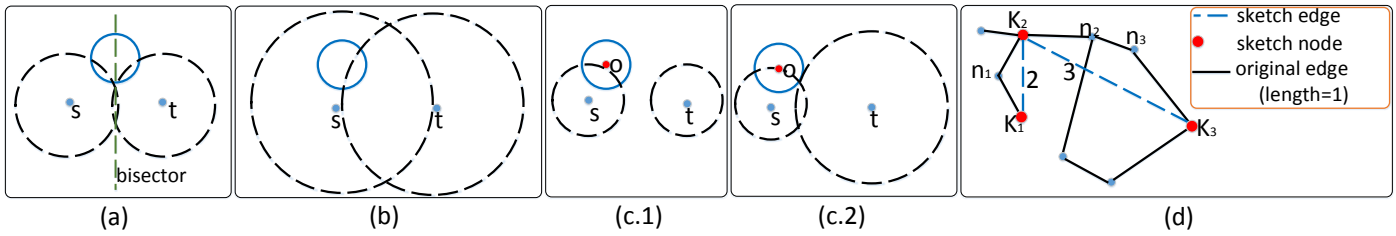


Fig. 2: (a) The cases where simple bi-directional search is favorite, as the search space (black dashed) is small. (b) The cases where simple bidirectional search is not favorite, as its search space (black dashed) is large. (c.1, c.2) Procedures of bidirectional search with temporary stop. (d) Illustration of the sketch graph.

step, resulting in a high cost if $dist(s,t)$ is large. This cost can be reduced by introducing a bi-directional search (abbr. bi-search) from s and t. The bi-search performs two Dijkstra searches, one originates from s and another from t. The two Dijkstra searches are executed concurrently in alternating lockstep fashion. The motivating case in Figure 2 (a) explains why a bi-search is helpful. In the figure, the given circle is close to the bisector of the line segment connecting s and t, where the bisector refers to the line that goes through the midpoint of the line segment connecting s and t while being perpendicular to the line segment. Now, consider that there is a bi-search that takes turns to expand the Dijkstra balls from s and t. Once there is a node $u \in circ(o,r)$ found inside the two search balls, the search will stop if $(s \rightsquigarrow u) \cup (u \rightsquigarrow t)$ is a ρ-route. Consequently, the search stops in a case where only small portions of $ball(s,(1+\rho)dist(s,t))$ and $ball(t,(1+\rho)dist(s,t))$ have been explored (as shown in black dashed circles in Figure 2 (a)), resulting in a search cost that is in general smaller than fully exploring $ball(s,(1+\rho)dist(s,t))$.

While the aforementioned bi-search is more effective than *Basic*, it is still not flexible enough to perform well in all cases and we show that it is necessary to enhance it by considering the location of $circ(o,r)$. As shown in Figure 2 (b), $circ(o,r)$ is quite close to node s. In this case, the search still incurs a large search space. To explain, in the case of Figure 2 (b) the search will not stop unless a node $u \in circ(o,r)$ inside the two search balls has been found. At the time when such a node u has been witnessed, however, the search balls are (almost) fully explored (as shown by the black dashed circles), due to the fact that $circ(o,r)$ is near to s. This again results in a high cost. To address this issue, we incorporate into the bi-search a more reasonable strategy by considering the position of circle center o. We still begin with two explorations from s and t. The difference is, a temporarily stop of the search from one side would be issued if the corresponding Dijkstra ball touches the circle center node o. In the mean while, the search from the other side goes on. The reason is that, when one-side search has visited o, it indicates that an enough search has been done for this side. We name this approach *BIS* (Bi-directional Iterative Search). A simplified procedure is shown in Figure 2 (c.1) and (c.2). At first, a ball centering at s temporarily stops its exploration when its boundary touches o. Later, the exploration from the other side touches the circle area and finds a node satisfying the positive condition rule (Claim 1).

The following Lemma 1 allows us to further improve the *temporary stop* based search. (The proofs of lemmas and theorems in this paper can be found in Section 10.) Lemma 1 points out that *BIS* can have a chance to stop when both balls from s and t have scanned o.

Lemma 1. *Denote the nearest node in $circ(o,r)$ to s (resp. t) by node u_s (resp. u_t). When the circle center o is scanned in both searches from s and t, the search can stop if*

$$\min\{dist(t,o)+dist(s,u_s), dist(s,o)+dist(t,u_t)\} > (1+\rho)dist(s,t).$$

Resumption of search. The conjunction of two temporary stops may cause the resumption of searches, so as to avoid skipping some candidate routes. Unless Lemma 1 is able to be applied, there is no evidence that all the candidate routes have been explored when the searches are suspended due to temporary stops. To handle this, we apply resume-checking for those stopped searches. The pesudocode of the *BIS* algorithm can be referred to Algorithms 6, 7 in Section 10.

BIS outperforms *Basic* significantly, making it particularly interesting for applications that are with stringent physical memory constraints.

5 The Sketch Framework

Previously, we have introduced index-free algorithms for ROAM queries. As we mentioned, those algorithms are useful when memory is precious. In this section, we focus on index-based approaches, which are even more efficient than *BIS*, and would be useful when the memory constraint is not the first consideration. We present a general framework to facilitate the index-based ROAM query, by looking at handling ROAM queries in a different angle: can we project the query into a smaller graph, where query processing becomes simpler? To this end, two questions have to be answered: 1) how to extract such a smaller graph? 2) how to project the query? These questions will be answered in the following sections. In particular, we design a *sketch graph* (presented in this section) and propose the *Sketch Search* (presented in Section 6) for handling a ROAM query based on the *sketch graph*.

5.1 Basic Idea of the Sketch Graph

The *sketch graph* aims to transfer the query processing to the *sketch graph* dimension, for better efficiency. The high-level idea resembles our behavior in looking up an online road network: we often first browse an abstract level of the road network (a road network sketch) to locate the most interesting area, then examine the details of the interesting area by adjusting the resolution of the network. Let us denote the *sketch graph* by $\mathcal{S}$. The goal is to preserve such a property: given a value $\rho > 0$, if there is a ρ-route intersecting a given area $circ(o,r)$ in the original network $\mathcal{G}$, then there must be a ρ-route intersecting a relevant area $circ(o,r')$ in $\mathcal{S}$. (How much larger r' compared to r depends on the value

of r. The precise relationship between r' and r will be given in Theorem 1.) This property, interestingly, enables a projection between $\mathcal{S}$ and $\mathcal{G}$ in ROAM query processing, as we will see shortly in Section 6. Formally, we define a *sketch graph* of the road network graph $\mathcal{G}$ as follows.

Definition 3 (Sketch Graph)**.** *Given a graph $\mathcal{G}(V, E)$, another graph $\mathcal{S}(V', E')$ is a Sketch Graph of $\mathcal{G}$, if $V' \subset V$, and $\forall u, v \in V'$, the shortest distance between u and v in $\mathcal{S}$ is equal to their shortest distance in $\mathcal{G}$. We refer to the nodes in V' as Sketch Nodes, and the edges in E' Sketch Edges.*

This definition implies a distance invariant property, which is crucial to the correctness of our querying algorithm in Section 6. An example of *sketch graph* is shown in Figure 2 (d). The bigger red nodes are sketch nodes, while the dashed lines represent the sketch edges. For example, edge (K_1, K_2) shortcuts path (K_1, n_1, K_2) and is therefore associated with a weight 2. (Suppose each edge in the original graph has a weight of 1.)

Next, we will present a detailed algorithm to construct a *Sketch Graph* conforming to the definition.

5.2 Sketch Construction in Preparation

In this section, we introduce some concepts that are necessary to understand our *sketch graph* construction algorithm.

As argued in [13], [15], [26], one important property of the road network graph is: given a spatial range, say a square, the relatively longer shortest paths in the range can be *covered* by a small set of nodes, where a node set *covers* a shortest path implies that the path passes through at least one of the nodes in the set. Especially, the authors of [13], [15] show that, if one imposes the road network with an $M \times M$ grid ($\forall M \geq 5$), then the set of shortest paths starting from certain grid cell C_1 and ending at a grid cell C_2 that is at least 2 cells apart (horizontally or vertically) from C_1, can be covered by a small number of nodes. For example, Figure 3 (a) shows the shortest paths $n_1 \rightsquigarrow n_2$ and $n_1 \rightsquigarrow n_3$ that are covered by node set $\{n_4\}$. We summarize this observation by a novel notion called *cover dimension* ϕ that is similar in spirit to the arterial dimension in [15]. We alternatively propose the cover dimension because it is more suitable for the analysis of our algorithms. The following are the formal definitions.

Definition 4 (Grid Distance)**.** *Suppose we impose the road network with an $M \times M$ grid. Given a node u located at i_1^{th} row and j_1^{th} column, and another node v located at i_2^{th}-row and j_2^{th} column. Then, the grid distance between u and v, $gd(u, v)$, is* $1 + max(|i_1 - i_2|, |j_1 - j_2|)$.

In Figure 3 (a), $gd(n_1, n_2) = 4$ because they are gapped by 4 cells (containing n_1 and n_2) horizontally.

Definition 5 (Inner Shell and Outer Shell [13])**.** *For any grid cell C, the 3×3 (resp. 5×5) rectangle centered at C is called the inner shell (resp. outer shell) of grid cell C. For any node n_0 inside cell C, its inner shell (resp. outer shell) is the inner shell (resp. outer shell) of C.*

For example, in Figure 3 (a), the dashed 3×3 rectangle containing cell C_1 is the inner shell of C_1, and it is also the inner shell of node n_1. The 5×5 rectangle centering at cell C_1 is the outer shell of cell C_1, and it is also the outer shell of node n_1. In the following, we will use C_3 to denote an inner shell, while using C_5 to denote an outer shell.

Definition 6 ($\mathcal{G}_i$ subgraph)**.** *The $\mathcal{G}_i$ subgraph of node u, denoted by $\mathcal{G}_i(u)$, is the subgraph whose edges have at least one end node in the $i \times i$ sub-grid centered at the cell containing node u.*

Definition 7 (Cover Path)**.** *Given a 5×5 region centered at cell C, we call a path a cover path if it is the shortest of the paths satisfying 1) the path starts from inside the cell C and reaches the outside of the outer shell of cell C; 2) the path has all nodes (except the last one) locate within the 5×5 region.*

An example is shown in Figure 3 (a), the shortest path from node n_1 to node n_2, which is colored green, starts inside of cell C_1 (the cell with blue boundary) and goes outside of the outer shell of C_1. Then, the shortest path is a cover path of cell C_1.

Definition 8 (Transit Node and Transit Edge [13])**.** *Given one edge, denoted as (n_0, n_0'), cutting across the boundary of the inner shell of C such that n_0 is inside of the inner shell, if edge (n_0, n_0') exists in any cover path of cell C, then node n_0 is called a transit node of cell C. (n_0, n_0') is called a transit edge.*

As shown in Figure 3 (a), the node n_4 is a transit node of cell C_1 since edge (n_4, n_5) cuts the boundary of the inner shell of C_1 and the edge lies in the cover path from node n_1 to node n_2. (n_4, n_5) is the corresponding transit edge. Node n_8 is not a transit node as the path $n_1 \rightsquigarrow n_9$ cannot reach outside of the outer shell. Next, we introduce the concept of *cover dimension*.

Definition 9 (Cover Dimension)**.** *Suppose that we impose a $M \times M$ grid on a road network, where $M \geq 5$. Let ϕ_C be the number of transit edges of a grid cell C. Then, the cover dimension of road network $\mathcal{G}$ with respect to the $M \times M$ grid is $\phi_M = \max\{\phi_C | C \text{ is a cell}\}$. In addition, $\phi = \max\{\phi_M | M \geq 5\}$ is called the cover dimension of the road network.*

Take $M = 5$ as an example. There are 25 cells in the grid and therefore 25 corresponding numbers of transit edges. Then, the maximal of the 25 numbers is the value of ϕ_5. The cover dimension ϕ is the maximal value after examining every $M \geq 5$.

We adopt the assumption similar to [15] as follows, and this assumption has been justified by experiments (see experiments in Section 7.5).

Assumption 1. *The cover dimension of a road network graph $\mathcal{G}$, ϕ, is a small constant.*

5.3 The Construction of Sketch Graph

With the aforementioned definitions and assumptions, in this section, we present the construction of the sketch graph. We respectively discuss the selection of sketch nodes and creation of sketch edges.

As hinted in [13], transit nodes are structurally important. Therefore, the transit nodes are reasonably included as a subset of sketch nodes. There are a few other nodes selected to be sketch nodes as well. These additional nodes are important to guarantee that the constructed sketch graph conforms to Definition 3, as can be observed in our proof of Lemma 3. To be more specific, the following strategy is employed:

Strategy 1 (Sketch Node Selection)**.** *1) For a 5×5 sub-grid whose center cell is C, the transit nodes with respect to the sub-grid are selected as sketch nodes. 2) If one end node of a transit edge locates within the cell C, then both end nodes of the transit edge are selected to be sketch nodes.*

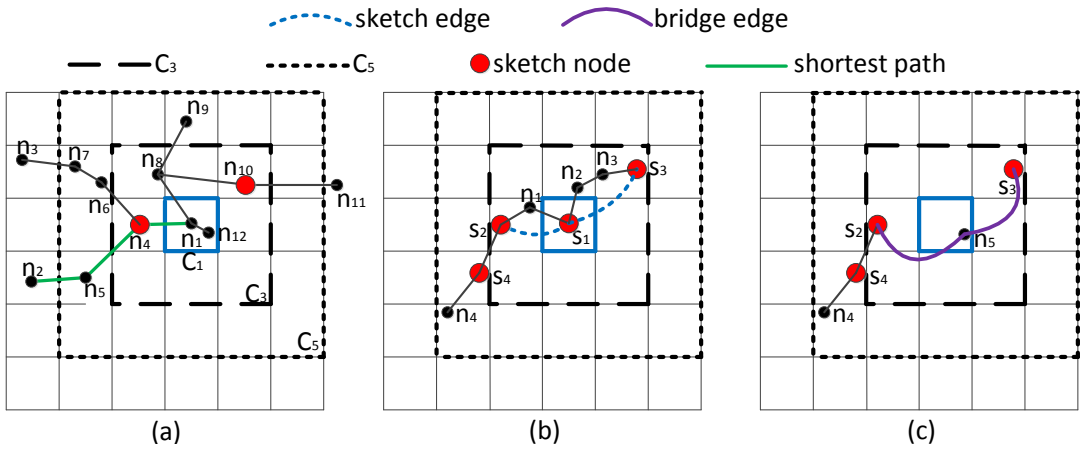


Fig. 3: (a) Examples of sketch graph, regional graph, cover paths. (b) Illustrating sketch edges. (c) Illustrating bridge edges.

A question followed by employing above selection strategy is how to extract sketch nodes (in fact, mostly are transit nodes). Let us focus on a specific 5×5 sub-grid. Based on transit node's definition (Definition 8), a straightforward method is to conduct Dijkstra searches from every node p in the central cell to explore the cover paths sourced at p, and pinpoint the transit nodes in those cover paths. This method, however, incurs as many runs of Dijkstra searches as the number of nodes in the central cell, which is often prohibitive. A better approach is to only conduct Dijkstra searches from the nodes with an outgoing edge cutting the boundary of the central cell. The rationale behind is that any cover path of the central cell cuts its boundary. In particular, every cover path must contain a subpath that starts with an edge cutting the boundary of the central cell C, and ends at an edge cutting the boundary of the outer shell[1]. For example, in Figure 3 (a), path $\{n_1, n_4, n_5, n_2\}$ is a such subpath of the cover path $\{n_{12}, n_1, n_4, n_5, n_2\}$. Hence, we only need to enumerate all such subpaths, as illustrated by Algorithm 1.

Algorithm 1: `SketchNodeSelect(`$\mathcal{G}$`)`

```
1  V' = ∅
2  for each 5 × 5 sub-grid C5 do
3      Extract the subgraph G5 corresponding to C5;
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 G5;
6          for each node q outside C5 that is visited by the
             Dijkstra search do
7              Backtrack along the shortest path from node q to
                 node p and locate the transit edges;
8              for each transit edge (u, v) do
9                  if u (resp. v) is in C3 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 of sketch node selection. Call back to Figure 3 (a). The figure shows a regional graph (corresponding to $\mathcal{G}_5$ in Algorithm 1). Edges (n_1, n_4) and (n_1, n_8) cut across the boundary of cell C. The cover paths from n_1 include $\{n_1, \mathbf{n_4}, n_5, n_2\}$, $\{n_1, \mathbf{n_4}, n_6, n_7, n_3\}$ and $\{n_1, n_8, \mathbf{n_{10}}, n_{11}\}$. The symbols in bold (i.e., n_4 and n_{10}) represent transit nodes, which have an outgoing edge cutting the boundary of the inner shell.

Now, it remains to construct the sketch edges. Recall that the *sketch graph* is required to maintain the invariance of the shortest distance between two sketch nodes (Definition 3). This motivates us to do the following: we create sketch edges from a sketch node to its nearest (measured in $dist(\cdot)$) sketch nodes. Specifically, we employ the following strategy and a pseudo-code is shown in Algorithm 2.

Strategy 2 (Regional Sketch Edge Selection)**.** *For a sketch node u which is located in cell C, link u to its nearest sketch nodes within $\mathcal{G}_5(u)$.*

Algorithm 2: `SketchEdgeCreate(`$\mathcal{G}$`)`

```
1  E' = ∅
2  for each 5 × 5 sub-grid C5 do
3      Extract the subgraph G5 corresponding to C5;
4      for each Sketch node u* in the central cell do
5          Conduct Dijkstra search from u* in G5;
6          for each leaf node q of the Dijkstra's search tree do
7              Backtrack from node q to node u* along the
                 shortest path between them, and locate the u*'s
                 nearest Sketch node v* in the path;
8              E' = E' ∪ {(u*, v*)};
9  return E'
```

Example procedure of sketch edge selection. Algorithm 2 shows the procedure of sketch edge creation. As shown in Figure 3 (b), S_1 is a sketch node inside the central cell. Then, conduct *Dijkstra* search from S_1 (Algorithm 2 line 5). For each leaf node (n_4 and S_3), backtrack the path to the source node S_1. The backtracked paths are $\{n_4, S_4, \mathbf{S_2}, n_1, S_1\}$ and $\{\mathbf{S_3}, n_3, n_2, S_1\}$. The bold ones are sketch nodes nearest to S_1 along the paths. Hence, we link two sketch edges from S_1 to its nearest sketch nodes, namely S_2 and S_3 (Algorithm 2 lines 7-8).

The complexities of the algorithm are shown by the following lemma.

Lemma 2. *The construction time of the sketch graph is $O(\Delta \max\{\phi n \log n,\ n \cdot \max_C\{b_C\} \cdot \max_C\{log(n_C)\}\})$, where n is the number of nodes in $\mathcal{G}$, Δ is the maximal degree of the road network graph, b_C is the number of edges cutting the boundary*

[1]If there exist multiple cuts of the boundary, then we select the subpath with the shortest length.

of cell C and n_C is the number of the nodes within cell C. In addition, the storage cost of the sketch is $O(min\{M^2\phi^2, n\phi\})$.

Remarks. In real road networks, Δ is often a small constant. (Recall that it represents the number of roads starting from a junction.) Also practically, $b_C < \frac{n}{M^2}$. (Note that the road network is divided into M^2 cells and b_C only counts the nodes close to the cell borders.) Therefore, the construction is practically efficient. Meanwhile the storage is approximately linear, which is close to optimal.

The following lemma gives that the constructed graph is indeed a graph conforming to Definition 3.

Lemma 3. *The constructed graph by Strategy 1 and Strategy 2 is a sketch graph conforming to Definition 3.*

6 The Sketch Search

To leverage the *sketch graph* for a ROAM query, the basic idea is to transfer querying workloads from the original graph to the *sketch graph*. One issue is that, in ROAM query nodes s and t may not be sketch nodes. We thus need to relate the non-sketch nodes to sketch nodes. To this end, we introduce a set of *bridge* edges, $B(u)$, associated with each non-sketch node u. Intuitively, we require the bridge edges in $B(u)$ to connect node u to its neighboring sketch nodes (Definition 10). The following Strategy 3 explains the bridge edge creation steps.

Definition 10 (Neighboring sketch nodes)**.** *A sketch node v is a neighboring sketch node of node u, if all the intermediate nodes inside the shortest path from u to v are not sketch nodes.*

Strategy 3 (Bridge edge creation)**.** *For each non-sketch node u which is located in cell C, we create bridge edges from node u to its neighboring sketch nodes. The weight of the bridge edge is the shortest distance between its two end nodes.*

Example procedure of bridge edge construction. In Figure 3 (c), the bridge edges connect n_5 to the transit nodes (namely, S_2, S_3 in the example), forming the set of bridge edges $B(n_5) = \{(n_5, S_2), (n_5, S_3)\}$.

A problem followed is to construct bridge edges efficiently. A naive approach is to run *Dijkstra* searches from each non-sketch nodes to search their neighboring sketch nodes. However, this approach runs n rounds of *Dijkstra* searches, which is rather costly. A better approach is to run *Dijkstra* search reversely from each of the transit nodes within the regional graphs. Since the number of transit nodes is at most ϕ in each 5×5 region by definition, we only need to run ϕ rounds of *Dijkstra* searches within each $\mathcal{G}_5$ subgraph corresponding to the 5×5 region. Interestingly, the total cost is only around pseudo-linear to the number of nodes in $\mathcal{G}$, as shown in the following lemma.

Lemma 4 (Bridge edge construction/storage)**.** *The time complexity of bridge edge creation is $O(\Delta\phi n \log n)$ and the space complexity of bridge edges is $O((\phi + \Delta)n)$, where Δ is the maximal degree of $\mathcal{G}$.*

With the bridge edges of node s and node t, we introduce an augmented *sketch graph*, called (s,t)*-sketch graph* (abbreviated as (s,t)-sketch) to handle ROAM query with respect to node pair (s,t). The (s,t)-sketch, composed of the *sketch graph*, $B(s)$ and $B(t)$, gives us an interesting projection theorem that facilitates efficient ROAM queries.

Theorem 1 (Sketch Projection Theorem)**.** *If $gd(o,s) \geq 5 + \lceil r/\delta \rceil$ and $gd(o,t) \geq 5 + \lceil r/\delta \rceil$, then ROAM$(s,t,o,r+2^{1.5}\delta,\rho)$ returning false in (s,t)-sketch implies that ROAM(s,t,o,r,ρ) returns false in $\mathcal{G}$, where δ is the width of a grid cell, gd denotes the grid distance and r is the radius of the given circular area.*

The theorem allows to employ the *sketch pruning strategy*: the search is first performed in the (s,t)-sketch, if ROAM$(s,t,o,r+2^{1.5}\delta,\rho)$ in the sketch returns $false$, then the original ROAM query must return $false$. Given that the (s,t)-sketch has a much smaller size compared with the original network $\mathcal{G}$, the pruning can be conducted efficiently. In addition, the most interesting part of the theorem is that, the pruning is *algorithm independent*. In other words, any possible ROAM query algorithm $\mathcal{M}$ can be performed in *sketch graph* to prune. This forms a general sketch search framework described in Algorithm 3.

Algorithm 3: `SketchSearch`$(\mathcal{G}, s, t, o, r, \rho)$

```
1 exist ← true;
2 get (s,t)-sketch from index;
3 if gd(o,s) ≥ 5 + ⌈r/δ⌉ and gd(o,t) ≥ 5 + ⌈r/δ⌉ then
4     exist ← M(s,t,o,r + 2^1.5 δ,ρ) in (s,t)-sketch; /* M
        is any approach to process ROAM
        query. */
5 if exist = false then
6     return false ;
7 else
8     return M(s,t,o,r,ρ) on G;
```

6.1 Optimization

This section presents an enhanced ROAM query method $\mathcal{M}$, with the help of the goal directed traversal. This method is not an index-free method, as it requires pre-computation of the distances between a selected set of sketch nodes. The structurally important sketch nodes allow good distance estimations between two nodes. With this observation, we pre-compute the shortest distances between a set of selected sketch nodes $(l_1, \ldots, l_k)$ and all the other nodes. By the triangle inequality, we have $dist(p,q) \geq |dist(l_i,p) - dist(l_i,q)|$. This gives us

$$dist(p,q) \geq \max_{1 \leq i \leq k} \{|dist(l_i,p) - dist(l_i,q)|\}, \qquad (2)$$

The above lower bound distance estimation (Equation 2) further allows us to perform a *goal directed* search (like $A^\star$ search [8]). In what follows, we first revisit $A^\star$ search, and then point out our non-trivial results in handling ROAM query (Lemmas 5 and 6).

In contrast to the classical *Dijkstra* search, $A^\star$ search advocates that the search should be influenced by the location of t. This strategy is referred to as the *goal directed search*, where the goal refers to the target node t. Specifically the *Dijkstra* search explores the node in an ascending order of the shortest distance from the source node s. In contrast, $A^\star$ search always explores the node u with the current smallest value of $dis[u] + lower\_dist[u]$, where $dis[u]$ is the currently found distance of node u from the source node s, and $lower\_dist(u,t)$ is a lower bound distance between node u and the target node t. As such, the search towards the opposite directions (i.e., deviating faraway from t) is reduced considerably. The search is stopped when t is scanned by the search. A useful property is that, the only difference between *Dijkstra* search and $A^\star$ search is the procedure of the edge relaxation (shown in Algorithm 4). Compared with the edge

relaxation of *Dijkstra* search, the differences lie in Lines 3 and 8, where the value of $dis[v] + lower\_dist[v]$ is set as the priority value for the PriorityQueue Q.

Algorithm 4: `Relax(u,v)`

```
1 if state[v] = unseen then
2     dis[v] ← dis[u] + w(u,v);
3     Q.push(v, dis[v] + lower_dist(v,t));
4     state[v] ← labeled;
5 if state[v] = labeled then
6     if dis[v] < dis[u] + w(u,v) then
7         dis[v] ← dis[u] + w(u,v);
8         Q.decreaseKey(v, dis[v] + lower_dist(v,t)));
```

If we extend the aforementioned A* searching algorithm for ROAM query, the search stopping criterion will be too conservative. We give the following new stop criterion for handling ROAM query using A* search.

Lemma 5 (Stop rule)**.** *When the visiting node u has $dis[u] + lower\_dist(u,t) > (1+\rho)dist(s,t)$ where $dis[u]$ is the distance associated with u during the search, then each node of any ρ-route has been scanned.*

We also show that the positive condition rule (Claim 1) is still applicable here, by the following Lemma.

Lemma 6. *Positive condition rule (Claim 1) holds in this goal directed search, if the employed estimation method $lower\_dist$ is monotone, i.e., $\forall$ edge $(u,v) \in E$, $lower\_dist(u,t) \leq w(u,v) + lower\_dist(v,t)$. Further, the $lower\_dist$ computed by Equation 2 is monotone.*

In order to apply the positive condition rule (Claim 1) indicated by Lemma 6, we employ a consecutive two runs of the goal directed searches respectively from s and t. As shown in Algorithm 5, a goal directed search first starts from s, with a stopping condition in Lemma 5 applied. After that, a checking from t is initiated, with the aim of extracting a ρ-route with the positive condition rule in Lemma 6.

6.2 Extensions and Discussions

Directed road networks. There are two directed edges (u,v) and (v,u) in a directed road network, such that the weights of (u,v) and (v,u) can be different. The adaptation relies on two kinds of *Dijkstra*'s searches, i.e., *forward Dijkstra*'s search and *backward Dijkstra*'s search. The forward *Dijkstra*'s search only explores each edge (u,v) when node u is visited, while the backward *Dijkstra*'s search only explores each edge (v,u) when node u is visited. Based on these two concepts, we conveniently adapt the ROAM query algorithms to directed road networks. For instance, in *Basic*, we use forward *Dijkstra*'s search for node s and backward *Dijkstra*'s search for node t. Then, during the step of *Checking from* t, for each visited node u, $d(s,u) + d(u,t)$ (instead of $d(s,u) + d(t,u)$) is computed. Following this idea, we further demonstrate the adaptation needed for the construction of $\mathcal{S}$. Similarly, for the *Sketch* based method, we create both *forward Sketch edges* and *backward Sketch edges*, which are respectively created by conducting forward *Dijkstra*'s search and backward *Dijkstra*'s search. As such, all the lemmas still hold, with slight modification to involve the backward *Sketch* edges.

Algorithm 5: `GoalDirectedSearch` $(\mathcal{G}, s, t, o, r, \rho)$

```
1  dis_s[s] ← 0, dis_t[t] ← 0;
2  stdis = ∞;
3  while some node has not been visited from the search of s do
4      compute lower_dist(u,t) by Equation 2;
5      u ← argmin_u{dis_s[u] + lower_dist(u,t)};
6      if u = t then
7          stdis ← dis_s[u];
8      if dis_s[u] + lower_dist(u,t) > (1+ρ)stdis then
9          Break;
10     for neighbor v of u do
11         Relax (u,v) by Algorithm 4;
12 while some node has not been visited from the search of t do
13     compute lower_dist(u,s) by Equation 2;
14     u ← argmin_u{dis_t[u] + lower_dist(u,s)};
15     if dis_s[u] + dis_t[u] ≤ (1+ρ)stdis then
16         return (s ⇝ u) ∪ (u ⇝ t);
17     if dis_t[u] + lower_dist(u,s) > (1+ρ)stdis then
18         return false;
19     for neighbor v of u do
20         Relax (u,v) by Algorithm 4;
21 return false;
```

Non-circular areas. For a non-circular area $\mathcal{A}$, the *sketch graph* based pruning can still be used, with slight modification of Theorm 1. In particular, we can find a circle that covers $\mathcal{A}$, 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 Theorm 1 can still be applied.

Dynamic road networks. A dynamic road network refers to a road network whose edge weights are gradually changing. When the weight of an edge (u,v) changes, the *sketch graph* can be updated *locally* and thus efficiently. In fact, only a small number of C_5 sub-grids, which contain u or v, would be affected. In particular, there are at most 50 such sub-grids as u (or v) is contained by at most $5 \times 5 = 25$ sub-grids. Thus, we can simply recompute the sketch edges originated from these sub-grids for an edge weight update.

7 Experiments

In this section, we demonstrate a systematic experimental study on the proposed methods, in terms of efficiency (Section 7.2), index cost (Section 7.3) and effectiveness (Section 7.4).

TABLE 2: Datasets

ID	Road network	#Edges	#Nodes	Query set
D1	New York (NY)	733,846	264,346	Green cab
D2	New York (NY)	733,846	264,346	Yellow cab
D3	Florida (FLA)	2,712,798	1,070,376	POI sets
D4	Northeast (NE)	3,897,636	1,524,453	Random
D5	Great Lakes (LK)	6,885,658	2,758,119	Random
D6	Western (WU)	15,248,146	6,262,104	Random
D7	Full USA (FU)	58,333,344	23,947,347	Random

7.1 Setup

The evaluation is conducted on 7 real and public datasets (Table 2, downloaded from website [2]). The query set for D1 (resp. D2) uses the green (resp. yellow) cab trajectory record in Jan 2015 [3]. Green cabs have restricted pick-up areas compared with Yellow

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

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

cabs, and therefore green caps and yellow cabs are divided into two datasets. The (pick-up, drop-off) locations are set to be the (s, t) pairs, and the areas are generated uniformly. For D3, 60 POIs are extracted from the Florida road network. The origin s, destination t and the area center o are randomly selected from the POI set. Datasets D4 to D7 are used to test the scalability of the approaches. To our knowledge, D7 (full USA) is one of the biggest road networks published. Experiments are conducted on an Intel i7-4870HQ 2.5GHz CPU with 16GB RAM running Mac OS X (10.10.4). The three methods proposed in this paper, namely *Basic*, *BIS*, and *Sketch*, are compared, in terms of the efficiency, index sizes and indexing time. The value of ρ is selected within a range of $[0.1 - 0.9]$, which means that a ρ-route does not stretch up to double of the exact shortest path. Unless otherwise specified, the running time reported is the average of running times of 400 queries in each setting; the grid resolution for *Sketch* is 100×100; the deviation ρ is set to 0.5.

7.2 Efficiency

Figures 4∼5 show the efficiency comparisons among the 3 approaches with various parameter settings.

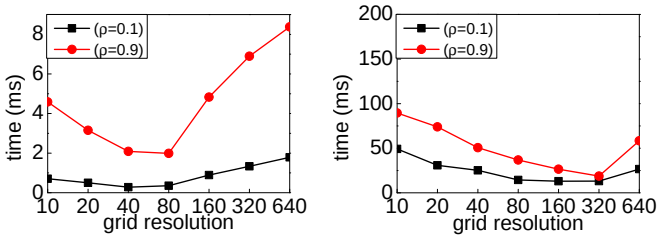
Sketch consistently performs the best in efficiency, followed by *BIS*, and then *Basic*. For D1 and D2, in particular, *Sketch* is up to 30 times faster than *Basic* (See Figures 4(a.1-a.3) and (b.1-b.3)). The superior performance of *Sketch* is two-fold. First, it owes to the efficient query processing on the smaller *sketch graph*. As shown in Table 3, the size of *sketch graph* is much smaller compared with the original network, and therefore processing ROAM query on *sketch graph* is much faster. Second, the goal directed search takes effect, and often this leads to a smaller search space compared with *Basic*.

Interestingly, the performance improvement of *Sketch* is relatively stable, regardless of what parameters are set. As shown in Figure 4, changes of ρ only slightly affect the running time of *Sketch*. In contrast, the performances of *Basic* and *BIS* are more sensitive to ρ. This owes to the fact that *Sketch* has a smaller search space than *Basic* and *BIS*, as the search is goal directed. Such an optimized search tends to be less sensitive to the changes of ρ.

To study the effect of area sizes, we perform queries with area radius in $[0, 50L]$, where L is the average edge length. In *Basic* and *BIS*, the search periphery touches the area earlier if the area is larger, and this typically means earlier stops of searches. As a result, the running times drop slightly with the increase the area sizes (see Figure 4 (a.4) (b.4) and (c.4)). Such changes have slight effects on *Sketch* since it is already largely optimized.

Furthermore, *Sketch* is scalable to large road networks and consistently outperforms the competitors by around one order of magnitude (see Figure 5).

Grid resolution study. How to determine a reasonable grid resolution is an interesting question. Here we give an experimental analysis. Figure 6 (a) shows the results of our tests on dataset D2, for which we vary the resolution from 10×10 to 640×640. The results demonstrate that, for D2, the resolution 80×80 (corresponding to the valley of the curve) achieves the best performance. Similar results are observed on dataset D3, as shown in Figure 6 (b). Such "valley curves" are formed for the following reasons: When the grid is finer, more nodes will be selected as *Sketch* nodes, and therefore the size of the *sketch graph* $\mathcal{S}$ will become larger. As a result, there are two contradictory effects: 1) *performance worsen:* executing queries on $\mathcal{S}$ becomes slower since the size of $\mathcal{S}$ is larger; 2) *performance enhanced:* more queries will satisfy the pruning condition in Theorem 1 (line 3 in Algorithm 3), and those queries enjoy faster processing. When the grid resolutions are smaller (i.e., the grids are coarser), the latter effect dominates, and the performance improves; As the resolution increases, the former effect starts to dominate, and the performance becomes worse. In practice, since the road network structure and historical query sets are typically available, and therefore this data can be used to study the "valley curve" and determine a reasonable resolution.



(a) D2 (b) D3

Fig. 6: (Performance V.S. grid resolution) Performance first improved and then degraded with the increase of the grid resolution.

7.3 Index Cost

Table 3 shows the index size and indexing time for *Sketch*. The grid resolution is set to 100×100. Naturally, the construction time increases with the size of road networks. For moderate sized datasets (up to millions of edges) such as NY, FLA, NE and LKS, the index construction is finished within a few minutes. For larger datasets (up to tens of millions 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. The sizes range from 3.1 to 12.3 times that of input size (i.e., original graph size). This agrees with our theory that the space cost is linear to the input size (see Lemma 2). Table 4 shows the index construction time and space cost when using different grid resolutions. The size of the *sketch graph* increases when the grid resolution becomes higher (from 20×20 to 100×100). The reason is, when the grid is finer, the more sketch nodes are selected and hence the larger of the *sketch graph* size. The number of the bridge edges is not sensitive to the changes of the grid size.

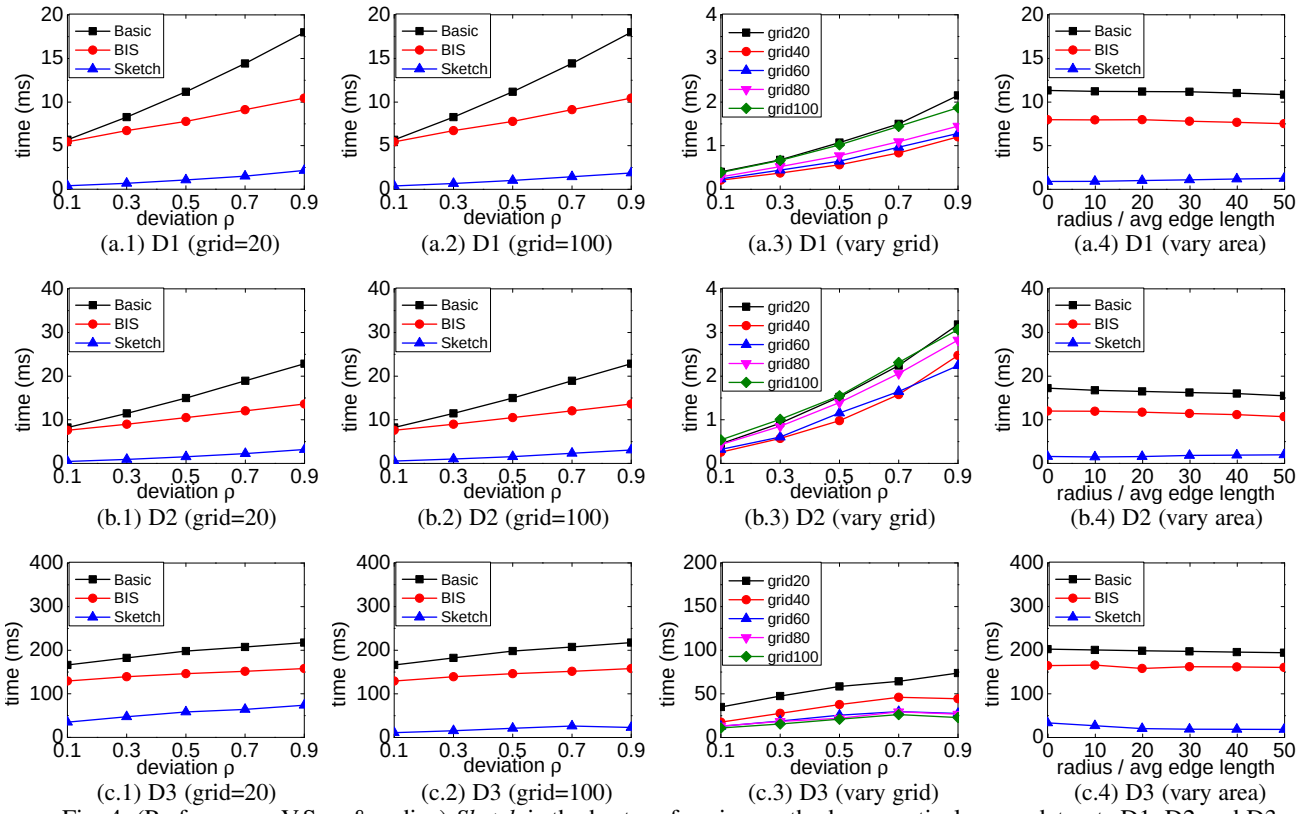
TABLE 3: Index size and indexing time (M=10^3KB, G=10^3MB).

Dataset	Input Graph Size	Sketch		
		Time	Sketch	Bridge-edges / $\lvert E \rvert$
NY (D1, D2)	22.1M	1.9mins	4.2M	6.7
FLA (D3)	86.6M	8.1mins	1.3M	3.1
NE (D4)	126.9M	8.3mins	3.5M	8.6
LKS (D5)	232.2M	13.0mins	3.1M	9.4
WU (D6)	534.6M	55.0mins	7.3M	12.3
FU (D7)	2.1G	4.5hours	4.2M	6.1

TABLE 4: Index size and indexing time w.r.t. grid sizes.

Dataset	Grid	Time	Sketch	Bridge-edges/$\lvert E \rvert$
FLA (D3)	20×20	596s	0.1M	3.3
	40×40	563s	0.3M	3.3
	60×60	549s	0.6M	3.3
	80×80	505s	0.9M	3.2
	100×100	487s	1.3M	3.1

Sketch graph update. As discussed in Section 6.2, at most 50 sub-grids (of size 5×5) need to be updated when an edge

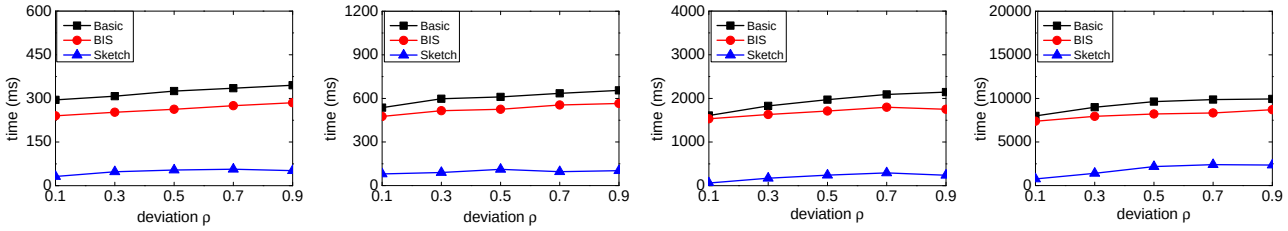


(a.1) D1 (grid=20) (a.2) D1 (grid=100) (a.3) D1 (vary grid) (a.4) D1 (vary area)

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

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

Fig. 4: (Performance V.S. ρ & radius) *Sketch* is the best performing method on practical query datasets D1, D2 and D3.



(a) D4 (b) D5 (c) D6 (d) D7

Fig. 5: (Performance V.S. ρ) *Sketch* performs the best on large road networks (D4-D7).

changes its weight. The can be done very efficiently. For moderate sized city-scale road network like NY, it only takes expected 0.6 seconds to conduct an update; for road network for large cities such as FLA, it takes around 2.4 seconds for an update. Considering that the changes of road networks traffics are typically not instantaneous, the update efficiency well suits practical needs.

7.4 Effectiveness

As we mentioned, using an area to model a service is necessary in many routing applications. An experiment is further conducted to show that modeling services using areas makes a difference. To explain, we conduct experiments to simulate a scenario of selecting one moving object to handle the service: there are c moving objects (e.g., taxis) with known destinations, and 100 servicing areas (e.g., 100 querying passengers) are distributed on the road network. For an area, it is regarded as *served* if there is at least one objects that can follow a ρ-route passing through its servicing area (e.g., at least one taxis can serve the passenger with a small detour cost). The moving objects ((s,t) pairs) are drawn from dataset D1. ρ is in [0, 0.2] and the 100 moving objects have identical ρ values. If there are p areas that can be served, we say the servicing rate is $p\%$. We compare the serving rate of area-modeling and point-modeling. The results for $c = 500$ and $c = 5000$ are shown in Figure 7 (a) and (b). The impact of area radius is significant on servicing rate (note that radius=0 indicates the service is modeled as an exact point). When ρ is close to 0, the servicing rate increases from 3% (radius= 0) to 21% (radius/unit-radius= 4) for the case of 500 objects, and increases from 12% to 53% for the case of 5000 objects. The result explains there can be deficiencies of the point-modeling for a service in routing applications: when a service in practice is an area, simplifying it to an exact point often reduces its possibility to be served, in a routing application where detour cost is a major concern.

7.5 Experiments on Cover Dimension

We experimentally estimate the cover dimension. We impose a grid with resolution $2^{r+2} \times 2^{r+2}$, where r ranges in $[1, 10]$. That is, the grid resolution ranges from 8×8 to 4096×4096. With each grid imposed, we count the number of transit edges within every 5×5 sub-grid. We figure out the maximal number and the average number of transit edges for all our test datasets. The results are shown in Figure 8. The x-axis is the resolution parameter r and "MAX" (resp. "AVG") is the maximal (resp. average) number of transit edges within each 5×5 sub-grid. All the results show that

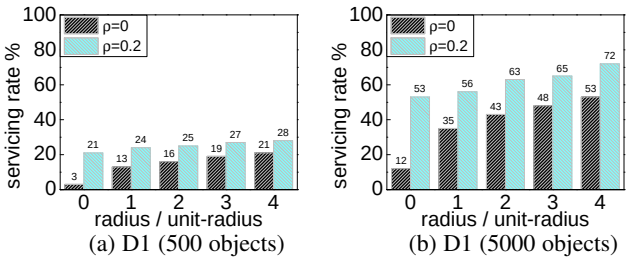


Fig. 7: (Servicing rate V.S. radius) Larger areas bring in higher servicing rates. One unit-radius $= 5L$.

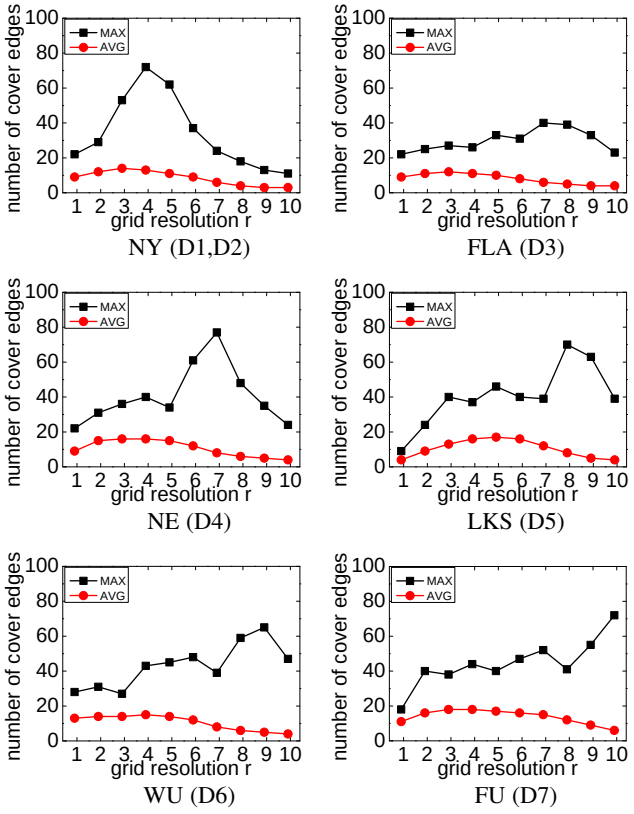


Fig. 8: (Transit edges V.S. grid resolution $2^{r+1} \times 2^{r+1}$) Illustrating small cover dimension.

in practical settings, the number of transit edges is bounded by a relatively small constant.

8 Related Work

Related algorithms. The shortest path query [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19] is related to the ROAM query, because the ρ-route is an approximate shortest path. However, extending these solutions to handle ROAM queries is not straightforward, due to the presence of ρ and area. For example, the algorithms [10], [11], [12] exploit a spatial coherent property that, two shortest paths with close origins and destinations typically share many edges. However, this spatial coherent property is invalid for the ρ-routes, because there can be two ρ-routes with close origins and destinations, but do not share any intermediate nodes. Our algorithms are also related to the hierarchical shortest path algorithms [13]. Particularly, *sketch graph* is built upon *transit nodes* [13]. Nevertheless, the key idea of the *Sketch Graph* is significantly different from that of [13]. The *Sketch Graph* connects the transit nodes which are closely located, rather than storing (almost) pairwise distances between transit nodes as in [13]. Using *Sketch Graphs* thus significantly reduces the space cost. Meanwhile, leveraging the *sketch graph* to answer a ROAM query is a unique problem of this paper and therefore new.

Related applications. Taxi-sharing systems have been studied recently [20], [22], [23], [24]. Some of them consider a detour constraint similar to the ROAM query within a shared ride, but simply employ less efficient shortest path based algorithms for detour constraint checking, compared with the *Sketch* method (referred to our additional experiments in Section 10). While in these systems more constraints such as time threshold may be considered for route selection, the constraint for the detour cost (as in ROAM query) is typically of a higher priority and acts as an independent pruning for routes. This proves the fundamentality of ROAM queries and the significance of evaluating them efficiently.

Existing spatial crowdsourcing studies [3], [4], [5], [6], [27], [28] often compute the spatial distance based on Euclidean distance. Euclidean distance, as pointed out in [10], is often inaccurate, since an object's movement is constrained by a road network. Hence, the aforementioned existing algorithms are not applicable to answer a ROAM query.

Besides, the existing studies for above applications rarely consider area based queries, and POIs are assumed to be exact points. However, as we pointed out in Section 1, services can be covered by an area (e.g., task area in crowd-sourcing). Extending existing point-based algorithms to handle area-based problems is often not simple, and a sophisticated adaptation of the existing algorithms may be required to handle a ROAM query.

Road network dimensions. The dimension models for road networks have been discussed [15], [26]. The highway dimension [26] states that there exists a small set S of nodes, such that any shortest path connecting two distant nodes must contain some nodes in S. While it is elegant in theoretical sense, the empirical results for the highway dimension is missing. Arterial dimension [15] describes a similar dimension to highway dimension based on grid partitions of the road network. It uses a 4×4 sub-grid (instead of 5×5 sub-grid in cover dimension) and counts the number of edges needed to cover all paths from one side of the window to its opposite side. The cover dimension we have adopted is similar in spirit to the arterial dimension. Cover dimension is introduced in this paper because it simplifies our theoretical analysis.

9 Conclusions

Providing service recommendation in location-based services has become an important topic. In this paper, we propose the ROAM query, which can be used to determine whether a service can be recommended to a object moving on a road network. We develop efficient algorithms and data structures for this query, and validate its efficiency and effectiveness on large real road network datasets.

Acknowledgments

Reynold Cheng was supported by the Research Grants Council of Hong Kong (RGC Projects HKU 17229116, 106150091, and 17205115) the University of Hong Kong (Projects 104004572, 102009508, and 104004129), and the Innovation and Technology Commission of Hong Kong (ITF project MRP/029/18). Ben Kao was supported by Hong Kong University Grant Council grants

17253616 and 17254016. Shuigeng Zhou was partially supported by National Natural Science Foundation of China under grant No. U1636205.

REFERENCES

[1] "http://www.rydesharing.com/sg/home/."
[2] "https://play.google.com/store/apps/details?id=com.sdu.didi.psnger."
[3] L. Kazemi and C. Shahabi, "Geocrowd: enabling query answering with spatial crowdsourcing," in *GIS*, 2012, pp. 189–198.
[4] 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.
[5] H. Yu, C. Miao, Z. Shen, and C. Leung, "Quality and budget aware task allocation for spatial crowdsourcing," in *ICAAMS*, 2015, pp. 1689–1690.
[6] Z. Chen, R. Fu, Z. Zhao, Z. Liu, L. Xia, L. Chen, P. Cheng, C. C. Cao, Y. Tong, and C. J. Zhang, "gmission: A general spatial crowdsourcing platform," *VLDB*, vol. 7, no. 13, pp. 1629–1632, 2014.
[7] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.
[8] A. V. Goldberg and C. Harrelson, "Computing the shortest path: A search meets graph theory," in *SODA*, 2005, pp. 156–165.
[9] M. Hilger, E. Köhler, R. H. Möhring, and H. Schilling, "Fast point-to-point shortest path computations with arc-flags," *Ninth DIMACS Implementation Challenge*, vol. 74, pp. 41–72, 2009.
[10] H. Samet, J. Sankaranarayanan, and H. Alborzi, "Scalable network distance browsing in spatial databases," in *SIGMOD*, 2008, pp. 43–54.
[11] J. Sankaranarayanan, H. Alborzi, and H. Samet, "Efficient query processing on spatial networks," in *ACM workshop on GIS*, 2005, pp. 200–209.
[12] J. Sankaranarayanan, H. Samet, and H. Alborzi, "Path oracles for spatial networks," *VLDB*, vol. 2, no. 1, pp. 1210–1221, 2009.
[13] 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.
[14] P. Sanders and D. Schultes, "Highway hierarchies hasten exact shortest path queries," in *Algorithms–Esa 2005*. Springer, 2005, pp. 568–579.
[15] 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.
[16] 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.
[17] A. D. Zhu, W. Lin, S. Wang, and X. Xiao, "Reachability queries on large dynamic graphs: a total order approach," in *SIGMOD*, 2014, pp. 1323–1334.
[18] B. Ding, J. X. Yu, and L. Qin, "Finding time-dependent shortest paths over large graphs," in *EDBT*, 2008, pp. 205–216.
[19] 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.
[20] S. Ma, Y. Zheng, and O. Wolfson, "Real-time city-scale taxi ridesharing," *TKDE*, vol. 27, no. 7, pp. 1782–1795, 2015.
[21] ——, "T-share: A large-scale dynamic taxi ridesharing service," in *ICDE*, 2013, pp. 410–421.
[22] 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.
[23] B. Shen, Y. Huang, and Y. Zhao, "Dynamic ridesharing," *SIGSPATIAL Special*, vol. 7, no. 3, pp. 3–10, 2016.
[24] 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.
[25] P. Kalnis, G. Ghinita, K. Mouratidis, and D. Papadias, "Preventing location-based identity inference in anonymous spatial queries," *TKDE*, vol. 19, no. 12, pp. 1719–1733, 2007.
[26] I. Abraham, A. Fiat, A. V. Goldberg, and R. F. Werneck, "Highway dimension, shortest paths, and provably efficient algorithms," in *SODA*, 2010, pp. 782–793.
[27] 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.
[28] L. Kazemi, C. Shahabi, and L. Chen, "Geotrucrowd: trustworthy query answering with spatial crowdsourcing," in *GIS*, 2013, pp. 314–323.

Algorithm 6: `Resume_Check`(p)

```
if p is s (resp. t) then
    q ← t (resp. s);
if search from q is stopped then
    return true
else
    return false;
```

Algorithm 7: `BIS`$(\mathcal{G}, s, t, o, r, \rho)$

```
best_dist← ∞
temp_stop_s ← false; temp_stop_t ← false;
/* indicators of temporary stops */
radi_stop_s ← false; radi_stop_t ← false ;
/* indicators of the radius stops, i.e.,
   whether the search radiuses exceeding
   (1+ρ)dist(s,t). */
for v ∈ V do
    dis_s[v] ← 0, dis_t[v] ← 0
while true do
    go_s=temp_stop_s is false and radi_stop_s is false;
    go_t=temp_stop_t is false and radi_stop_t is false;
    if go_s then
        u ← next visit node by Dijkstra(s);
        best_dist← min{best_dist, dis_s[u] + dis_t[u]};
        if u = o then
            temp_stop_s ← true;
        if dis_s[u] > (1+ρ)best_dist then
            radi_stop_s ← true;
        if u ∈ circ(o,r) and
          dis_s[u] + dis_t[u] ≤ (1+ρ)best_dist;
          /* Positive condition rule */
        then
            return (s ⇝ u) ∪ (u ⇝ t);
        if the condition in Lemma 1 holds then
            return false;
    if go_t then
        /* Do the same by switching s and t
           */
    if go_s is false and go_t is false then
        if temp_stop_s and resume_check(s) then
            temp_stop_s ← true
        if temp_stop_t and resume_check(t) then
            temp_stop_t ← true
        update go_s and go_t
    if go_s is false and go_t is false then
        Break;
return false;
```

10 PROOFS, CODES, ADDITIONAL EXPERIMENTS

Proof sketch of Lemma 1.

Proof. We show that when the condition in the lemma is met, all intersecting nodes between $circ(o, r)$ and a ρ-route have been scanned by both searches, and therefore it is safe to stop searching. We assume by contradiction that there is a node u^* which has at least not been scanned by the search from one side (say, the search from s). Then, we consider following two cases.

Case (i): If u^* has also not been scanned by the search from t. Then, we have $dist(s, u^*) + dist(t, u^*) \geq dist(s, o) + dist(t, o) \geq dist(s, u_s) + dist(t, o) > (1 + \rho)dist(s, t)$, implying that u^* is not in a ρ-route, violating to the assumption.

Case (ii): If u^* has only been scanned by the search from t. In this case, $dist(s, u^*) + dist(t, u^*) \geq dist(s, o) + dist(t, u^*) \geq dist(s, o) + dist(t, u_t) > (1 + \rho)dist(s, t)$, implying that u^* is not in a ρ-route, and violation ensues. □

Proof sketch of Lemma 2.

Proof. Sketch node selection time. Consider sketch node selection in a regional graph of cell C. There are at most b_C boundary nodes cutting the boundary of cell C. Let n'_C be the number of nodes within 5×5 sub-grid centered at cell C. Then, the sketch node selection consumes $O(\Delta b_C n'_C \log(n'_C))$ time. By summing over all the cells, the sketch node selection complexity is $O(\Delta \sum_C b_C n'_C \log(n'_C)) = O(\Delta n \cdot \max_C\{b_C\} \cdot \max_C\{\log(n_C)\})$.

Sketch edge selection time. There are at most $O(\phi)$ sketch nodes selected within a 5×5 sub-grid. Within each regional graph, there are $O(\phi)$ runs of Dijkstra searches. The Dijkstra searches in total consume $O(\phi \Delta n'_C \log(n'_C))$ time. Summing over all the regional graphs, the cost is $O(\phi \Delta n \log n)$.

Storage. For each sketch node p, it links to at most $O(\phi)$ nodes, thus consuming $O(\phi n)$ storage. On the other hand, each 5×5 sub-grid creates $O(\phi^2)$ sketch edges, and hence consuming $O(M^2\phi^2)$ storage. □

Proof sketch of Lemma 3.

Proof. For two reachable sketch nodes s, t, we suppose that the shortest path in $\mathcal{G}$ between s and t, denoted as $s \rightsquigarrow t$, consists of nodes $s = v_0, v_1, \ldots, v_h = t$. Let the nearest sketch node to s along the path $s \rightsquigarrow t$ be a node v_1^*. Then we consider two cases to show (s, v_1^*) is a sketch edge.

Case 1: node v_1^* is outside of the outer shell of s. Then node s must connect to node v_1^* in the original graph $\mathcal{G}$. Otherwise, we denote the transit edge of the shortest path from s to v_1^* as (v', v'') where v' is inside the inner shell.

- If $v' \neq s$, then v' must be a sketch node by sketch node selection strategy (Strategy 1 (1)). This contradicts to the fact that v_1^* is the nearest sketch node to s.
- If $v' = s$, then by Strategy 1 (2), v'' is selected as a sketch node, then $v_1^* = v''$, otherwise v'' is a closer sketch node to s than v_1^*. $v' = s$ and $v_1^* = v''$ imply that node s connects to node v_1^* in the original graph $\mathcal{G}$. Consequently, by our sketch edge selection strategy (Strategy 2), s also connects to node v_1^* in the *sketch graph*.

Case 2: node v_1^* is inside the outer shell of s. In this case, by our sketch edge selection strategy (Strategy 2), node s connects to node v_1^* in the *sketch graph*.

In a nutshell, s connects to its nearest sketch node along the shortest path to t. Go on finding the nearest sketch node v_2^* for v_1^* along the path, we can also show that (v_1^*, v_2^*) is a sketch edge. With this repeatable procedure, we conclude that there is a shortcut version of the shortest $s \rightsquigarrow t$ in the *sketch graph*. Hence the distance between s and t in the *sketch graph* is the same as their shortest distance in the original graph. And hence the *sketch graph* conforms to Definition 3. □

Corollary 1. *For two sketch nodes s and t, let the shortest path from s to t in $\mathcal{G}$ contain sketch nodes $s = v_0^*, v_1^*, \ldots, v_h^* = t$. Then, $\forall 0 \leq i \leq h-1$, (v_i^*, v_{i+1}^*) is a sketch edge such that either $gd(v_i^*, v_{i+1}^*) \leq 3$ or edge (v_i^*, v_{i+1}^*) exists in $\mathcal{G}$.*

Proof. Lemma 3 has proven (v_i^*, v_{i+1}^*) is a sketch edge. It remains to show either $gd(v_i^*, v_{i+1}^*) \leq 3$ or edge (v_i^*, v_{i+1}^*) exists in the original network $\mathcal{G}$. We consider (v_i^*, v_{i+1}^*) to be (s, v_1^*) in the proof of Lemma 3, and hence we have two cases as discussed in the proof. For case 2, i.e., v_{i+1}^* is in the outer shell of v_i^*, clearly we have $gd(v_i^*, v_{i+1}^*) \leq 3$. For case 1, only the second sub-case is feasible, which states that (v_i^*, v_{i+1}^*) exists in $\mathcal{G}$. □

Proof sketch of Lemma 4.

Proof. We first show the space complexity. Each outer shell overlaps with at most 25 outer shells, each of which generates $O(\phi)$ transit nodes within its shell range. Hence in total the number of transit nodes in a shell is $O(\phi)$.

We consider the *Dijkstra* searches for a specific $\mathcal{G}_5$ subgraph with respect to a cell C. The number of transit nodes within the outer shell of C is $O(\phi)$. Suppose the number of nodes in the outer shell of cell C is n'_C, then the cost of the *Dijkstra* search conducted in the $\mathcal{G}_5$ subgraph corresponding to the outer shell is $O(\Delta n'_C \log n'_C)$. Summing over the costs for every cell C, we have $O(\sum_C \Delta n'_C \log n'_C) = O(\Delta n \log n)$.

As for the space complexity, we only need to show the number of bridge edges associated with a node u, $|B(u)|$, is $O(\phi + \Delta)$. This directly follows from Strategy 3. □

Lemma 7. *If $gd(s,t) \geq 4$, the shortest path between s and t in (s,t)-sketch, denoted as $s \rightsquigarrow_{\mathcal{S}} t$, is a shortcut shortest path of $s \rightsquigarrow t$ in $\mathcal{G}$. That is, for each edge (u,v) in $s \rightsquigarrow_{\mathcal{S}} t$, $(u,v) \in B(s) \cup \mathcal{S} \cup B(t)$. Furthermore, either 1) $gd(u,v) \leq 3$; or 2) edge (u,v) exists in $\mathcal{G}$.*

Proof. We consider the following cases: (1) s and t are sketch nodes; (2) one of $\{s,t\}$ is a sketch node, while the other is not; (3) both s and t are not sketch nodes. The proof for case (1) is implied by Corollary 1. We next focus on proofs for case (3), since the proof for case (2) is easily derived from that for case (3).

We first show each edge $(u,v) \in s \rightsquigarrow_{\mathcal{S}} t$ is in $B(s) \cup \mathcal{S} \cup B(t)$. Let us consider the original shortest path between s and t, denoted as $s \rightsquigarrow t = \{s = v_0, v_1, \ldots, v_h = t\}$. First, $gd(s,t) \geq 4$ implies that $s \rightsquigarrow t$ must contain a cover path as a subpath, because there is a shortest path starting from s reaches the outside of the outer shell of s. Consequently, the cover path contained in $s \rightsquigarrow t$ must contain one sketch node u_1 closest to s and therefore $(s, u_1) \in B(s)$. Similarly, $s \rightsquigarrow t$ must also contain one sketch node u_2 closest to t and therefore $(t, u_2) \in B(s)$. Combining Corollary 1, we have for each edge (u,v) in $s \rightsquigarrow_{\mathcal{S}} t$ that $(u,v) \in B(s) \cup \mathcal{S} \cup B(t)$. Similar to Corollary 1, we can show that edge $(u,v) \in s \rightsquigarrow_{\mathcal{S}} t$ satisfies either 1) $gd(u,v) \leq 3$; or 2) edge (u,v) exists in $\mathcal{G}$. We omit the details here. □

Proof sketch of Theorem 1.

Proof. For ease of proof, we conduct the following perturbation: if one node is on the grid cell border, we deviate its location slightly to let it locate inside a certain cell. Now, it is easy to show for three nodes a, b and c, we have $gd(a,b) \geq gd(a,c) - gd(b,c) + 1$.

Next, we prove the Theorem. If ROAM(s,t,o,r,ρ) 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 $gd(o,s) \geq 5 + \lceil r/\delta \rceil$ and $gd(o,o') = \lceil r/\delta \rceil$, we have

$$gd(o',s) \geq gd(o,s) - gd(o,o') + 1 \geq 5 + \lceil r/\delta \rceil - \lceil r/\delta \rceil + 1 = 6$$

Similarly, we have $gd(o',t) \geq 6$. Then, there exists a sketch node p (can be o' itself) along the shortest path from o' to s such that $gd(o',p) < 3$ (note: either p is o' itself or p is in C_3 of o'). Next, we show the following two results: 1) p resides in a ρ-route of (s,t)-sketch and 2) $p \in circ(o, r + 2^{1.5}\delta)$.

For 1), due to $gd(o',s) \geq 6$ and $gd(o',p) < 3$, we have

$$gd(p,s) \geq gd(o',s) - gd(o',p) + 1 > 6 - 3 + 1 = 4,$$

Similarly, $gd(p,t) \geq gd(o',t) - gd(p,o') + 1 > 6 - 3 + 1 = 4$

Hence by Lemma 7, we can exactly compute $dist(p,s)$ and $dist(p,t)$ in (s,t)-sketch. In addition,

$$\begin{aligned} dist(p,s) + dist(p,t) \leq & dist(p,s) + (dist(p,o') + dist(o',t)) \\ = & (dist(p,s) + dist(p,o')) + dist(o',t) \\ = & dist(s,o') + dist(o',t) \\ \leq & (1+\rho) dist(s,t) \end{aligned} \quad (3)$$

Note that, the distance computed in (s,t)-sketch is at least $dist(s,t)$, thus Equation 3 implies that node p must reside in a ρ-route of (s,t)-sketch. For 2),

$$\begin{aligned} dist_E(o,p) \leq & dist_E(o,o') + dist_E(o',p) \\ \leq & r + 2^{0.5}\delta \cdot gd(o',p) \leq r + 2^{1.5}\delta \end{aligned}$$

□

Proof sketch of Lemma 5.

Proof. If node u^* in a ρ-route and $u*$ has not been scanned, then 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 $dist(s,u_i) + dist(u_i,t) \leq (1+\rho) dist(s,t)$ as u^* is in a ρ-route. This implies if u_i is in the queue, then u_{i+1} must be also in the queue. Hence all u_i must have been scanned before visiting u. □

Proof sketch of Lemma 6.

Proof. Positive condition rule. It is sufficient to show the claim that, if $lower\_dist$ is monotone, for any node u scanned during the goal directed search, its true distance from s, $dist(s,u)$, is exactly calculated. We next prove the claim. For any two consecutive nodes u', u'' along the shortest path from node s to node u, it can be shown that node u' must be scanned before u''. This is due to $dis[u''] + lower\_dist[u''] = dis[u'] + dist(u',u'') + lower\_dist[u''] \geq dis[u'] + lower\_dist[u']$. The last step is due to the monotonicity of $lower\_dist$. Therefore, $s \rightsquigarrow u$ cannot be missed during the search, thus follows the first part of the lemma.

Monotonicity of Equation 2. We first show that when $k = 1$ the estimation is monotone. To see this, $lower\_dist(u,t) = |dist(l_1,t) - dist(l_1,u)|$, and $lower\_dist(v,t) = |dist(l_1,t) - dist(l_1,v)|$. Then, $|dist(l_1,t) - dist(l_1,u)| - |dist(l_1,t) - dist(l_1,v)| \leq |(dist(l_1,t) - dist(l_1,u)) - (dist(l_1,t) - dist(l_1,v))| = |dist(l_1,u) - dist(l_1,v)| \leq w(u,v)$. Monotonicity holds for $k = 1$.

Next, we show the monotone property for $k > 1$. Let $A_i = |dist(l_i,s) - dist(l_i,u)|$, $B_i = |dist(l_i,s) - dist(l_i,v)|$, for $1 \leq i \leq k$. Then, due to the case of $k = 1$, we have $A_i - B_i \leq w(u,v)$. It is sufficient to show $max\{A_i\} - max\{B_i\} \leq w(u,v)$ by two cases: 1) $max\{A_i\} - max\{B_i\} \leq 0$. Obviously, $max\{A_i\} - max\{B_i\} \leq w(u,v)$ in this case; 2) $max\{A_i\} - max\{B_i\} > 0$. Let $A_j = max\{A_i\}$, then we have $max\{A_i\} - max\{B_i\} = A_j - max\{B_i\} \leq A_j - B_j \leq w(u,v)$. Therefore, the monotonicity holds. □

Additional experiments. A detour constraint checking algorithm is hinted in a ride-sharing system [24], which uses pairwise shortest distance query to verify whether the detour route exceeds that of $(1+\rho) \cdot dist(s,t)$. As [24] did not aim at solving the same problem as ROAM, we adapt its technical idea behind to answer ROAM queries, and we name the method SG-Ride. The comparison among the approaches are shown in Figure 9 (with the default parameters). The performance of *Sketch* is generally better than SG-Ride. For D1 and D2, SG-Ride is much less efficient than

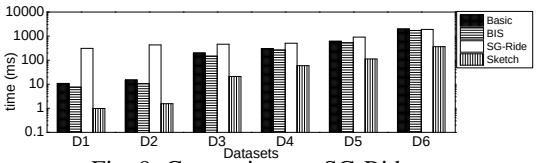
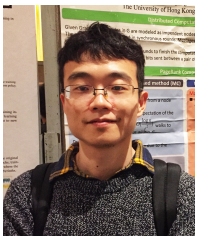


Fig. 9: Comparison to SG-Ride.

Basic. This owes to that s and t are typically close in D1 and D2, in which case conducting one pairwise shortest path query is not faster compared with the Dijkstra search.



Siqiang Luo received the BEng and MS degrees from Fudan University in 2010 and 2013 respectively. He is a Ph.D. candidate in the Department of Computer Science, University of Hong Kong (HKU) under the supervision of Prof. Ben Kao and Dr. Reynold Cheng. His research interests are in the areas of adaptive optimization techniques, spatio-temporal data management and graph algorithms.

Reynold Cheng received the PhD degree in computer science from Purdue University, in 2005. He is an associate professor in the Department of Computer Science, University of Hong Kong. He received an Outstanding Young Researcher Award in 2011-2012 from HKU. He has served as a PC member and reviewer for international conferences (e.g., SIGMOD, VLDB) and journals (e.g., TKDE, TODS). He is an associate editor of TKDE.

Ben Kao received the BSc degree in computer science from the University of Hong Kong, in 1989, and the PhD degree in computer science from Princeton University, in 1995. He is currently a professor in the Department of Computer Science with the University of Hong Kong. His research interests include database management systems, data mining, real-time systems, and information retrieval systems.

Xiaokui Xiao received the PhD degree in computer science from the Chinese University of Hong Kong in 2008. He is currently an Associate Professor at the National University of Singapore (NUS), Singapore. From 2009 to 2017, he was a faculty member at the Nanyang Technological University (NTU), Singapore. His research interests include data privacy, spatial databases, graph databases, and parallel computing. He is an associate editor of TKDE and VLDBJ.

Shuigeng Zhou received the bachelor's degree from the Huazhong University of Science and Technology (HUST), in 1988, the master's degree from the University of Electronic Science and Technology of China (UESTC), in 1991, and the PhD degree in computer science from Fudan University, Shanghai, China, in 2000. He is currently a professor in the School of Computer Science, Fudan University. His research interests include data management, data mining, and bioinformatics. He is a member of the IEEE.

Jiafeng Hu received the BEng degree from Jilin University, in 2011, the M.E. degree from the University of Chinese Academy of Sciences (UCAS), in 2014, and the PhD degree in computer science from the University of Hong Kong (HKU), in 2018. His research interests include spatio-temporal data management and graph databases.