An Efficient Sampling Method for Characterizing Points of Interests on Maps

Pinghui Wang #1, Wenbo He #2, Xue Liu #3

#School of Computer Science, McGill University, Montreal, Canada

[1]wangpinghui369@gmail.com

[2]wenbohe@cs.mcgill.ca

[3]xueliu@cs.mcgill.ca

***Abstract*—Recently map services (e.g., Google maps) and location-based online social networks (e.g., Foursquare) attract a lot of attention and businesses. With the increasing popularity of these location-based services, exploring and characterizing points of interests (PoIs) such as restaurants and hotels on maps provides valuable information for applications such as start-up marketing research. Due to the lack of a direct fully access to PoI databases, it is infeasible to exhaustively search and collect all PoIs within a large area using public APIs, which usually impose a limit on the maximum query rate. In this paper, we propose an effective and efficient method to sample PoIs on maps, and give unbiased estimators to calculate PoI statistics such as sum and average aggregates. Experimental results based on real datasets show that our method is efficient, and requires six times less queries than state-of-the-art methods to achieve the same accuracy.**

I. Introduction

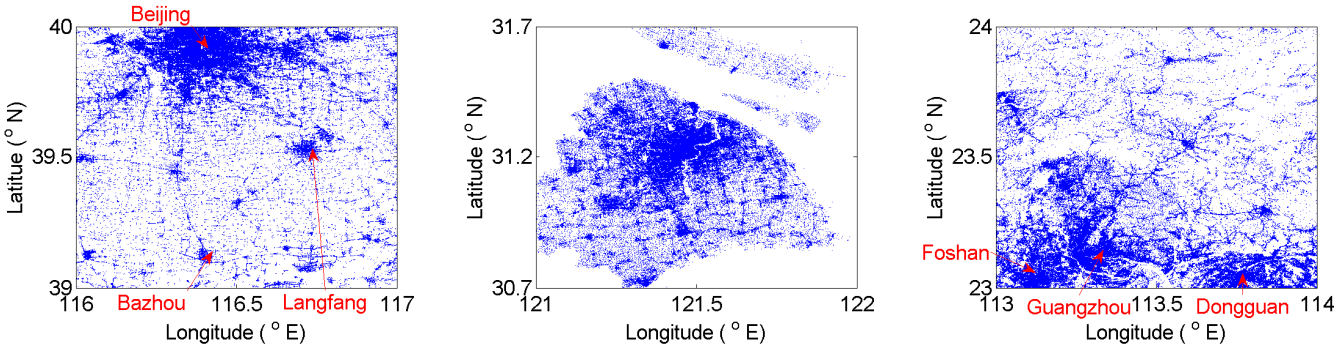
Aggregate statistics (e.g., sum, average, and distribution) of points of interests (PoIs), e.g., restaurants and hotels on map services such as Google maps [1] and Foursquare [2], provide valuable information for applications such as marketing decision making. For example, the knowledge of the PoI rating distribution enables us to evaluate a particular PoI's relative service quality ranking. Moreover, a restaurant start-up can infer food preferences of people in a geographic area by comparing the popularity of restaurant PoIs serving different cuisines within the area of interest. Meanwhile, it can also estimate its market size based on PoI aggregate statistics, such as the number of Foursquare users checked in PoIs within the area. Similarly, a hotel start-up can utilize hotel PoIs' properties such as ratings and reviews to understand its market and competitors.

To exactly calculate the above aggregate statistics, it requires to retrieve all PoIs within the area of interest. However most map service providers do not provide the public with a direct fully access to their PoI databases, so we can only rely on public map APIs to explore and collect PoIs. Moreover, public APIs usually impose limits on the maximum query rate and the maximum number of PoIs returned in a response to a query, therefore it is costly to collect PoIs within a large area. For example, Foursquare map API [3] returns up to 50 PoIs per query and it allows 500 queries per hour per account. To collect PoIs within 14 cities in Foursquare, Li et al. [4] spent almost two months using 40 machines in parallel.

To address the above challenge, ***sampling*** is required. That is, a small fraction of PoIs are sampled and used to calculate PoI statistics. Due to the lack of a direct fully access to PoI databases, one cannot sample over PoIs in a direct manner, so it is hard to sample PoIs uniformly. The existing sampling methods [5], [6] have been proved to sample PoIs with biases. After sampling a fraction of PoIs using these two methods, one has no guarantees whether the PoI statistics obtained directly are to be trusted. To solve this problem, Dalvi et al. [5] propose a method to correct the sampling bias. However the method is costly because it requires a large number of queries for each sampled PoI (e.g., on average 55 queries are used in their paper). The method in [6] samples PoIs with unknown bias, so it is difficult to remove its sampling bias.

In this work we propose an efficient method to eliminate the estimation bias. Our idea is to sample a set of sub-regions from an area of interest at random and then collect PoIs within sampled regions. However, when we query a sampled sub-region including a large number of PoIs, an unknown sampling bias is introduced if we only collect PoIs returned. Otherwise, we need to further divide the sampled sub-region to exhaustively collect all PoIs within it. It requires a large number of queries. To solve this problem, we divide the area of interest into fully accessible sub-regions without overlapping, where a region is defined as a fully accessible region if it includes PoIs less than the maximum number of PoIs returned for a query. Then it is efficient to collect PoIs within a sampled sub-region, which requires just one query. To sample a fully accessible region, our method works as follows: From a specified area, our method divides the current queried region into two sub-regions without overlapping, and then randomly selects a non-empty sub-region as the next region to query. It repeats this process until it observes a fully accessible region. We show that our sampling method is efficient, and it requires only several queries to sample a fully accessible region. Besides its efficiency, the sampling bias of our method is easy to be corrected, which requires no extra queries in comparison with the existing methods [5], [6]. We perform experiments using a variety of real datasets, and show that our method dramatically reduces the number of queries required to achieve the same estimation accuracy of state-of-the-art methods.

The rest of the paper is organized as follows. The problem



(a) Beijing (b) Shanghai (c) Guangzhou

Fig. 1. (Baidu maps) PoI geographical distributions.

is formulated in Section II. Section III presents our method for sampling PoIs on maps and estimating the aggregate statistics of PoIs within the area of interest. Performance evaluation and real applications on Foursquare and Google maps are presented in Section IV. Section V summarizes related work. Concluding remarks then follow.

II. Background

In this section, we first formulate the problem, and then conduct an in-depth analysis of several real datasets to show the challenges of solving the problem.

A. Objectives

Our aim is to estimate aggregate statistics (e.g., sum, average, and distribution) of PoIs' attributes. Formally, let $\mathbb{A}$ be the area of interest. Denote by $\mathbb{P}$ the set of PoIs within $\mathbb{A}$. For example, $\mathbb{P}$ can be the set of hotels within $\mathbb{A}$. We want to estimate the following statistics over $\mathbb{P}$.

1) ***Sum aggregate***. For any function $f : \mathbb{P} \to \mathbb{R}$, where $\mathbb{R}$ is the set of real numbers, the sum aggregate is defined as

$$f_s(\mathbb{P}) = \sum_{p \in \mathbb{P}} f(p).$$

If $f(p)$ is the number of rooms a hotel p has, then $f_s(\mathbb{P})$ corresponds to the total number of hotel rooms within $\mathbb{A}$. If $f(p)$ is the constant function $f(p) = 1$, then $f_s(\mathbb{P})$ corresponds to $|\mathbb{P}|$, the number of hotels within $\mathbb{A}$.

2) ***Average aggregate***. For any function $f : \mathbb{P} \to \mathbb{R}$, the average aggregate is defined as

$$f_a(\mathbb{P}) = \frac{1}{|\mathbb{P}|} \sum_{p \in \mathbb{P}} f(p).$$

If $f(p)$ is the price per room per night for a hotel p, then $f_a(\mathbb{P})$ corresponds to the average price for hotels within $\mathbb{A}$.

3) ***PoI distribution***. Let $L(p)$ be the label of a PoI p specifying a certain property of p. For example, $L(p)$ can be the star rating of a hotel p. Denote the range of PoI labels as $\{l_1, \ldots, l_J\}$. Let $\theta = (\theta_1, \ldots, \theta_J)$ be the distribution of a set of PoIs, where θ_j $(1 \le j \le J)$ is the fraction of PoIs with label l_j. Formally,

$$\theta_j = \frac{1}{|\mathbb{P}|} \sum_{p \in \mathbb{P}} \mathbf{1}(L(p) = l_j), \qquad 1 \le j \le J,$$

where $\mathbf{1}(L(p) = l_j)$ is the indicator function that equals one when predicate $L(p) = l_j$ is true, and zero otherwise. If $L(p)$ is the star rating of a hotel p, then θ is the star rating distribution of hotels within $\mathbb{A}$.

As alluded before, we focus on designing efficient ***sampling*** methods to estimate the above statistics.

B. Datasets

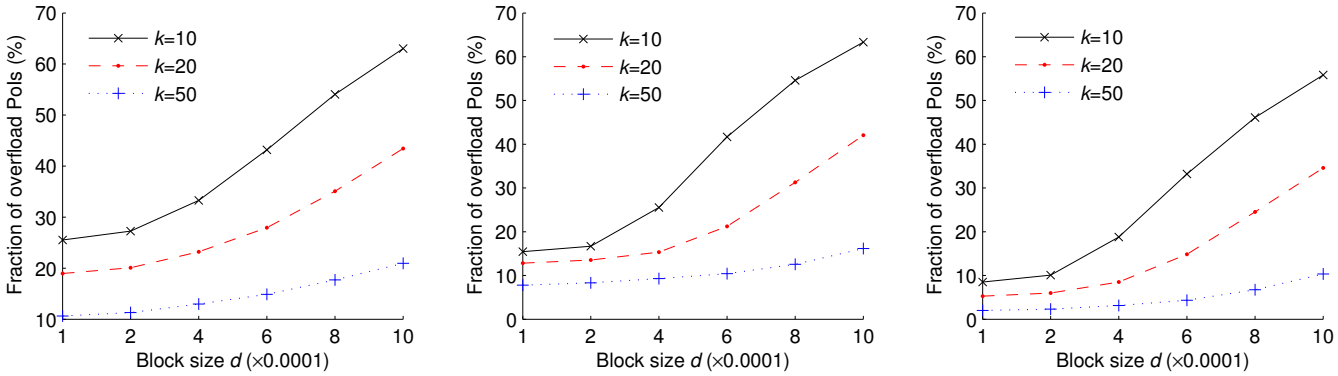
Table I summarizes the public dataset [7] used in this paper, which is collected from Baidu maps for five areas in China. Each area spans a degree of longitude and a degree of latitude. They cover five popular cities in China: Beijing, Shanghai, Guangzhou, Shenzhen, and Tianjin. Each dataset consists of more than 200 thousand PoIs.

Table I
Overview of Baidu PoI Datasets Used in Our Experiments.

Area	Latitude	Longitude	PoIs
Beijing	39° N–40° N	116° E–117° E	606,306
Shanghai	30.7° N–31.7° N	121° E–122° E	470,676
Guangzhou	23° N–24° N	113° E–114° E	554,114
Shenzhen	22° N–23° N	114° E–115° E	322,677
Tianjin	39° N–40° N	117° E–118° E	212,838

C. Challenges

Figure 1 shows the geographical distribution of PoIs within Beijing, Shanghai, and Guangzhou. We can see that PoIs are not evenly distributed, but are clustered into sparse and dense areas. The lack of the PoI geographical distribution makes it challenging to accurately estimate PoI statistics, since it is hard to sample PoIs from $\mathbb{A}$ uniformly. Suppose that one divides $\mathbb{A}$ evenly into non-overlapping sub-regions with size: d degrees of longitude $\times$ d degrees of latitude. Then sample a set of sub-regions at random and collect PoIs within sub-regions sampled. Next, we show that this straightforward sampling method cannot be implemented in practice.



(a) Beijing (b) Shanghai (c) Guangzhou

Fig. 2. (Baidu maps) Fractions of inaccessible PoIs.

Let k be the maximum number of PoIs returned for a query. A region is defined as an inaccessible sub-region when it includes more than k PoIs. Fig. 2 shows the fraction of inaccessible PoIs, where inaccessible PoIs refer to the PoIs within inaccessible sub-regions. We can see that the number of inaccessible PoIs increases with d. When $d = 0.001$, more than 30% of PoIs are inaccessible for $k = 10$ and $k = 20$. For a sampled inaccessible sub-region r, therefore, it might include a large number of PoIs. Then we need to further divide r and explore all PoIs within r, which might require a large number of queries since r might include a lot of PoIs. Otherwise, k PoIs within r are returned by public APIs and they might not be randomly selected from the PoIs within r. This introduces unknown errors into estimates of PoI's statistics. Fig. 2 shows that almost 10% of PoIs are within inaccessible regions even when $d = 0.0001$ for the Beijing and Shanghai areas. Thus, a quite small value of d is required to ensure there is no inaccessible sub-region.

However a small d dramatically increases the number of queries required to sample a non-empty sub-region (i.e., a region with at least one PoI). Define ρ as the fraction of non-empty sub-regions among all sub-regions. Then, the hit ratio defined as the probability of sampling a non-empty sub-region is ρ. To sample a non-empty sub-region, on average we need to sample and query $1/\rho$ sub-regions at random. Fig. 3 shows the hit ratios for different d. We can see that ρ increases with d. When $d = 0.0001$, almost 99.8% of sub-regions do not include any PoI. Therefore, on average more than 500 queries are required to sample a non-empty sub-region for $d \leq 0.0001$.

In summary, the above straightforward sampling method is not easy to be implemented, so designing unbiased and efficient sampling methods for estimating PoI statistics is a much challenging task.

III. Random Region Zoom-in on Maps

In this section, we present our sampling method to estimate PoI aggregate statistics defined in Section II. We first propose a random region zoom-in (RRZI) method to sample PoIs within an area $\mathbb{A}$ of interest, and give our unbiased estimators of PoI

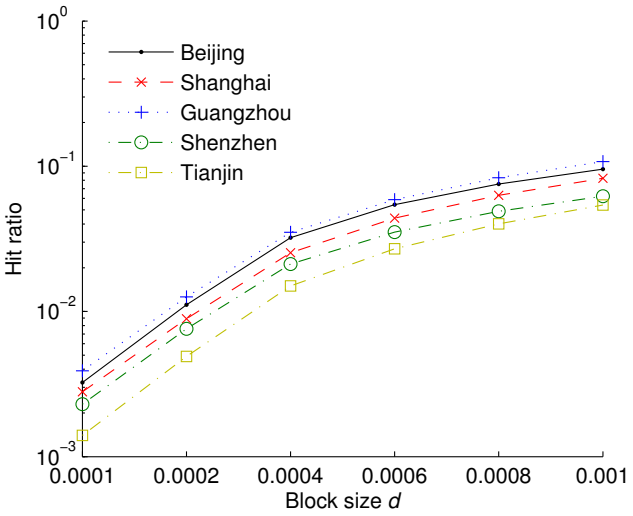


Fig. 3. (Baidu maps) Hitting ratios.

statistics. To improve the accuracy of RRZI, we then propose a method RRZIC by utilizing the meta information (i.e., the total number of PoIs within an input search region) returned for a query, which is provided by map services such as Google maps. To further reduce the number of queries of RRZI and RRZIC required, we propose mix methods RRZI_URS and RRZIC_URS, which first pick a small sub-region from $\mathbb{A}$ at random and then sample PoIs within the sub-region using RRZI and RRZIC respectively. For ease of reading, we list notations used throughout the paper in Table II.

A. Random Region Zoom-in

As shown in Fig. 4, RRZI($\mathbb{A}$) works as follows: From the initial region $\mathbb{A}$, RRZI divides the current queried region into two sub-regions without overlapping, and then randomly selects a non-empty sub-region as the next region to zoom in and query. Repeat this procedure recursively until a fully accessible region is finally observed. The reason of why we divide the current region into just two sub-regions is because

Table II
TABLE OF NOTATIONS

$\mathbb{A}$	area of interest
$\mathbb{P}$	set of PoIs within $\mathbb{A}$
k	maximum number of PoIs returned in a response to a query
$\tau(Q, A)$, $Q \subseteq A$	probability of sampling a region Q from A
$n(Q)$	number of PoIs within a region Q
$\chi_0(Q)$, $\chi_1(Q)$	two sub-regions obtained by dividing a region Q
δ	minimum acceptable latitude and longitude precision of map APIs
L	parameter to control the size of regions sampled by URS
B_L	set of sub-regions obtained by iteratively applying L times region division operations into $\mathbb{A}$
B_L^*	set of non-empty regions in B_L
$P(r)$	set of PoIs within a region r
V	set of fully accessible regions obtained by iteratively applying region division operations into $\mathbb{A}$
m	number of sampled fully accessible regions

it minimizes the number of queries required at each step, which will be discussed later. Algorithm 1 shows the pseudo-code of RRZI($\mathbb{A}$). Initially RRZI sets region $Q = \mathbb{A}$. At each step, RRZI uses a query $searchPoI(Q)$ to explore PoIs within Q, and then determines whether Q is a fully accessible region depending on the value of $|O|$, the number of PoIs returned by $searchPoI(Q)$. When $|O|$ equals k, Q might not be a fully accessible region, and RRZI divides Q into two non-overlapping sub-regions Q_0 and Q_1 to further explore. If Q_0 and Q_1 are both non-empty regions, RRZI randomly selects one as the next region, that is, $Q \leftarrow random(Q_0, Q_1)$. Otherwise, RRZI sets Q as the non-empty region among Q_0 and Q_1. RRZI repeats this procedure until a fully accessible region Q is observed.

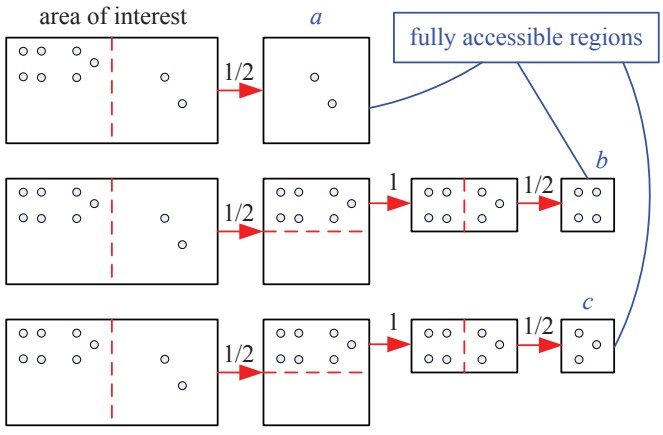


Fig. 4. An example of applying RRZI into the area of interest, where $k = 5$. The number above a red arrow refers to the probability of selecting a sub-region of the current queried region to zoom in.

Next, we answer the following three ***critical*** questions.

- ***How to divide Q into two non-overlapping regions Q_0 and Q_1?*** For simplicity, we denote $Q = [(x_{SW}, y_{SW}), (x_{NE}, y_{NE})]$ as a quadrangle region with south-west corner (x_{SW}, y_{SW}) (latitude and longitude pair) and north-east corner (x_{NE}, y_{NE}). Let δ be the minimum acceptable latitude and longitude precision of map APIs. Let $\beta(x_{SW}, x_{NE})$ be a function whose values are positive real numbers. We define functions $\chi_0(Q)$ and $\chi_1(Q)$ as follows: If $|x_{SW} - x_{NE}| \geq \beta(x_{SW}, x_{NE})|y_{SW} - y_{NE}|$, then

$$\begin{cases} \chi_0(Q) = [(x_{SW}, y_{SW}), (\lceil \frac{x_{SW} + x_{NE}}{2\delta} \rceil \delta - \delta, y_{NE})] \\ \chi_1(Q) = [(\lceil \frac{x_{SW} + x_{NE}}{2\delta} \rceil \delta, y_{SW}), (x_{NE}, y_{NE})]. \end{cases} \quad (1)$$

Otherwise,

$$\begin{cases} \chi_0(Q) = [(x_{SW}, y_{SW}), (x_{NE}, \lceil \frac{y_{SW} + y_{NE}}{2\delta} \rceil \delta - \delta)] \\ \chi_1(Q) = [(x_{SW}, \lceil \frac{y_{SW} + y_{NE}}{2\delta} \rceil \delta), (x_{NE}, y_{NE})]. \end{cases} \quad (2)$$

Using functions χ_0 and χ_1 we divide Q into two sub-regions $\chi_0(Q)$ and $\chi_1(Q)$ without overlapping, which almost have the same size. $\beta(x_{SW}, x_{NE})$ is used to control the shape of a sub-region finally sampled. For instance, we let $\beta(x_{SW}, x_{NE}) = 1/\cos(0.5(x_{SW} + x_{NE}))$, and then the shape of a fully accessible region sampled by RRZI approaches to a square, since the physical distance between two points on the earth with the same latitude x but different longitudes y and $y + d$ is $111d \times \cos x$ kilometers, and the distance between two points with the same longitudes y but different longitudes x and $x + d$ is $111d$.
- ***How to determine whether $\chi_0(Q)$ and $\chi_1(Q)$ are empty regions or not using a minimum number of queries?*** Let O be the set of PoIs within the region Q observed by previous queries. If O includes PoIs within both $\chi_0(Q)$ and $\chi_1(Q)$, then neither of $\chi_0(Q)$ and $\chi_1(Q)$ is empty. Otherwise, we query the sub-region with no PoIs in O to determine whether it is a truly empty region. Therefore RRZI needs at most one query at each step.
- ***Does RRZI sample PoIs uniformly? If not, how to remove the sampling bias?*** Fig. 4 shows an example of applying RRZI into $\mathbb{A}$, where $k = 5$. We can see that there exist three fully accessible regions a, b, and c, which can be observed and sampled by RRZI. The probabilities of sampling a, b, and c are $1/2$, $1/4$, and $1/4$ respectively. The sampling bias might introduce large errors into the measurement of PoI statistics. To solve this problem, we use a counter τ to record the probability of sampling a region from $\mathbb{A}$, which is used to correct the sampling bias later. τ is initialized with 1, and updated as follows: At each step, we set $\tau = \tau/2$ if both $\chi_0(Q)$ and $\chi_1(Q)$ are non-empty, otherwise τ keeps unchanged. Finally τ records the probability of sampling a fully accessible sub-region from $\mathbb{A}$.

At last, we analyze the query cost of RRZI and present our unbiased estimators of PoI aggregate statistics under study. We can easily find that the number of queries required to sample

a fully accessible region is smaller than

$$H_{max} = \log(L_x/\delta) + \log(L_y/\delta), \tag{3}$$

where L_x and L_y are the length and width of the region $\mathbb{A}$. For instance, H_{max} equals 60 when $\delta = 10^{-6}$ and $\mathbb{A}$ is a region with size: one degree of longitude $\times$ one degree of latitude. Experimental results in a later section show that the associated average number of queries required is about 13, which is much smaller than H_{max}.

Denote $\tau(Q, \mathbb{A})$ as the probability of sampling a fully accessible region Q from $\mathbb{A}$. Let $P(Q)$ be the set of PoIs within Q. Denote V as the set of fully accessible regions, which can be observed by RRZI. Then we have $\sum_{r \in V} \tau(r, \mathbb{A}) = 1$. Given a set of fully accessible regions r_i $(1 \le i \le m)$ sampled by calling the function RRZI($\mathbb{A}$) m times, we collect PoIs within r_i, and estimate the sum aggregate $f_s(\mathbb{P})$ as

$$\tilde{f}_s(\mathbb{P}) = \frac{1}{m} \sum_{i=1}^{m} \sum_{p \in P(r_i)} \frac{f(p)}{\tau(r_i, \mathbb{A})}.$$

We can easily find that $\tilde{f}_s(\mathbb{P})$ is an **unbiased** estimator of $f_s(\mathbb{P})$, that is, $E[\tilde{f}_s(\mathbb{P})] = f_s(\mathbb{P})$. The variance of $\tilde{f}_s(\mathbb{P})$ is

$$Var(\tilde{f}_s(\mathbb{P})) = \frac{1}{m} \left(\sum_{r \in V} \frac{\left(\sum_{p \in P(r)} f(p) \right)^2}{\tau(r, \mathbb{A})} - f_s^2(\mathbb{P}) \right).$$

Similarly, we estimate the PoI distribution $\theta = (\theta_1, \ldots, \theta_J)$ as follows

$$\tilde{\theta}_j = \frac{1}{\tilde{H}} \sum_{i=1}^{m} \sum_{p \in P(r_i)} \frac{\mathbf{1}(L(p) = l_j)}{\tau(r_i, \mathbb{A})}, \qquad 1 \le j \le J.$$

where $\tilde{H} = \sum_{i=1}^{m} \frac{n(r_i)}{\tau(r_i, \mathbb{A})}$ In this section, we do not present our method for estimating the average aggregate $f_a(\mathbb{P})$. $f_a(\mathbb{P})$ is easily computed based on our estimates of sum aggregates since $f_a(\mathbb{P}) = f_s(\mathbb{P})/|\mathbb{P}|$.

B. Random Region Zoom-in with Count Information

In this subsection we propose a method, named random region zoom-in with count information (RRZIC), to further improve the accuracy of RRZI for map services such as Google maps, where results from a query include the number of PoIs within the input search region. Compared to RRZI, RRZIC tends to sample PoIs uniformly, giving us smaller estimation errors for PoIs statistics. The pseudo-code RRZIC($\mathbb{A}$) is shown as Algorithm 2. Initially we set $Q = \mathbb{A}$. Denote z as the number of PoIs within the current queried region Q. Let z_0 and z_1 be the number of PoIs within the two sub-regions $\chi_0(Q)$ and $\chi_1(Q)$ of Q respectively. If $z > k$, Q is not a fully accessible region, and RRZIC queries sub-region $\chi_0(Q)$ to obtain z_0. With probability z_0/z, RRZIC then selects $\chi_0(Q)$ to further explore. That is, set $Q = \chi_0(Q)$ and $z = z_0$. Or, RRZIC sets $Q = \chi_1(Q)$ and $z = z_0$ with probability z_1/z. RRZIC repeats this procedure until a fully accessible Q is observed. We can easily find that RRZIC samples Q from $\mathbb{A}$

Algorithm 1: RRZI($\mathbb{A}$) pseudo-code.

```
/* A is the region of interest.                                   */
input : A
/* Q is a sub-region sampled from A at random,
   and τ records the probability of sampling Q
   from A.                                                        */
output: Q and τ
Q ← A, τ ← 1, and l ← 0;
/* searchPoI(Q) is the set of PoIs returned for
   querying the region Q                                          */
O ← searchPoI(Q);
/* k is the maximum number of PoIs returned in a
   response to a query.                                           */
while |O| = k do
    /* χ0(Q) and χ1(Q) are the two sub-regions of Q
       defined as (1) and (2)                                     */
    Q0 ← χ0(Q) and Q1 ← χ1(Q);
    /* If O includes no PoI within the region Q0/Q1,
       then emptyRegion(Q0, Q1, O) returns 0/1.
       Otherwise, both Q0 and Q1 are non-empty, and
       emptyRegion(Q0, Q1, O) returns -1                          */
    i ← emptyRegion(Q0, Q1, O);
    if i ≠ −1 then
        O' ← searchPoI(Qi);
        if |O'| = 0 then
            Q ← Q1−i;
        else
            /* random(Q0, Q1) returns Q0 and Q1 at
               random                                             */
            Q ← random(Q0, Q1);
            O ← O' and τ ← τ/2;
        end
    else
        Q ← random(Q0, Q1);
        O ← searchPoI(Q) and τ ← τ/2;
    end
end
```

with probability $\tau(Q, \mathbb{A}) = n(Q)/n(\mathbb{A})$, where $n(Q)$ is the number of PoIs within Q.

Given m fully accessible regions r_i $(1 \le i \le m)$ sampled by RRZIC, we estimate $f_s(\mathbb{P})$ as

$$\bar{f}_s(\mathbb{P}) = \frac{1}{m} \sum_{i=1}^{m} \sum_{p \in P(r_i)} \frac{f(p) n(\mathbb{A})}{n(r_i)}, \tag{4}$$

and estimate $\theta = (\theta_1, \ldots, \theta_J)$ as

$$\bar{\theta}_j = \frac{1}{m} \sum_{i=1}^{m} \sum_{p \in P(r_i)} \frac{\mathbf{1}(L(p) = l_j)}{n(r_i)}, \qquad 1 \le j \le J. \tag{5}$$

Similar to $\tilde{f}_s(\mathbb{P})$ and $\tilde{\theta}_j$, $\bar{f}_s(\mathbb{P})$ and $\bar{\theta}_j$ are **unbiased** estimators

of $f_s(\mathbb{P})$ and θ_j, $1 \le j \le J$. Their variances are

$$Var(\bar{f}_s(\mathbb{P})) = \frac{1}{m}\left(\sum_{r\in V}\frac{\left(\sum_{p\in P(r)} f(p)\right)^2 n(\mathbb{A})}{n(r)} - f_s^2(\mathbb{P})\right);$$

$$Var(\bar{\theta}_j) = \frac{1}{m}\left(\sum_{r\in V}\frac{\left(\sum_{p\in P(r)} \mathbf{1}(L(p)=l_j)\right)^2}{n(r)n(\mathbb{A})} - \theta_j^2\right).$$

Algorithm 2: RRZIC($\mathbb{A}$) pseudo-code.

```
/* A is the region of interest.                                   */
input : A
/* Q is a sub-region sampled from A at random,
   and τ records the probability of sampling Q
   from A.                                                        */
output: Q and τ
Q ← A and τ ← 1;
/* countPoI(Q) returns the number of PoIs within Q.              */
z ← countPoI(Q);
/* k is the maximum number of PoIs returned in a
   response to a query.                                           */
while z > k do
    /* χ0(Q) and χ1(Q) are the two sub-regions of Q
       defined as (1) and (2).                                    */
    Q0 ← χ0(Q) and Q1 ← χ1(Q);
    /* z0 and z1 are the numbers of PoIs within the
       regions Q0 and Q1 respectively.                            */
    z0 ← countPoI(Q0) and z1 = z − z0;
    /* U(0,1) is a uniform (0,1) random sample.                   */
    u ← U(0,1);
    if u < z0/z then
        Q ← Q0, τ ← τ × z0/z, and z ← z0;
    else
        Q ← Q1, τ ← τ × z1/z, and z ← z1;
    end
end
```

C. Random Region Zoom-in Combined with Uniform Region Sampling

Public map APIs might impose a limit on the size of input regions. For example, Foursquare returns an error message "Your geographic boundary is too big. Please search a smaller area." for a query specified with an input quadrangle region with size: 3 degrees of longitude $\times$ 3 degrees of latitude, although it does not literally state any limit on the input quadrangle region. Thus, RRZI and RRZIC cannot be directly applied to sample PoIs from a large area on Foursquare. To solve this problem, we propose mix methods, which first pick a small sub-region from $\mathbb{A}$ at random and then sample PoIs within the sub-region using RRZI and RRZIC. Moreover, we show that our mix methods also reduce the number of queries required to sample a fully accessible region in comparison with RRZI and RRZIC.

We first introduce a uniform region sampling (URS) method, which is used to sample sub-regions from $\mathbb{A}$ uniformly. Let L be a parameter to control the size of sub-regions sampled by URS. Denote B_L as the set of sub-regions of $\mathbb{A}$ obtained by iteratively applying L times region division operations defined as (1) and (2) into $\mathbb{A}$. Formally,

$$B_L = \{\chi_{i_1}(\chi_{i_2}(...(\chi_{i_L}(\mathbb{A})))) : i_1, i_2, \ldots, i_L \in \{0,1\}\}.$$

B_L consists of 2^L regions. That is, $|B_L| = 2^L$. Regions in B_L are nearly 2^L times smaller than $\mathbb{A}$. We assume that they are small enough to be used as input regions for public APIs. Let B_L^* be the set of non-empty regions in B_L. To sample a region from B_L^* uniformly, URS repeats to sample regions from B_L at random until a non-empty region is observed.

Our mix methods RRZI_URS and RRZIC_URS are modifications of RRZI and RRZIC. Algorithm 3 shows their pseudo-codes. RRZI_URS and RRZIC_URS first randomly select a non-empty region b from B_L using URS, and then sample a fully accessible region from b using RRZI(b) and RRZIC(b) respectively. Let $|\hat{B}_L^*|$ be an estimate of $|B_L^*|$, which can be easily estimated based on the hit ratio of sampling a non-empty region from B_L using URS. Given m fully accessible regions r_i ($1 \le i \le m$) sampled by RRZI_URS or RRZIC_URS, we estimate $f_s(\mathbb{P})$ and $\theta = (\theta_1, \ldots, \theta_J)$ as follows

- **Estimators for RRZI_URS:**

$$\acute{f}_s(\mathbb{P}) = \frac{|\hat{B}_L^*|}{m}\sum_{i=1}^{m}\sum_{p\in P(r_i)}\frac{f(p)}{\tau(r_i,b_i)};$$

$$\acute{\theta}_j = \frac{1}{\acute{H}}\sum_{i=1}^{m}\sum_{p\in P(r_i)}\frac{\mathbf{1}(L(p)=l_j)}{\tau(r_i,b_i)}, \qquad 1 \le j \le J,$$

  where $\acute{H} = \sum_{i=1}^{m}\frac{n(r_i)}{\tau(r_i,b_i)}$.
- **Estimators for RRZIC_URS:**

$$\grave{f}_s(\mathbb{P}) = \frac{|\hat{B}_L^*|}{m}\sum_{i=1}^{m}\sum_{p\in P(r_i)}\frac{f(p)n(b_i)}{n(r_i)};$$

$$\grave{\theta}_j = \frac{1}{\grave{H}}\sum_{i=1}^{m}\sum_{p\in P(r_i)}\frac{\mathbf{1}(L(p)=l_j)n(b_i)}{n(r_i)}, \qquad 1 \le j \le J,$$

  where $\grave{H} = \sum_{i=1}^{m} n(b_i)$.

Finally, we analyze the query costs of the above mixing methods. To sample a non-empty region from B_L at random, on average URS needs to sample and query $|B_L|/|B_L^*|$ regions from B_L, but RRZI and RRZIC require L queries. Thus, the mixing methods are more efficient than RRZI and RRZIC when $|B_L|/|B_L^*| < L$, which is true for small values of L.

IV. Data Evaluation

In this section, we first conduct experiments on real world datasets listed in Table I to evaluate the performance of our methods for estimating PoI aggregate statistics. Then we apply our methods to characterize PoIs on Foursquare and Google maps in the wild.

Algorithm 3: RRZI_URS/RRZIC_URS pseudo-code.

```
/* A is the region of interest.                                   */
input : A and L
/* r_i, 1 ≤ i ≤ m, are sub-regions sampled from A
   at random, and τ(r_i, b_i) records the probability
   of sampling r_i from a region b_i, where b_i is
   sampled from the set B*_L at random.                           */
output: r_1, ..., r_m and τ(r_1, b_1), ..., τ(r_m, b_m).
i ← 1;
while i ≤ m do
    /* URS(A) returns a region sampled from the
       set B*_L at random.                                        */
    b_i ← URS(A);
    /* The following statement is used for
       RRZI_URS. For RRZIC_URS, it should be
       [r_i, τ(r_i, b_i)] ← RRZIC(b_i)                            */
    [r_i, τ(r_i, b_i)] ← RRZI(b_i);
    i ← i + 1;
end
```

A. Results of Estimating $n(\mathbb{A})$

We evaluate the performance of our methods RRZI and RRZI_URS for estimating $n(\mathbb{A})$, the total number of PoIs within the area of interest $\mathbb{A}$, which is an important statistic studied in [5], [6]. In this subsection, our sampling methods using the PoI count information (i.e., RRZIC and RRZIC_URS) are not studied, since we assume that map services do not provide a public API where results from a query include the number of PoIs within an input search region (otherwise $n(\mathbb{A})$ can be obtained in a direct manner). We define the normalized root mean square error (NRMSE), that is $\text{NRMSE}(\hat{n(\mathbb{A})}) = \sqrt{E[(\hat{n}(\mathbb{A}) - n(\mathbb{A}))^2]}/n(\mathbb{A})$, be a metric to measure the relative error of the estimate $\hat{n}(\mathbb{A})$ with respect to its true value $n(\mathbb{A})$. In the following experiments, we average the estimates and calculate their NRMSEs over 10,000 runs. Fig. 5 shows the NRMSEs of estimates of $n(\mathbb{A})$ for RRZI with different m, the number of fully accessible regions sampled. We can see that the NRMSEs decrease as m increases, and are roughly proportional to $1/\sqrt{m}$. Meanwhile, RRZI exhibits smaller errors for larger k, the maximum number of PoIs returned in a response to a query. Fig. 6 shows the average number of queries required to sample a fully accessible region for RRZI_URS with different L, where the y-axis is in log scale. We can see that the query cost of RRZI_URS first decreases with L and then increases with L. It is because URS needs only one or two queries to sample a non-empty sub-region for small and medium L, but a large number of queries is required for large L, which is shown in Fig. 7. In practice, we can conduct a pilot study to estimate the optimal value of L. Fig. 8 shows the compared NRMSEs of estimates of $n(\mathbb{A})$ for RRZI_URS with different L under the same number of queries (10,000 queries). We can see that the NRMSEs first decrease with L and then increase with L. RRZI_URS with $L = 10$, $L = 15$, and $L = 20$ are almost 2 times more accurate than RRZI_URS with $L = 0$, which is equivalent to RRZI. It indicates that RRZI_URS requires nearly 4 times less queries than RRZI to achieve the same estimation accuracy, since the NRMSEs are roughly proportional to $1/\sqrt{m}$, which is observed in Fig. 5.

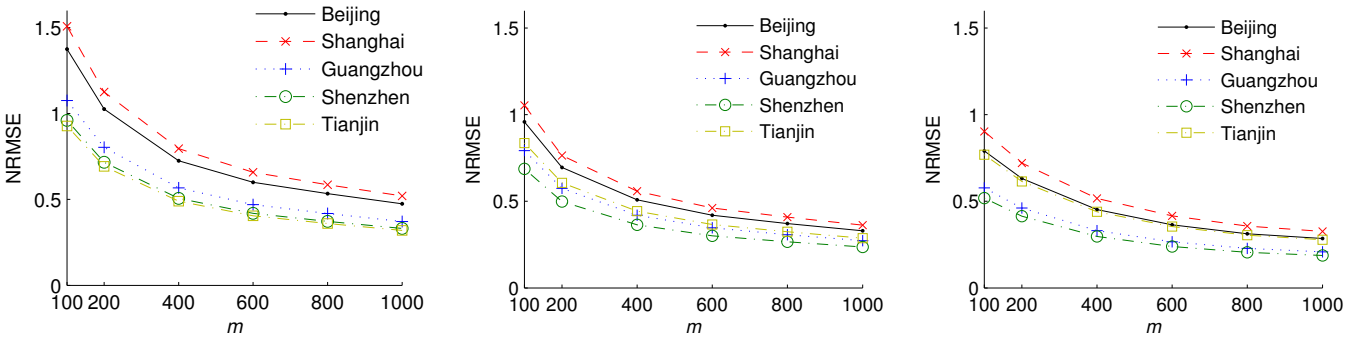
We compare our methods with the state-of-the-art methods *nearest-neighbor search* (NNS) [5] and *random region sampling* (RRS) [6]. Fig. 9 shows the average number of queries required to obtain an estimate of $n(\mathbb{A})$ with NRMSE less than 0.1 for RRZI and RRZI_URS in comparison with NNS and RRS, where we set $L = 15$ and $k = 20$. We can see that RRZI_URS requires 8 and 6 times less queries than RRS and NNS respectively. Here the results of NNS are obtained based on the assumption that a NNS API is supported by map service providers. When map services such as Foursquare do not provide such a NNS API, the PoI closest to a randomly sampled point (x, y) can be obtained by applying a binary search to find a radius r, which ensures at least one and at most k PoIs are within the circle with center (x, y) and radius r. Thus, on average $\log(W/\delta)$ queries are required to get the PoI closest to (x, y), where δ be the minimum acceptable precision for public APIs and W is the diameter of $\mathbb{A}$. Let $D(p)$ be the set of points in the plane $\mathbb{A}$ that are closer to p than the other PoIs, which is used to correct the sampling bias of the NNS method. Table III shows that on average a thousand queries are required to sample a PoI p and determine $D(p)$, which indicates that the NNS method requires much more queries than our methods to achieve the same estimation accuracy when the NNS API is not available.

Table III
(BAIDU MAPS) AVERAGE NUMBER OF QUERIES REQUIRED FOR SAMPLING A POI p AND DETERMINING $D(p)$ WHEN THE NNS API IS NOT AVAILABLE.

Area	$k = 10$	$k = 20$	$k = 50$
Beijing	1,196	1,079	995
Shanghai	1,176	1,076	982
Guangzhou	1,172	1,043	969
Shenzhen	1,154	1,038	931
Tianjin	1,106	1,034	909

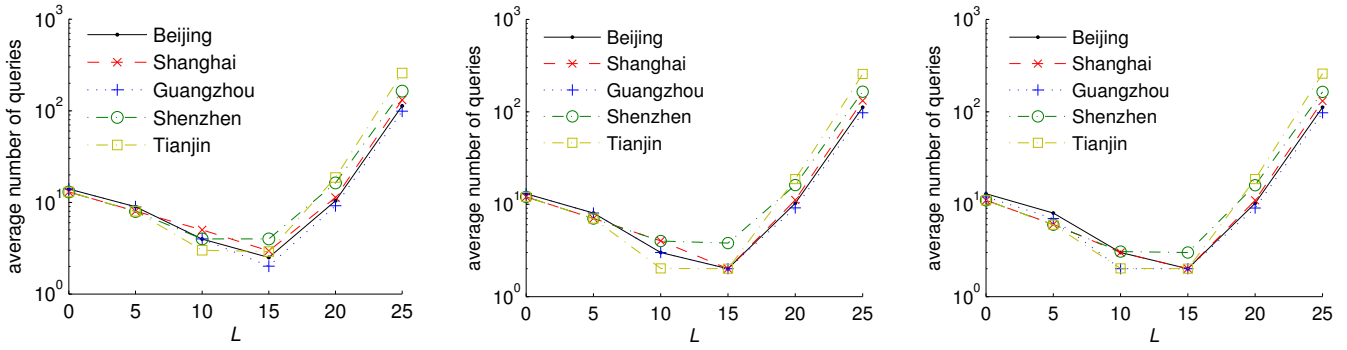
B. Results of Estimating Average and Distribution Statistics

In this subsection, we conduct experiments to evaluate the performance of our methods for estimating PoIs' average and distribution statistics. For the Baidu PoI datasets we used, PoIs are classified into different types such as restaurant, hotel, and shopping. The numbers of restaurant-tpye PoIs are 75,255, 36,417, 24,353, 16,025, and 10,032 for datasets Beijing, Shanghai, Guangzhou, Shenzhen, and Tianjin respectively. We use these restaurant-type PoIs to generate benchmark datasets for our following experiments.We manually generate a cost for each restaurant-type PoI using two different cost distribution schemes CDS_UNI and CDS_NOR. For CDS_UNI, the cost of a PoI is uniformly selected from the range (0, 300) at random. For CDS_NOR, the cost of a PoI is a positive



(a) $k = 10$ (b) $k = 20$ (c) $k = 50$

Fig. 5. (Baidu maps) NRMSEs of estimates of $n(\mathbb{A})$ for RRZI with different m, where m is the number of fully accessible regions sampled.



(a) $k = 10$ (b) $k = 20$ (c) $k = 50$

Fig. 6. (Baidu maps) Average number of queries required to sample a fully accessible region for RRZI_URS.

number randomly selected from $(0, +\infty)$ according to a normal distribution with mean 150 and stand deviation 100.

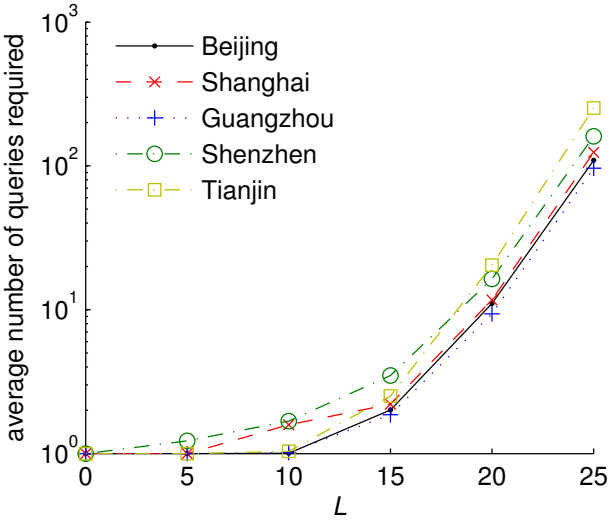


Fig. 7. (Baidu maps) Average number of queries required to sample a non-empty sub-region for URS with different L.

Figures 10 and 11 show our experimental results. Fig. 10 shows the results for estimating the average cost of restaurant-type PoIs for different methods under 1,000 queries, where we set $L = 9$ for our mix methods RRZI_URS and RRZIC_URS. We can see that RRZI_URS is almost 2 times more accurate than RRZI. RRZIC and RRZIC_URS utilizing PoI count infor-

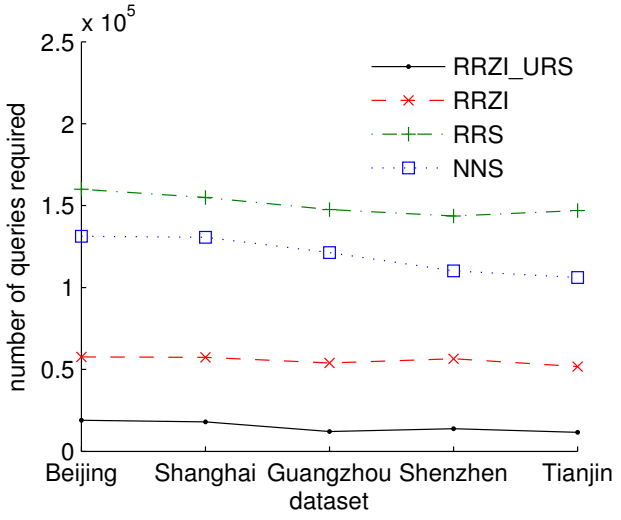
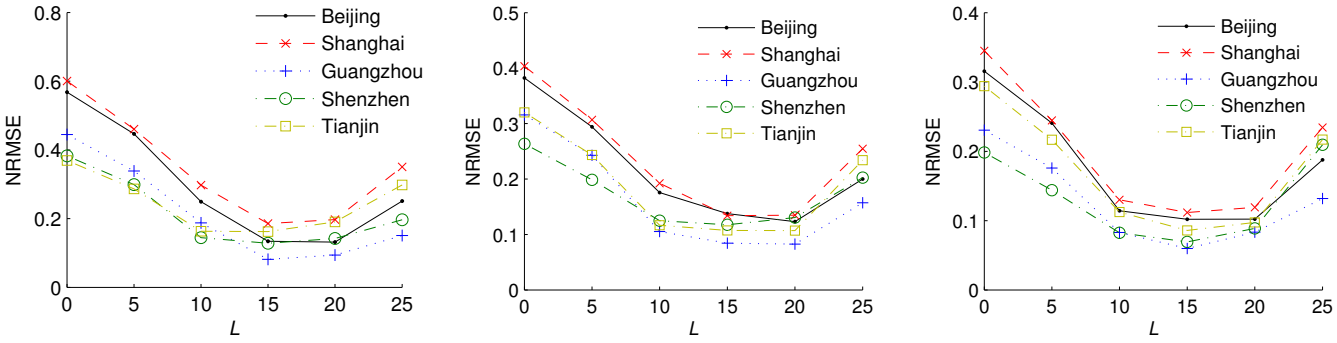
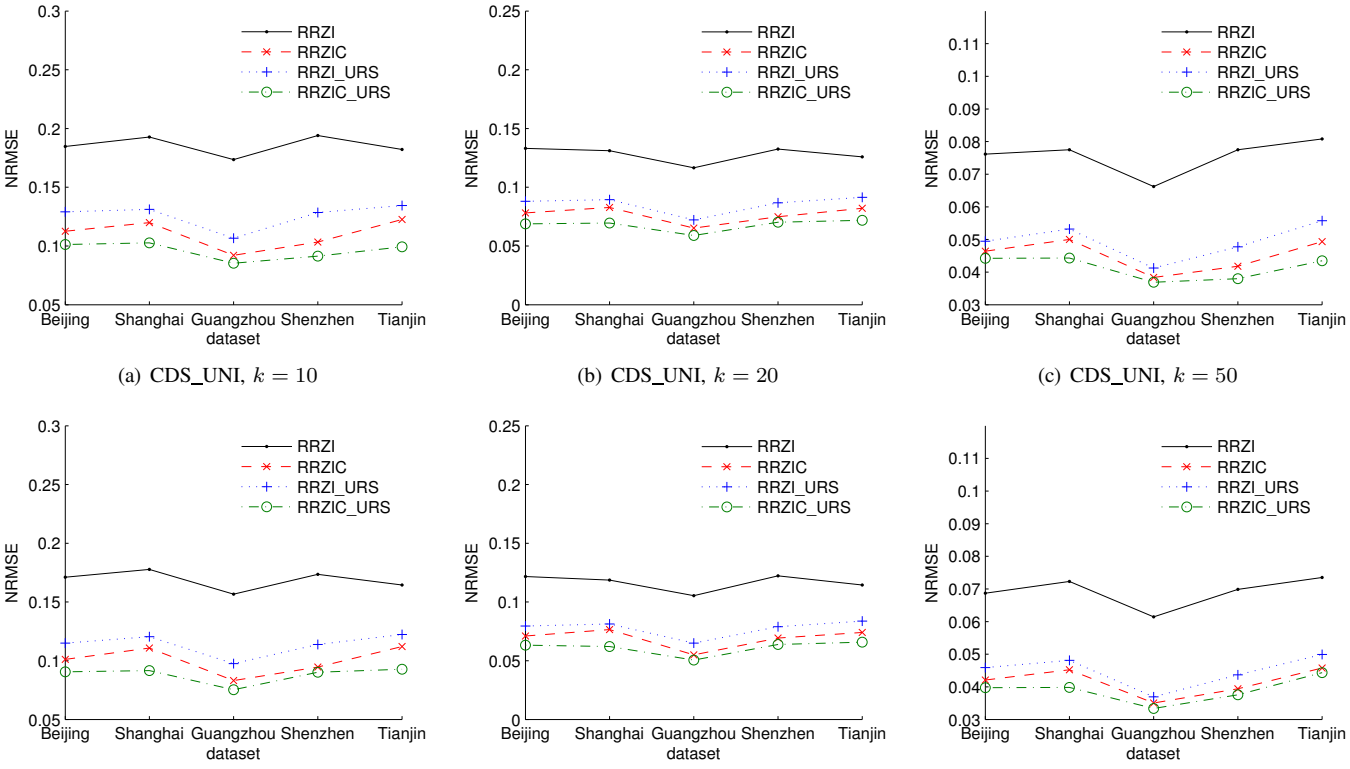


Fig. 9. (Baidu maps) The number of queries required to obtain an estimate of $n(\mathbb{A})$ with NRMSE less than 0.1 for RRZI_URS and RRZI in comparison with the state-of-the-art methods NNS [5] and RRS [6].



(a) $k = 10$ (b) $k = 20$ (c) $k = 50$

Fig. 8. (Baidu maps) Compared NRMSEs of estimates of $n(\mathbb{A})$ for RRZI_URS with different L under 10,000 queries.



(a) CDS_UNI, $k = 10$ (b) CDS_UNI, $k = 20$ (c) CDS_UNI, $k = 50$

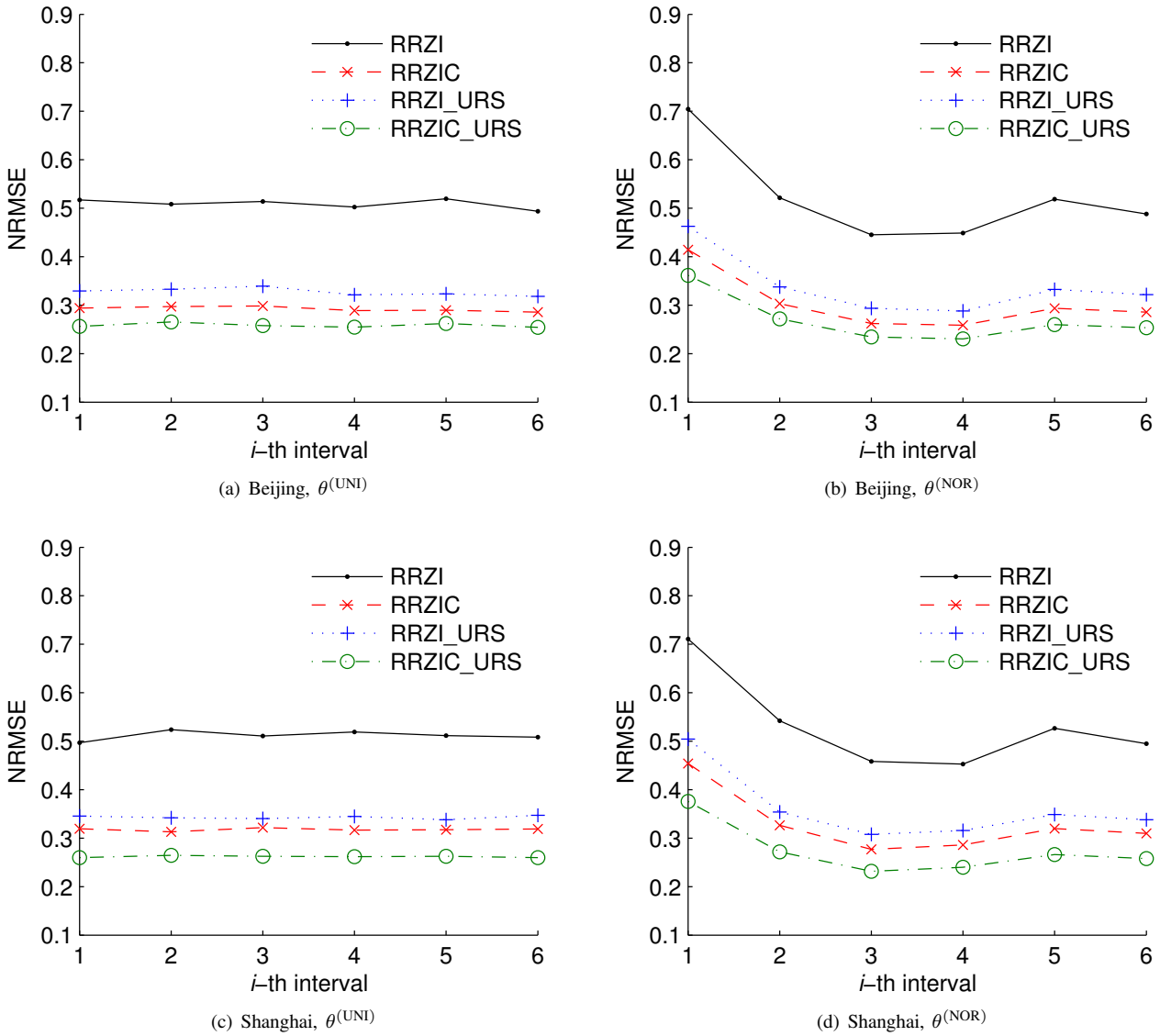
(d) CDS_NOR, $k = 10$ (e) CDS_NOR, $k = 20$ (f) CDS_NOR, $k = 50$

Fig. 10. (Baidu maps) Compared NRMSEs of estimates of restaurant PoIs' average cost for different methods under $1,000$ search queries.

mation are more accurate than RRZI and RRZI_URS. Next, we evaluate the performance of our methods for estimating the cost distribution of restaurant-type PoIs. For simplicity, we divide the values of PoIs' costs into six intervals: 1) (0, 50], 2) (50, 100], 3) (100, 150], 4) (150, 200], 5) (200, 250], and 6) (250, $+\infty$). Denote by $\theta = (\theta_1, \ldots, \theta_6)$ the cost distribution, where θ_i is the fraction of PoIs with cost within the j-th interval, $1 \leq j \leq 6$. For CDS_UNI and CDS_NOR, their associated distributions are $\theta^{(\mathrm{UNI})} = \{1/6, \ldots, 1/6\}$ and $\theta^{(\mathrm{NOR})} = (0.09, 0.016, 0.20, 0.21, 0.16, 0.18)$ respectively. Fig. 11 shows the NRMSEs of estimates of $\theta^{(\mathrm{UNI})}$ and $\theta^{(\mathrm{NOR})}$ for different methods under 1,000 queries, where we set $k = 20$ and $L = 8$. Similar to the average statistic, we can see that RRZI_URS and RRZIC_URS are more accurate than RRZI and RRZIC respectively.

C. Real Applications

In this subsection, we present our real applications on Foursquare and Google maps. The following results are obtained based on PoIs sampled on April 21–30, 2013. Foursquare provides a public API [3], which allows developers to explore a particular category of PoIs by setting the parameter "categoryId". It defines a three-level hierarchical structure of PoIs categories. The nine top-level categories



(a) Beijing, $\theta^{(\mathrm{UNI})}$

(b) Beijing, $\theta^{(\mathrm{NOR})}$

(c) Shanghai, $\theta^{(\mathrm{UNI})}$

(d) Shanghai, $\theta^{(\mathrm{NOR})}$

Fig. 11. (Baidu maps) Compared NRMSEs of estimates of restaurant PoIs' cost distribution for different methods under $1,000$ search queries.

are listed in Table IV. If "categoryId" is specified as a top-level category, all sub-categories will also match the query. Let $\mathbb{A}$ be the area with latitude from 26° N to 49° N and longitude from 125° W to 67° W, which covers almost the entire US territory. We apply our method RRZI_URS to sample PoIs within $\mathbb{A}$. For each top-level category, we sampled about 1.0×10^5 fully accessible regions. Table IV shows our estimated statistics of PoIs within $\mathbb{A}$. In terms of the average statistics (i.e., the average numbers of check-ins, tips, and users), *Food*, *Nightlife Spot*, and *Shop & Service* are the top-3 popular categories. Moreover, we can see that 25.8% of PoIs belong to the *Residence* category, which includes four sub-categories: *Living*, *Home (Private)*, *Housing Development*, and *Residential Buildings (Appartments/Condos)*. It indicates that a large number of users have revealed their exact living places to Foursquare.

Similarly we apply our methods to sample and characterize food-type PoIs within US on Google maps. The Google public API [8] returns results including PoIs' ratings and price levels. A PoI's rating and price level are calculated based on its user reviews, where a PoI's rating ranges from 1.0 to 5.0, and a PoI's price level is an integer with the range 0 (most affordable) to 4 (most expensive). Among 4.7 million food-typed PoIs sampled, 15.5% and 18.2% of PoIs' ratings and price levels are available respectively. Based on these PoIs, we estimate the PoI price level distribution and find that 0.05%, 72.5%, 25.8%, 1.6%, and 0.06% of PoIs' price levels are 0 to 4 respectively. Fig. 12 shows our estimate of the PoI rating distribution. We can see that most PoIs have high ratings. The average rating is 3.9, and nearly 90% of PoIs have ratings larger than 3.0. Thus, it is hard to evaluate a PoI's relative service quality just depending on its rating value.

Table IV
(OUR REAL APPLICATION ON FOURSQUARE) STATISTICS OF POIS IN US.

Category	Fraction (%)	Average statistics (per PoI)		
		# tips	# check-ins	# users
Food	10.4	6.6	757	304
Nightlife Spot	6.4	3.4	422	166
Shop & Service	14.1	1.9	526	141
Travel & Transport	7.3	0.8	278	77
Arts & Entertainment	3.7	1.8	370	194
College & University	2.2	1.0	353	59
Outdoors & Recreation	16.0	0.7	207	64
Residence	25.8	0.2	83	5
Professional & Others	14.0	0.7	237	45

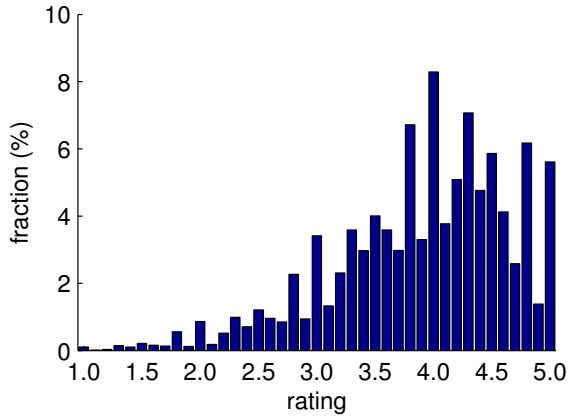


Fig. 12. (Our real application on Google maps) Rating distribution of food-type PoIs within US.

V. RELATED WORK

Recently a lot of attention has been paid to study hidden databases using public search interfaces. Previous work focuses on crawling, retrieving, and mining information from web search engines [9]–[12], text-based databases [13]–[19] and form-based [20]–[28] databases. Several sampling methods are given in [11], [29]–[33] to estimate a form-based hidden database's size (number of tuples). These methods are designed for search engines with inputs specified as categorical data, and their performances depend on the range of input values, so they cannot be directly applied to sample PoIs using map-based search engines, which have a quite large number of input values (latitude and longitude pairs within the area of interest). To address this challenge, two methods in [5] and [6] are given to sample PoIs using public map APIs. Next we discuss these two methods in detail.

A. Nearest-Neighbor Search

Dalvi et al. [5] propose a method to sample PoIs using the nearest-neighbor search (NNS) API, which returns the several closest PoIs to an input location specified as a pair of latitude and longitude. Their method works as follows: At each step, it first randomly selects a point (x, y) from $\mathbb{A}$, the area of interest. Then it finds and samples the closest PoI p to (x, y) using a NNS query. Denote by $D(p)$ the points in the plane $\mathbb{A}$ that are closer to p than the other PoIs. An example is shown in Fig. 13. Dalvi et al. [5] prove that their method samples PoIs with biases, and the probability of sampling p is $\gamma(p) = \frac{area(D(p))}{area(\mathbb{A})}$. To remove the sampling bias, a large number of NNS queries (e.g., on average 55 queries are used in [5]) are required to determine the boundary of $D(p)$ for each sampled PoI p. Therefore the method in [5] is quite expensive especially for service providers such as Foursquare, which does not provide a NNS API to the public.

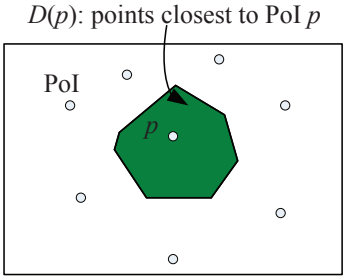


Fig. 13. $D(p)$ of a PoI p.

B. Random Region Sampling

Li et al. [6] propose a random region sampling (RRS) method to estimate the number of PoIs within $\mathbb{A}$. RRS first picks a point (x, y) from $\mathbb{A}$ at random. Then, it computes an estimate $\hat{d}$ of the PoI density around (x, y) based on the PoI densities of regions sampled previously, and initializes a new search region as a square with center (x, y) and length $\sqrt{k/\hat{d}}$, where k is the maximum number of PoIs returned for a query. If the new region has overlapping with any region sampled previously, then RRS cuts the new region to make it not collide with any sampled regions. At last RRS exhaustively searches and collects PoIs within the new region. Let $n(Q)$ be the number of PoIs within a region Q. Given $m > 0$ sampled regions Q_i $(1 \le i \le m)$, Li et al. [6] estimate $n(\mathbb{A})$ as follows

$$\hat{n}(\mathbb{A}) = \frac{area(\mathbb{A})}{m} \sum_{i=1}^{m} \frac{n(Q_i)}{area(Q_i)}. \quad (6)$$

We find that RRS has following drawbacks: 1) At the beginning of RRS, its PoI density estimate might exhibit a large error, so a large and dense region might be determined to explore. It requires a large number of queries to collect all PoIs within this region. 2) A very small region might be sampled from sparse areas due to RRS' region cutting operation. Then the number of PoIs within this small region is overestimated in equation (6). 3) $\hat{n}(\mathbb{A})$ is not an unbiased estimator of $n(\mathbb{A})$ and it might exhibit a large error. As shown in Fig. 14, $\mathbb{A}$ is specified as an $1 \times T$ quadrangle, where $T \gg 1$. $\mathbb{A}$ has two PoIs p_1 and p_2. Suppose that $k = 1$ and the distance between p_1 and p_2 is larger than 1. Let Q_1 and Q_2 be the regions sampled by RRS, which include p_1 and p_2 respectively. Clearly $area(Q_1)$ and $area(Q_2)$ are not larger than 1. Therefore, we have $\hat{n}(\mathbb{A}) \ge T \gg n(\mathbb{A}) = 2$.

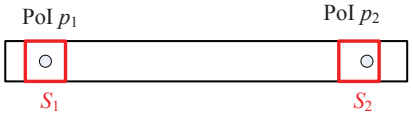


Fig. 14. An example of sampling PoIs using RRS.

VI. Conclusions

In this paper, we propose methods to sample PoIs on maps, and give unbiased estimators of PoI aggregate statistics. We show that the mix method RRZI_URS is more accurate than RRZI under the same number of queries used. When PoI count information is provided by public APIs, RRZIC_URS utilizing this meta information is more accurate than RRZI_URS. The experimental results based on a variety of real datasets show that our methods are efficient, and they sharply reduce the number of queries required to achieve the same estimation accuracy of state-of-the-art methods.

Acknowledgment

We thank the anonymous reviewers as well as Prof. Bettina Kemme and Junzhou Zhao for helpful suggestions.

References

[1] "Google maps," https://maps.google.com/, 2012.
[2] "Foursquare," http://www.foursquare.com.
[3] "search venues on foursqure," https://developer.foursquare.com/docs/venues/search, 2013.
[4] Y. Li, M. Steiner, L. Wang, Z.-L. Zhang, and J. Bao, "Exploring venue popularity in foursquare," in *The Fifth IEEE International Workshop on Network Science for Communication Networks*, 2013, pp. 1–6.
[5] N. Dalvi, R. Kumar, A. Machanavajjhala, and V. Rastogi, "Sampling hidden objects using nearest-neighbor oracles," in *Proceedings of ACM SIGKDD 2011*, December 2011, pp. 1325–1333.
[6] Y. Li, M. Steiner, L. Wang, Z.-L. Zhang, and J. Bao, "Dissecting foursquare venue popularity via random region sampling," in *Proceedings of the 2012 ACM conference on CoNEXT student workshop*, 2012, pp. 21–22.
[7] "Baidu poi datasets," http://ishare.iask.sina.com.cn/f/34170612.html, 2013.
[8] "search places on google maps," https://developer.foursquare.com/docs/venues/search, 2013.
[9] P. Rusmevichientong, D. M. Pennock, S. Lawrence, and L. C. Giles, "Methods for sampling pages uniformly from the world wide web," in *AAAI Fall Symposium on Using Uncertainty Within Computation*, November 2001, pp. 121–128.
[10] Z. Bar-Yossef and M. Gurevich, "Efficient search engine measurements," in *Proceedings of WWW 2007*, 2007, pp. 401–410.
[11] ——, "Mining search engine query logs via suggestion sampling," *Proceedings of VLDB Endowment*, vol. 1, no. 1, pp. 54–65, Aug. 2008.
[12] M. Zhang, N. Zhang, and G. Das, "Mining a search engine's corpus: efficient yet unbiased sampling and aggregate estimation," in *Proceedings of SIGMOD 2011*, New York, NY, USA, 2011, pp. 793–804.
[13] J. Callan and M. Connell, "Query-based sampling of text databases," *ACM Trans. Inf. Syst.*, vol. 19, no. 2, pp. 97–130, Apr. 2001.
[14] P. G. Ipeirotis and L. Gravano, "Distributed search over the hidden web: hierarchical database sampling and selection," in *Proceedings of VLDB 2002*, 2002, pp. 394–405.
[15] E. Agichtein, P. G. Ipeirotis, and L. Gravano, "Modeling query-based access to text databases," in *WebDB*, 2003, pp. 87–92.
[16] L. Barbosa and J. Freire, "Siphoning hidden-web data through keyword-based interfaces," in *SBBD*, 2004, pp. 309–321.
[17] A. Ntoulas, P. Zerfos, and J. Cho, "Downloading textual hidden web content through keyword queries," in *Proceedings of JCDL 2005*, New York, NY, USA, 2005, pp. 100–109.
[18] K. Vieira, L. Barbosa, J. Freire, and A. Silva, "Siphon++: a hidden-webcrawler for keyword-based interfaces," in *Proceedings of CIKM 2008*, 2008, pp. 1361–1362.
[19] X. Jin, N. Zhang, and G. Das, "Attribute domain discovery for hidden web databases," in *Proceedings of SIGMOD 2011*, 2011, pp. 553–564.
[20] N. Bruno, L. Gravano, and A. Marian, "Evaluating top-k queries over web-accessible databases," in *Proceedings of ICDE 2002*, 2002, pp. 369–380.
[21] M. Álvarez, J. Raposo, A. Pan, F. Cacheda, F. Bellas, and V. Carneiro, "Crawling the content hidden behind web forms," in *Proceedings of ICCSA 2007*, 2007, pp. 322–333.
[22] A. Calì and D. Martinenghi, "Querying the deep web," in *Proceedings of EDBT 2010*, New York, NY, USA, 2010, pp. 724–727.
[23] S. Raghavan and H. Garcia-Molina, "Crawling the hidden web," in *Proceedings of VLDB 2001*, 2001, pp. 129–138.
[24] C. Sheng, N. Zhang, Y. Tao, and X. Jin, "Optimal algorithms for crawling a hidden database in the web," in *Proceedings of the VLDB Endowment 2012*, vol. 5, 2012, pp. 1112–1123.
[25] T. Liu, F. Wang, and G. Agrawal, "Stratified sampling for data mining on the deep web," in *Proceedings of the ICDM 2010*, 2010, pp. 324–333.
[26] T. Liu and G. Agrawal, "Active learning based frequent itemset mining over the deep web," in *Proceedings of ICDE 2011*, 2011, pp. 219–230.
[27] ——, "Stratified k-means clustering over a deep web data source," in *Proceedings of KDD 2012*, 2012, pp. 1113–1121.
[28] S. Thirumuruganathan, N. Zhang, and G. Das, "Digging deeper into deep web databases by breaking through the top-k barrier," *CoRR*, vol. abs/1208.3876, 2012.
[29] A. Dasgupta, G. Das, and H. Mannila, "A random walk approach to sampling hidden databases," in *Proceedings of SIGMOD 2007*, 2007, pp. 629–640.
[30] A. Dasgupta, N. Zhang, and G. Das, "Leveraging count information in sampling hidden databases," in *Proceedings of ICDE 2009*, 2009, pp. 329–340.
[31] A. Dasgupta, X. Jin, B. Jewell, N. Zhang, and G. Das, "Unbiased estimation of size and other aggregates over hidden web databases," in *Proceedings of SIGMOD 2010*, 2010, pp. 855–866.
[32] A. Dasgupta, N. Zhang, and G. Das, "Turbo-charging hidden database samplers with overflowing queries and skew reduction," in *Proceedings of EDBT 2010*, 2010, pp. 51–62.
[33] F. Wang and G. Agrawal, "Effective and efficient sampling methods for deep web aggregation queries," in *Proceedings of EDBT/ICDT 2011*, 2011, pp. 425–436.