Sampling Hidden Objects using Nearest-Neighbor Oracles

Nilesh Dalvi Ravi Kumar Ashwin Machanavajjhala Vibhor Rastogi
Yahoo! Research
701 First Ave
Sunnyvale, CA 94089
{ndalvi,ravikumar,mvnak,rvibhor}@yahoo-inc.com

ABSTRACT

Given an unknown set of objects embedded in the Euclidean plane and a nearest-neighbor oracle, how to estimate the set size and other properties of the objects? In this paper we address this problem. We propose an efficient method that uses the Voronoi partitioning of the space by the objects and a nearest-neighbor oracle. Our method can be used in the hidden web/databases context where the goal is to estimate the number of certain objects of interest. Here, we assume that each object has a geographic location and the nearest-neighbor oracle can be realized by applications such as maps, local, or store-locator APIs. We illustrate the performance of our method on several real-world datasets.

Categories and Subject Descriptors. H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; H.4.m [**Information Systems Applications**]: Miscellaneous; G.3 [**Mathematics of Computing**]: Probability and Statistics—*Probabilistic algorithms*

General Terms. Algorithms, Experimentation, Performance, Theory

Keywords. Sampling, Voronoi cell, Nearest-neighbors, Size estimation

1. INTRODUCTION

Consider the following scenarios. The owner of a fledgling cafe chain wishes, for competitive reasons, to estimate the number of Starbucks in a strategic geographic region. A search engine company is apprehensive about the quality of its local data and especially would like to compare its coverage of hospitals with that of Bing. A restaurant portal start-up wants to estimate the fraction of restaurants in Yelp that serves Ethiopian cuisine, for business intelligence purposes. An insurance company wants to know the number of dentists in a city. A pervading theme in these scenarios, among other things, is the implicit role of geography.

These scenarios abstract the problem of estimating an aggregate statistic over an unknown collection of objects, where each object has an associated geographic location. It is crucial to note that in each of these scenarios, one does not have access to the full collection (which is often a database) in a direct manner. In fact, it is hard to imagine Starbucks or Bing or Yelp willing to share their collection. Instead they allow a web query interface to their collection: using the store locator or local or maps API, one can specify an arbitrary location as input and the (ten) closest Starbucks/hospitals/Ethiopian restaurants will be returned by the API as the output. In other words, these APIs are nearest-neighbor oracles. Furthermore, these APIs allow the location to be specified as a latitude-longitude pair, thereby allowing the query to be an arbitrary point on the map, which, for simplicity, can be treated as a plane.

The question then is how to use such a nearest-neighbor oracle in order to estimate an aggregate statistic over the object collection? Since the estimation problem can be solved by computing the estimate on a uniform random sample, an equivalent question is the following: how to sample an object uniformly at random from the object collection? Such questions have been studied in broader contexts and various guises such as estimating the size of a search engine index [14, 13, 3, 5, 1], estimating the size of the hidden Web [8, 9, 22], and measuring the recall in Web document retrieval engines [7, 12]. The techniques in these papers are based on querying the search engine with carefully constructed queries, by conducting random walks on suitable graphs, by automatically filling fields in web forms or a combination of these. In this work we propose a completely different family of solution to the aggregate estimation problem, where we crucially exploit the geometry of our setting and the availability of a nearest-neighbor oracle.

Our solution is based on a careful sampling of points from the plane and using the nearest-neighbor oracle. To illustrate the importance of exercising caution in the approach, consider the following method for estimating a statistic $f(\cdot)$ of the objects: first pick a random point p on the plane and then use the nearest-neighbor oracle to find the object d that is closest to p and use $f(d)$ as an estimator. While the points p are picked uniformly at random, unfortunately, the resulting sample $f(d)$ is not uniform. To see this, consider Figure 1, which has eight points on a plane. For a point picked uniformly at random, d_7 is more likely to be its nearest neighbor than d_1. Therefore, we need to account for this bias in order to get an unbiased estimator of $f(\cdot)$ from the collection.

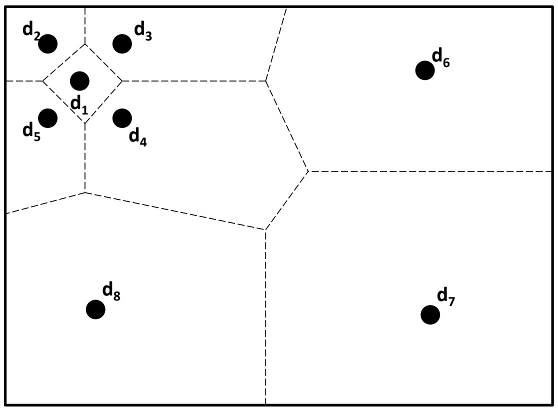

Figure 1: Voronoi partition of points $d_1, \ldots, d_8$.

1.1 Main contributions

Our main contribution is a geometric method to estimate the size and other properties of an unknown collection of objects. The method is based on considering the Voronoi partition of the plane by the objects and using the nearest-neighbor oracle (see Figure 1 for a depiction of the Voronoi partition). Suppose that for any object d, we have a black-box that computes the area of the Voronoi cell area(d) that contains d. Then, an estimate of the statistic $f(\cdot)$ can be obtained as follows: pick a random point p, use the nearest-neighbor oracle to find the closest object d, compute $f(d)$, and output $f(d)/\text{area}(d)$. For this method, we obtain analytic bounds on the variance of the estimator.

The question then is how to implement the black-box for computing the area of a Voronoi cell containing the given point? We show that this black-box can be efficiently realized by once again using the nearest-neighbor oracle. The idea behind this is a step-by-step identification of the facets of the Voronoi cell, using the information from the nearest-neighbor oracle and performing binary searches on both angles and distances. While the method in general can take time that is linear in the number of facets of the cell, the planarity of the partition comes to our rescue: on average, each cell has around six facets. While our problem is an extreme special case of estimating the volume of convex bodies (which has a rich and illustrious literature [11, 19, 20, 15]), we believe our solution is desirable since it fully exploits the simplicity of the setting.

From an implementation standpoint, we present several heuristics to improve the overall performance of our method. If the objects are distributed on the plane in a highly non-uniform manner, then the Voronoi cells will have high variance (see Figure 3), which in turn increases the variance of our estimate. To alleviate this, we present a weighted-sampling method that uses auxiliary information such as the population density; this can reduce the variance. We show another heuristic improvement to the basic area estimation algorithm that allows us to use all the top closest objects returned by the nearest-neighbor oracle instead of just the closest.

We conduct experiments on several real-world datasets to study the performance of our methods; these datasets include several popular chain-store locations, all of which can be accessed through search engine maps APIs or store-locator APIs. We study the variance, the number of samples and nearest-neighbor oracle calls for the unweighted and weighted versions of the algorithm. For the Voronoi cell area estimation, we study the effect of the oracle power (in terms of the number of nearest-neighbors it can return) and the effect of number of sides of the cell itself. Our experiments show that our methods are very efficient in practice, on average requiring around 30 nearest-neighbor oracle calls to estimate the area of a cell.

1.2 Related work

Our problem is closely related to that of estimating the overlap between search engine indices and estimating the size of the Web. There have been several papers on this topic and they are all based on the search engine query interface. Bharat and Broder [4] were the first to study the overlap of search engine indices; their method was based on issuing the same query to both search engines and measuring the fraction of the overlapping results; similar query-based methods were also studied in [16, 6, 24, 3, 5]. As noted earlier, the Web size estimation problem is equivalent to the problem of uniformly sampling the URLs. Random walks have been a natural heuristic tool for this problem and in many cases, they can be modified to obtain a provably uniform URL sample; see [17, 2, 14, 23, 3]. None of these papers, however, considers our problem of sampling objects as opposed to sampling web pages; furthermore, their methods are too general to exploit the geometry that is available in our geographic setting.

A related problem is that of estimating the size of the hidden Web, which includes hidden databases and tables. Some of these methods depend on knowing the exact number of results for a query [10]; this is typically not available in our setting. Recently, Dasgupta et al. [9] propose a random-walk based unbiased estimator for the size of a hidden database. Their method is based on a random drill-down of queries from broad to narrow, and backtracking whenever there is an "underflow." They use this method for categorical data and the performance depends on the number of values. It is not clear how to modify their method to geographic data since the number of values is quite large. A similar idea, but with rejection sampling, was used earlier [8]; but this approach needs a lot of queries. The capture-recapture method from statistics has also been used for this purpose [21, 22]. Liu et al. [18] develop special stratified sampling techniques for data mining on the deep web.

Techniques-wise, our Voronoi cell area estimation problem is related to the classical problem of estimating the volume of a high-dimensional convex body given a separation oracle; the critical parameter there is n, the number of dimensions. Dyer, Frieze, and Kannan [11] proposed a random walk method for approximating the volume; their algorithm took $O(n^{26})$ time. Subsequent improvements by Lovász and Simonovits [19] and more recently by Lovász and Vempala [20] have brought it down to $\tilde{O}(n^4)$. For more details on the history of this problem, see [15]. These algorithms are not very practical and their parameter of focus is n, which is immaterial in our case.

Algorithm 1 ESTIMATESUM

Input: A set of objects D inside $\mathbb{B}$

Output: Estimate of $f(D)$

```
1: for $i = 1$ to $N$ do
2:     Pick a random point $p_i$ in the plane
3:     Compute closest point $d_i$ to $p_i$ using $\Gamma_k$
4:     $a_i \leftarrow$ FINDVORONOIAREA($d_i$)
5:     $S_i \leftarrow (\text{area}(\mathbb{B})/a_i) \cdot f(d_i)$
6: end for
7: Return $\sum_{i=1}^{N} S_i/N$
```

2. AGGREGATE STATISTICS ESTIMATION

2.1 Problem definition

Let D be an unknown set of objects drawn from a universe $\mathcal{D}$. We want to estimate aggregate statistics over this set. More precisely, for any function $f : \mathcal{D} \to \mathbb{R}$, we want to estimate the sum

$$f(D) = \sum_{d \in D} f(d).$$

For instance, let D be the set of all restaurants in the United States; this will serve as our running example. If f is the constant function 1, then $f(D)$ corresponds to the number of restaurants in the US. Likewise, if f_I is the indicator function for Italian restaurants, then $f_I(D)$ corresponds to the total number of Italian restaurants in the US.

We assume that each object $d \in D$ is associated with a point in the Euclidean space. In our example, each restaurant can be associated with a latitude and longitude. For simplicity, we assume that the surface of the US is in the Euclidean plane. We also assume that we have a *nearest-neighbor oracle* Γ_k that, given any query point in the plane, gives us the k objects in D that are the nearest to the query point. This oracle faithfully captures a large class of hidden datasets that serve data using a map or a store-locator interface.

Given this, we wish to solve the following problem.

PROBLEM 1. *For a function f, given access to Γ_k, estimate $f(D)$ with a minimum number of calls to the oracle.*

2.2 Approach

The basic idea is as follows: generate a random point p on the plane and using Γ_k, obtain the object $d \in D$ that is closest to p. However, as we discussed in Section 1.1, this does not result in a uniform sample from D. In order to precisely compute this bias, we look at the *Voronoi cells* of objects as defined below.

DEFINITION 2 (VORONOI CELL). *Given an object $d \in D$, the* Voronoi cell *of d, denoted* vor(d)*, is the set of points in the plane that are closer to d than any other object in D.*

Voronoi cells imply a natural decomposition of the Euclidean plane.

DEFINITION 3 (VORONOI DECOMPOSITION). *Given a set of objects D, the* Voronoi decomposition *of D is a partition of the plane into the Voronoi cells of objects in D.*

Figure 1 shows the Voronoi decomposition of a set of eight points. It is easy to see that a Voronoi cell is always a convex polygon, which might be bounded (e.g., vor(d_4)) or unbounded (e.g., vor(d_2)).

For the rest of the paper we assume that there is a known bounding box $\mathbb{B}$ such that all the points in D lie inside the box. For instance, if D contains locations in the United States, we can take $\mathbb{B}$ to be the smallest rectangle enclosing the boundaries of the United States. Given a Voronoi decomposition of D, we will consider its restriction to $\mathbb{B}$; thus, every Voronoi cell becomes a bounded convex polygon.

The basic intuition behind our approach is the following simple observation.

LEMMA 4. *Let $D = \{d\}$ be a set of points inside $\mathbb{B}$. Let X be a random variable defined by the following process: uniformly select a point in $\mathbb{B}$ and compute its closest point in D. Then,*

$$\Pr[X = d] = \frac{\text{area}(\text{vor}(d))}{\text{area}(\mathbb{B})}.$$

Suppose we have an oracle FINDVORONOIAREA that, given a point d, gives us an estimate of area(vor(d)). Then, we can use Algorithm 1 to solve Problem 1. The algorithm picks a point p_i in $\mathbb{B}$ uniformly at random, finds the object d_i that is closest to p_i using Γ, and computes a_i the area of the Voronoi cell containing d_i. Define $\hat{f}(D)$ as follows.

$$\hat{f}(D) = \frac{1}{N} \sum_{i=1}^{N} \frac{\text{area}(\mathbb{B})}{a_i} \cdot f(d_i). \tag{1}$$

We then show that $\hat{f}(D)$ is an unbiased estimator for $f(D)$.

LEMMA 5. $E[\hat{f}(D)] = f(D)$.

PROOF. For $d \in D$, define

$$w(d) = \frac{\text{area}(\mathbb{B})}{\text{area}(\text{vor}(d))}.$$

Let X be the random variable as defined in Lemma 4. Then, (1) simply computes the expectation of the random variable $w(X)f(X)$, i.e.,

$$\begin{aligned} E[\hat{f}(D)] &= E[w(X)f(X)] \\ &= \sum_{d \in D} w(d) \Pr[d] f(d) \\ &= \sum_{d \in D} f(d) \\ &= f(D), \end{aligned}$$

where the third step follows from Lemma 4. □

Also, it is easy to show the following.

LEMMA 6. *The variance of $\hat{f}(D)$ is given by*

$$\frac{1}{N}\left(\left(\sum_{d \in D} w(d) f(d)^2\right) - (f(D))^2\right). \tag{2}$$

In the next section, we describe how to implement the FINDVORONOIAREA oracle.

3. VORONOI CELL ESTIMATION

In this section we consider the following problem, which we believe is of independent interest.

PROBLEM 7. *Given a nearest-neighbor oracle Γ_k over D, and a point $d \in D$, compute the Voronoi cell* vor(d) *using a minimum number of calls to the oracle.*

We first describe some known techniques from the literature that are applicable to this problem, and state their limitations. Then we describe our algorithm that makes oracle calls linear in the number of sides of the Voronoi cell to estimate it.

3.1 Techniques from literature

As we stated earlier, a Voronoi cell defines a convex region. The problem of estimating the area (or volume in the case of higher dimensions) of a convex region has been studied extensively. These work assumes that there is an unknown convex body specified solely by a *separation oracle.*

DEFINITION 8 (SEPARATION ORACLE). *A* separation oracle*, given any point, tells whether the point is inside or outside the convex body, and if it is outside, returns a hyperplane separating the point and the convex body.*

LEMMA 9. *A separation oracle can be simulated using a nearest-neighbor oracle.*

PROOF. Support we want to determine the Voronoi cell of a point d. Given any query point p, we can check whether it is inside or outside vor(d) by issuing a nearest-neighbor query from p. If the nearest point is d, then p is in the Voronoi cell of d. If the nearest point is $d' \neq d$, then p is outside the cell, and in addition, the perpendicular bisector of d and d' serves as a separating line between p and vor(d). □

As we mentioned in Section 1.2, there are polynomial-time approximation schemes for the problem of estimating the volumes of convex bodies using a separation oracle. There are three general reasons why these techniques are not suitable for our setting. First, even for planar problems, where $n = 2$, these techniques still require a very large number of oracle calls to estimate the areas with high confidence and low errors. Second, while most of the techniques deal with general convex shapes, Voronoi cells have the property of being polygons with a small number of sides, which can be exploited. Third, while a separation oracle can be simulated using a nearest-neighbor oracle that just returns a single result, these techniques cannot effectively exploit an oracle that return top-k results for $k > 1$.

In the next section, we propose a new algorithm that can compute Voronoi cells very efficiently.

3.2 Our algorithm

Let d be a point whose voronoi cell, vor(d), we want to compute. As we discussed in Section 2.2, we assume that there is a known bounding box $\mathbb{B}$ such that all the hidden points lie inside the box. Let L be the diameter of $\mathbb{B}$.

We start by defining some primitive operations that we will use to build our solution. The basic primitive is, given any point x, check whether x is inside vor(d); we denote this by the Boolean function INSIDE$_d(x)$. The second primitive is, given a point x inside vor(d) and a direction ϑ, compute the point where the ray given by x and ϑ intersects the boundary of vor(d); we denote this point by INT$_d(x, \vartheta)$. The third primitive is, given an edge a, a' of vor(d), find the direction of the next edge of vor(d) at point a'. We denote this by DIR$_d(a, a')$.

Algorithm 2 FINDVORONOIAREA(EDGECHASE)

Input: An object $d \in D$
Output: Area of vor(d)

```
1: $e_0 \leftarrow$ nearest point to $d$
2: $a_0 \leftarrow$ mid-point($d, e_0$)
3: $\vartheta_0 \leftarrow$ perperdicular direction to $(e_0, d)$
4: repeat
5:     $a_{i+1} \leftarrow \text{INT}_d(a_i, \vartheta_i)$
6:     $\vartheta_{i+1} \leftarrow \text{DIR}_d(a_{i+1}, a_i)$
7: until $(a_i, a_{i+1})$ intersects $(d, e_0)$
8: return area(polygon($a_1, a_2, \ldots$))
```

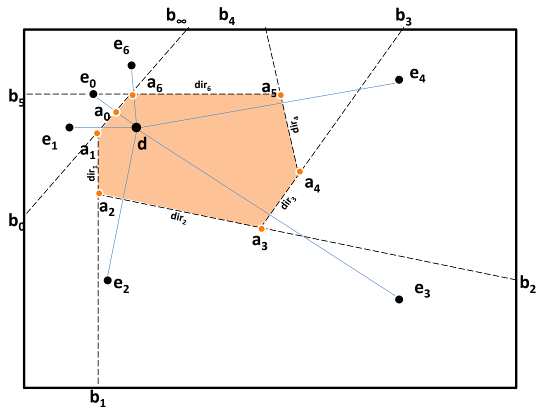

Figure 2: Working of FINDVORONOIAREA algorithm.

LEMMA 10. INSIDE$_d(x)$ *can be computed using a single call to the oracle Γ_k.*

As shown in the previous section, we simply need to query at point x and check whether the nearest point is d.

LEMMA 11. INT$_d(x, \vartheta)$ *can be approximated to within ϵ error using $O(\log(L/\epsilon))$ calls to the oracle Γ_k.*

We give a sketch of the proof here. Let x' be the point where the arc defined by x and ϑ intersects $\mathbb{B}$; clearly, x' is outside vor(d). Since vor(d) is convex, INT$_d(x, \vartheta)$ divides the line segment between x and x' into two such that one segment is completely inside vor(d) and the other is completely outside. Thus, we can use a binary search on the interval $[x, x']$ to search for INT$_d(x, \vartheta)$, using the INSIDE$_d$ primitive. Initially, the length of the interval is at most L, and each iteration reduces the interval by a factor of 2.

LEMMA 12. DIR$_d(a, a')$ *can be approximated to within ϵ error using $O(\log^2(L/\epsilon))$ calls to the oracle Γ_k.*

Again, we omit a formal proof for lack of space, but give a sketch to explain the intuition. Consider three arcs, ℓ_0, ℓ_1, ℓ_2, each starting from d, with ℓ_0 pointing towards a', ℓ_1 at an angle θ to ℓ_0 on the opposite side of a, and ℓ_2 at angle θ to ℓ_1 on the opposite side of l_0. We know that ℓ_0 intersects the boundary of vor(d) at a'. We use the INT primitive to find the points c_1, c_2 where the arcs ℓ_1 and ℓ_2 intersect the boundary of vor(d). If a', c_1, c_2 are collinear, then both c_1

and c_2 lie on the next edge of $\mathrm{vor}(d)$, and we get the direction of the next edge. If not, then we decrease θ by a factor of 2 and repeat until we are within our error bounds.

Given the primitives $\textsc{Int}_d$ and $\textsc{Dir}_d$, we can compute $\mathrm{vor}(d)$ by chasing the edges of the Voronoi polygon. To get started, we use the oracle to find the nearest point e_0 to d. It is easy to show that $a_0 = (e_0 + d)/2$ must lie on the boundary of $\mathrm{vor}(d)$ and the perpendicular bisector of d and e_0 defines one edge of the Voronoi polygon of d. We start from the point a_0 and find the first vertex in the direction of the perpendicular bisector using $\textsc{Int}_d$. Then we find the next edge direction using $\textsc{Dir}_d$, and repeat till we return to the starting point. The algorithm is outlined in Algorithm 2.

THEOREM 13. *Given a parameter ϵ,* EDGECHASE *computes the area of an s-sided* $\mathrm{vor}(d)$ *with a relative error of less than ϵ using $O(s \log^2(L/\epsilon))$ oracle calls to Γ_k, where L is the diameter of the bounding box.*

3.3 Optimizations

Replacing angle search. In practice, we can get rid of the Step 6 of Algorithm 2 and replace it by a single oracle call. Instead of using the primitive $\textsc{Dir}_d(a_i, a_{i+1})$ to find the next direction ϑ_{i+1}, we query the oracle at the point a_{i+1}. When points are in general position, we expect to obtain three points as the output: d, e_i, and e_{i+1}. For example, in Figure 2, querying for point a_1, gives d, e_0, and e_1 as output. Now the perpendicular bisector of d and e_{i+1}, gives the next direction. Again, in Figure 2, the perpendicular bisector of d and e_1 gives ϑ_1.

Refined upper and lower bounds. As mentioned in Step 5 of the EDGECHASE algorithm, we do a binary search on (a_i, ϑ_i) to find the next Voronoi vertex. As shown in Figure 2, let b_i be the intersection of the the direction $\textsc{Dir}_i$ with the boundary. The segment (a_i, b_i) can be very long, and thus the binary search can take several iterations to converge on a_{i+1}. We briefly describe an optimization to find a much smaller segment on which to do a binary search. For the optimization, we cache all the objects returned in the results by oracle Γ_k so far. Each cached object d' together with the starting object d partitions the space into two half-spaces. Points on one side are closer to d' and the points on the other side are closer to d; we call the half-space of points closer to d as the relevant half-space. For the given segment (a_i, b_i) we find the intersection of the edge with all relevant half-spaces to get a smaller segment, where the right end of the smaller segment is the farthest point from a_i on (a_i, b_i) contained in all relevant half-spaces and the left end of the smaller segment is the nearest.

4. EXTENSIONS

In this section we describe two extensions to our algorithm. First, we discuss a technique to reduce the variance in $\hat{f}(D)$ using publicly available datasets that are correlated with the distribution of D on the plane. Next, we show how to estimate the average function $f_{\mathrm{avg}}(D)$.

4.1 Reducing variance using weighted sampling

Recall from (2) that the variance of the sum estimator $\hat{f}$ depends on the variance of the Voronoi cell areas. To illustrate this, consider a distribution of objects such that the Voronoi cell of d_1 covers half of A, and the rest of the $n-1$ points in D are evenly distributed in the other half of the plane. Suppose f is the constant function (we are estimating total number of objects in D), then the variance is given by:

$$2 + \sum_{i=2}^{|D|} 2(|D|-1) - |D|^2 = O(|D|^2).$$

A skewed distribution amongst Voronoi sizes is not unrealistic. Figure 3 provides a graphical illustration of the Voronoi cell sizes for the restaurants in the United States.

The variance can be reduced by ensuring that each Voronoi cells is picked with roughly the same probability. This can be done by not picking points on the plane uniformly at random, but rather according to a distribution that resembles D to the best of our knowledge. In the case of restaurants (Figure3), note that the density of restaurants is highly correlated with the population density. Therefore, weighting each sample by an amount proportional to the population density can help reduce the variance. We call this scheme WEIGHTED.

More precisely, let ρ be any probability distribution over the plane. Given any polygon Π, define

$$\rho(\Pi) = \int_{p \in \Pi} \rho(p).$$

Then, the weighted sample estimator for $f(D)$ is given by

$$\hat{f}^{\rho}(D) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{\rho(\mathrm{vor}(d_i))} f(d_i), \qquad (3)$$

where $d_1, \ldots, d_N$ are the samples drawn in Algorithm 1 according to the distribution ρ. As before, we can show that $\hat{f}^{\rho}(D)$ is an unbiased estimator for $f(D)$.

LEMMA 14. $E[\hat{f}^{\rho}(D)] = f(D)$.

A special case of $\hat{f}^{\rho}(D)$ is the uniform sampling over bounding box $\mathbb{B}$, where the probability $\rho(\mathrm{vor}(d_i))$ reduces to $\mathrm{area}(\mathrm{vor}(d_i))/\mathrm{area}(\mathbb{B})$.

We compare the weighted ($\hat{f}^{\rho}$) and uniform ($\hat{f}$) estimators in Section 5.

4.2 Estimating averages

We may also want to compute average statistics over the set, i.e., for any function f, we want to compute

$$f_{\mathrm{avg}}(D) = \frac{f(D)}{|D|}.$$

For example, if f is the indicator function for *Italian* restaurants, then f_{avg} computes the fraction of restaurants that are *Italian*.

One could estimate the average value of a function on D ($f_{\mathrm{avg}}(D)$) in one of two ways. The first method is to use Algorithm 1 and estimate the average using the following estimator:

$$\hat{f}_{\mathrm{avg}}(D) = \frac{\sum_{i=1}^{N} f(d_i)/a_i}{\sum_{i=1}^{N} 1/a_i}. \qquad (4)$$

Unfortunately, (4) is not an unbiased estimator. For instance, when $N = 1$, the expected value of $\hat{f}_{\mathrm{avg}}(D)$ is given by

$$\sum_i \frac{a_i}{A} \cdot f(d_i) \neq f_{\mathrm{avg}}(D).$$

Nevertheless, we can show that $\hat{f}_{\text{avg}}(D)$ is asymptotically unbiased, i.e., as the number of samples tends to infinity, the estimate tends to the actual value.

LEMMA 15. $\lim_{N\to\infty} \hat{f}_{\text{avg}}(D) = f_{\text{avg}}(D)$.

PROOF. When $N \to \infty$, for each object d_i, we expect $N \cdot a_i/\text{area}(\mathbb{B})$ samples to be drawn from the Voronoi cell of d_i. Therefore,

$$\lim_{N\to\infty} \frac{\sum_i N \frac{a_i}{\text{area}(\mathbb{B})} \cdot \frac{1}{a_i} f(d_i)}{\sum_i N \frac{a_i}{\text{area}(\mathbb{B})} \cdot \frac{1}{a_i}} = f_{\text{avg}}(D). \quad \square$$

The second method is to evaluate the average on a uniform random sample of objects from the database. A uniform random sample of objects from D can be computed using *rejection sampling*, which is a standard technique to generate uniform random samples from biased random samples. Use Algorithm 1 to sample $d \in D$; retain d in the sample with probability $\propto \frac{1}{a_i}$. An average computed on this sample is indeed an unbiased estimator of f_{avg}. However, this would require many more oracle calls per sample.

5. EXPERIMENTS

We first describe the setup of our experiments and our datasets.

Datasets. We consider several datasets, each containing objects having a geographical location in the United States. Our primary dataset is called `Restaurants`, which is a collection of 182,510 restaurants in United States from Yahoo! Local (`local.yahoo.com`). Yahoo! Local serves data using a web interface that allows users to specify any location and returns the restaurants closest to the location. In addition, we consider several other datasets. These include `Starbucks`, containing the locations of Starbucks outlets, `McDonalds`, containing the locations of McDonalds outlets, `Dentists` containing locations of dentists in United States, `Walmart`, containing Walmart store locations, `GasStations`, containing locations of gas stations in United States, and finally, `Random`, which is a synthetic dataset that we created by choosing 10,000 points uniformly and randomly across the entire United States. Each of them, with the exception of `Random`, is a real dataset accessible via a corresponding interface that supports nearest-neighbor queries, e.g., `Starbucks` has a locator page `http://www.starbucks.com/store-locator` that, given any point, returns the closest cafes to that point.

To obtain the ground truth for the evaluation of our methods, we obtained a complete copy of each dataset, either by a full crawl or by acquiring them from known sources. Table 1 lists the size of each dataset.

For each dataset, we estimate the size of the dataset using our sampling techniques. To compute the Voronoi cells using the appropriate nearest-neighbor oracle, we use the EDGECHASE algorithm. We consider two variants for sampling points as discussed in the paper: UNIFORM (given by (1)), where we uniformly sample points from United States, and WEIGHTED (given by (3)), where we sample points from United States based on population densities. For population densities, we use the publicly available 2010 Census data containing the list of counties in USA along with their geographical center and the population.

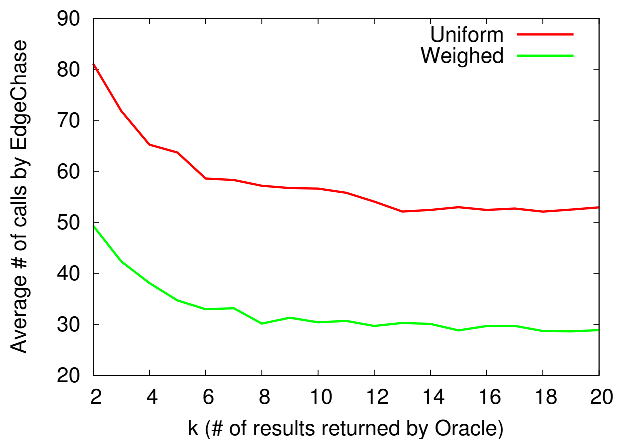

Figure 4: Number of oracle calls vs. the number of results returned by oracle.

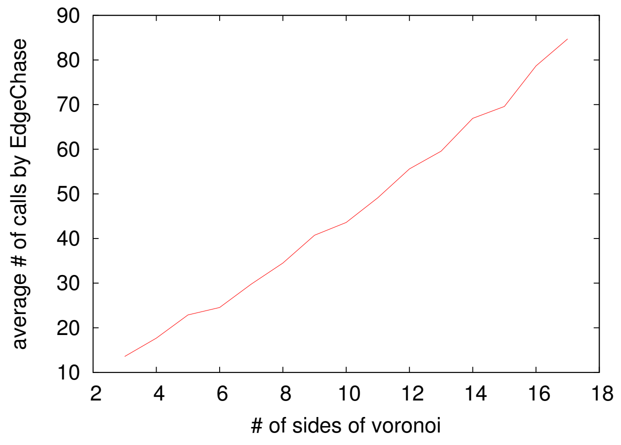

Figure 5: Number of oracle calls vs. the number of sides of the Voronoi polygons.

5.1 Computing Voronoi cells

In this experiment, we evaluate the performance of our EDGECHASE algorithm for computing Voronoi cells.

We first consider the `Restaurants` dataset, which contains 182510 points, and hence, divides the plane into 182510 Voronoi cells. The average number of sides is 6 and the max number of sides is 29. Figure 3 shows the Voronoi decomposition of `Restaurants`. We consider a top-10 oracle, that, given any point p, returns the 10 nearest restaurants to p. In the first experiment, we study the average number of oracle calls needed using the top-10 oracle to compute the Voronoi cell as a function of the number of sides of the cell. For this experiment, for each t, we randomly pick restaurants $r_1^t, \ldots, r_m^t$ whose true Voronoi cells have exactly t sides, and apply EDGECHASE. We report the number of oracle calls required averaged across $r_1^t, \ldots, r_m^t$. Figure 5 plots the resulting number of oracle calls for each t. We observe that EDGECHASE makes a very small number of calls to the oracle, and has almost linear dependence on the number of sides of the polygon.

In the second experiment, we vary the number of top-k results returned by the oracle, and study the effect of k on

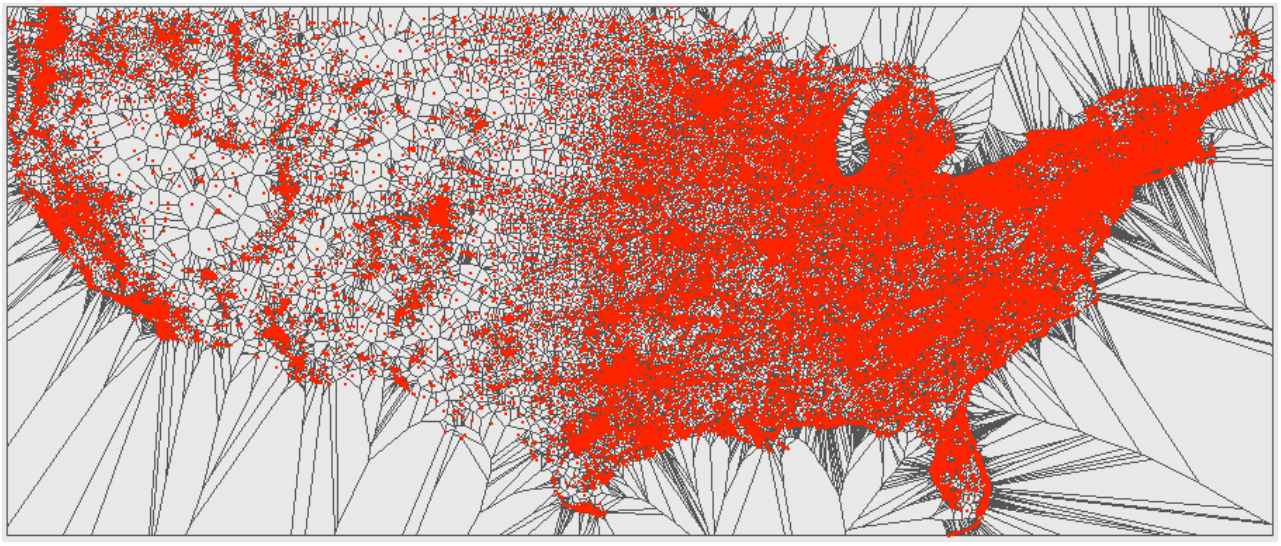

Figure 3: The Voronoi decomposition of the restaurants in the US.

Dataset	Size	Voronoi edges		Uniform sampling		Weighted sampling	
		max	avg	# samples	avg oracle calls per sample	# samples	avg oracle calls per sample
`Restaurants`	182510	29	5.99	434	55	72	30
`Starbucks`	6914	14	5.98	1895	55	24	38
`McDonalds`	14124	18	5.99	466	52	4	32
`Walmart`	4722	14	5.98	160	51	9	34
`GasStations`	82636	27	5.99	371	52	7	28
`Dentists`	66663	21	5.99	795	57	10	32
`Random`	9999	12	5.96	4	33	3096	34

Table 1: Summary of experiments on all datasets. For each sampling technique, it lists the # of samples required to estimate size with less than 10% error, and the average number of oracle calls per sample.

the number of calls made by EDGECHASE. Here, instead of picking a random Voronoi cell with a given number of sides, we instead randomly sample points according to either UNIFORM or WEIGHTED, and compute the average number of oracle calls as a function of the number of results k returned by the nearest-neighbor oracle. The resulting plot is shown in Figure 4. We observe that for each k, average number of calls for WEIGHTED is much less than that for UNIFORM. This can be explained as follows: WEIGHTED tends to draw points from dense regions, where the average number of sides of Voronoi polygons is small, and results in lesser number of oracle calls. UNIFORM, on the other hand, frequently draws points from empty spaces, which tend to have many-sided Voronoi polygons.

For both the curves, we observe that the number of calls fall sharply at the beginning, but after $k = 6$, increasing k does not help much. We postulate the following reason for this behavior. A significant fraction of Voronoi cells in the `Restaurants` dataset has at most 6 sides. Hence, in most cases, an object d's six nearest neighbors are sufficient to completely determine vor(d). Thus, the additional nearest neighbors provided by a top-7 or a top-10 oracle might provide no additional help in computing the Voronoi cell. Hence, the trend seen in Figure 4.

Table 1 shows the average number of calls per sampling step for the two sampling methods for each of the datasets using a top-10 oracle. Note that except in the case of `Random`, the WEIGHTED algorithm makes significantly lesser number of oracle calls per sample compared to UNIFORM. This is because, in `Random`, most Voronoi cells have a small number of edges (maximum is only 12). Moreover, unlike in `Restaurants`, the weights used by WEIGHTED are no longer correlated to Voronoi cell sizes or the number of edges. Therefore, both the sampling strategies see roughly the same distribution of Voronoi cells edges, explaining the same number of oracle calls per sample.

5.2 Estimating size of datasets

Next, we look at using our sampling techniques to estimate the size of a dataset. Again we look at both UNIFORM and WEIGHTED. While both the expressions provide an unbiased estimate of the size, they have different variance characteristics, and we compute the variance in the estimation as a function of the number of samples. Figure 6 contains the resulting plot for `Restaurants`.

We see that even UNIFORM requires only a small number of samples for obtaining a good estimate. E.g., to obtain the size within an error of 10%, one only requires 434 samples. Also, we can see that UNIFORM indeed has a much lower variance, and can estimate the size of the dataset within 10% error using only 72 samples.

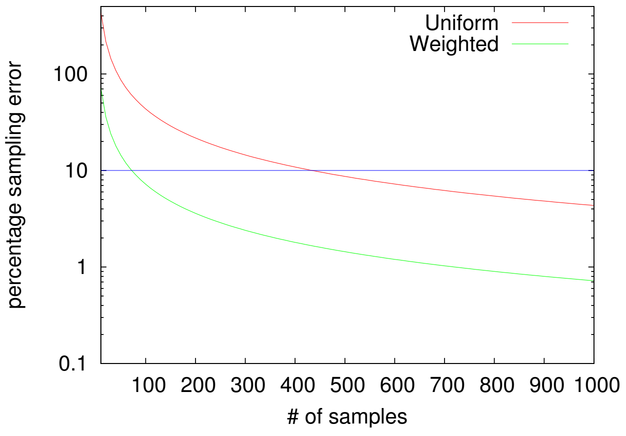

Figure 6: Size estimation of restaurants.

Table 1 contains a summary of the number of samples required for each approach for all the datasets. Except `Random`, we see a similar trend in each dataset. The number of oracle calls using uniform sampling is consistently around 55 and using weighted sampling, its around 33. Further, weighted sampling vastly reduces the number of samples required to get a good size estimate. As noted before, `Random` has a very different behavior. The uniform sampling has very low variance, but the weighted sampling requires a very large number of samples, as it weights the samples in a manner that is completely uncorrelated to the distribution of Voronoi cells.

We also see that the number of samples required does not correlate with the size of the dataset. For example, even though the number of objects in the `Starbucks` and `Walmart` datasets is comparable, the number of samples required for `Starbucks` (especially under uniform sampling) is significantly higher. A possible explanation for this behavior is that `Walmart` locations are much more spread out than `Starbucks`. In fact the distribution of Starbucks locations is even more skewed than population density – Starbucks locations are even more denser in densely populated areas. This explains why more samples are required for `Starbucks` even in the WEIGHTED case.

6. CONCLUSIONS

In this paper, we considered the problem of estimating aggregates over hidden data objects on a plan using a top-k nearest neighbor oracle. This required us to develop techniques for sampling uniformly from the set of objects. The key technical contribution of this paper lies in a novel algorithm EDGECHASE to compute the area of a Voronoi cell of an object using the nearest neighbor oracle. That the number of oracle calls made by EDGECHASE is linear in the number of edges of the Voronoi cell, makes this technique efficient. Using this tool an aggregate can be estimated by sampling a random point, finding the nearest object and dividing the value of the function at that object by the area of the Voronoi cell of the same object. We also present several heuristics to improve the overall performance of our method in practice, like a weighted sampling variant that uses auxiliary information such as population density to reduce the variance of our estimates, and an optimization to fully utilize all the k results returned by the nearest neighbor oracle.

Acknowledgments

We thank Prabhakar Raghavan and Raghu Ramakrishnan for several discussions that lead to us studying this question.

7. REFERENCES

[1] J. Bar-Ilan. Size of the web, search engine coverage and overlap — Methodological issues. Unpublished, 2006.

[2] Z. Bar-Yossef, A. Berg, S. Chien, J. Fakcharoenphol, and D. Weitz. Approximating aggregate queries about web pages via random walks. In *Proc. 26th VLDB*, pages 535–544, 2000.

[3] Z. Bar-Yossef and M. Gurevich. Random sampling from a search engine's index. *J. ACM*, 55(5), 2008.

[4] K. Bharat and A. Z. Broder. A technique for measuring the relative size and overlap of public web search engines. *Computer Networks*, 30(1-7):379–388, 1998.

[5] A. Z. Broder, M. Fontoura, V. Josifovski, R. Kumar, R. Motwani, S. U. Nabar, R. Panigrahy, A. Tomkins, and Y. Xu. Estimating corpus size via queries. In *Proc. 15th CIKM*, pages 594–603, 2006.

[6] J. Callan and M. Connell. Query-based sampling of text databases. *ACM TOIS*, 19(2):97–130, 2001.

[7] S. Clarke and P. Willett. Estimating the recall performance of web search engines. *Aslib Proceedings*, 49(7), 1997.

[8] A. Dasgupta, G. Das, and H. Mannila. A random walk approach to sampling hidden databases. In *Proc. SIGMOD*, pages 629–640, 2007.

[9] A. Dasgupta, X. Jin, B. Jewell, N. Zhang, and G. Das. Unbiased estimation of size and other aggregates over hidden web databases. In *Proc. SIGMOD*, pages 855–866, 2010.

[10] A. Dasgupta, N. Zhang, and G. Das. Leveraging COUNT information in sampling hidden databases. In *Proc. 25th ICDE*, pages 329–340, 2009.

[11] M. E. Dyer, A. M. Frieze, and R. Kannan. A random polynomial time algorithm for approximating the volume of convex bodies. *J. ACM*, 38(1):1–17, 1991.

[12] M. Fricke. Measuring recall. *Journal of Information Science*, 24(6), 1998.

[13] A. Gulli and A. Signorini. The indexable web is more than 11.5 billion pages. In *Proc. 14th WWW (Special interest tracks & posters)*, pages 902–903, 2005.

[14] M. R. Henzinger, A. Heydon, M. Mitzenmacher, and M. Najork. On near-uniform URL sampling. *Computer Networks*, 33(1-6):295–308, 2000.

[15] M. Jerrum. *Counting, Sampling, and Integrating: Algorithms and Complexity*. Birkhauser, 2003.

[16] S. Lawrence and C. Giles. Accessibility of information on the web. *Intelligence*, 11(1):32–39, 2000.

[17] S. Lawrence and C. L. Giles. Searching the world wide web. *Science*, 280(5360):98–100, 1998.

[18] T. Liu, F. Wang, and G. Agrawal. Stratified sampling for data mining on the deep web. In *Proc. 10th ICDM*, pages 324–333, 2010.

[19] L. Lovász and M. Simonovits. Random walks in a convex body and an improved volume algorithm. *Random Structures and Algorithms*, 4:359–412, 1993.

[20] L. Lovász and S. Vempala. Simulated annealing in convex bodies and an $\tilde{O}(n^4)$ volume algorithm. *JCSS*, 72(2):392–417, 2006.

[21] J. Lu. Efficient estimation of the size of text deep web data source. In *Proc. 17th CIKM*, pages 1485–1486, 2008.

[22] J. Lu. Ranking bias in deep web size estimation using capture recapture method. *Data Knowl. Eng.*, 69(8):866–879, 2010.

[23] P. Rusmevichientong, D. M. Pennock, S. Lawrence, and C. L. Giles. Methods for sampling pages uniformly from the world wide web. In *AAAI Fall Symposium on Using Uncertainty Within Computation*, pages 121–128, 2001.

[24] S. Wu, F. Gibb, and F. Crestani. Experiments with document archive size detection. In *Proc. 25th ECIR*, pages 294–304, 2003.