Hyper-Local, Directions-Based Ranking of Places

Petros Venetis
Stanford University
venetis@cs.stanford.edu

Hector Gonzalez
Google Inc.
hagonzal@google.com

Christian S. Jensen
Aarhus University
csj@cs.au.dk

Alon Halevy
Google Inc.
halevy@google.com

ABSTRACT

Studies find that at least 20% of web queries have local intent; and the fraction of queries with local intent that originate from mobile properties may be twice as high. The emergence of standardized support for location providers in web browsers, as well as of providers of accurate locations, enables so-called hyper-local web querying where the location of a user is accurate at a much finer granularity than with IP-based positioning.

This paper addresses the problem of determining the importance of points of interest, or places, in local-search results. In doing so, the paper proposes techniques that exploit logged directions queries. A query that asks for directions from a location a to a location b is taken to suggest that a user is interested in traveling to b and thus is a vote that location b is interesting. Such user-generated directions queries are particularly interesting because they are numerous and contain precise locations.

Specifically, the paper proposes a framework that takes a user location and a collection of near-by places as arguments, producing a ranking of the places. The framework enables a range of aspects of directions queries to be exploited for the ranking of places, including the frequency with which places have been referred to in directions queries. Next, the paper proposes an algorithm and accompanying data structures capable of ranking places in response to hyper-local web queries. Finally, an empirical study with very large directions query logs offers insight into the potential of directions queries for the ranking of places and suggests that the proposed algorithm is suitable for use in real web search engines.

1. INTRODUCTION

With the proliferation of high-end mobile devices and improvement to the wireless infrastructure, accessing the Internet from mobile phones is becoming commonplace. Current projections are that in the near future, the sales of high-end mobile devices with capable browsers will outnumber the sales of desktop computers, and the mobile Internet is slated to become bigger than the conventional, wired Internet. Recent studies find that at least 20% of web queries have local intent, and the fraction of queries with local intent that originate from mobile properties may be twice as high.

Proceedings of the VLDB Endowment, Vol. 4, No. 5

The emergence of standardized support for location providers in web browsers, as well as of providers of accurate locations, enables so-called *hyper-local* web querying where the location of a user is accurate at a much finer granularity than with IP-based positioning.

This paper addresses a fundamental problem that arises in mobile search: determining the importance of points of interest in local-search results. We consider a setting in which a mobile user queries the web for popular near-by places. The user may optionally also specify a category of places of interest (e.g., restaurants, attractions, shopping). We assume that a collection of geo-referenced, near-by places is available, but we are agnostic as to how they were collected (e.g., from a yellow pages service or a web crawl). The challenge is to produce a good ranking of the places and to do so in a scalable fashion so that the places can be ranked for any query location. While previous work on local-web querying (e.g., [3, 22]) assumes IP-based positioning and therefore knows the location of the user within a ZIP code, we assume that the user's location is known within tens to hundreds of meters, leading to a fundamentally different problem.

Our first contribution is to show that logs of *directions queries* are a promising source of user-generated content that may enable the desired ranking. A directions-query log entry contains an origin, a destination, and the time at which the query was issued. With the rise in queries to map-based services, such logs are voluminous. In particular, they are much larger and much more current than online reviews of points of interest, which would be another source for determining user interest in places. Although one can imagine many reasons for issuing directions queries (e.g., to determine mileages for expense reports) a query suggests that there is an interest of some kind in the destination. Also, aggregate figures across large numbers of queries serve to eliminate noise (as confirmed by our studies).

Intuitively, we interpret a query for directions to a point b as a vote by a user that point b is of interest. However, we also allow our scoring functions to take several other aspects into consideration. First, not all votes are equal: a vote from an origin farther away from b could be considered as more important than a vote from nearby b, as it suggests that the user was willing to travel a longer distance to reach b. Second, when candidate places are considered in response to a point-of-interest query from a user location l, the distance to a should be a factor in the ranking. Third, as directions queries have a temporal component, it is relevant to consider whether temporal patterns in the directions queries can be exploited. For example, users may have different intentions in the morning than in the evening, or on weekdays versus weekend days.

We believe that this is the first research that explores the potential of directions logs for the ranking of places in local search. Studies have been reported that aim to identify places that individuals visit

from GPS logs and related data sources [7, 17, 18, 19, 25]. Some of these studies also study link-based ranking of places. This paper does not consider the identification of places; and it considers ranking based on directions queries, not link-based techniques.

Our second contribution is a scalable query processing technique for a general hyper-local location-ranking architecture. The architecture, shown in Figure 1, operates in two phases. In the offline phase, we compute a score for each place in our directory. In the online phase, we rank places relative to a user's query location and the distance the user is willing to travel [16].

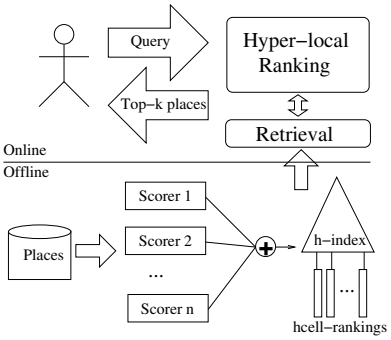


Figure 1: Architecture for hyper-local ranking

The offline score can combine scores from several scorers, thus leveraging different information sources. In addition to driving directions, we can consider the rankings of web pages (or just PageRank) associated with places, or reviews that places have received in Google or `yelp.com`. Combining multiple scorers has been considered in previous work [11] and is not considered here.

Our query processing technique superimposes a grid on the space. For each grid cell, called an *hCell*, we store a list of the places that fall into the cell, sorted by their (offline) score. At query time, we retrieve the set of hCells relevant to the query and then compute the top-k places, adjusting the score of each candidate place according to its distance to the user's query location.

We report on experiments that show that real (driving) directions logs are a viable source for the scoring of places and that the proposed framework scales to large data sources and is capable of fast query response, enabling integration with an existing search engine infrastructure.

Roadmap: Section 2 presents the problem setting and proposes techniques that use driving directions for place ranking. Section 3 then discusses the spatial indexing required to follow our work and presents a baseline algorithm and a more efficient algorithm for place ranking. Section 4 considers the suitability of a real driving directions log for ranking of places. Section 5 presents performance experiments for the ranking algorithms we propose. We cover related work in Section 6 and conclude in Section 7. An appendix contains a detailed description of our data sources, some examples of ranking functions that take into account the directions query logs, pseudocode for the baseline algorithm, proofs, and additional experimental results.

2. RANKING OF PLACES

2.1 Problem Setting

A query takes four arguments: (i) the user's location, (ii) the time that the query is issued, (iii) the maximum distance the user is willing to travel, and (iv) a query string. The query string can be one of a predefined set of categories, e.g., museums, restaurants, and shopping. This query is capable of supporting a class of popular services. The result of a query is a ranked list of nearby places that match the query string, where the ranking aims to reflect the interestingness of the places.

To formalize the key concepts of the problem setting, we assume that a set of *places* $\mathcal{P}$ is given and has signature $\mathcal{L} \times \mathcal{C}$, where $\mathcal{L}$ is the set of locations in Euclidean space and $\mathcal{C}$ is a set of categories (e.g., restaurant, cafe). Places thus model points of interest that a user can visit, and they are categorized. We use Google's business directory as $\mathcal{P}$.

We also assume that a set $\mathcal{D}$ of *directions query log entries* is given, and we use the Google Maps query log. An entry $d \in \mathcal{D}$ is a record $\langle t, a, b, \|a, b\| \rangle$ with signature $\mathcal{T} \times \mathcal{L} \times \mathcal{L} \times \mathbb{R}_+$. Here, $\mathcal{T}$ is the domain of query times and t is the time when the query was issued; $\mathcal{L}$ was defined earlier, and a is the "from" and b is the "to" of the query; $\|a, b\|$ is the distance between a and b.

A user query $q \in \mathcal{Q}$ is a quadruple $\langle l, t, D, \vec{q} \rangle$ with signature $\mathcal{L} \times \mathcal{T} \times \mathbb{R}_+ \times \mathcal{Q}_q$. Here, $\mathcal{Q}_q$ is the set of all possible strings, i.e., a set of predetermined categories $\mathcal{C}$, with the empty string denoting all categories. Also, $q.D$ is the distance that the user is willing to travel. This parameter may be specified by the user, or it may be derived from the travel pattern of the user (e.g., walking, driving) using the query logs.

2.2 Scoring Functions

We use a scoring function $\mathcal{S} : \mathcal{Q} \times \mathcal{P} \times \mathcal{D}$ to rank an argument place according to a set of log entries and a user query. Since we use scoring functions for ranking, the absolute values returned by a scoring function do not matter—only the relative values matter.

Specifically, our framework supports the following kind of scoring function:

$$\mathcal{S}(q, p, \mathcal{D}) = \mathcal{S}(p, \mathcal{D}) \times weight_{q.D}(\|q.l, p.l\|).$$

This function separates the offline scoring from the online scoring, as discussed in Section 1. Thus, $\mathcal{S}(p, \mathcal{D})$ represents the offline part of the scoring, while the weight $weight_{q.D}(\cdot)$ is computed in the online part. This arrangement enables maximum precomputation for the problem considered. At query time, the offline score needs only be adjusted by a simple multiplication with a score that depends on the distance between a query and a place. This is important to achieve low query latency.

We restrict the function $weight_{q.D}(\cdot)$ to be non-increasing, as we discuss in Section 3.3. This simply means that a user is assumed to prefer to travel a shorter rather than a longer distance to achieve the same reward. This assumption also underlies k nearest neighbor querying. Appendix A presents a range of possible scoring and weight functions and an experimental comparison of these functions.

2.3 Time-Aware Scoring

Time-aware scoring may be applied to any scoring function we just described. To account for the intuition that users' behavior vary across the day and between weekdays and weekend days, we assign different weights to different directions queries according to the temporal match between their time and the time in the user's query.

Thus, we can obtain a temporal counterpart of each instance of the previous kind of scoring function as follows:

$$\begin{aligned} \mathcal{S}'(q, p, \mathcal{D}) \quad = \quad & \mathcal{S}(q, p, \mathcal{D}) + \alpha \times \mathcal{S}^{\mathrm{tod}(q.t)}(q, p, \mathcal{D}) + \\ & + \beta \times \mathcal{S}^{\mathrm{dow}(q.t)}(q, p, \mathcal{D}). \end{aligned}$$

Here, α and β are positive constants. Function $\mathcal{S}^{\text{tod}(\cdot)}$ restricts the scoring function to the query log tuples that occur during a particular time of the day. We divide a day into disjoint intervals for morning (06.00–10.00), lunch (10.00–14.00), afternoon (14.00–17.00), dinner (17.00–20.00), evening (20.00–23.00), and night (23.00–06.00). Similarly, function $\mathcal{S}^{\text{dow}(\cdot)}$ restricts the scoring function to the query log tuples that occur during particular sets of days of the week: either weekends or weekdays.

We note that the directions query logs can be used in this time-aware setting in contrast to other methods used until now (e.g., number of reviews, sentiment of reviews).

3. RANKING ALGORITHMS

We assume that a set of places has been retrieved and scored in an offline phase. We describe two algorithms that then compute the top-k most relevant places for a query issued by a user at a known location.

We first provide background information on our spatial indexing approach. Because major search engines (e.g., Google) use space-filling curves to index their geo content, our methods rely on these techniques for better integration.

3.1 Spatial Indexing

To efficiently retrieve places located in a given geographic region, we use a space-filling curve that maps locations on Earth to a one dimensional curve.

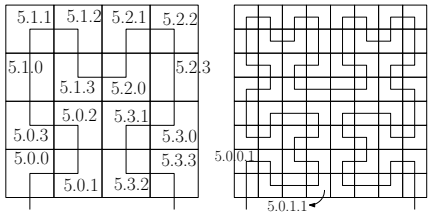


Figure 2: Example of a Hilbert curve at two levels

We project the Earth's surface onto the six faces of a unit cube (applying standard transformations to account for distortion). On each face of the cube we use the Hilbert curve [21] to map points from 2-D to 1-D. The Hilbert curve is defined recursively, as exemplified in Figure 2 that considers part of the 5^{th} face of the Earth. Each cell is subdivided into four smaller cells at each subsequent level (identified by the numbers 0, 1, 2, 3 every time). The process stops when the desired granularity has been achieved. We use cells from level 1 (a face in the unit cube) to level 23 (approximately a 1 m $\times$ 1 m cell). We refer to cells in a Hilbert Curve as hCells in the following. Also, we denote as hCellID_l the ID of a hCell at level l. Example hCellIDs can be seen in Figure 2 for $l = 3$ (left) and $l = 4$ (right)[1]. Advise Appendix B for additional details.

Each (latitude, longitude) location is mapped to an hCell at level 23. Addresses are mapped into hCells by using a standard *geocoder*. A geocoder is a piece of software that maps various geographic data (e.g., address, ZIP code) to (latitude, longitude) pairs.

3.2 Table-Based Algorithm (Baseline)

We first describe a table-based ranking algorithm. The algorithm uses a table `BusinessListing`, which contains information about $\mathcal{P}$. The schema of `BusinessListing` is `(PlaceID, Location, Category, Score)`, where `PlaceID` is an ID of a place, `Location` is its location (expressed as an hCell_{23} identifier), `Category` is the category to which it belongs (e.g., restaurant), and `Score` is the offline score assigned to this place (see Figure 1), which can be the result of a combination of scores from multiple scorers.

The table-based algorithm takes as input a subset of the tuples in `BusinessListing`, namely the places that are within the range the user is willing to travel. We use standard spatial indexes to find such tuples/places. From this subset of tuples, the algorithm computes a place ranking by sorting the result tuples according to their distance-weighted scores (online part of Figure 1). The pseudocode can be found as Algorithm 2 in Appendix C. The following example illustrates the ranking.

EXAMPLE 1. Assume that we have a database with the businesses in Table 1 and that Table 2 contains the information necessary for the algorithm, computed from the query log.

Table 1: Example business directory

Loc	Name	Type	Category
5.1...	Petros' place	Greek restaurant	Restaurant
5.1...	Christian's place	Danish restaurant	Restaurant
5.0...	Hector's place	Colombian restaurant	Restaurant
5.0...	Alon's place	Coffee shop	Restaurant
5.3...	Jack's place	American restaurant	Restaurant

Table 2: Statistics of interest for Example 1

BusName	Offline score	Distance (km)
Petros' place	1000	2.2
Christian's place	700	1.2
Hector's place	200	1.5
Alon's place	500	1.0
Jack's place	550	1.2

We assume that the query q is $\langle$5.2..., 2009/Jun/15 14:23:24.412 PST, 2 km, restaurant$\rangle$. The relevant set of places then excludes Petros' place, which is too far away (further than the 2 km that the user is willing to travel). Assuming a simple linear weighting function ($weight_{q.D}(d) = 1 - d/D$), the algorithm ranks the results as follows: Christian's place (score = $(1 - 1.2/2.0) \times 700 = 280$), Alon's place (score = $(1 - 1.0/2.0) \times 500 = 250$), Jack's place (score = $(1 - 1.2/2.0) \times 550 = 220$), Hector's place (score = $(1 - 1.5/2.0) \times 300 = 75$). □

Although this naive algorithm is fast when the set of relevant places is small (i.e., the distance the user is willing to travel is small or the density of places is low), it rapidly degrades as we need to examine more places. One reason is that the distance function call is expensive.

3.3 Threshold-Based Algorithm

The algorithm described next adopts the approach of the threshold algorithm [13] and is capable of handling a very large number of places. This algorithm integrates well with current search engines that use space filling curves to index their geo content.

Supported scoring functions. The algorithm accepts any scoring of places that is adjusted with a weighting function $weight_{q.D}(d)$ that decreases monotonically with d, i.e., for $d_i < d_j$ we always have $weight_{q.D}(d_i) \geq weight_{q.D}(d_j)$.

A simple example in our case would be a scoring function of the form: $\mathcal{S}(p, \mathcal{D}) = |\{d|\ d \in \mathcal{D} \wedge d.b = p.l\}|$, where $d.b$ is the

[1]We have that $\text{hCellID}_1 \in \{0, \ldots, 5\}$ (face of the Earth) and $\text{hCellID}_{l+1} = \text{hCellID}_l.s$ where $s \in \{0, 1, 2, 3\}$ tells us the order by which the Hilbert curve crosses the four smaller squares of the hCellID_l.

destination for the directions query d and $p.l$ is the location of the place p.[2]

The set of possible scorers is very general, and arbitrary scorers can be combined to compute a query-independent score for each place. At query time, the (offline) scores are adjusted according to the query location.

Place ranking in hCells. We assume that the space has been partitioned into hCells at a given level. For each hCell, we precompute a sorted list of places that map to it. This sorting is done offline according to $\mathcal{S}(p, \mathcal{D})$ as it is independent of the query location. Given a preselected set of categories, we carry out this procedure for each category (i.e., having places of the same category in each sorted list).

When receiving a user query that contains a location $q.l$ and the distance $q.D$ the user is willing to travel, we select the hCells that intersect the circle with center $q.l$ and radius $q.D$. Assume that we obtain n hCells and thus n ranked lists L_i, $i \in \{1, 2, \ldots, n\}$, of places. Algorithm 1 computes the top-k places from these lists and returns them in priority queue L.

The algorithm maintains a priority queue PQ of the lists L_i, where the priority of a list is the best possible score that an unseen place in the list can have. This number is easily computed from the shortest distance from the user to the hCell of the list and the top scoring unexamined place in the list. In each iteration, we pick the most promising place from the list that is the head of PQ that we (according to its offline score) have not yet examined and add it to the results (L). The algorithm terminates early if the score of the k-th element of the results L is higher than the maximum possible (online) score of the first element in the list at the head of the queue PQ.

This algorithm is an adaptation of the threshold algorithm (TA; or more specifically NRA) [13]. First, it does not rank objects for which each list contains a separate dimension; rather, each list contains completely different objects. Second, the score for each place in each list is adjusted dynamically according to its distance to the user.

The following theorems establish the main properties of our algorithm: the algorithm correctly finds the top-k list of places; and for a particular level of hCells, it examines the minimum number of places possible.

THEOREM 1. *Algorithm 1 finds the correct top-k list if function $weight_{q.D}(\cdot)$ is non-increasing, i.e., if for every $d_i \geq d_j$ we have $weight_{q.D}(d_i) \leq weight_{q.D}(d_j)$.*

THEOREM 2. *Algorithm 1 finds the top-k list in the minimum number of steps for a given partition of space (and only sorted access to our lists L_i).*

The proofs, in Appendix D, adapt proofs of similar properties by Fagin et al. [13], to our setting and assumptions.

In the experimental section, we show that it is possible to keep the ranked lists according to the offline score for *only one* particular level of the hCells; this yields good results in all possible scenarios.

EXAMPLE 2. We continue with Example 1. Assume that the businesses are located as shown in Figure 3. We use level 2 hCells, and aim to find the top-3 results.

Initially PQ and L are empty. We have three lists as input to the algorithm: L_0, L_1, and L_3. We know that the minimum distance between the user and the hCells are: $m_0 = 0.5$ km, $m_1 = 0.4$ km, and $m_3 = 0.3$ km.

[2]In our case, the scoring function in the right part of the equation depends on the query logs; thus, it is $\mathcal{S}(p, \mathcal{D})$. In general, any function that does not depend on the user's query q can be used.

Algorithm 1: Threshold-based algorithm

Input: A query $q = \langle l, t, D, \vec{q} \rangle$, lists $(L_1, \ldots, L_n)$, minimum distances for every list's hCell $(m_1, \ldots, m_n)$, k
Output: List L with all the k highest ranked places according to user query q

```
activeLists ← {1, 2, ..., n};
/* Priority queues (PQ and L) have 2
   operations: getNext() returns the next
   element from the queue and pops the element;
   insert(element, priority) inserts an element into
   the queue with a specific priority                  */
PQ ← ∅;
L ← ∅;
/* lists L_i have 3 operations: getNext()
   returns the next element in the list and
   removes the element; pollNext() returns the
   next element without removing the element
   from the list; hasMore() returns true iff
   the list is non-empty                               */
for i = 1 to n do
    /* The prioritization is over the maximum
       possible score achieved for that list           */
    p = L_i.pollNext();
    PQ.insert(i, S(p, D) × weight_{q.D}(m_i));
while activeLists.size() > 0 do
    nextListToCheck ← PQ.getNext();
    if nextListToCheck ∉ activeLists then
        continue;
    if L_{nextListToCheck}.hasMore() then
        p = L_{nextListToCheck}.getNext();
        /* notice that we are using the exact
           score                                       */
        if ‖p.l, q.l‖ ≤ q.D then
            L.insert(p, S(p, D) × weight(‖p.l, q.l‖));
        if L_{nextListToCheck}.hasMore() then
            p = L_{nextListToCheck}.pollNext();
            PQ.insert(nextListToCheck, S(p, D) ×
            weight(m_{nextListToCheck}));
    else
        activeLists.remove(nextListToCheck);
    if the k-th element in L has a score greater than the first list in
    PQ maximum expected score then
        break;
return L
```

The algorithm first initializes priority queue PQ. Taking into account the minimum distances, the lists, and the simple linear weight function from Example 1, we have:

$$\begin{aligned} PQ &= \langle (L_1, (1 - \frac{0.5}{2.0}) \cdot 1000), (L_3, (1 - \frac{0.3}{2.0}) \cdot 550) \\ &\quad (L_0, (1 - \frac{0.4}{2.0}) \cdot 500) \rangle \\ &= \langle (L_1, 750), (L_3, 467.5), (L_0, 375) \rangle. \end{aligned}$$

Since the hCell with ID 1 has the highest priority in PQ, it is examined first. List L_1 contains Petros' place first, which has a distance from user's location that exceeds $q.D$. Thus it is ignored, but the priority queue is updated. Since Christian's place is next in L_1, after this step, PQ will be $\langle (L_1, 560), (L_3, 467.5), (L_0, 375) \rangle$.

Again, list L_1 is examined as it has the highest priority in PQ. Christian's place is added to priority queue L, and $L = \langle (C, 280) \rangle$. Since we extracted list L_1 from PQ, we have $PQ = \langle (L_3, 467.5), (L_0, 375) \rangle$.

We then examine list L_0, since it is next in PQ. After one iteration of the main while loop, we get $PQ = \langle (L_0, 375) \rangle$ and $L = \langle (C, 280), (A, 220) \rangle$.

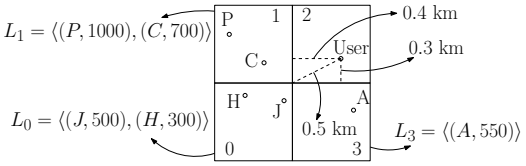


Figure 3: Threshold-based algorithm example

We now examine list L_0 since it is next in PQ. After the iteration, we get $L = \langle (C, 280), (J, 250), (A, 220) \rangle$ and $PQ = \langle (L_0, 150) \rangle$. We now have three elements in priority queue L. Since the last element has an actual score that is greater than the maximum possible score of any remaining element in list L_0, we have found the top-3 elements and are done. □

4. DIRECTIONS LOGS EVALUATION

We now describe key characteristics of our data sources (more details can be found in Appendix E) and show that the directions log can be a very effective signal for place ranking. Additional experiments can be found in Appendix F.

4.1 Key Data Source Statistics

We use a sample of queries from a travel directions log that contains all the queries issued at Google Maps during July 2009. We focus on queries in a sub-region of the United States and thus consider the business listings located in this sub-region. We limit the set of business listings to a subset of categories including museums, hotels, restaurants, night clubs, and landmarks.

The query log that we use ($\mathcal{D}$) contains queries for 18,968,123 different locations. For the categories of interest we have 151,721 business listings ($\mathcal{P}$). It is clear that most of the destinations of driving directions queries are not businesses in the directory.

We perform a join between the destination query log and the business directory, on the log destination and the business location attributes. There are 128,159 businesses for which there is at least one query with a destination matching the business location. That is, around 84.47% of all business listings in the directory are queried in the driving directions queries; 0.676% of driving directions queries in $\mathcal{D}$ have a business in $\mathcal{P}$ as their destination. Although, 0.676% is a small percentage, the actual number of queries issued for the 128,159 businesses is 49,533,223; this number is almost two orders of magnitude greater than the number of reviews for the same places (which was around 550,000). Interestingly, 22.5% of the places for which we have directions queries do *not* have any reviews, showing that the abundance of directions queries offers additional coverage.

In the business directory available, there were multiple businesses located in the same location (hCell_{23}). This happens because the business directory merges several data sources and then performs an imperfect Entity Resolution (ER). Furthermore, the information extracted from web pages is not always completely accurate. The problem of ER in the business listing is orthogonal to our study (see Appendix E.3 for additional details).

Figure 4 shows the distribution of the queries across locations for the locations that "survived" the join between $\mathcal{D}$ and $\mathcal{P}$, i.e., locations in which at least one business is located and for which at least one directions query has the location as its destination. For example, in the lower right corner of the figure, we can observe that there is a location that has received $\sim 8 \times 10^6$ queries. In the top left corner of the figure, we can see that there are around 8,000 locations that have received 1 query each.

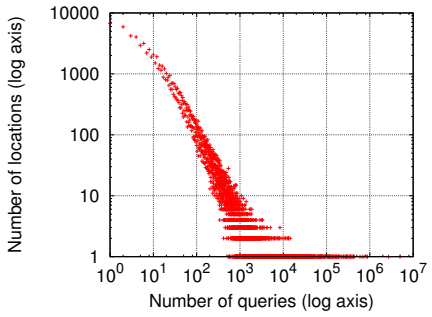


Figure 4: Distribution of queries across locations

4.2 Utility of Directions Logs

This experiment investigates the utility of the directions logs signal. We compare the sets of the top-k results produced and the rankings of the results with the corresponding sets and ranking produced by other methods.

We have created a system where a user is able to select a location the user knows very well and then select a category among the following three: restaurant, point of interest, and hotel. Finally, the user selects a maximum travel distance. The system then computes the top-5 places that three methods produce:

1. DL: number of directions log entries that have the place as their destination,
2. NR: number of reviews for the place under consideration, and
3. AR: the average score (sentiment) that reviews have assigned to the place under consideration.

Finally, the resulting 15 places (top-5 from each method) are shuffled and shown to the user, without the user knowing the origin of each place[3]. We did not incorporate distance between the user and a place since we wanted to compare the usefulness of our (offline) signal against two other standard signals. The user is then asked to evaluate these 15 places with a score between 0 and 4:

Score	Specification
0	I have no idea of what this is
1	Not interesting to most people in general and not recommended
2	Neutral to most people in general
3	An OK location to most people in general
4	Very interesting to most people in general and recommended

Since users are experts in the areas they selected (e.g., around their houses or work), their assessments are considered to be of high quality.

We had 10 users select locations, issue queries, and evaluate 15 places for each query. A total of 45 queries and 675 places were evaluated during this process.

We compared the average score that the evaluators assigned to the top-5 places of each method: our proposed signal (DL) had an average score of 1.960, while AR had 1.498, and NR had 1.453. This means that our signal is better at reporting the best places in its top-5 list than the other two signals.

We also evaluated the rankings produced by each method using the nDCG metric [15]. This metric can be defined for any list of evaluated objects; it compares the ranking a method has done (list of objects) to an optimal list of objects according to the evaluation scores. We evaluated nDCG for the top-5 places of each of the methods we examined. The metric takes values between 0 and 1: 1 means that the ranking is as expected according to the evaluations,

[3]If there are duplicates among the top-5 lists, we continue deeper into the lists so that we always show 15 places.

and 0 means that the ranking is very bad. DL had nDCG equal to 0.787, AR had 0.845, and NR had 0.827.

This study shows that driving directions logs can serve as a strong signal, on par with reviews, for place ranking. This is an important finding because log data offer a number of advantages over reviews, as mentioned in Section 4.3. The fact that the directions-based signal is comparable to the review-based signals is surprising. It is up to a scorer that takes into account multiple signals to decide how the signals should be combined based on their characteristics.

4.3 Correlation with Number of Reviews

In this experiment, we evaluate the feasibility of using driving directions logs as a proxy for the popularity/importance of a place. We compare the correlation of the driving directions based signal with the number of reviews for a place, which is a commonly accepted measure of popularity. The number of reviews was extracted via Google's business directory, and it is the total number of reviews found in various data sources on the Web.

For this experiment, we choose 100 random user locations. We then issue the category query "food" and set the distance the user is willing to drive to 2 km. Each of the 100 queries has at least 100 ranked results (otherwise, we choose a new random user location). We had to consider 514 distinct places to find 100 user locations with at least 100 food-related places within a radius of 2 km ($\sim$20% of the randomly selected places had at least 100 results).

We then find the number of reviews of each ranked place for all the queries. We partition the ranked results into batches of ten results (the top-10, the top-11–20, etc.). For each batch, we sum the number of reviews for the places ranked in those places for the 100 rankings that we have; the result can be seen in Figure 5. There is a clear correlation between the rankings of the results and the number of reviews that a place has, which shows that the directions logs are indicative of the popularity of a place.

These findings are important because driving directions logs are cheap to collect and are orders of magnitude more frequent than user reviews, which are expensive to obtain. Further, the logs provide near real-time evidence of changing sentiment, an aspect that is usually hard to capture with other signals (e.g., the reviews that a place has received; even a newly added web page of a restaurant will need time to increase its PageRank), and they are available for broader types of locations.

The scoring function we used for this experiment is $\mathcal{S}(p, \mathcal{D}) = |\{d = \langle t, a, p.l, \|a, p.l\| \rangle | \ d \in \mathcal{D}\}|$ and $weight_{q.D}(d) = 1 - \frac{d}{q.D}$. However, similar results are obtained when using other scoring and weighting functions (like the ones described in Appendix A).

5. PERFORMANCE EVALUATION

We now proceed to report on the evaluation of the runtime performance of the proposed table-based and threshold-based ranking algorithms in the presence of very large log and place databases.

In the experiments, we assume that the user issues an empty query string (retrieve all possible places around me) in order to have more results to rank. Also, the user is located in a region with high business density.

We use the same datasets as described in Section 4.1, and we run all the experiments on a single machine with two AMD dual-core Opteron CPUs (we only use one of the cores) at 2.2 GHz and with 8 GB of RAM.

For the table-based algorithm, we use MySQL as the DBMS with indexing for efficiently determining the places near a user. For the threshold-based algorithm, we load the ranked lists from the offline computation into memory.

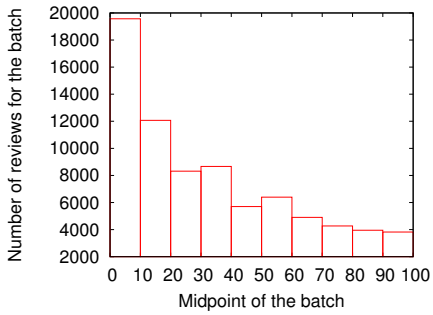


Figure 5: Number of reviews for our ranked list of results

5.1 Table- vs. Threshold-Based Algorithm

For the threshold-based algorithm we use hCells at level 7 (for reasons that will be clear later on) and compute the top-100,000 results.

Figure 6 presents the ranking time vs. the distance that the user is willing to travel. It can be observed that the table-based algorithm

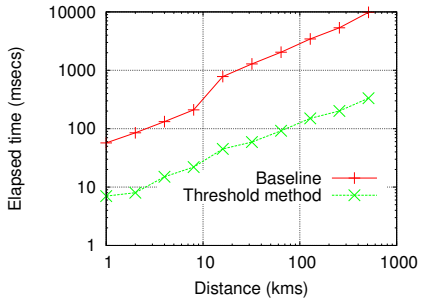


Figure 6: Performance comparison for our two algorithms

(baseline) takes up to 9.5 secs for large distances (which translates into a lot of places), while the threshold-based (efficient) algorithm takes around 350 msecs in the worst case.

5.2 Varying the hCell Level

In this experiment, we vary the hCell level from 3 to 23. We start at 3 because the areas the query logs cover are at this level. We measure the time required to find the top-1,000 results. The measurements can be seen in Figure 7 for varying distances ($q.D$) that the user is willing to travel and for all hCell levels.

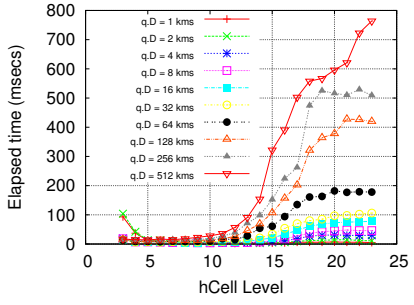


Figure 7: Elapsed time vs. hCell level

We observe that the higher the hCell level (i.e., the smaller our hCells are), the more time is required for our computation for all

distances d. This is so because we have to handle more ranked lists of results, which makes the handling of the priority queue PQ slower.

We also see that for small distances (like $q.D = 1$ km and $q.D = 2$ km), the elapsed time for ranking increases significantly if one uses a very low hCell level. This happens because we have to process large lists in order to find results that belong to the small region of interest to the user. Put differently, we have to consider places that lie in relatively large regions, and that will eventually be filtered given the user's low willingness to travel.

It seems that a very good trade-off for all possible distances is to use hCell levels 6 or 7. Thus, a search engine can keep statistics for only one of these two hCell levels and not for every possible level, thus achieving important savings in storage and computation.

5.3 Varying the Value of k

Next, we evaluate the time required to find the top-k places for a fixed maximum travel distance and different values of k. We chose the distance $q.D = 512$ km, for a user located in a region with high business density. This setting maximizes the number of places to rank, thus imposing a greater burden on the ranking algorithm. We experiment with hCell level 7, since the previous experiment shows that the threshold-based algorithm performs very well at this level.

Figure 8 presents the time required to compute the top-k lists for several values of k. As expected, the total time increases with k. We note that the process is incremental, meaning that we can first find the top-10 results and then, using the same data structures, continue running the algorithm up to the desired level of k.

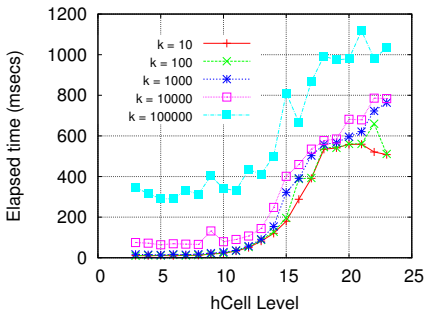


Figure 8: Time required to find the top-k results for varying values for k

5.4 Performance of Offline Procedures

For this experiment, we measure the storage and time requirements of the offline part of our two algorithms. The MySQL table `BusinessListing` is around 89 MB for the table-based algorithm (it contains some metadata about the places); the total memory required for the creation of all the ranked lists (based on the offline score) is around 100.7 MB for all possible levels we have considered.

Both algorithms need to compute the offline score; the additional cost that the threshold-based algorithm imposes is that one must create ranked lists for each hCell beforehand, which is not required in the table-based algorithm. This additional cost (when everything run on one machine) is around 25–26 secs for hCell levels 3–8 and then steadily increases to 28.3 secs for level 9, 31.7 secs for level 10, 38.8 secs for level 11, ..., 329.5 secs for level 23.

We can see that for low levels, the additional offline time is insignificant; as the level of detail increases, the overhead also increases. However, the cost is not prohibitive for our method, especially when we are at levels 6 and 7 where, as we saw earlier, we achieve the best performance. Also, this computation is done offline and its cost will be amortized across millions of queries.

6. RELATED WORK

Finding interesting locations and/or destinations: A number of studies have aimed to identify important locations using primarily GPS data. Our setting assumes that we know the important locations and that they are classified (e.g., restaurants, hotels); this information is found in business directories.

In the location identification setting, several studies aim to identify places visited by individuals from GPS logs. Ashbrook and Starner [1, 2] present techniques capable of learning significant user locations and predicting user movement based on GPS data. Brilingaite et al. [4, 5, 6] capture routes and (start and end) destinations from GPS trajectories and use these along with temporal usage information for predicting the routes and destinations of users. Experiments with data from users demonstrate that the techniques are effective. Krumm and Horvitz [17, 18] also propose techniques for destination prediction based on GPS data.

Liu et al. [19] and Cao et al. [7] extract so-called stay points from the trajectories of users; they propose clustering and reverse geocoding techniques that aim to derive semantically meaningful locations from the stay points. Zheng and Xie [25] also identify stays from GPS logs, propose a hierarchical containment ordering for the geographical extents of stays, and then mine so-called life patterns of individuals from the resulting location histories. Along the same lines Zheng et al. [25], mine locations from GPS trajectories. Here, focus is on identifying public locations, which are then ranked using link-based techniques. Cao et al. [7] offer improved link-based ranking techniques.

Our approach can use the results from this line of research as another source of places (instead of a business directory). Our approach does not use GPS data and link-based techniques for its ranking, but considers a different alternative: that of using driving directions ranking. GPS datasets typically contain many positions for few users; in contrast, directions queries are derived from larger user populations, with each user issuing few queries.

Ranking functions in a Spatial Context: Other related work deals with the problem of place ranking [24], where the ranking of objects (e.g., houses) is defined with respect to other qualities around them (e.g., restaurants, schools, hospitals) within a distance range. Also, a framework for the efficient computations of this ranking is provided. These methods do not try to determine the inherent value of a place; they determine the value of a place given the fact that other places are around it. Thus, one could apply our algorithms and then, as a next step, the algorithms described in [24].

In a more general context, substantial research has considered local search where the results that a user sees for a query depend on the user's location; the results though are documents and not ranked places/businesses (for example, one result could be a blog post about a particular restaurant), which makes the setting very different. Two recent works in this field propose indexes that are evaluated in terms of efficiently in their settings [8, 26].

In other work [10], NLP techniques are used for geo-searches on the web. The idea is to find phrases like "next to Eiffel tower" in a hotel's web document and to thus be able to answer queries like "hotel in Paris" more accurately. Again the output is ranked documents, not ranked places. Our work differs from other techniques in that no geocoding is used, but only NLP techniques. Means of detecting and extracting geographical information (e.g., addresses) from web documents has also been studied [20]. The idea is to

extract addresses, phones, ZIP codes and other useful geographical information from a web document and then to augment the location information with keywords from the content of the web document.

Query log analysis: Backstrom et al. [3] aim to determine the region of importance for a query, using a probabilistic model. This work does not use positioning, but rather depends on IP-based (10×10 km^2) positioning. We try to determine good places, which is different from trying to determine the region to which a query is relevant. Other work [22] in this line of research aims to determine the so-called dominant location for each query (the location most important for a particular query). The techniques used include query tokenization, query log analysis, and exploration of the snippets of search results for the top-k results.

Work on mobile search log analysis [9] aims to determine how users search using mobile devices. Some of the explored user behaviors are the number of keywords in queries, number of queries per user per day, and search topics.

Finally, there is some work on categorization of queries depending on their scope (local, neighborhood, etc.) [14, 23]. Other work in this area [16] explores the distances that one is willing to travel in order to get to a particular place. This work can be used in our framework to automatically find a maximum threshold for travel distance for a given query.

TA/NRA comparison: The threshold-based algorithm proposed in Section 3.3 is an adaptation of the threshold algorithm (TA) and more specifically, a variation of TA called "no random accesses" (NRA) [13]. Both TA and NRA rank objects. Each object has m scores for m different attributes. The overall score of an object is determined by a combination of the scores for the m attributes, using some monotone aggregation function. For each of the m attributes there is a ranked list of the objects according to their score for that attribute. TA and NRA find the top-k objects according to the overall score. TA and NRA differ in that NRA assumes sorted access to the ranked lists, i.e., to see the object in the l^{th} position of a sorted list, one must have seen all $l-1$ objects before it. Our algorithm differs from NRA in that each list contains different objects in our setting, not a ranking of the same objects for a different attribute. In addition, in our setting, the score for each place in each list is adjusted dynamically according to its distance to the user, which we believe has not been explored in any other work.

7. CONCLUSIONS AND DIRECTIONS

Given the ability to accurately determine the location of a mobile user and the obvious revenue possibilities involved, it is likely that answering hyper-local queries will receive increasing attention. Our first contribution is to present a new signal, directions logs, that can be used for scoring places in response to hyper-local queries. We present key statistics about these logs and show that direction queries have a good correlation with the number of reviews for places. But unlike reviews and ratings, they are inexpensive to collect and readily available.

We present an architecture for answering hyper-local queries that is the first to take into account the distance between a user and a place and to adjust the score of the place accordingly. To support this framework, we propose a threshold-based algorithm that scales very well with the number of places under consideration. This algorithm returns *exact rankings* under assumptions that are reasonable in practice.

In terms of future research, we see interesting issues in exploring additional scoring functions and methods for combining these functions. In particular, some scoring functions may be more appropriate than others for specific types of user queries. Another interesting aspect of our work is the personalization of the ranking depending on a user's specific search history.

Acknowledgments. Christian S. Jensen is an Adjunct Professor at University of Agder, Norway.

8. REFERENCES

[1] D. Ashbrook and T. Starner. Learning Significant Locations and Predicting User Movement with GPS. In *ISWC*, pp. 101–108, 2002.

[2] D. Ashbrook and T. Starner. Using GPS to Learn Significant Locations and Predict Movement across Multiple Users. *Personal and Ubiquitous Computing*, 7(5):275–286, 2003.

[3] L. Backstrom, J. M. Kleinberg, R. Kumar, and J. Novak. Spatial Variation in Search Engine Queries. In *WWW*, pp. 357–366, 2008.

[4] A. Brilingaitė. *Location-Related Context in Mobile Services*. PhD thesis, Aalborg University, 2006.

[5] A. Brilingaitė and C. S. Jensen. Enabling Routes of Road Network Constrained Movements as Mobile Service Context. *GeoInformatica*, 11(1):55–102, 2007.

[6] A. Brilingaitė, C. S. Jensen, and N. Zokaitė. Enabling Routes as Context in Mobile Services. In *GIS*, pp. 127–136, 2004.

[7] X. Cao, G. Cong, and C. S. Jensen. Mining Significant Semantic Locations from GPS Data. *PVLDB*, 3(1):1009–1020, 2010.

[8] Y.-Y. Chen, T. Suel, and A. Markowetz. Efficient Query Processing in Geographic Web Search Engines. In *SIGMOD*, pp. 277–288, 2006.

[9] K. Church, B. Smyth, K. Bradley, and P. Cotter. A Large Scale Study of European Mobile Search Behaviour. In *MobileHCI*, pp. 13–22, 2008.

[10] T. M. Delboni, K. A. V. Borges, and A. H. F. Laender. Geographic Web Search Based on Positioning Expressions. In *GIR*, pp. 61–64, 2005.

[11] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank Aggregation Methods for the Web. In *WWW*, pp. 613–622, 2001.

[12] R. Fagin, R. Kumar, and D. Sivakumar. Comparing Top k Lists. *SIAM J. Discrete Math.*, 17(1):134–160, 2003.

[13] R. Fagin, A. Lotem, and M. Naor. Optimal Aggregation Algorithms for Middleware. *J. Comput. Syst. Sci.*, 66(4):614–656, 2003.

[14] L. Gravano, V. Hatzivassiloglou, and R. Lichtenstein. Categorizing Web Queries according to Geographical Locality. In *CIKM*, pp. 325–333, 2003.

[15] K. Järvelin and J. Kekäläinen. Cumulated Gain-Based Evaluation of IR Techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, 2002.

[16] R. Jones, W. V. Zhang, B. Rey, P. Jhala, and E. Stipp. Geographic Intention and Modification in Web Search. *Int. J. Geogr. Inf. Sci.*, 22(3):229–246, 2008.

[17] J. Krumm and E. Horvitz. Predestination: Inferring Destinations from Partial Trajectories. In *Ubicomp*, pp. 243–260, 2006.

[18] J. Krumm and E. Horvitz. Predestination: Where Do You Want to Go Today? *IEEE Computer*, 40(4):105–107, 2007.

[19] J. Liu, O. Wolfson, and H. Yin. Extracting Semantic Location from Outdoor Positioning Systems. In *MDM*, pp. 73, 2006.

[20] Y. Morimoto, M. Aono, M. E. Houle, and K. S. McCurley. Extracting Spatial Knowledge from the Web. In *SAINT*, pp. 326–333, 2003.

[21] H. Sagan. *Space-Filling Curves*. Springer-Verlag, Berlin/Heidelberg/New York, 1994.

[22] L. Wang, C. Wang, X. Xie, J. Forman, Y. Lu, W.-Y. Ma, and Y. Li. Detecting Dominant Locations from Search Queries. In *SIGIR*, pp. 424–431, 2005.

[23] X. Yi, H. Raghavan, and C. Leggetter. Discovering Users' Specific Geo Intention in Web Search. In *WWW*, pp. 481–490, 2009.

[24] M. L. Yiu, X. Dai, N. Mamoulis, and M. Vaitis. Top-k Spatial Preference Queries. In *ICDE*, pp. 1076–1085, 2007.

[25] Y. Zheng, L. Zhang, X. Xie, and W.-Y. Ma. Mining Interesting Locations and Travel Sequences from GPS Trajectories. In *WWW*, pp. 791–800, 2009.

[26] Y. Zhou, X. Xie, C. Wang, Y. Gong, and W.-Y. Ma. Hybrid index structures for location-based web search. In *CIKM*, pp. 155–162, 2005.

APPENDIX

A. ADDITIONAL SCORING AND WEIGHT FUNCTIONS

We present some specific examples of possible scoring functions that can be used in practice. We recall that a place $p = \langle l, c \rangle$ has a signature $\mathcal{L} \times \mathcal{C}$, where $\mathcal{L}$ is the set of locations in Euclidean space and $\mathcal{C}$ is a set of categories that could be of interest to the users.

A.1 Scoring Functions

Count-based Scoring: The simplest scoring function assigns a score to a place p that is equal to the count of directions to p:

$$\mathcal{S}_{\mathrm{C}}(p, \mathcal{D}) = |\{d = \langle t, a, p.l, \|a, p.l\| \rangle | \ d \in \mathcal{D}\}|.$$

The intuition is that the importance of a place increases with the number of users that are willing to travel to reach the place.

Distance-Aware Scoring: With distance-aware scoring, we weigh the contribution to a count by a directions query $\langle t, a, b, D \rangle$ by $\|a, b\|$, and do not just use a plain count:

$$\mathcal{S}_{\mathrm{D}}(p, \mathcal{D}) = \sum_{\langle t, a, p.l, \|a, p.l\| \rangle \in \mathcal{D}} \|a, p.l\|.$$

The intuition is that the importance of a place increases with the distance that users are willing to travel to reach the place. If a user is willing to travel long distances, the place is more important than if the user is willing to travel only short distances to reach it.

Locality-Aware Scoring: With locality-aware scoring, we only take into account queries with a distance $\|a, b\|$ similar to the one the user is willing to travel:

$$\mathcal{S}_{\mathrm{L}}(p, \mathcal{D}) = |\{\langle t, a, p.l, \|a, p.l\| \rangle \in \mathcal{D} | \ \|a, p.l\| \approx \|q.l, p.l\|\}|.$$

The intuition behind this scoring function is that people who are willing to drive similar distances, will more probably want to go to similar places.

A.2 Weighting Functions

Linear Weight: We used two linear weight functions:

$$weight_{q.D}\left(\|q.l, p.l\|\right) = 1 - \frac{\|q.l, p.l\|}{q.D}$$

and

$$weight_{q.D}\left(\|q.l, p.l\|\right) = 1 - \frac{\|q.l, p.l\|}{2 \times q.D}.$$

The first one takes the value 1 for $\|q.l, p.l\| = 0$ and decreases linearly to the value 0 for $\|q.l, p.l\| = q.D$. The second one takes again the value 1 for $\|q.l, p.l\| = 0$ and decreases linearly to the value $\frac{1}{2}$ for $\|q.l, p.l\| = q.D$.

Parabolic Weight: We used two parabolic weight functions:

$$weight_{q.D}\left(\|q.l, p.l\|\right) = 1 - \frac{\|q.l, p.l\|^2}{q.D^2}$$

and

$$weight_{q.D}\left(\|q.l, p.l\|\right) = 1 - \frac{\|q.l, p.l\|^2}{2 \times q.D^2}.$$

The first one takes the value 1 for $\|q.l, p.l\| = 0$ and decreases parabolically to the value 0 for $\|q.l, p.l\| = q.D$. The second one takes again the value 1 for $\|q.l, p.l\| = 0$ and decreases parabolically to the value $\frac{1}{2}$ for $\|q.l, p.l\| = q.D$.

A.3 Experimental Study

To study the differences of the various scoring and weighting functions, we randomly chose 1,000 query locations and ranked all points of interest, within a radius of 5 kms with all combinations of the scoring and weight functions we defined in Appendix A.1 and A.2.

We used three metrics to compare the ranking produced by different scoring functions [12]:

Kendall's Tau: This metric counts the number of times that a pair of entries (α, β) appears in reverse order in the two rankings, i.e., $(\alpha, \ldots, \beta)$ in ranking one, and $(\beta, \ldots, \alpha)$ in ranking two, normalized by the total number of possible pairs.

Spearman's Footrule: This metric counts is defined as the sum of the absolute differences between the ranks of an item in two lists (e.g., if place p is 1^{st} in list 1 and 4^{th} in list 2, place p contributes $|1 - 4| = 3$ to the total sum. We normalize it by dividing by the maximum possible value this sum can have [12].

Intersection Metric: When comparing two top-k lists, this metric gives the sum of the sizes of the symmetric differences of the two lists when considering the top-1, top-2, ..., top-k elements of the two lists, normalized by the maximum possible value of this sum [12].

For all three metrics a value close to 0 signals similar rankings, while a value close to 1 signals different rankings. Using as a weight function a constant function (e.g., $weight_{q.D}(\|q.l, p.l\|) = 1$), the comparison for the (offline) scoring functions gave the results contained in Table 3, where the Spearman's footrule for the top-10 lists is contained (C.-b. stands for Counts-based, D.-a. for Distance-aware, and L.-a. for Locality-aware). Very similar results

Table 3: Comparison of offline scoring functions

	Count-based	*Distance-aware*	*Locality-aware*
C.-b.	0	0.430	0.064
D.-a.	0.430	0	0.470
L.-a.	0.064	0.470	0

were obtained for Kendall's tau and the intersection metric, and also for other values of top-k.

Similarly, for the count-based scoring functions we defined in Section A.1, Kendall's tau metric for the top-20 lists is given in Table 4 (where L.-1 stands for Linear-1, L.-$\frac{1}{2}$ for Linear-$\frac{1}{2}$, P.-1 for Parabolic 1 and P.-$\frac{1}{2}$ for Parabolic-$\frac{1}{2}$).

Table 4: Comparison of weight functions

	Linear-1	*Linear-*$\frac{1}{2}$	*Parabolic-1*	*Parabolic-*$\frac{1}{2}$
L.-1	0	0.044	0.001	0.060
L.-$\frac{1}{2}$	0.044	0	0.048	0.006
P.-1	0.001	0.048	0	0.064
P.-$\frac{1}{2}$	0.060	0.006	0.064	0

Very similar results were obtained for Spearman's footrule and the intersection metric, other values of top-k and other offline scoring functions.

We observe that the actual definitions of the offline scoring functions may give substantially different results (look for example the comparison between the Count-based and the Distance-aware functions). Furthermore, the weight functions can also change the ranked lists by a fair amount. While we do not explore with user studies how to select the appropriate functions, the space is very large and many different aspects of ranking can be captured in our setting.

B. SPACE PARTITIONING

The following example shows how a (latitude, longitude) pair is mapped to an hCell's ID for any possible hCell level.

EXAMPLE 3. We denote the hCell ID at level j, where $j \in \{1, 2, \ldots, 23\}$, by hCellID_j. Assume that we are given a location with coordinates $(37.2^o, -119.34^o)$, as seen in Figure 9. We want to find the hCell ID of these coordinates at all levels. In this example, we restrict ourselves to levels 1 and 2.

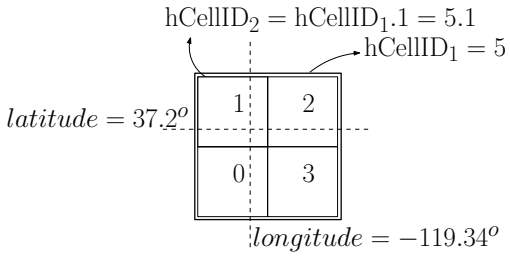


Figure 9: Example of hCells

The process is recursive. We first find the hCell ID at level 1 of these coordinates; this ID is a number between 0 and 5 (since at level 1, we have a cube with six faces). We assume $\text{hCellID}_1 = 5$. We then take the rectangle and use the Hilbert curve to number its four smaller rectangles. Then, the hCell ID for level 2 becomes $\text{hCellID}_2 = \text{hCellID}_1.1 = 5.1$, as can be seen in the figure. □

C. PSEUDOCODE FOR THE TABLE-BASED ALGORITHM

Algorithm 2 contains the pseudocode the straightforward table-based algorithm.

Algorithm 2: Table-based algorithm

Input: A query $q = \langle l, t, D, \bar{q} \rangle$, a set of relevant places `RP` $\subseteq$ `BusinessListing`
Output: A ranked list of the places in `Result`

```
Result ←
SELECT PlaceID, Score ×weight_{q.D}(‖q.l, RP‖) AS s
FROM RP
WHERE RP.Category = q̄
ORDER BY s;
```

The algorithm assumes that all places have been scored and that the retrieval module has selected a subset of relevant places, e.g., places within 10 km of the user's location. The SQL query simply returns the places sorted by their distance-adjusted scores.

D. THEOREM PROOFS

Here is the proof of Theorem 1:

Proof: At each point of our algorithm, priority queue PQ contains all the active lists. These are ordered by the maximum possible score that a list can contain after removal of the elements we have seen so far from the list.

The priority queue L contains a ranked list of places that we have seen in the lists. The ranking for L is over the score $\mathcal{S}(q, p, \mathcal{D})$.

The algorithm may stop for two reasons:

1. There are no more elements to see in the lists L_i: Then priority queue L contains all the elements ranked by their actual score, and we have solved the problem correctly.

2. The k^{th} element in L has a score greater than the first list in PQ. Assume the score of the k^{th}-th element in L has score s_k and the first list in PQ has a maximum possible score s_l. Since the algorithm stopped, we have $s_k \geq s_l$. All the places that have been examined so far (and have been included in L) have been ranked according to their actual score. Also, all the places that have not been examined in the lists have a score that is at most $s_l \leq s_k$. Thus, we know that apart from the current top-k places in L, there is no other place that can have a score that places it higher than the k^{th} place in L. Thus, we have computed the top-k places.

We need the non-increasing property of function $weight(\cdot)$ to ensure that PQ always *overestimates* the maximum score for the next element in the list. □

Here is a sketch of the proof for Theorem 2:

Proof: Assume we have the ranked lists L_i, for $i = 1, 2, \ldots, n$ to which we only allow sorted access; sorted access means that we have to retrieve elements from the top of each list L_i before retrieving elements below. Also, assume that the minimum distance between the user and the hCell that list L_i represents is m_i.

Let's assume that the ranked (by its offline score) content of list L_i is $\langle p_{i,1}, p_{i,2}, \ldots, p_{i,|L_i|} \rangle$, for all i. Now assume that we decay the places in lists L_i in order to create the conceptual lists $L'_i = \langle p'_{i,1}, p'_{i,2}, \ldots, p'_{i,|L_i|} \rangle$; the score of place $p'_{i,j}$ is $weight_{q.D}(m_i)$ times less than the score of $p_{i,j}$. Note that the relative order of places in each list does not change due to the conceptual decay process; all places are decayed by the same factor if they are in the same list.

Each place's $p'_{i,j}$ offline score has an "error" from its online score that is bounded by a function that takes into account the $weight(\cdot)$ function, the diameter of the hCell and the exact user's location. Our algorithm keeps a threshold (let's call it T_{on} for this proof) which is the maximum k^{th} online score seen so far in places of lists L_i (and L'_i). At each step, we pick the top place p in lists L'_i which has maximum possible offline score s_{off}. If $s_{off} < T_{on}$ and we have k examined elements, then we have found the correct top-k elements.

Now assume that the number of places our algorithm has to examine before it finds the correct top-k places is d. Assume that there is another algorithm A that correctly finds the top-k places in our setting with at most $d - 1$ steps. Obviously, algorithms A does not examine at least one place that our algorithms examines. Let's call this place $\hat{p}$. Say that at that point that algorithm A stops the threshold for our algorithm was T'_{on}. Then, since our algorithm did not stop at that point and some of the lists L'_i had at least the place $\hat{p}$, it was the case that $s'_{off} \geq T'_{on}$. Thus, if $\hat{p}$ actually had an online score T'_{on} (which is possible when place $\hat{p}$ lies on the point with minimum distance from the user's location and the hCell of the list where $\hat{p}$ is located), then algorithm A would miss a place that should have been included in the output top-k places. □

E. DATA SOURCES

We describe the two main data sources used for place scoring: the business directory data and the driving directions logs.

E.1 Business Directory

The business directory stores information regarding businesses and their location. An important consideration is high coverage, which is achieved by combining various data sources.

After the business listings have been retrieved, an extensive attempt to do Entity Resolution (ER) takes place. A particular restaurant may have one entry in `yelp.com`, one in `yellowbook.`

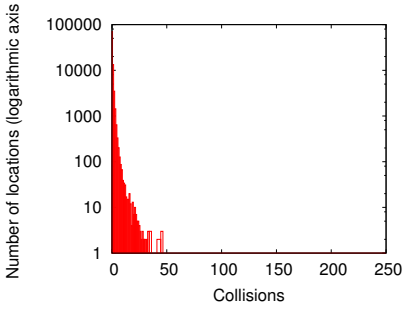


Figure 10: Histogram of collisions

com, one in a blog post, and it may have its own web page. This information has to be retrieved, cleaned, and then merged into (ideally) one entity. Google is very aggressive with respect to ER, in order to maintain high quality business listings.

After the cleaning and ER, the most accurate address for a business is geocoded. Thus, we have a very good estimate of the actual location of the business (with an accuracy of ∼10 m). Since the geocoder is the same as the one that was used for the addresses in the direction logs, we can perform a join between the two data sources.

Google's business directory is a table `BusinessListing` with schema `(PlaceID, Name, Location, Category)`, where `PlaceID` is an ID for a place, `Name` is the name of the place, `Location` is a hCellID_{23} that captures the location of the place (details on hCellIDs can be found in Section 3.1), and `Category` is one of a few predefined business categories, e.g., museum, restaurant, university. Table 1 is an example business directory and can be found in Section 3.2.

Because we have accurate locations of businesses, it is easy to distinguish between two different "branches" of the same chain, e.g., different McDonald's restaurants.

We derive places from business listings:

Definition 1. We define a `place` p as a tuple $p = \langle l, c, n \rangle$, where l is a location, c is a (business) category, and n is the (business) name. We may also use the tuple $\langle l, c \rangle$ as a place, if the name of the business under examination is not important.

E.2 Directions Logs

The main data source is the directions logs from Google maps. A log entry contains a timestamp, an origin (or source), a destination, and the distance from the source to the destination. The following is an example of a log entry: ⟨`2009/Jun/15 16:24:43.956 PST, 5.2.2.0.3..., 5.2.2.3.2..., 61,043 m`⟩.

The timestamp is the time when the query was issued and has an accuracy of 1 ms. The source and destination are given as hCellID_{23} identifiers. These values are obtained by geocoding the query strings provided by the user. If a geocode cannot be obtained (e.g., if the source string is not an address but the string "chocolate"), no source value is logged for this attempt to get directions. Thus, we can be certain that if we have valid location IDs then the original query contained actual addresses. The distance between the source and the destination is given in meters and has an accuracy of some tens of meters.

E.3 Joining the Two Data Sources

We performed a join between the destination query log and the business directory, on the log destination and the business location attributes as we described in Section 4.1.

Let us now examine the number of places that have been assigned in the same location as other places have been assigned to.

Definition 2. There are k collisions in a particular location l_i if there are exactly $(k+1)$ places for which $p.l = l_i$.

Figure 10 shows the histogram of collisions that appear in the locations that we have in hand after we perform the join between the query log ($\mathcal{D}$) and the business directory ($\mathcal{P}$).

Most locations contain exactly one place (business listing). There are some locations, though, with more than one business. From the empirical evaluation, we have determined that most collisions occur because the entity resolution (ER) is not perfect, especially when one has to take into account multiple data sources of business listings. For example, we have seen examples of two places in the same location that have the exact same name, or that have a slightly different name (e.g., "La Strada Italian Restaurant" and "La Strada Ristorante Italiano"), which are the same place, but were not merged during the ER.

We also note that there are locations with more than 5 business collisions. This happens, for example, when the only information that we have for a restaurant is the city in which it is located, in which case the location of the restaurant is assumed to be the center of that city. We ignore all the places at locations with 5 or more collisions in them. We only eliminate 1,733 places with this method, which is less than 1.14% of the total business listings in the region we examine.

The focus of this paper is not on entity resolution, and the business directory at our disposal is clean enough for place ranking.

F. ADDITIONAL EXPERIMENTAL RESULTS

F.1 Directions Log Time Dependence

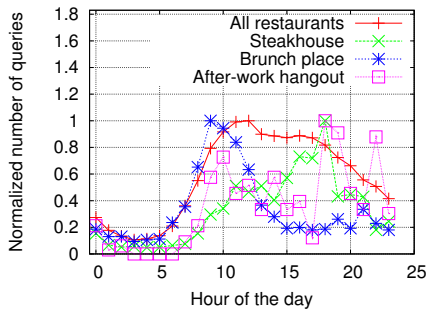
We study the fluctuations in the number of driving directions queries for different hours of the day and days of the week. We examined restaurants familiar to us: a steakhouse where people often go for dinner on weekdays (it is generally avoided during weekends since it is slightly isolated), a famous brunch place where people go during weekends between the hours of 10 am and 2 pm, and a place where people gather for drinks and food. Figure 11(a) shows the normalized number of queries for different hours of the day for these three places and for all the restaurants in our business directory. Figure 11(b) shows the normalized number of queries for different days of the week (day 1 is Sunday).

These graphs reveal quite distinct temporal variation. For example, the brunch place gets the majority of its queries during the mornings of the weekends. Similarly, the (isolated) steakhouse receives few queries during the weekends, and the after-work hangout receives many queries during Friday's before dinner and lunch time.

F.2 Directions Logs Contribution

In this experiment we aim to understand whether a place ranking signal based on driving directions adds new information to an existing place ranking system. The existing place ranking system may take into account aspects such as the number of reviews, sentiment of reviews, and the PageRank of web pages related to a place. The purpose of this experiment is not to quantify the quality of the new signal, but merely to determine whether it can bring new information to ranking.

We use Kendall's tau, described in Appendix A.3, to measure the distance between two rankings, and we compare place rankings



(a) Hour of the day

(b) Day of the week (day 1 is Sunday)

Figure 11: Normalized number of queries

generated using our scoring function with rankings generated using Google local search (GLS). For our ranking, we use the count-based scoring function (see Appendix A.1) and $weight_{q.D}(d) = 1 - \frac{d}{q.D}$.

Figure 12 uses box-whiskers plots to compare top-k rankings for $k = 5, 10, 15, \ldots, 100$ using Kendall's tau. The boxes contain the values between the first quartile and the third quartile. The end points represent the minimum and maximum values observed for the measurements, while the line inside the box is the median. We see that the median of new information is around 50% for all the values of k considered. Also, it tends to stabilize as k gets larger.

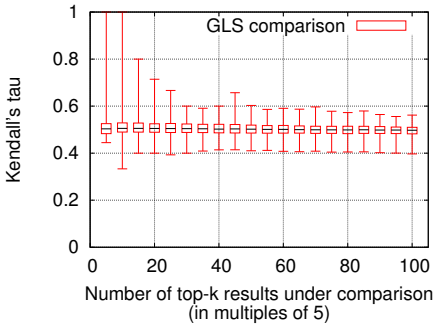


Figure 12: Count-aware scoring versus GLS ranking

We note that no existing system explicitly uses a $weight_{q.D}(\cdot)$ function for the distance. Usually current systems just rank places that are in the user's ZIP code, irrespectively of the user's distance to the places, or they use other heuristics similar to this. Thus, it is expected that the use of directions logs will add new information to any existing ranking system.

F.3 Sensitivity to the User's Location

In this experiment, we examine how sensitive our ranking functions are to the exact location of a user.

For examining the sensitivity of our ranking functions to the exact location of a user, we use the simple $weight_{q.D}(\cdot)$ function $weight_{q.D}(d) = 1 - \frac{d}{q.D}$ and the count-based scoring function (see Appendix A.1), which basically just counts the queries for a particular location.

The settings for the experiment are as follows. We have 100 hypothetical users. Each user is located in a randomly chosen location, the user issues the query "food" (one of the predetermined categories), and the user is willing to travel 2 km. For each one of the 100 users, we select 400 locations, such that their distances to the user are less than ∼1.4 km.

For each one of the 400 locations, we re-rank the results that were initially ranked for the associated user's location, using the ranking function $\mathcal{S}(q', p, \mathcal{D})$, where q' and q are only different in the user's location ($q.l$ is the user's original location and $q'.l$ is one of the 400 locations around the user). Then we compare the top-k lists of the two lists using Kendall's tau (with parameter $p = 0$).

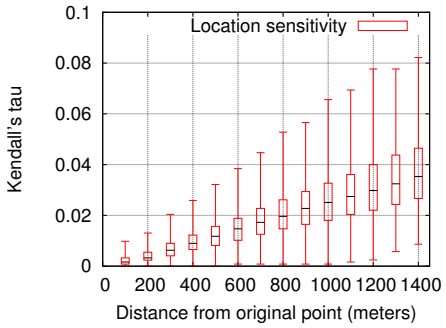


Figure 13: Location sensitivity of our ranking functions

We have included all the comparisons of the top-50 lists for various distances and we present a box-whiskers plot (as described in Appendix F.2) for distances that are less than 100 m, 200 m, ..., 1,400 m in Figure 13.

One can observe that two ranked lists are very similar (there is a difference that is less than 1%) for distances of around 100 m. The similarity decreases rapidly with distance, to around 8% for a distance of 1.4 km. This provides evidence of the importance of hyper-local ranking. When the precise location of a user is known, we can do a better job of ranking locally interesting places.

The results for other scoring functions (e.g., the ones described in Appendix A.1) and different values of k (for the top-k lists) were very similar. We also experimented with some other weight functions (e.g., the ones described in Appendix A.2); again, the results are very similar to the ones presented here. Of course, the graph depends on how fast the $weight_{q.D}(\cdot)$ function decreases, but no qualitative differences were observed during our experiments.