Estimating the Scoring Functions Used by Google Maps in Top-k Spatial Keywords Search

by

Siying Lyu

A technical report submitted to the graduate faculty in partial fulfillment of the requirements for

the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:
Dr. Ying Cai, Major Professor
Jin Tian
Simanta Mitra

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this technical report. The Graduate College will ensure this technical report is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University

Ames, Iowa

2020

TABLE OF CONTENTS

Page

LIST OF FIGURES

Page

LIST OF TABLES

Page

ACKNOWLEDGMENTS

I would like to thank my committee chair, Dr. Ying Cai, and my committee members, Dr. Jin Tian, and Dr. Simanta Mitra, for their guidance and support throughout the course of this research.

In addition, I would like to thank my friends, colleagues, the department faculty and staff for making my time at Iowa State University a wonderful experience. I want to also offer my appreciation to those who were willing to participate in my surveys and observations, without whom, this thesis would not have been possible.

ABSTRACT

A popular search on Google Maps is top-k spatial keywords query. Given a value of k, a set of keywords, and a reference point, Google Maps returns the k spatial objects whose description and location are most relevant to the search keywords and the reference point. Here the relevance is computed using a scoring function. While Google Maps supports top k queries, it keeps the scoring functions in secret. Our research is interested in this secret. Specifically, we want to know the scoring functions used by Google Maps processing top k spatial keywords queries. We believe that knowing the ranking behaviors of a search engine such as Google Maps will make it possible to leverage the engine to search for the information not available from the standard services it provides. To estimate the scoring functions, we develop a process that analyzes the top-k query results returned by Google Maps. Based on the monotonicity properties of geo-proximity and relevance of the textual description to the search keywords, we estimated the linear format of scoring functions used in Google Maps. The linear inequality systems are leveraged to iteratively solve the coefficients and the constants in scoring functions. By combining the methods of computing the textual relevance and solving the linear inequality systems, we estimated six sets of scoring functions. The estimated scoring functions are used to perform the top-k spatial keywords query in order to evaluate its performance. The query result of the estimated scoring functions is compared with the one given by Google Maps in terms of the number of the common spatial objects and the ranking order. In this report, we will present the implementation and examine the performance results.

CHAPTER 1. INTRODUCTION

Top-k Spatial Keywords Search

The *top-k* spatial keywords query is one of the most commonly used queries in our daily life. It ranks the objects based on the distance between the object and the reference point, and the textual relevance of the query keywords. The *top-k* spatial keywords query is widely supported by all the online search engines, such as Google Maps.

A typical input of the *top-k* spatial keywords query includes a reference point *p*, at where the query gets performed, a set of keywords *W* and a number *k*, which is used to determine the number of the objects returned by the query. The result of the query gives a set of *k* objects ranked on the spatial distance to the reference point *p* and the textual relevance to the query keywords *W*.

Figure 1 gives an example of a *top-k* spatial keywords query. The blue point indicated the reference point *p*. The keywords are "Chinese Restaurant", and the number *k* is 3. Therefore, there are 3 objects returned by the query, which indicates the top-3 objects that are most close to the blue point and most relevant to "Chinese Restaurant".

Figure 1 Top-k spatial keywords query

A ranking problem always comes with a comparison between objects. As an online search engine that provides the *top-k* spatial keywords query, every online search engine has its own method to rank the spatial objects. It is reasonable to assume that there is a numerical measurement for an online search engine to help the evaluation of each object.

This numerical measurement is the score of the object. We found that such score must represent the way that the online search engine evaluates an object. As a result, it is essential to understand the how the scores are computed, in the other words, we need to find the scoring function. Unfortunately, there is no explicit information about the scoring function used by any online search engine, considering it is a commercial secret to every company.

Importance and Challenges of Finding the Scoring Functions

The importance of the scoring function is obvious. For a user of the online search engine, the scoring function will give a clear picture of the weight of each factor of a spatial object. It will help the user to understand the rank of a spatial object. Furthermore, making the decision based on the factor he/she values most. For a spatial objects (business) in the online search engine, knowing the scoring function helps it to find the shortcut to improve its rank. For developers, the scoring function provides latent information of the database used by the online search engine. This information facilitates the study of some novel queries, such as finding the region in which a spatial object is always ranked in *top-k*.

However, the challenges in finding the scoring function is difficult to overcome. First, we need to find the format of the scoring function. Second, since the absolute scoring used by the online search engine is unknow, the testing/training dataset is unavailable for us. Third, as the result of the absence of the ground truth, the error function becomes a mystery as well. Although these challenges prevent us from getting the scoring functions directly, we can discover valuable information from the data returned by the online search engine.

Purposed Solution

In our research, we selected Google Maps as the online search engine. As we stated previous, the input of a *top-k* spatial keywords query requires the reference point and the set of keywords, which provides the measurements of the geometric closeness and the textual relevance. Therefore, we believe that the geometric closeness and the textual relevance play essential roles in the scoring functions. By analyzing the *top-k* spatial keywords query result returned by Google Maps, we discovered the monotonicity properties of the geometric closeness and the textual relevance. Given the monotonicity properties, we are able to purpose the linear format of the scoring functions, which takes the geometric closeness and the textual relevance as the input.

Because of the absence of the testing/training dataset, we cannot solve the linear scoring functions with the linear regression method or other supervised learning methods. Instead, we leveraged the linear inequality systems constructed based on the *top-k* spatial keywords query result returned by Google Maps. The inequality relationship is guaranteed by the nature of ranking order in the query result. The error function is formed by the relationship between the rank and score because the ranking order is the only concrete information provided by Google Maps. Once we have an error function, we can solve the linear inequality systems iteratively to find the global minimal of the error function.

By combining the three methods of computing the textual relevance and two methods of solving the linear inequality systems, we estimated six sets of scoring functions. By leverage each set of the scoring functions to perform the *top-k* spatial keywords query, we can evaluate their performance.

CHAPTER 2. API Data Analysis

Google Maps APIs

As one of the most widely application, Google Maps, which launched in 2005, made a revolutionized contribution to the online mapping service applications on the World Wide Web [1]. Beside the new type client/server interaction it introduced based on Asynchronous JavaScript and XML (AJAX), Google Maps also provides us its extensive sources of code called the Application Programming Interface (API). The API consists of a set of data structures, object classes or functions that can be used by us using Python, PHP or other scripting language [2]. The data given by APIs contains the query results which is computed by Google. Google Maps provides various APIs to fulfill the requirements of many frequently retrieved queries.

Table 1 List of Google Maps APIs

Category	API	Description
Maps	Maps SDK for Android/iOS	Add a map to your Android/iOS app
	Maps Static API	Add simple, embeddable map images to your website with minimal code
	Maps JavaScript API	Add an interactive map to your website. Customize it with your own content and imagery
	Street View Static API	Embed real-world imagery with 360° panoramas
	Maps URLs	Launch Google Maps and initiate an action, like search or directions, using a cross-platform URL scheme
	Maps Embed API	Add an interactive map, or Street View panorama to your site, using a simple HTTP request
Routes	Directions API	Provide directions for transit, biking, driving, or walking between multiple locations
	Distance Matrix API	Calculate travel times and distances for multiple destinations
	Roads API	Determine the precise route of a vehicle
Places	Places SDK for Android/iOS	Add rich details for millions of places to your Android/iOS app. Provide

		autocomplete results for user queries. Convert between addresses and geographic coordinates
	Places Library, Maps JavaScript API	Add rich details for millions of places to your website. Provide autocomplete results for user queries. Convert between addresses and geographic coordinates
	Places API	Get up-to-date information about millions of locations using HTTP requests
	Geocoding API	Convert addresses to geographic coordinates or the reverse
	Geolocation API	Return the location of a device without relying on GPS, using location data from cell towers and WiFi node
	Time Zone API	Get the time zone for a specific latitude and longitude coordinate

For example, one of the most used queries is the *top-k* spatial keyword query. The *top-k* spatial keyword query focuses on finding the k spatial objects whose location and related texts (descriptions) are most relevant to a given reference point and certain keywords. A spatial object is associated with a geolocation and some related texts helping to describe it (e.g. some keywords or reviews from visitors). The name of the query is given based on the fact that it is computed in terms of both the spatial closeness and the relevance between the descriptions and query keywords [3] [4]. In Google Maps, the following requests under Places API category are found helpful to accomplish *top-k* spatial keyword query and collecting other information of a spatial object.

Table 2 Candidate APIs for *Top-k* spatial keywords query

API	Requests	Description
Place Search	Find Place requests	A Find Place request takes a text input and returns a place
	Nearby Search requests	A Nearby Search lets you search for places within a specified area, the search request can be refined by supplying keywords or specifying the type of place you are searching for

	Text Search requests	The service responds with a list of places matching the text string and any location bias that has been set.
Place Details	Place Details Requests	A Place Details request returns more comprehensive information about the indicated place such as its complete address, phone number, user rating and reviews

Although the API provides us a way to get access to part of the data Google used to accomplish the services of Google Maps, the full access to the data of Google Maps is still limited. It will be a great benefit for programmers if there is a way to take a glimpse on the unreleased data. It will have a huge research potential because we can develop new types of queries which are not provided by Google Maps.

Choice of APIs and API data analysis

Let $D = \{o_1, o_2, \ldots, o_n\}$ denote the set of spatial objects on the Internet. Each object $o_i \in D$ has a spatial attribute that consists of the object's latitude and longitude, denoted as $(o_i.x, o_i.y)$. Moreover, each object oi has a textual attribution that collected from the reviews of the object, denoted as $o_i.text$, which consists of a sequence of words. We consider all the Google Maps APIs discussed in the previous session. The following APIs serve our goal best:

$nearbysearch(p, R, W)$: This API supports searching for the objects according to the reference point p and the set of keywords W. The API returns the set of spatial objects who locate inside a range of R to p and match W. The objects are ranked by prominence by default. For each object returned from the API, it is attached with the latitude, longitude and place id which is assigned by Google.

$details(place\_id)$: This API supports searching the detail information of a spatial object. The information includes the rating and reviews from the visitors, price level, opening hours etc. By specifying the field of information, the API can return only the reviews for us.

In our research, we leverage these two APIs to collect the data we needed for the scoring function estimation. For every point *p* on the map, we call $nearbysearch(p, R, W)$ to execute the *top-k* spatial keywords query. For each object in the result set of the query, we call $details(place\_id)$ to get the reviews for the object, which is used to compute the relevance of the object to the keywords set.

For the *top-k* spatial keyword query, simply refer to *top-k* in this report, there are two factors making a spatial object to obtain a higher rank in such kind of query: 1) from a viewpoint of geolocation, higher closeness to the given reference point, 2) from a viewpoint of reviews, higher relevance to the given keywords. Based on the fact that Google Maps APIs provides the data to support the *top-k* query, we can verify the data collected from Google Maps APIs has the monotonicity properties on the two parameters we studied, namely the distance to the reference point p and the relevance to the keywords set W.

Monotonicity Verification - Distance

To verify the distance monotonicity of the data, we send query to the Google maps APIs on the random points between any pair of spatial objects with a fixed set of keywords. The x-axis indicates the longitude of the spatial objects and the y-axis indicated the latitude. The blue point in the graph indicates one of the spatial objects in the pair and the red point indicates the other one. All the other black and green points represent the reference points chosen between the blue and red points. At a certain point, if the blue point obtains the higher rank than the red one, we mark the point to black. Meanwhile, a point is colored to green if the red point ranks higher than the blue one. The title gives the name of the spatial objects.

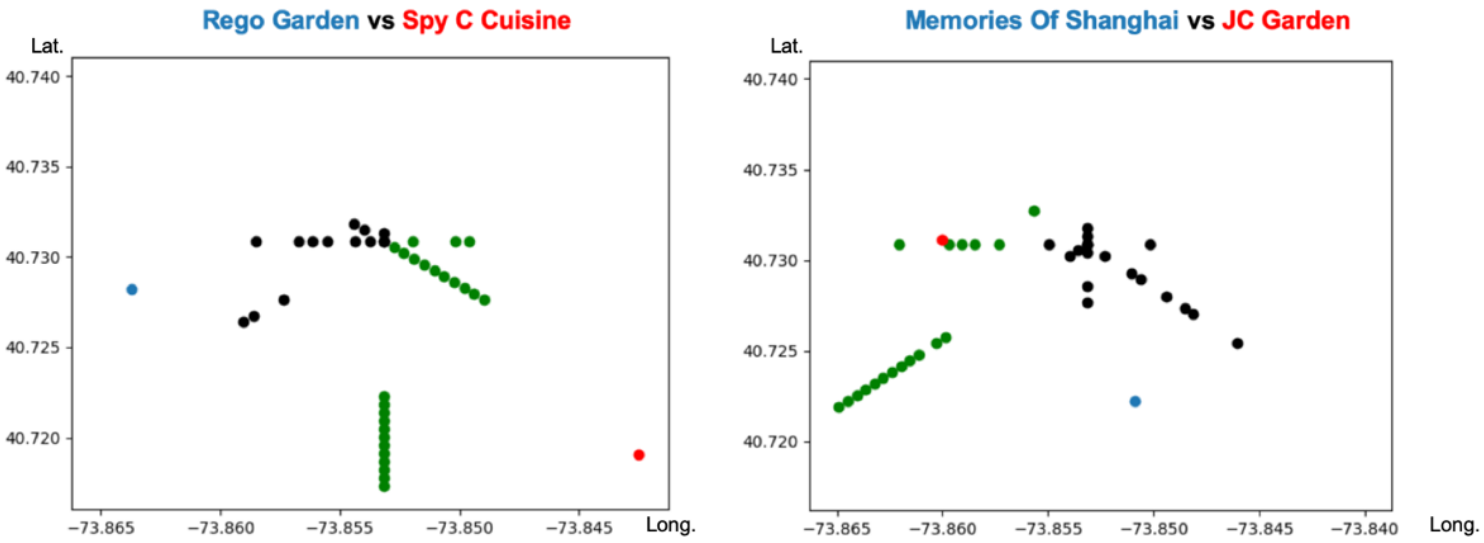


Figure 2 Monotonicity of Distance

From the Figure 2 shown above, we can observe a boundary between the black points and the green points. It tells us that, once the relative rank order between two spatial objects swaps, it will not swap back, given a fixed set of keywords.

We can observe that the distance between certain green point to the red point is always smaller than the distance between the green point to the blue point. For a black point, the distance between the blue point to it is smaller than the distance between the red point to it as well. It indicates that the as the distance between a reference point p to a spatial object shrinks, the rank of the spatial object at p will grow.

Monotonicity Verification - Relevance

Similarly, we choose different keywords sets to verify the relevance monotonicity. The reference point p is fixed so that the distance of each object to p is fixed. The objects we chosen for observation are found in the returned API data with the same keywords set. In the graph, the x-axis is the relevance between the keywords set and the reviews of the object. The relevance is computed by the frequency of the word in keywords set appearing in the reviews. The y-axis is

the ranking of an object. The larger the number is, the lower the ranking will be. The color of the point corresponds to the name of the object. The keywords used for Google Maps query is shown in the graph accordingly.

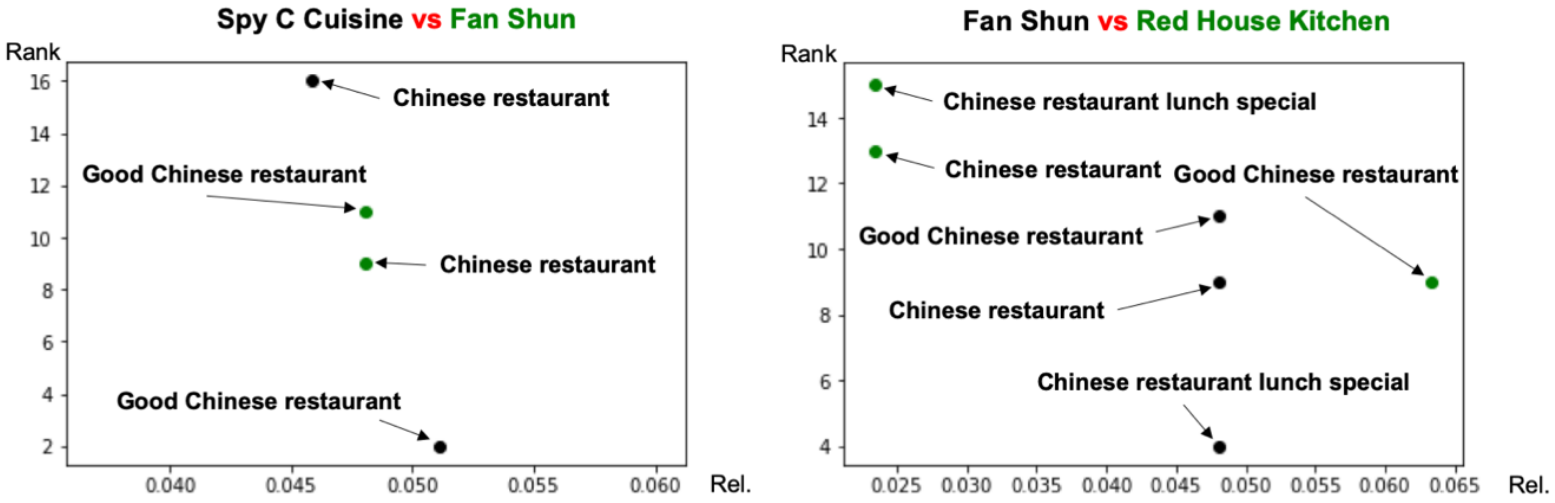


Figure 3 Monotonicity of Relevance (Word Frequency)

As shown in the Figure 3, as the relevance of the keywords set and reviews of the object increases, the ranking of the object increases (the number of ranking decreases). Given a pair of objects, the relevance could affect the relative ranking between the objects. For example, ‘Spy C Cuisine’ ranks higher than ‘Fan Shun’ with the keywords ‘good Chinese restaurant’, but ‘Fan Shun’ obtains a higher ranking with the keywords ‘Chinese restaurant’. The same relevant may results to different rankings with a fixed distance to the reference point. This is because that the ranking is based on the total number of objects returned from Google Maps API. This observation explained that for some object, although their relevance to different keywords sets are similar, the ranking varies a lot. For a single object, the monotonicity of the relevance is observed clearly. Beside by computing relevance with word frequency, the Jaccard similarity and cosine similarity are also applied (Figure 4). The monotonicity property conserves for the data even the similarity is computed differently.

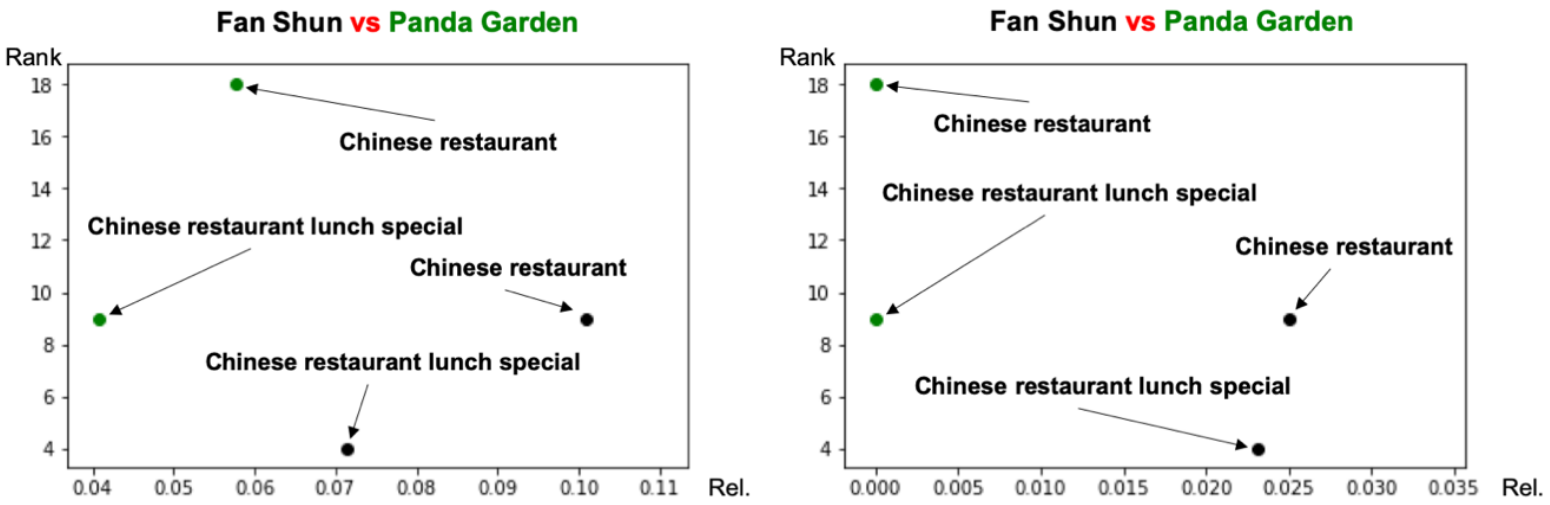


Figure 4 Monotonicity of Relevance, Jaccard Similarity and Cosine Similarity

CHAPTER 3. METHOD AND EXPERIMENTS

Estimating Linear Ranking Function

In the previous chapter, the ranking function is only discussed with the aspects of spatial closeness and textual relevance. But Google Maps may also consider other unknow features such as the rating, advertisement fee paid to Google, etc. Since we showed that the monotonicity of spatial closeness and textual relevance, it makes possible for us to estimate the actual ranking function used by Google Maps. Let $f(\cdot) = f(rel(W, o_i), dist(p, o_i), o_i.E)$ be the ranking function used to compute the score of an object o_i to a reference point p and a set of keywords W. The $rel(W, o_i)$ is the relevance between W and the reviews we collected for o_i, the $dist(p, o_i)$ is the distance between p and geolocation of o_i, and the $o_i.E$ is the unknown constant used in ranking o_i. We consider $f(\cdot)$ to be in a linear form and will discuss how to estimate the unknown parameters in this chapter.

We first assume $f(\cdot)$ takes the following linear form:

$$f(rel(W, o_i), dist(p, o_i), o_i.E) = \beta_1 * dist'(p, o_i) + \beta_2 * rel(W, o_i) + \beta_3 * o_i.E \quad (1)$$

Equation 1 is the linear combination of the spatial closeness, textual relevance and the unknown constant, where $\beta = \{\beta_1, \beta_2, \beta_3\}$ are the unknown coefficients to be estimated.

For $dist'(p, o_i)$, this term is a result of the normalized distance with a range of [0,1]. The relationship between $dist(p, o_i)$ and $dist'(p, o_i)$ is as following:

$$dist'(p, o_i) = 1 - \frac{dist(p,o_i)}{\sqrt{1+dist(p,o_i)^2}} \quad (2)$$

According to Equation 2, $dist'(p, o_i) = 1$ indicates o_i overlaps with the reference point p, and as $dist(p, o_i)$ decreases, the $dist'(p, o_i)$ increases. The original spatial distance is normalized in such a way because the monotonicity characteristic is better explained if the

ranking score is monotonically non-increasing with respect to $dist'(p, o_i)$. The original spatial distance measurement is the Euclidean distance between the reference point p and the object o_i.

For $rel(W, o_i)$, we consider three different canonical functions: Cosine similarity, Jaccard similarity, and word frequency. Although the actual $rel(W, o_i)$ used by Google Maps is unknown, but the absolute value of $rel(W, o_i)$ is less important for us. We examined that the comparison result is not affected by the way we compute the $rel(W, o_i)$ in the monotonicity validation part.

Since β_3 and $o_i.E$ are both unknown, we treat the third term as a single latent parameter for each object and thus can simply use the denotation of $o_i.E$. Thus, we can avoid $o_i.E$ in the input of scoring function. Therefore, we can rewrite

The ranking function as:

$$f\big(rel(W, o_i), dist(p, o_i)\big) = \beta_1 * dist'(p, o_i) + \beta_2 * rel(W, o_i) + o_i.E \quad (3)$$

As a result of the property of the parameters, we need to find the score function for each object. Instead of a universal scoring function for all the objects, we need to estimate a pair of coefficients β_1 and β_2 for all the objects and a set of $o_i.Es$. As the fact that we have no idea how $o_i.E$ is computed, we will estimate $o_i.E$ with uniform distribution initially and followed with an iterative solving manner. The method we applied for improving this estimation will be described in the implementation part.

There are several challenges we need to solve in the above problem. First, we need to estimate both the function parameters and a latent parameter for each object. Instead of estimating a scoring function for all the objects, we need to estimate the scoring function for each of the object appearing in the result of the *top-k* query result. Second, there is not a given training dataset that we can optimize for. Because the actual absolute score for each of the object

is unknow in our case. The only information is the ranking order of the result objects set of the *top-k* query. Third, our goal is to minimize the error between the actual score and the estimated score. From the second challenge, we encountered the difficulty of directly compute this error. All these challenges make our problem different from standard linear regression problem.

We developed following approach in our research to overcome the challenges. Let $top(p, k, W)$ denote a *top-k* query issued to the search engine and $\{o_1, o_2, \dots, o_k\}$ be the query result, where o_i $(1 \leq i \leq k)$ is the *i*-th ranked object in the result. Using the ranking result of $\{o_1, o_2, \dots, o_k\}$ as an observation, we can formulate the following linear inequality system, where f_i is the score function for object o_i $(1 \leq i \leq k)$:

$$\begin{cases} f_1(p) \geq f_2(p) \\ f_2(p) \geq f_3(p) \\ \vdots \\ f_{k-1}(p) \geq f_k(p) \end{cases} \qquad (4)$$

In the other words, we can solve the linear inequality system formed by the data from the search engine query result. Although we have the linear inequality system, the target function is unknow in our case. It makes the linear programming methods such as Simplex [5] not good choices for our problem. In order to solve the linear inequality system, we observe the feasible region of the linear inequality system first.

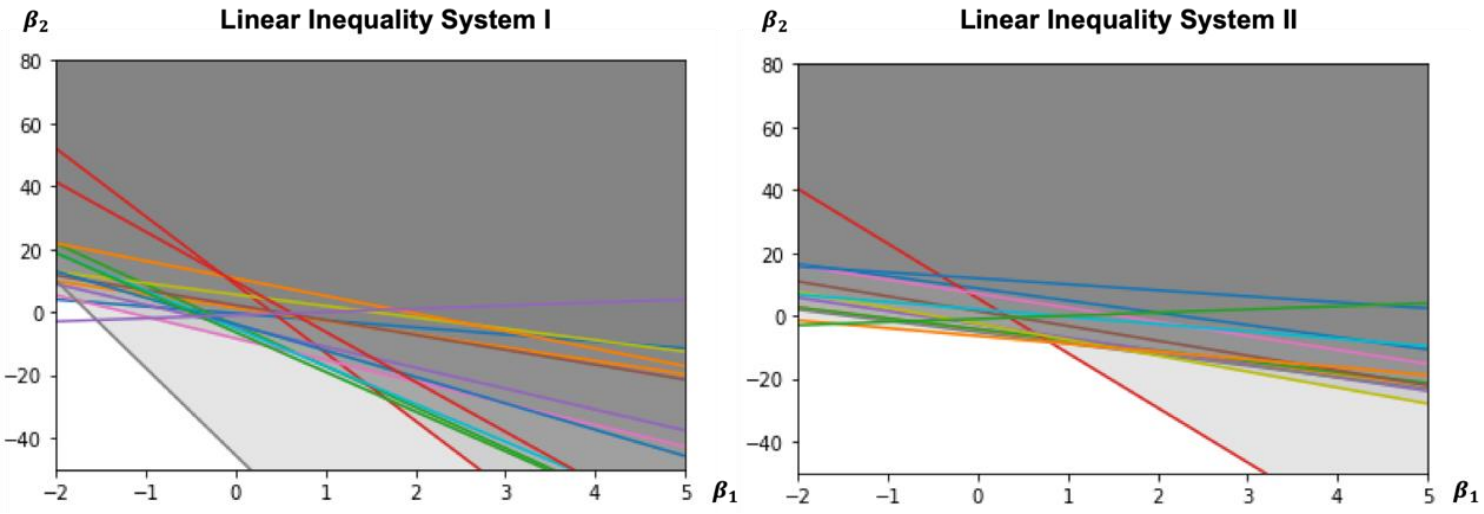


Figure 5 Feasible Region of Linear Inequality System

In Figure 5, we plotted the feasible region of two linear inequality systems, which are formulated by two randomly chosen *top-k* queries. The x-axis and y-axis represent β_1 and β_2, respectively. Each line in the graph indicates the function $\beta_1 * (dist'(p, o_{i-1}) - dist'(p, o_i)) + \beta_2 * (rel(W, o_{i-1}) - rel(W, o_i)) + (o_{i-1}.E - o_i.E) = 0$, where $(2 \leq i \leq k)$. In the colored region, we have $\beta_1 * (dist'(p, o_{i-1}) - dist'(p, o_i)) + \beta_2 * (rel(W, o_{i-1}) - rel(W, o_i)) + (o_{i-1}.E - o_i.E) > 0$. The most saturated region is resulted from the overlap of all the feasible region of each line. Therefore, the most saturated region is the feasible region of the linear inequality system. We purpose two methods to solve the linear inequality system.

First, since all the feasible regions are roughly with the same shape, we are able to come up a simply target function. Once the target function is known, we are still able to apply a method which is used to solve linear programming problem. The target function we chosen is $\beta_1 + \beta_2$, which is used as a minimization linear programming problem. The final β_1 and β_2 are determined by applying the k-mean clustering method on the solved parameters from all the linear inequality systems.

The second way to solve the linear inequality system is to solve the pairs in inequalities. For instance, the objects set $\{o_1, o_2, o_3\}$ is the *top-k* query result returned from Google Maps. We can expend the linear inequality system into the following format:

$$\begin{cases} \beta_1 * \big(dist'(p, o_1) - dist'(p, o_2)\big) + \beta_2 * \big(rel(W, o_1) - rel(W, o_2)\big) + (o_1.E - o_2.E) \geq 0 \\ \beta_1 * \big(dist'(p, o_2) - dist'(p, o_3)\big) + \beta_2 * \big(rel(W, o_2) - rel(W, o_3)\big) + (o_2.E - o_3.E) \geq 0 \end{cases}$$

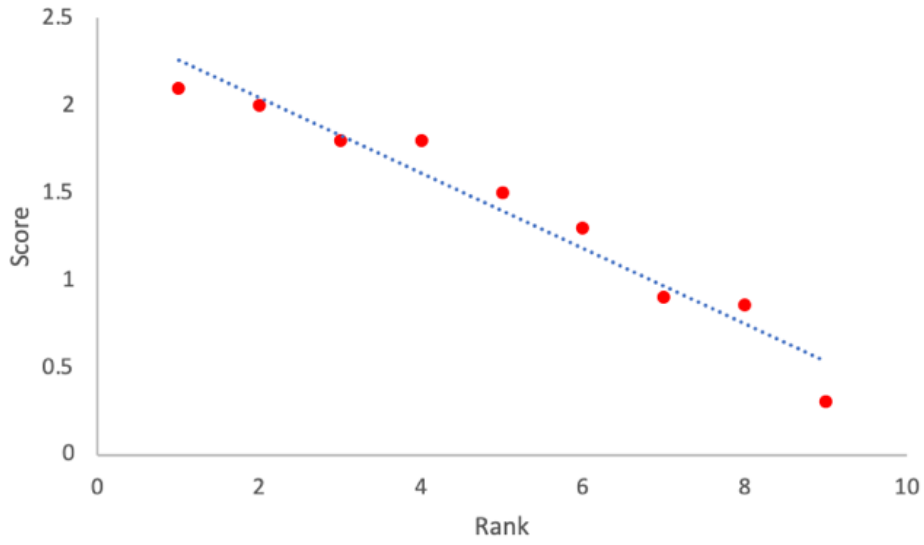
Since $o_1.E$, $o_2.E$ and $o_3.E$ is estimated by the uniform distribution, there are only two unknown variables to be solved. Therefore, we are able to solve the exact β_1 and β_2 parameters based on this pair of inequalities. In the result object set of a top-k query, we expected to have $k-1$ inequalities and at most $k-2$ solved parameters for each linear inequality system. Among the $k-2$ β_1s and β_2s, we select the minimal and maximal values, respectively, to indicate the range of β_1 and β_2 that fulfill the linear inequality system. To obtain the final β_1 and β_2, we find the intersection of the ranges solved from all the linear inequality system. When there is no overlap for all the solved ranges, we select the range that occurs in most of the solved ranges. The final β_1 and β_2 is the average of the final β_1 range and β_2 range.

Given a target object o_i and a set of keywords, we can issue different *top-k* queries with different reference points. For each query result, we can build one inequality system and perform parameter estimation. More queries will introduce more constraints to the systems and thus result in better estimation.

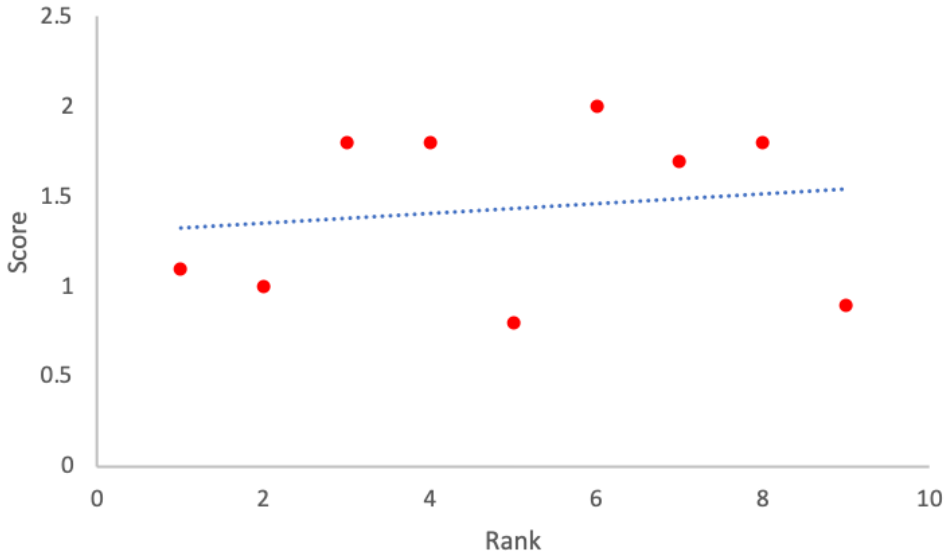
Here we state the details of our algorithm. Let p be an arbitrary reference point on the map. A rectangle region R is defined based on the position of p. The four corners are posed with a distance of 5km to p. Our algorithm starts by issuing a query at p with the given keywords. According to the result returned from API, we generated the reference points around each object in the result set. These reference points are generated based on eight directions around each object. The step is selected as same as the parameter we send to Google Maps API, which is used

to help $nearbysearch(p, R, W)$ API to perform the search within a local cyclic region. Beside the reference points near each object from the *top-k* query result set at p, we include some other reference points that are slight remote from the object. These remote reference points are included to add more constrains. All the reference points are selected within R. Each result set returned from Google Maps API at each reference point formed a linear inequality system. Once the linear inequality systems are solved, the parameters, β_1, β_2 and $o_i.E$, are ready to form the scoring functions.

As analyzed above, the error between the actual score and the estimated score cannot be computed. Therefore, we check the relationship of the ranking order and the score we computed for the objects instead. For all the objects in a *top-k* query result, which comes from the Google Maps API and can be treated as benchmark, we expect the score of each object leads to the same ranking order. For example, given all the objects in a *top-k* query result set, Figure 6 (a) is the correct relationship of the ranking order and the score of all the objects. The x-axis indicates the number for ranking and the y-axis indicates the score computed by the scoring function. The higher the score is, the lower the number for ranking will be. If we fit a trendline for the data point, the gradient of the trendline should be negative.



(a) Negative Trendline



(b) Positive Trendline

Figure 6 Rank-Score Relationship

However, among all the *top-k* query result sets, half of the *top-k* query result sets show incorrect relationship after applying our scoring function, as shown in Figure 6 (b). This may result from the following aspects: 1) the parameters solved for each linear inequality system do not cluster well enough or the common overlap should cover more parameter ranges. 2) the

initial $o_i.E$ needs to be improved. Since we did not provide a way to modify the $o_i.E$ part, we will purpose an algorithm to optimize the $o_i.E$ here. The goal for the optimization is to increase the number of the *top-k* query result sets which obtain the correct relationship with after applying the scoring function.

As we state before, the initial value of $o_i.E$ is generated by the normal distribution. The algorithm will solve the final β_1, β_2 and $o_i.E$ iteratively. For the first iteration, β_1 and β_2 are solved by the methods we proposed for solving the linear inequalities. Once β_1 and β_2 are computed, we apply the current scoring function with β_1, β_2 and the initial $o_i.E$ to each top-k query result set to check the gradient of the trendline we shown in Figure 6. If the trendline is with a negative gradient, we collect the difference between the object's score and its projection on the trendline, shown as Figure 7.

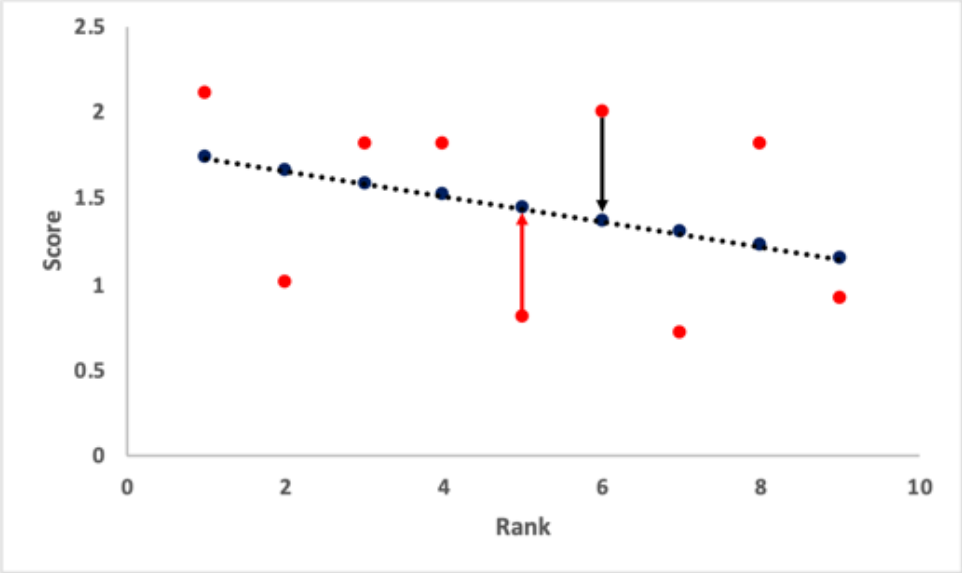


Figure 7 Correction on the Rank-Score Relationship

All the red points are the scores computed by the scoring function and all the blue points are the projections of the red points on the trendline. The dashed line indicates the trendline. For the red points r_i and r_{i+1} (objects o_i and o_{i+1}), which are with the rank of i and $i+1$, the score

of r_i should be greater than r_{i+1}. As the fact that the score of r_i is smaller than r_{i+1}, we try to move r_i and r_{i+1} towards the trendline. The distances should be moved are indicated by the length of the red arrow and the blue arrow, respectively. The red arrow indicates that $o_i.E$ needs to add the length of the red arrow, which is defined as positive shift. The blue arrow indicates that $o_{i+1}.E$ needs to subtract the length of the blue arrow, which is defined as negative shift. We can collect a set of positive shifts and a set of negative shifts for each object after checking all the data. The $o_i.E$ is modified by either the intersection of all its positive shifts or the intersection of all its negative shifts. The choice is made based on the number of positive/ negative shifts and the range of their intersections. After the modification of all the $o_i.E$, the algorithm goes into the next iteration. A number of iterations is set for the algorithm. The parameters which maximize the number of negative gradient trendline for the *top-k* query result set are selected for constructing the set of scoring functions.

Implementation of the Algorithm

We implement the proposed techniques, i.e., estimating the linear scoring functions and evaluating their performance. We discuss our implementation and evaluation as follows.

Before we perform the implementation, we need to find a proper area for the estimation algorithm. Since we need enough data for the scoring function estimation, we decided to choose an area where the density of the spatial objects is high. Therefore, we choose to perform our estimation algorithm in New York City (NYC), a large city with a high spatial object density (about 35/km^2). The default k value is set to 20 for Google Maps API, which is to guarantee that the target object will appear on the first page of results. We limit the type of spatial objects to Chinese restaurants.

The process of the program is described as following. The parameters we send to the Google Maps API for the *top-k* query are:

keywords="good Chinese restaurant"

location="40.7282122, -73.8637183"

The initial region R and the points for more queries are generated, as shown in Figure 8. The total number of points generated is 351. The x-axis indicates the longitude and the y-axis indicates the latitude. The red point in the center represents the location of the initial reference point p. The other four points on the corners shown the corners of the region R.

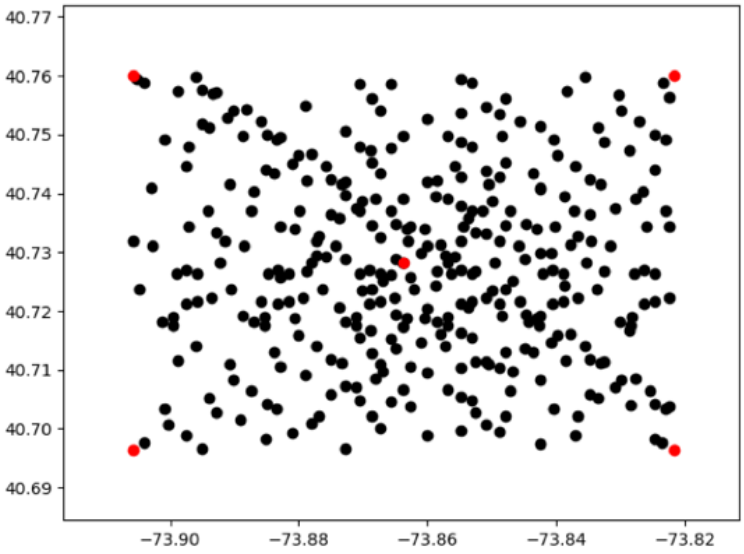


Figure 8 Initial Reference Points

The next step is to run *top-k* query on all the points generated from the previous step. Then the Place Details API is called for each object to get the corresponding reviews. These data are further processed to get the distance variable and the textual relevance variable for each of the object. Three types of textual relevance are computed, namely the word frequency, the Jaccard similarity and Cosine similarity. Once the data is preprocessed, the linear inequalities are built for each object set from the *top-k* query result. The parameters for the linear scoring

function are solved in an iterative manner. Since we provided two methods to solve the linear inequalities, we set six experiments to run the data, which correspond to the combination of the three textual relevance and two linear inequality solving method.

Table 3 Solved Parameters

Solving Inequalities	Textual relevance method	β_1	β_2	Highest percentage of negative gradience
Range Merging	Cosine	0.01338	11.26243	84.9%
	Jaccard	0.04334	56.36923	93.6%
	Word Frequency	0.60883	25.12657	86.2%
Linear Programming	Cosine	4.08170	39.70540	67.4%
	Jaccard	0.07351	98.27260	84.4%
	Word Frequency	0.00416	47.01687	86.2%

Performance of the Estimated Scoring Functions

There are two measurements we selected to evaluate the estimated scoring functions. The first one is number of the common spatial objects and the second one is the ranking order. The first measurement compares the common spatial objects of the *top-k* result set from Google Maps API with the ones computed by the estimated scoring functions. The second measurement compares the ranking order in the *top-k* result set from Google Maps with the one resulted by our scoring functions.

The number of the common spatial objects is calculated as following. Given a result set of a *top-k* query (**result set**), we computed the score of each object with the estimated scoring functions and then order the objects by the new scores. The reordered result set is called **reordered set**. A number n, where $n \leq k$, is selected for obtaining the subsets of size n from the result set and the reordered set, respectively. The number of the common spatial objects is computed from the subset of the result set and the reordered set. This measurement is used to

evaluate whether the estimated scoring functions can result a same object set of a *top-k* query as Google Maps does.

When $n = 20$, the number of the common spatial objects is 20, because the input parameter for the *top-k* query is 20. The percentage of the common spatial objects is 100%. In order to have an overall view of this measurement, for each set of scoring functions, we calculated the percentage of the common spatial objects for each *top-k* query result set. The number n is tested from 1-20. For each n, an average of the percentage is computed. The number n vs percentage of the common spatial objects plot is shown in Figure 9.

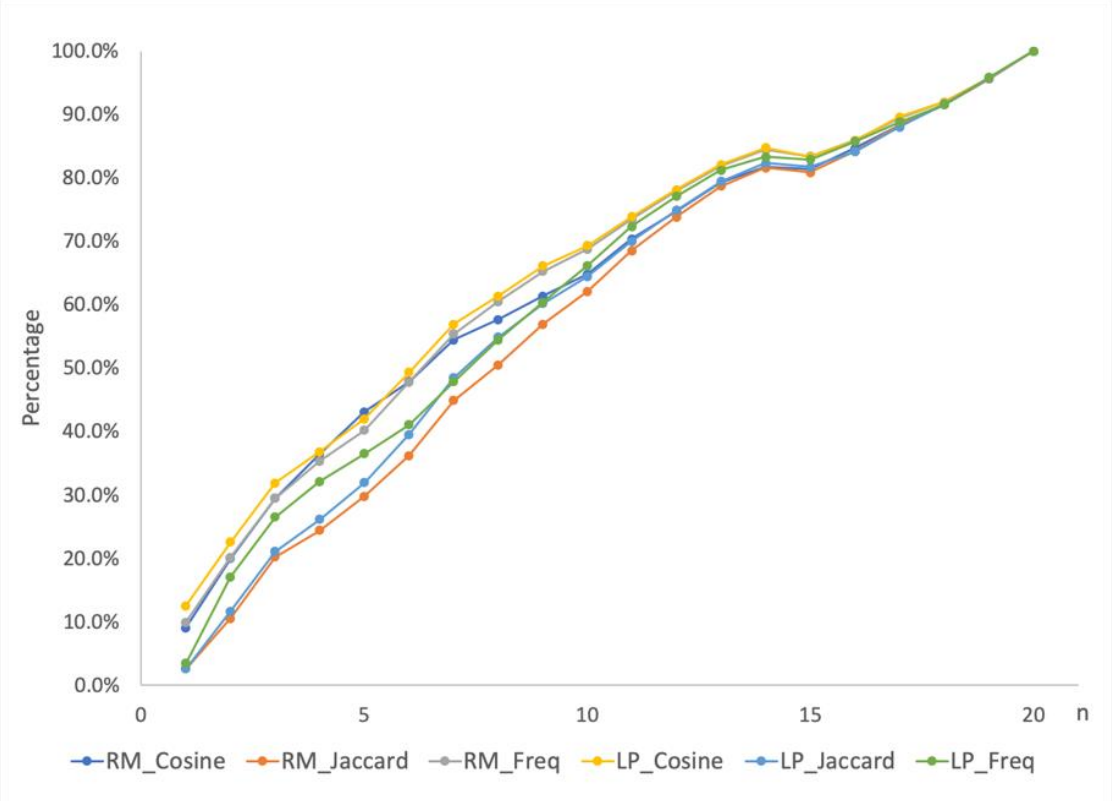


Figure 9 Percentage of the common spatial objects

We can observe that overall, as the number n increases, the percentage of the common spatial objects in the result set and the reordered set increases. When $n = 10$, the average percentage of all six sets of scoring functions is 66.0%. It indicates that if we are querying for

half of the objects in a given objects set, which is the best half of the whole set, our scoring function can get a result that roughly 66.0% close to the Google Maps result. Among all the sets of scoring functions, the one computed by Linear Programming and Cosine similarity (LP_Cosine) performs the best. The one computed by Range Merge and Word Frequency (RM_Freq) is good when $7 \leq n \leq 10$. We are focusing more on the data where $n \leq 10$, because in such case the number of target objects, the number of the objects in the *top-k* query result set, is only half or even less than half of the total number of the objects. Such situation is common in our daily life search.

The number of the common spatial objects is also compared between different methods. The methods for solving the linear inequality systems are studied first. The average percentages of three textual methods for the Range Merge method and the Linear Programming method are computed for $n \leq 10$. The data is illustrated in Figure 10.

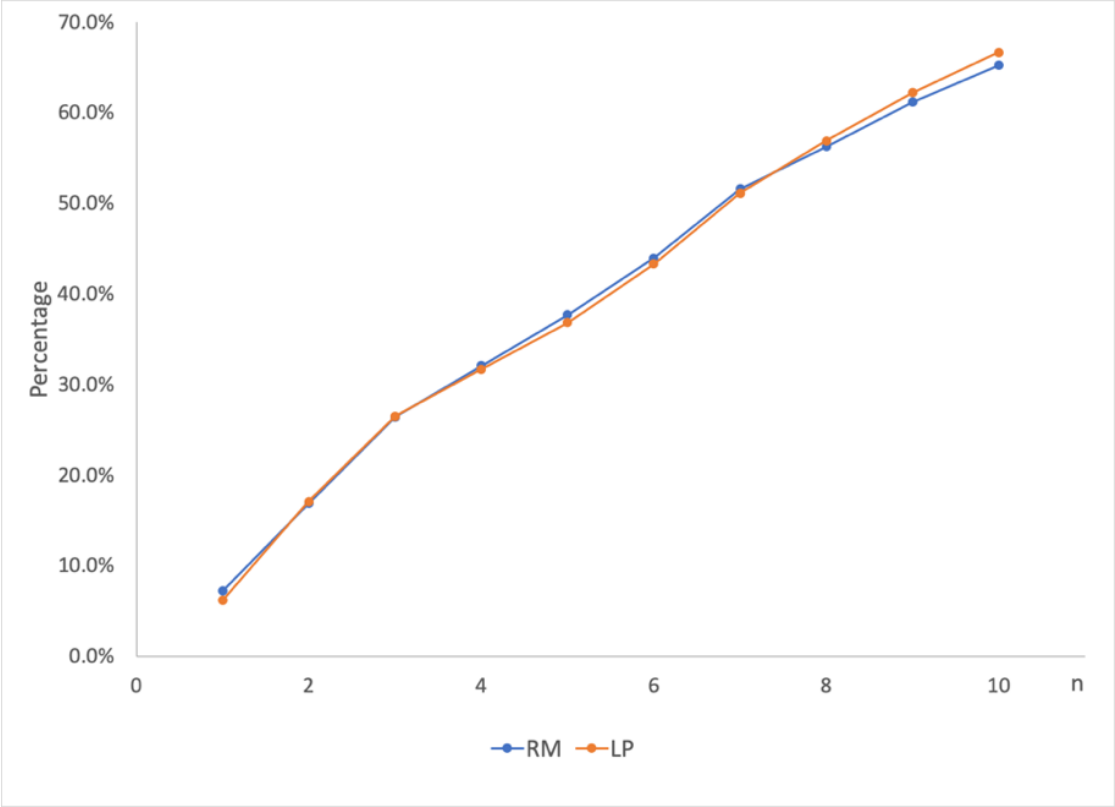


Figure 10 Number of the common spatial objects by inequality-solving methods

When n is less then 7, the Range Merge method performs slightly better. The percentage for the Range Merge method and the Linear Programming method is 51.6% and 51.1%, respectively. As the n increases, the Linear Programming method works slightly better than the other one. Therefore, the two methods perform similar with regard to the number of the common spatial objects.

The performance of the textual relevance methods is also evaluated. The average percentages for the inequality-solving methods for the Cosine similarity, the Jaccard similarity and the Word Frequency are computed for $n \leq 10$. Similarly, we plotted the data in Figure 11.

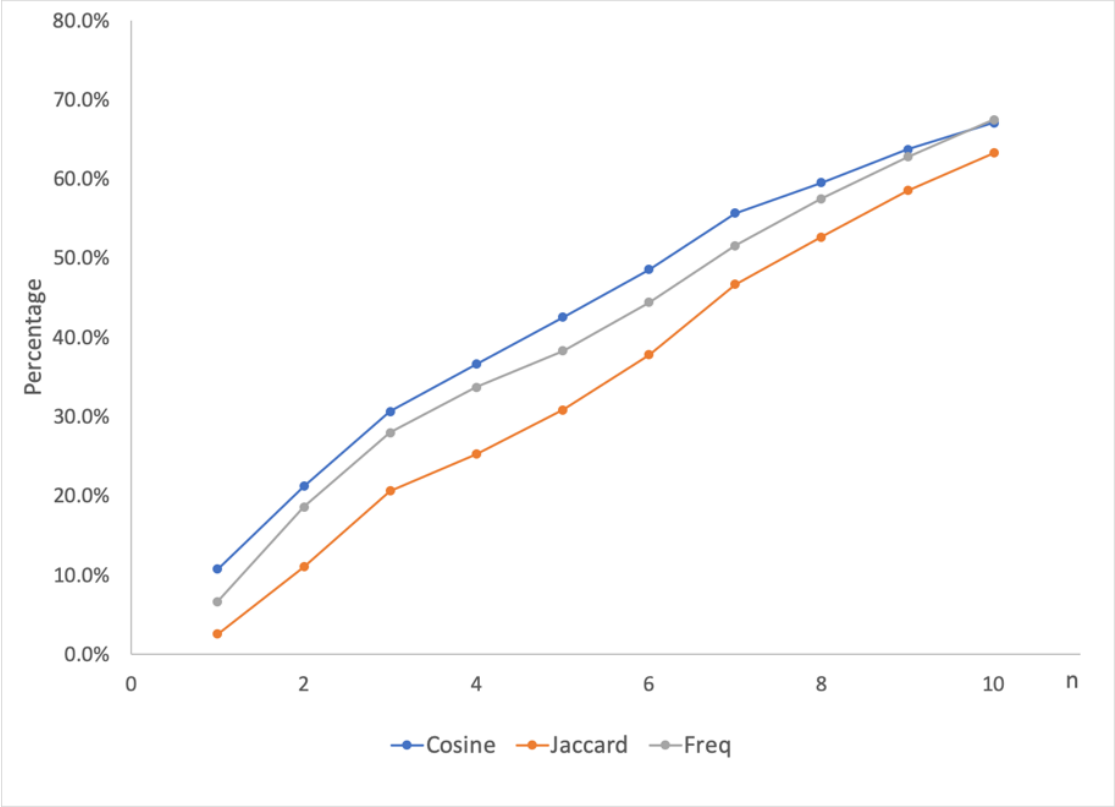


Figure 11 Number of the common spatial objects by textual relevance methods

Overall, the Cosine similarity outstands the other two methods, especially when n is less than 9. When $n = 9$, the percentage of Cosine similarity, Word Frequency and Jaccard similarity are 63.7%, 62.8% and 58.6%, respectively. The Word Frequency method is at the second place in terms of performance. When n is greater than 9, Word Frequency method outperforms the Cosine similarity. Considering the performance of the Word Frequency is outstanding in large n case, it may indicate that the combination of the Cosine similarity and the Word Frequency maybe pretty similar to the textual relevance method applied by Google Maps.

The second measurement we need to study is the ranking order. Similarly, the number n, the result set and reordered set are set up for the ranking order study. As we are focusing on the ranking order now, we will only study the objects sets where the result set and the reordered set are the same.

Given a result set and a reordered set, we are able to find the rank difference for each of the object because of set identity. For a given object appears in both result set and reordered set, if its rank in the result set is higher than the one in the reordered set, then our estimated scoring function underrates the object. Otherwise, the estimated scoring function overrates the object. If the two ranks are same, we say the rank obtained by the estimated scoring function is correct. By analyzing the ranks of each object in given result set and reordered set, we can have a glimpse of how the estimated scoring function ranks objects comparing with Google Maps.

As we studied the individual object, the number n is investigated via a range instead of an integer for the sake of object ranks analysis. Since we have stated the importance of $n = 10$ before, we will analyze the data for $1 \leq n \leq 10$ beside the one for $1 \leq n \leq 20$. We have a hypothesis that the larger the n is, the more different the order will be. However, we need to understand what results the difference as well.

To verify the hypothesis, we can check if the ranks of each objects is same in the result set and the reordered set. A percentage with regard to the number of the result set / reordered set is computed to show the closeness of the two ranks. For example, given three objects $\{o_1, o_2, o_3\}$, let's consider the following cases:

$$case\ 1: \begin{cases} result\ set: o_1, o_2, o_3 \\ reordered\ set: o_3, o_1, o_2 \end{cases}, \qquad case\ 2: \begin{cases} result\ set: o_1, o_2, o_3 \\ reordered\ set: o_3, o_2, o_1 \end{cases}$$

In case 1, there are no object having the same rank in the result set and the reordered set. While in case 2, there is o_2 ranked at the second place in both result set and reordered set. Therefore, we say the ranks difference in case 1 is larger than that in case 2. The difference percentages are calculated as 0.0% and 33.3% in case 1 and case 2, respectively.

The question of the reason for the differences can be simplified to the study of the percentage of the overrated cases and underrated cases. Either overrating or underrating can

induce the difference, but we still not sure which one plays a greater rule, or they affect the ranking order evenly. The percentages are computed with regard to the number of the result set / reordered set, and the number of overrating / underrating is counted by the ranks comparison.

Table 4 Percentage of the ranks difference by scoring functions

	n	RM_Cosine	RM_Jaccard	RM_Freq	LP_Cosine	LP_Jaccard	LP_Freq	Average
Correct	1-10	10.8%	13.5%	15.5%	15.0%	15.3%	15.2%	14.2%
	1-20	6.5%	5.3%	6.3%	7.0%	6.5%	7.1%	6.4%
Overrate	1-10	52.2%	57.4%	57.0%	58.1%	53.1%	50.8%	54.8%
	1-20	58.5%	63.3%	61.2%	60.9%	60.5%	59.6%	60.7%
Underrate	1-10	37.0%	29.1%	27.5%	26.8%	31.7%	34.0%	31.0%
	1-20	35.0%	31.4%	32.6%	32.0%	33.0%	33.3%	32.9%

Table 4 shows the percentage of the ranks difference for the six sets of scoring functions. The method of calculation is explained in the previous text. The data shown in the Correct rows indicates that the hypothesis of larger n leading a larger difference between the ranking orders in the result set and the reordered set is verified. The higher the Correct percentage is, the more similar the ranking orders in the result set and the reordered set will be. When the n is in the range of 1-10, the average Correct percentage is 14.2%. When the size of the object set increases (larger n), the average Correct percentage drops to 6.4%.

Beside the Correct rows, the average Overrate percentage and the average Underrate percentage indicates that the estimated scoring functions are overrating an object in general speaking. As the n increases, the overrating affects even more on the ranking order. Because the drop of the average Correct percentage (7.8%) is mainly contributed by the increasing of the

average Overrate percentage, which is 5.9%. The average Underrate percentage also increases by 1.9%, but it has much less affect to the average Correct percentage comparing with the average Overrate percentage.

Table 5 Percentage of the ranks difference by methods

	n	RM	LP	Cosine	Jaccard	Freq
Correct	1-10	13.3%	15.2%	12.9%	14.4%	15.3%
	1-20	6.0%	6.8%	6.8%	5.9%	6.7%
Overrate	1-10	55.5%	54.0%	55.1%	55.2%	53.9%
	1-20	61.0%	60.4%	59.7%	61.9%	60.4%
Underrate	1-10	31.2%	30.8%	31.9%	30.4%	30.7%
	1-20	33.0%	32.8%	33.5%	32.2%	33.0%

Table 5 shows the percentage of the ranks difference for the methods we applied. Two inequality-solving methods, the Range Merge method and the Linear Programming method, are compared with each other, and the three textual relevance methods form another comparison group. From the aspect of the inequality-solving method, the Range Merge method overrates and underrates an object more than the Linear Programming method does. When n is small, the degree of overrating and underrating for the Range Merge method is more obvious. As the n increases, the difference between the Range Merge method and the Linear Programming method becomes less.

Among the three textual relevance methods, the Jaccard similarity tends to overrate an object more than the other two methods do. This tendency is even more obvious when the n is large. The Word Frequency method performs pretty good in the case of both small and large n

cases. We also found the Cosine similarity is the most stable one among all the three methods, given then difference of the percentages between the small n case and large n case is smallest for the Cosine similarity method in both aspects of overrating and underrating. For the overrating, the differences are 4.6%, 6.7% and 6.5% for the Cosine similarity method, the Jaccard similarity method and the Word Frequency, respectively. For the underrating, the differences are 1.6%, 1.8% and 2.3%, respectively.

General speaking, the Linear Programming method is slightly better than the Range Merge method, especially in terms of the ranking order. The Word Frequency turned to be the best textual relevance method. The performance of the Cosine similarity is also remarkable, and it may be considered together with the Word Frequency if we apply future improvements on the estimated scoring functions.

CHAPTER 4. CONCLUSION

We have introduced a process to estimate the scoring functions used by Google Maps. Based on the monotonicity of the distance and the textual relevance, we estimated the linear scoring functions, which take the distance and the textual relevance as input. The coefficients of the distance and the textual relevance are estimated universally, while the constant in the scoring function is estimated for each object individually. As such, we generated a set of scoring functions, each of which is specific for one object.

The linear inequality systems are constructed from the *top-k* spatial keywords query result set. By solving the linear inequality systems iteratively, we are able to solve the scoring function of each object in the *top-k* spatial keywords query result set. We developed two methods to solve the linear inequality systems, namely Range Merge and Linear Programming. For the calculation of the textual relevance, we implemented three methods, namely Cosine similarity, Jaccard similarity and Word Frequency. These methods together produced six different sets of scoring functions.

To evaluate the performance of the estimated scoring functions, we came up with two measurements, namely the number of the common spatial objects and the ranking order. The number of the common spatial objects shows whether the estimated scoring functions can get the same *top-k* spatial keywords query result set as the Google Maps does without considering the ranking order. The number of the common spatial objects increases when we are querying a larger k for *top-k* spatial keywords query. The Range Merge method slightly outperforms the Linear Programming method with regard to the number of the common spatial objects. The Cosine similarity performs best when k is small, and the Word Frequency performs best when k is large.

The ranking order is compared by studying the individual object ranks change. As we expected, the larger the k is, the more different the order will be. Overall, the estimated scoring functions overrate the objects. As the k increases, the degree of overrating rising as well. The Range Merge method overrate an object more than the Linea Programming method. The Word Frequency performs the best in terms of ranking order, while the Cosine similarity is the most stable one when the value of k increases.

According to the analysis of the two measurements, the Linear Programming method and the Word Frequency can be considered the outperforming one in our research. The estimated scoring functions can be further applied to develop new types of queries.

CHAPTER 5. Works Cited

[1] T. D. Shunfu Hu, "Online Map Application Development Using Google Maps API, SQL Database, and ASP.NET," *International Journal of Information and Communication Technology Research,* pp. 102-110, 2013.

[2] M. P. Peterson, International perspectives on maps and the Internet, Springer, 2008.

[3] D. F. Ian, H. Vagelis and R. Naphtali, "Keyword search on spatial databases," in *2008 IEEE 24th International Conference on Data Engineering*, IEEE, 2008, pp. 656-665.

[4] R. Hariharan, B. Hore, C. Li and S. Mehrotra, "Processing spatial-keyword (SK) queries in geographic information retrieval (GIR) systems," in *19th International Conference on Scientific and Statistical Database Management (SSDBM 2007)*, IEEE, 2007, pp. 16-16.

[5] D. G. Luenberger, Introduction to linear and nonlinear programming, Addison-Wesley Reading, 1973.

[6] L. Chen, G. Cong, C. S. Jensen and D. Wu, "Spatial keyword query processing: an experimental evaluation," *Proceedings of the VLDB Endowment,* vol. 6, no. 3, pp. 217-288, 2013.