Spatial-Keyword Queries with Similarity Search

Le Xu
Dept. of Computing,
Hong Kong Polytechnic Univ.
le-simon.xu@connect.polyu.hk

***Abstract*—With the rapid growth of e-commerce, targeted marketing and recommendation systems have become increasingly important. One of the fundamental challenges in these systems is efficiently retrieving relevant products or services for a given user query. Top-k retrieval and similarity search have emerged as popular techniques for addressing this problem. In particular, spatial-keyword queries have been shown to be effective in modeling user preferences by capturing both the spatial and textual aspects of the query. In this paper, we review recent advances in similarity search for targeted marketing and recommendation, focusing on spatial-keyword queries. We will propose efficient techniques to process similarity search on single point, and attempt to improve the running time and queries' quality.**

***Index Terms*—Nearest-neighbor search, Similarity Search, Top-k retrieval**

I. Problem Definition

A single keyword point is created when the geo-textual object that uses this keyword as tag is created. In this paper, we study the similarity search problem on single keyword point, and it is also the query point from the user perspective.

We define that a query point has two attributes, keyword, and location, so it can be represented as $q = \langle keyword, location \rangle$, where $location$ is the geographical location in 2D dimension, and $location = \langle x, y \rangle$, and $keyword$ is a specific keywords.

A data set U_{data} is the collection of all posts, where each posts is created by content tagged with some keywords, and every post is associated with a location.

A point set S_{key} is defined as the set of all locations of the posts, which contains the same keyword key in the tag section of the post. Every post corresponds to a point.

The collection of point sets formed from U_{data} is defined as C_s so that if we have n different keywords in our current dataset, $\bigcup_{i=1}^{n} S_{key} = C_s$.

We define the similarity function of a point set S_{key} with respect to q at time as follow:

$$F(q, S_{key}) = \sum_{i=1}^{n} S_p(q, s_i) \cdot \mathrm{Rel}(q, s_i)$$

where $n = |S_{key}|$, which is the number of point in this point set. s_i is a specific point in S_{key}. $S_p(q, s_i)$ is the spatial proximity degree of (q, s_i), which is calculated by using standardized euclidean distance[1]:

$$S_p(q, s_i) = \sqrt{\sum_{j=1}^{2} \frac{\left(q_j - s_{i_j}\right)^2}{V\left[j\right]}}$$

where q_1, s_{i_1} are the longitude of q and s_i respectively, and q_2, $s_{i_2}$is the latitude. $V\left[1\right]$ is the variance of q_1 and s_{i_1}, and $V\left[2\right]$ is the variance of q_2 and s_{i_2}. It should be noted that the method for calculating variance here is not by dividing by n, but rather by dividing by n-1, which is one in our case. The range of $S_p(q, s_i)$ is from 0 to positive infinity, where 0 indicates the positions of the two points coincide.

S_{key} is divided into separate parts because more personalized information could be more valuable for individual users. Additionally, it is possible that different points within the same S_{key} may be widely dispersed, thereby reducing the effectiveness of the information. $\mathrm{Rel}(q, s_i)$ is text relevance between $q.keyword$ and the tag section of s_i, which can be modeled by the Jaccard distance [2]:

$$\mathrm{Rel}(q, s_i) = 1 - \frac{Num(q, s_i)}{|s_i|}$$

$\frac{Num(q,s_i)}{|s_i|}$ represents the rate at which the $q.keyword$ appears in the tag section of s_i. $Num\left(q, s_i\right)$ represents the number of $q.keyword$ that appears in the keyword list of s_i. $|s_i|$ is the size of the keyword list. The range of $\mathrm{Rel}(q, s_i)$ is $[0, 1]$.

A. Problem Statement

Collection of point sets C_s derived from current U_{all}, a query point Q_{kp}, and the user-specified K for the top-k ranking. The similarity search problem retrieves K point sets in C_s having the smallest $F(q, S_{key})$. The problem we are trying to solve focuses more on query requests issued by individual points rather than those issued by groups. This is one major difference from the problem statement in [2]. Our work is expected to obtain higher-quality answers and superior time complexity compared to the methods used in [2].

Figure 1 gives an example of how the similarity search is working. The edge costs are standardized euclidean distance. There are four point sets in C_s: S_{part}, S_{animal}, S_{lake}, $S_{mountain}$. Therefore, we could calculate the similarity as the following:

- q to park: $1.3 + 1.6 + 2.0 \times \frac{2}{3} = 4.23$

[1]Scipy: https://docs.scipy.org/doc/scipy-0.7.x/reference/spatial.distance.html

[2]https://en.wikipedia.org/wiki/Jaccard_index

- q to animal: $2.0 \times \frac{2}{3} = 1.33$
- q to lake: $2.0 \times \frac{1}{2} = 1.0$
- q to mountain: $0 + 2.0 \times \frac{1}{2} + 2.0 \times \frac{2}{3} = 2.33$

If $K = 2$, the results should be S_{lake} and $S_{mountain}$.

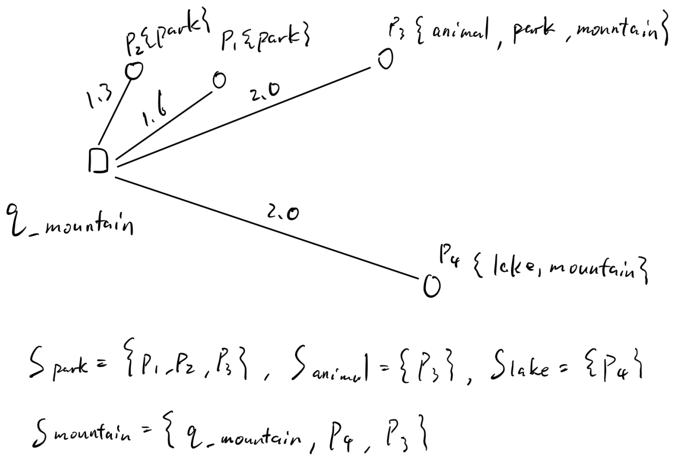

P2{park}
P1{park}
P3{animal, park, mountain}
1.3
1.6
2.0
q_mountain
2.0
P4{lake, mountain}
S_park = {P1, P2, P3}, S_animal = {P3}, S_lake = {P4}
S_mountain = {q_mountain, P4, P3}

Fig. 1. Similarity Search

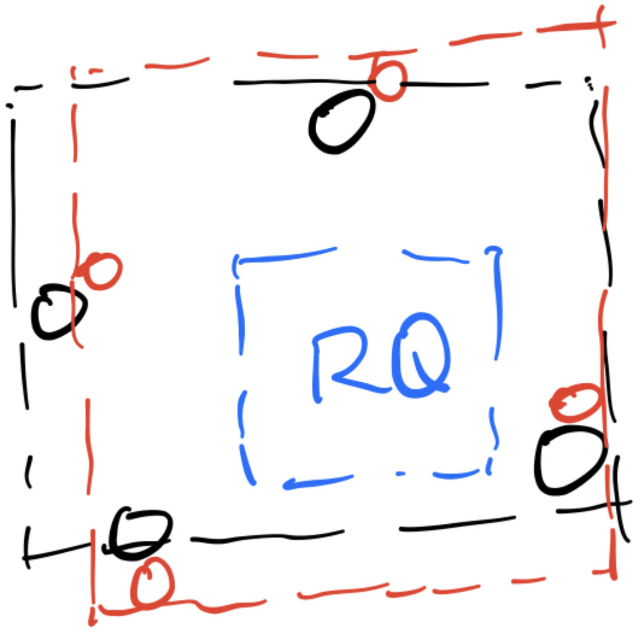


Fig. 2. Inefficiency of using MBRs

II. Existing Methods

One possible solution for addressing SKQ is to partition different point sets using R-trees [1]. During the querying process, all point sets serve as indexes for the R-tree, and then the minimum bounding rectangles (MBRs) are used to conduct pruning. However, in situations where the overlapping degree among point sets is high, the pruning effect of R-tree may not be optimal, and the complexity may degenerate into brute-force traversal in local areas with high overlap degree. Figure 2 shows that when two point sets are highly overlapping, the range query perform on their dead space will need to traverse both MBRs.

Another possible approach, which is also the state of art solution for our problem, is to use a representative lower bound for pruning and attempt to use some heuristics to better select the proxy [2], which can generate a tighter lower bound early on. Meanwhile, some filtering techniques can be attempted to retrieve some candidates from all point sets to avoid computational-intensive checking of inequalities. This will be discussed in detail in the section of proposed methods.

Our work is based on [2], starting from a domain-driven perspective [3], and using finer-grained analysis formulas to perform similarity queries on individual users. Additionally, the similarity measure that I have defined differs somewhat from that of [2].

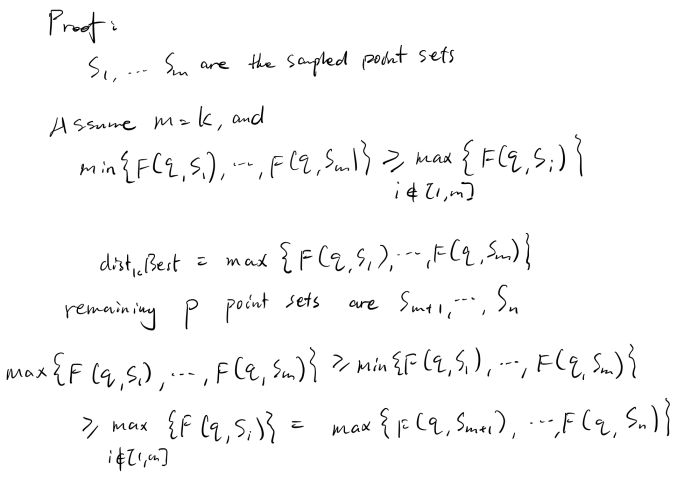
III. Proposed Methods

A. Greedy Candidate Retrieval

In [2], they mentioned that a parameter β can be introduced to represent the percentage of point sets sampled from all point sets as candidates, and the number of sampled point sets should be greater than or equal to K. Using these sampled candidates, the dist_{kBest} for the lower bound test can be quickly obtained and used for pruning in the subsequent candidate retrieval process. Unfortunately, the method of applying range search with radius as dist_{kBest} on different query representatives and taking the intersection of the results adopted in their work is not applicable to single-point queries. Since the object issuing the query is not a point set. Nevertheless, placing the point set to which the query point is referred into the candidate set earlier is still an obvious greedy step because when it is unknown whether other point sets contain $q.keyword$, the point set of $q.keyword$ is definitely included. This means that the point set of $q.keyword$ has a smaller text relevance distance relative to other "unknown" point sets. Therefore, compared to directly taking β percent of all point sets out, placing the point set of $q.keyword$ into candidate sets first can make dist_{kBest} converge faster, thus upgrade the filtering power.

B. Preprocessing and Memorization

One point that must be mentioned is that in this problem, after obtaining dist_{kBest}, we cannot simply traverse all points and calculate $S_p(q, s_i) \cdot \mathrm{Rel}(q, s_i)$ to compare with dist_{kBest} for pruning because our interest is of point sets (i.e. keywords). A point set S_{key} may have many points. Simply selecting m points from S_{key} and calculating the sum of similarity distance between m points and q may not provide a tight lower bound, especially when there are many points in S_{key}. Therefore, an easily conceivable incremental computation algorithm is to traverse point sets other than β sets we sampled and perform a lower bound test during the calculation of the similarity distance between q and the point set. This is also the basic idea used in [2]. However, this heuristic may have a relatively poor time complexity in some cases. For example, consider that we have n point sets. At the beginning, we sampled $m = \beta \times n$ $(m \geq K)$ point sets and there are still $p = n - m$ point sets left. If the similarity distances between these m point sets and q are exactly the largest m among these n. Then during our traversal of the remaining p point sets, there is no way to use our lower bound test (worst case). Figure 3 is the proof. However, if we keep updating dist_{kBest} when traversing the remaining p point sets, it is still an optimization, and we applied this strategy in our final implementation.



Proof:

$S_1, \cdots S_m$ are the sampled point sets

Assume $m = k$, and

$\min\{F(q,S_1), \cdots, F(q,S_m)\} \geq \max\limits_{i \notin [1,m]} \{F(q,S_i)\}$

$\text{dist}_{kBest} = \max\{F(q,S_1), \cdots, F(q,S_m)\}$

remaining p point sets are $S_{m+1}, \cdots, S_n$

$\max\{F(q,S_1), \cdots, F(q,S_m)\} \geq \min\{F(q,S_1), \cdots, F(q,S_m)\}$

$\geq \max\limits_{i \notin [1,m]} \{F(q,S_i)\} = \max\{F(q,S_{m+1}), \cdots, F(q,S_n)\}$

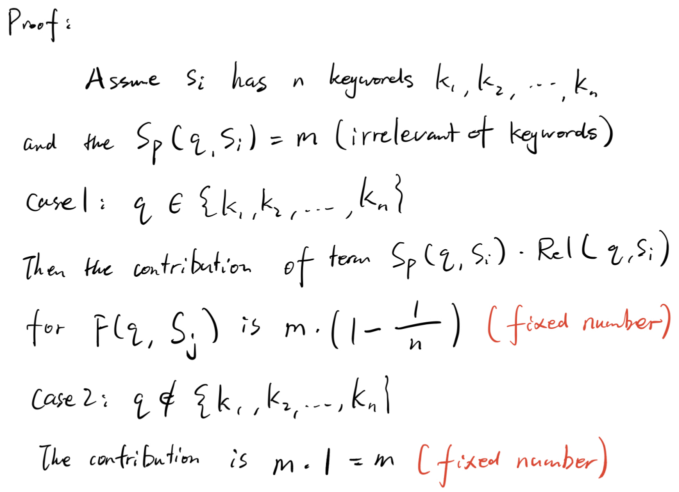
Fig. 3. Case could not use LB test

In addition, if many of the remaining p point sets have overlapping points, we may also encounter repeated calculations. The worst time complexity of this calculation overhead is $O(pd)$, where $d = |\bigcap_{i=1}^{p} S_i|$ is the size of the intersection of points in these p point sets. Taking the example from Figure 1 again, when we calculate the similarity distance from q to S_{park}, we need to calculate the term $2.0 \times 2/3$. When we calculate the similarity distance from q to $S_{mountain}$, we also need to calculate $2.0 \times 2/3$. However, calculating both of these items actually corresponds to calculating the similarity distance from q to any keywords in p_3. In other words, when $q.keyword$ and p are determined, the contribution of q to all the point sets corresponding to the keywords in p is the same. Figure 4 is the proof. Surprisingly, it is irrelevant of the type of keywords in p_3. Therefore, a good step here is to use memoization and pre-process the similarity distance from q to other points in advance. The time for this pre-processing is $O(n)$, where n is the size of the raw data. We can use this step of memoization in the function that calculates the similarity distance from q to all point sets, achieving an amortized $O(1)$ calculation time.

The implementation (pseudocode) of our algorithm is at appendix.

IV. Experimental study

We conducted all the experiments on macOS with 2 GHz Quad-Core Intel Core i5 processor and 16GB memory. Due to the miss of the original dataset in [2], this experiment use



Proof:

Assume S_i has n keywords $k_1, k_2, \cdots, k_n$

and the $Sp(q,S_i) = m$ (irrelevant of keywords)

Case 1: $q \in \{k_1, k_2, \cdots, k_n\}$

Then the contribution of term $Sp(q,S_i) \cdot Rel(q,S_i)$

for $F(q,S_j)$ is $m \cdot (1 - \frac{1}{n})$ (fixed number)

Case 2: $q \notin \{k_1, k_2, \cdots, k_n\}$

The contribution is $m \cdot 1 = m$ (fixed number)

Fig. 4. Irrelevant of keyword types

the random function in C standard library provided by lab1. In total, we use the random function to prepare $2e7$ entries of information of the post ($2e7 = U_{data}$), with each entry = $\langle keyword - list, longitude, latitude \rangle$. We assume the length of the keyword list is at most 10, and every keyword in the list is different with each other, which is also natural in real world. The way we form the collection of point sets C_s is to traverse $2e7$ entries, and for each keyword in the entry insert the point $\langle longitude, latitude \rangle$ to the corresponding point sets of the keyword. Finally, we get 6199528 point sets, and the Figure 5 is the distribution of the size of the point sets. I implement the existing methods in [2] by myself, and the source code[3] is written by C++.

Range of set size	number of point sets	percentage
[1, 9]	3393216	54.7%
[10, 99]	2606312	42.04%
[100, 999]	200000	3.23%

Fig. 5. Distribution of point set size

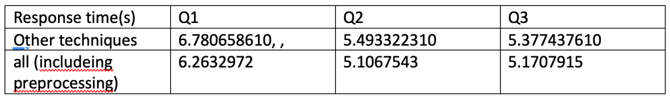
```
/**
 * @return double in [0, 1)
*/
double drand48(void) {
    return rand() / (RAND_MAX + 1.0);
}
```

A. *Performance comparison*

Effect of proposed methods. Table 6 shows that the pruning effect of reprocessing, after we combine all proposed methods, the effect is better, and we achieve faster response time. Q1 means the range of set size is [1,9], Q2 is [10, 99], Q3 is [100, 999].

Change query size. Figure 7 shows that when the size of $q.keyword$ varies, these two implementation are still stable.

[3]https://github.com/yoshino-polyu/similarity_s



Response time(s)	Q1	Q2	Q3
Other techniques	6.780658610, ,	5.493322310	5.377437610
all (includeing preprocessing)	6.2632972	5.1067543	5.1707915

Fig. 6. Effect of proposed methods.

It is shown that the size of the point set to which q.keyword is referred does not have a significant impact on query speed. At the same time, it can be observed that our implementation is 3.2 times faster in terms of response time compared to our competitors' implementation.

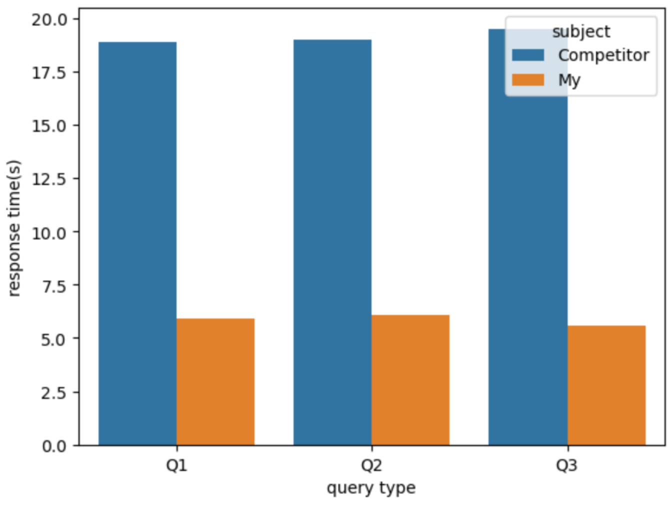


Fig. 7. Change the size of point set of $q.keyword$

Change K. Figure 8 shows that as K increases from 5 to 10, the response time of the competitor increases sharply, while my response time remains relatively constant. After K increases beyond 10, the growth rate of the competitor's response time slows down and approaches linear growth.

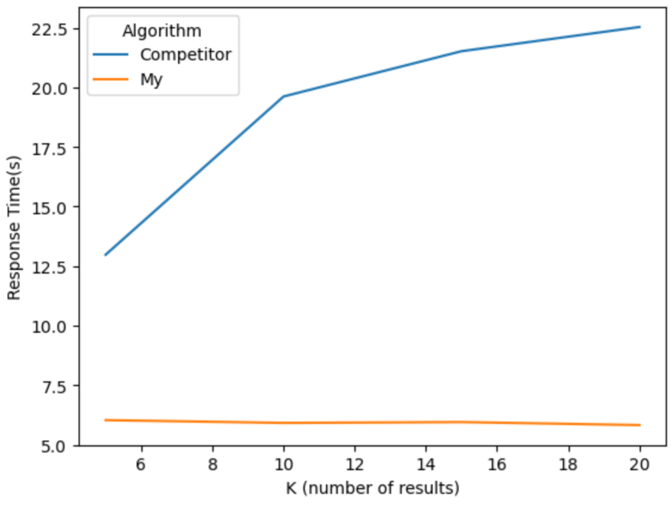


Fig. 8. Change K

References

[1] Marco D. Adelfio, Sarana Nutanong, and Hanan Samet. 2011. Similarity search on a large collection of point sets. In *ACM SIGSPATIAL*. 132–141.

[2] Z. Li, Y. Li, and M. L. Yiu, "Fast similarity search on keyword-induced point groups," in Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, Nov. 2018, pp. 109-118.

[3] E. Evans and E. J. Evans, "Domain-driven design: tackling complexity in the heart of software," Addison-Wesley Professional, 2004.

Algorithm - Similarity Search

Require: single query point q, number of results k, sampling ratio β, raw data U_{data}, all point sets C_s.

Ensure: $\beta \times |U_{data}| \geq k$, and $\beta \geq 10\%$

```
  resultSet $\leftarrow \emptyset$
  candSet $\leftarrow \emptyset$
  $dist_{kBest} \leftarrow \infty$
  for each point $p_j \in U_{data}$ do                ▷ Section III.B
      cacheSimilarity$(p_j) \leftarrow S_p(q, p_j) \cdot Rel(q, p_j)$
  candSet $\leftarrow S_{q.keyword}$                  ▷ Section III.A
  $dist_{kBest}$, candSet $\leftarrow$ Find-Candidates$(C_s, \beta)$
  for each point set $S_{key} \in C_s$ and $S_{key} \notin$ candSet do
          temp $\leftarrow$ Incremental_Distance$(q, S_{key}, dist_{kBest})$
          if temp $< dist_{kBest}$ then
              remove the largest distance from candSet
              update $dist_{kBest}$ and candSet
  return resultSet = candSet
```

```
Algorithm – Incremental_Distance
Require: query point $q$, point set $S_{key}$, $dist_k^{Best}$
  $d \leftarrow 0$
  for each point $s_i \in S_{key}$ do
      $d \leftarrow d + S_p(q, s_i) \cdot Rel(q, s_i)$
      if $d >= dist_{kBest}$ then
          return $d$
  return $d$
```