Nearest Neighbor Search with Keywords

Yufei Tao

KAIST

May 26, 2013

In recent years, many search engines have started to support queries that combine keyword search with geography-related predicates (e.g., Google Maps). Such queries are often referred to as spatial keyword search. In this lecture, we will discuss one such type of queries that finds use in many applications in practice.

Problem (Nearest Neighbor Search with Keywords)

Let P be a set of points in $\mathbb{R}^2$. Each point $p \in P$ is associated with a set W_p of terms. Given:

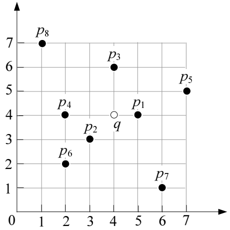
- a point $q \in \mathbb{R}^2$,
- an integer k,
- a real value r,
- a set W_q of terms

a k nearest neighbor with keywords (kNNwK) query returns the k points in $P_q(r)$ with the smallest Euclidean distances to q, where

$$P_q(r) \quad = \quad \{p \in P \mid W_q \subseteq W_p \text{ and } dist(p, q) \leq r\}.$$

where $dist(p, q)$ is the Euclidean distance between p and q.

Example: Suppose that P includes the black points $p_1, ..., p_8$.



p	W_p
p_1	$\{a, b\}$
p_2	$\{b, d\}$
p_3	$\{d\}$
p_4	$\{a, e\}$
p_5	$\{c, e\}$
p_6	$\{c, d, e\}$
p_7	$\{b, e\}$
p_8	$\{c, d\}$

- Given q as shown (the white point), $k = 1$, $r = 5$, and $W_q = \{c, d\}$, then a kNNwK query result returns p_6.
- Same query with $k = 2$ returns p_6 and p_8.

Think

What applications can you think of for this problem?

As a naive solution, we can first retrieve the set P_q of points $p \subseteq P$ such that $W_p \subseteq W_q$ (think: how to do so with an inverted index?). Then, we obtain $P_q(r)$ from P_q, and finally, obtain the query result by calculating the distances of the points in $P_q(r)$ to q.

In practice, the values of k and r are small, which makes it possible to do better than the above solution.

Let us first look at a simpler problem:

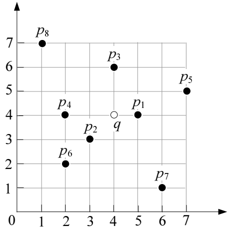
Problem (Nearest Neighbor Search)

Let P be a set of points in $\mathbb{R}^2$. Given:

- a point $q \in \mathbb{R}^2$,
- an integer k

a k nearest neighbor (kNN) query returns the k points in P with the smallest Euclidean distances to q.

Example: Suppose that P includes the black points $p_1, ..., p_8$.



p	W_p
p_1	$\{a, b\}$
p_2	$\{b, d\}$
p_3	$\{d\}$
p_4	$\{a, e\}$
p_5	$\{c, e\}$
p_6	$\{c, d, e\}$
p_7	$\{b, e\}$
p_8	$\{c, d\}$

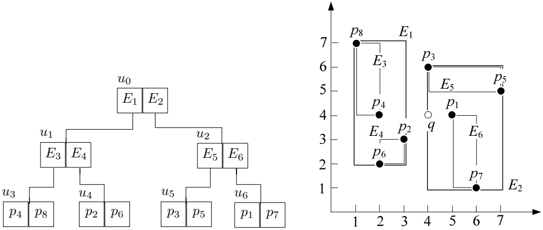
- Given q as shown (the white point) and $k = 1$, then a kNNwK query returns p_1.
- Same query with $k = 2$ returns p_1 and p_2.

Nearest neighbor search can be efficiently solved by indexing P with an R-tree T defined as follows:

- All the leaves of T are at the same level.
- Every point of P is stored in a unique leaf node of T.
- Every internal node stores the minimum bounding rectangle (MBR) of the points stored in its subtree.

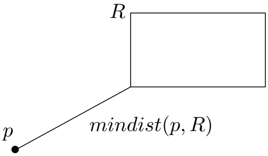
See the next slide for an example.

Example:



The left figure shows the tree whereas the right figure shows the points and MBRs.

The mindist of a point p and a rectangle R is the shortest distance between p and any point on R.



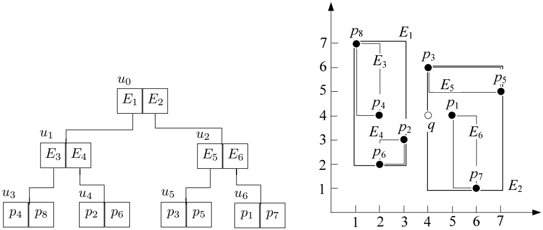
Think

How would you compute $mindist(p, R)$?

We can answer a 1NN query by a distance browsing (also called best first) algorithm:

```
algorithm best-first(T, q)
/* q is the query point; T is an R-tree */
S ← the MBR of the root of T
while (true)
    R ← the rectangle in S with the smallest mindist(q, R)
    if R is a data point p then
        return p
    elseif R is the MBR of an internal node u then
        insert to S the MBRs of all the child nodes of u
    else /* R is the MBR of a leaf node u */
        insert to S all points stored in u
```

Example:



Given a 1NN query with the query point q as shown, the algorithm accesses nodes u_0, u_2, u_1, u_4 before returning p_2.

Think

Why is the algorithm correct?

Think

How would you extend the algorithm (easily) to answer a kNN query?

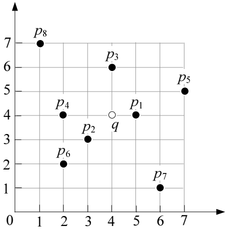
Think

If we only want to return the k nearest neighbors of a query point q that are within distance r from q, how would you extend the algorithm (easily)?

Now, let us get back to the kNNwK problem. We can create the following structure that combines the inverted index and the R-tree:

- For every term t in the dictionary, let $P(t)$ be the set of points $p \in P$ such that $t \in W_p$. Create an R-tree on $P(t)$, i.e., one R-tree per t.

Example:



p	W_p
p_1	$\{a, b\}$
p_2	$\{b, d\}$
p_3	$\{d\}$
p_4	$\{a, e\}$
p_5	$\{c, e\}$
p_6	$\{c, d, e\}$
p_7	$\{b, e\}$
p_8	$\{c, d\}$

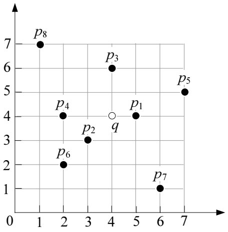
word	inverted list
a	p_1 p_4
b	p_1 p_2 p_7
c	p_5 p_6 p_8
d	p_2 p_3 p_6 p_8
e	p_4 p_5 p_6 p_7

Create an R-tree on the points in the inverted list of each word.

We now extend the best first algorithm to answer a 1-NNwK query:

algorithm best-first-1-NNwK(q, r, W_q)
/* q is the query point, r is a distance range, and W_q is a set of query words */
$S \leftarrow$ the root MBRs of the R-trees of the words in W_q
while (true)
    $R \leftarrow$ the rectangle in S with the smallest $mindist(q, R)$
    if $mindist(q, R) > r$ **then**
        return $\emptyset$
    if R is a data point p **then**
        $p.cnt++$
        if $p.cnt = |W_q|$ **then**
            return p
    elseif R is the MBR of an internal node u **then**
        insert to S the MBRs of all the child nodes of u
    else /* R is the MBR of a leaf node u */
        insert to S all points stored in u

Example:



p	W_p
p_1	$\{a, b\}$
p_2	$\{b, d\}$
p_3	$\{d\}$
p_4	$\{a, e\}$
p_5	$\{c, e\}$
p_6	$\{c, d, e\}$
p_7	$\{b, e\}$
p_8	$\{c, d\}$

word	***inverted list***
a	p_1 p_4
b	p_1 p_2 p_7
c	p_5 p_6 p_8
d	p_2 p_3 p_6 p_8
e	p_4 p_5 p_6 p_7

For q being the point shown, $r = 5$, $k = 1$, and $W_q = \{c, d\}$, the algorithm visits the points in this order: p_2, p_3, p_6, p_6, terminates after seeing the second p_6, and returns p_6.

Think

Why Line 8?

Think

This algorithm is typically much faster than the naive algorithm mentioned at the beginning when r is small. Why?

Think

How to extend the algorithm to kNNwK queries?