2010 12th International Asia-Pacific Web Conference

Aggregate Nearest Keyword Search in Spatial Databases

Zhicheng Li*, Hu Xu†, Yansheng Lu‡, Ailing Qian§
Huazhong University of Science and Technology, Wuhan, China
{**anders*, †*hxu*, §*qaling*}*@smail.hust.edu.cn*
‡*lys@mail.hust.edu.cn*

***Abstract*—Given a set of spatial points D containing keywords information, a set of query objects Q and m query keywords, a top-k aggregate nearest keyword (ANK) query retrieves k objects from Q with the minimum sum of distances to its nearest points in D such that each nearest point matches at least one of query keywords. For example, consider there is a spatial database D which manages facilities (e.g., school, restaurants, hospital, etc.) represented by sets of keywords. A user may want to rank a set of locations with respect to the sum of distances to nearest interested facilities. For processing this query, several algorithms are proposed using IR^2-Tree as index structure. Experiments on real data sets indicate that our approach is scalable and efficient in reducing query response time.**

***Keywords*-aggregate nearest neighbor; spatial keyword query; spatial databases;**

I. Introduction

Most traditional spatial queries on spatial databases such as nearest neighbor queries [1], [2], range queries [3], and spatial join [4], [5], [6] do not concern non-spatial information (e.g., name, description, and type etc.). Due to popularity of keyword search services on the Internet such as Google Earth and Yahoo Maps, many of these applications allow users to provide a list of keywords besides the spatial information of objects. Queries on spatial objects associated with textual information represented by sets of keywords, called *spatial keyword queries* [7], have received significant attention in recent years.

In this paper, we study an interesting type of spatial keyword query called *aggregate nearest keyword* (ANK) query. Given a set of data points D which contains keyword information, a group of query objects Q and m query keywords, a top-k ANK query retrieves k objects in Q with the minimum sum of distances to its nearest points in D such that each nearest point matches at least one of query keywords. It can be widely utilized in various decision support systems and multiple domains like service recommendation, investment planning, etc. For example, consider a spatial database D which manages facilities (or services) such as schools, restaurants and hospitals, represented by sets of keywords. A user wants to rank the locations with respect to the sum of distances to nearest interested facilities. The user may issue a set of locations and multiple query keywords representing his interested facilities, the result returns k best locations that minimize the summed distance to these facilities.

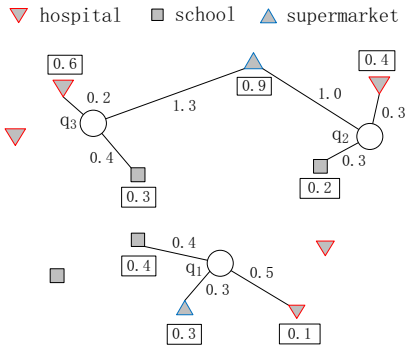


Figure 1: Example of top-k ANK query

Fig. 1 gives out a more concrete example of an ANK query. White points are apartments as query objects Q defined by a local resident. Gray points represent sample dataset of data points D with three keywords i.e., hospital, school and supermarket. The resident may be interested in the apartment q which minimizes the sum of distances to nearest hospital, school and supermarket. For instance, the distance between q_1 and its nearest neighbor which includes keyword hospital is 0.5. The distance between q_1 and its nearest school and supermarket is 0.4 and 0.3 respectively. The summed distance of q_1 is $\tau(q_1)$ = 0.5 + 0.4 + 0.3 = 1.2. Similarly, we obtain $\tau(q_2)$ = 0.3 + 0.3 + 1.0 = 1.6 and $\tau(q_3)$ = 0.2 + 0.4 + 1.3 = 1.9. $\tau(q_1)$ is minimum and hence q_1 is returned as the best result to the user. In this example, each gray point is associated with only one keyword. Actually a spatial object is associated with a set of keywords rather than one keyword in many real applications. Each point in D may be associated with multiple keywords. In this case, the nearest hospital and school of a query point may be the same point in D.

Top-k spatial preference query (SPQ) proposed by Yiu et al. in [8] ranks objects based on the qualities of features in their spatial neighborhood. The objects which maximize the aggregate function on the preference value of nearest neighbors are returned as result. In above example, the value in the box is the preference value of the data point. Then, q_3 with maximum value of 0.3 + 0.6 + 0.9 = 1.8 is returned as the best result for SPQ. However, the nearest

978-0-7695-4012-2/10 $26.00 © 2010 IEEE
DOI 10.1109/APWeb.2010.25

supermarket of q_3 is too far away and a resident may choose q_1 rather than q_3. Our query orders the objects by the summed distance of its nearest neighbors. The user may prefer the nearest distance rather than the preference value defined by others. In addition, the method using the upper bound of the summed maximum quality used in SPQ does not fit in our query.

The query combining k-NN query and keyword search is proposed by Felipe et al. [7], which uses IR^2-Tree to index spatial objects and keywords. The query specifies one query location and returns its nearest neighbor which must contain all query keywords. The naive way to solve ANK query is to compute the summed distance of each query object and select k query objects with the minimum summed distance as the result. For each query keyword w, first find the distance between one query object q and its nearest neighbor in D which contains query keyword w using the method in [7]. Then sum the distances and get the summed distance of q. At last it selects the object with the minimum summed value as the result from all query objects. Our example involves only three keywords, the database D is likely to involve more keywords and the number of query keywords m may be large in real scenario. Suppose the number of node accesses for finding the nearest neighbor of one query point which contains an query keyword w_i is $N(w_i)$. The cardinality of Q is $|Q|$. Then the number of node accesses in total is $|Q| \cdot \sum_{i=1}^{m} N(w_i)$. This will be prohibitively expensive when both $|Q|$ and m are large.

In this paper, we present an approach to efficiently answer top-k ANK query, which uses a pruning heuristic to filter large portion of Q and reduce the number of node accesses significantly. Our work has the following contributions:

- We propose an interesting type of spatial keyword query called aggregate nearest keyword (ANK) query.
- A heuristic is proposed to prune large portion of query objects and reduce the number of nodes accesses significantly.
- We propose a recursive hierarchical join algorithm to answer the ANK queries efficiently.
- The extensive experiments on real data sets demonstrate that our algorithm is not only effective in reducing ANK query response time, but also exhibits good scalability with respect to the cardinality of query objects and the number of query keywords.

The remainder of this paper is organized as follows. In Section II we review the related work. Section III formally defines the top-k ANK query and presents the suggested solutions. In Section IV, our algorithms are experimentally evaluated with real data sets. We concludes the paper in Section V.

II. Related Work

Nearest neighbor (NN) queries on a spatial database are a classical problem. The existing algorithms for point NN queries using R-trees [9] follow the branch-and-bound paradigm, utilizing lower and upper bounding functions to prune the search space. The k-NN algorithm for R-trees [1] traverses a R-tree while maintaining a list of k potential nearest neighbors in a priority queue in a Depth-First (DF) manner. The DF algorithm is sub-optimal, i.e., it accesses more nodes than necessary. The Best-First (BF) algorithm [2] achieves the optimal I/O performance by maintaining a heap with the entries visited so far, sorted by their $mindist$. As shown in [2], DF can be more I/O consuming than BF. However, DF requires only bounded memory and at most a single tree path resides in memory during search.

The closest pair queries (CPQ) are a combination of spatial join and nearest neighbor queries, which find the pair with the minimum distance among all pairs from two data sets. Hjaltason and Samet [6] proposed an incremental algorithm based on a priority queue for the closest pair query between two R-tree indexes. Non-incremental recursive and iterative branch-and-bound algorithms have been discussed by Corral et al. [5]. The difference between NN and CPQ is that the algorithms of the latter access two index structures (one for each data set) and utilize the distance function of two intermediate nodes to prune the pairs. Several techniques have been suggested in literature [10], [11] to solve group nearest neighbor (GNN) queries. GNN is the most similar one to our query, the difference is that GNN retrieves k points from D with the minimum sum of distances to all points in Q and our query returns k points from Q with the minimum summed distance to some points of D such that each point matches at least one of query keywords. ANK can be reviewed as a constrained GNN query essentially.

Top-k spatial preference query (SPQ) proposed by Yiu et al. in [8] ranks objects based on the qualities of features in their spatial neighborhood. The query orders the objects by an aggregate function on the preference value of its nearest neighbors. Our query orders the objects by summed distance of its nearest neighbors. In addition, the method using the upper bound of the summed maximum quality to process the top-k SPQ does not fit in our query.

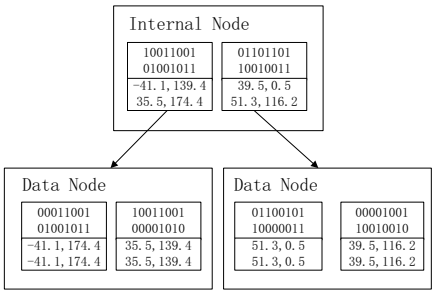


Figure 2: Example of IR^2-tree

Recently, queries on spatial objects which are associated with textual information represented by a set of keywords, have received significant attention. Several spatial keyword queries and different indexes have been proposed [12], [13],

[14], [15], [7], [16]. Hariharam et al. [15] introduced a type of query combining range query and keyword search (RK). The objects returned are required to intersect with the query region and contain all the user-specified keywords. A hybrid index of R*-tree and inverted index, called KR*-tree, is used during query processing. Another typical spatial keyword query called NNK which combines *k*-NN query and keyword search was proposed by Felipe et al. [7], which uses a hybrid index of R-tree and signature file, called IR^2-Tree. Fig. 2 shows an internal node and two data nodes of an IR^2-Tree. In particular, each node of IR^2-Tree contains both spatial and keyword information. The former is in the form of MBR and the latter is in the form of signature file. If there are n keywords in the database, keywords for each node can be represented using a bitmap of size n, with an 1 indicating its existence in the node and a 0 otherwise. This representation incurs little storage overhead. Moreover, it can accelerate the checking process of keyword constraints due to the relatively high speed of binary operations.

NNK specifies only one query location and our query specifies a set of query locations. RK and NNK both require the qualified points must contain all query keywords, our query just requests that the data point contains at least one query keyword . Zhang et al. [16] proposed a novel spatial keyword query called the mCK query and a new index bR*-tree for query processing. The mCK query aims at finding m closest keywords in the leaf nodes matching the query keywords. The bR*-tree is an extension of IR^2-tree. Besides the bitmap describing the keyword information, each node also stores the keyword MBR which summarizes the spatial location of each keyword w in the node. The bR*-tree contains more information than IR^2-Tree and can significantly strengthen the power of pruning rules. However the cost of storing the keyword MBRs can be very high when the number of keywords in the database is large. Hence our solutions use IR^2-Tree as the basic index structure.

III. Processing Aggregate Nearest Keyword Query

A. Problem Definition

We define a spatial object p as a pair in the form ($p.l$, $p.t$), where $p.l$ is a location descriptor in the multidimensional space, and $p.t$ is the textual description represented by sets of keywords. Let D be the universe of all objects in the database. Given a group of query points $Q = \{q_1, q_2, \ldots, q_n\}$ and a set of m query keywords $Q_w = \{w_1, w_2, \ldots, w_m\}$. A top-*k* spatial keyword query retrieves k query points from Q with the minimum sum of distances.

Definition 1: $\forall\ q \in Q$, the *nearest keyword* w of q is a point p_i ($p_i \in$ D) which contains keyword w such that $\forall\ p_j \in D/p_i$ and $w \in p_j.t$, we have $dist(p_i, q) \leq dist(p_j, q)$.

The function $dist(q, p_i)$ is the Euclidean distance between q and p_i. We use function $nearkey(q, w)$ to present the distance between q and its nearest keyword w. Then the summed distance of q is defined as $\sum_{i=1}^{m} nearkey(q, w_i)$, where $w_i \in Q_w$. The ANK query returns the k query points in Q with the minimum summed distance.

Here each query point in Q only contains the spatial information. In the remainder of this paper, we assume neither Q nor D fits in memory, indexed by R*-tree and IR^2-Tree respectively. We use notation R_Q to present the R*-tree of Q and R_D to present the IR^2-Tree of D. Based on the above indexing scheme, we develop various algorithms (PA, CPB and HJ) for processing the top-k ANK query.

B. Probing Algorithm

We firstly introduce a brute-force solution which computes the summed distance of every query object in Q. For each query point q, we utilize the distance-first IR^2-Tree algorithm in [7] to compute the distance to its nearest keyword w_i in Q_w. Then sum the distances and get the summed distance of q. At last the query object with the minimum summed distance will be returned as the result.

Algorithm 1: Probing Algorithm (PA)

```
algorithm PA(Node: node of R_Q)
begin
    if Node is not data node then
        read the child node N_c of Node;
        PA( N_c );
    else
        foreach entry e in Node do
            currdist = 0;
            for c = 1 to m do
                kwddist_c = IR2NearestNeighbor(e, w_c);
                currdist += kwddist_c;
                if currdist ≥ γ then
                    break;
            if currdist < γ then update W_k and γ;
end
```

Algorithm 1 is the pseudo-code of Probing Algorithm (PA), which retrieves the query result by computing the summed distance of every query point in Q. Here the second parameter of the function *IR2NearestNeighbor* [7] is only one keyword w_c. The function *IR2NearestNeighbor* returns the distance between q and its nearest keyword w_c, where q is a query point in a data node. Initially, the algorithm is invoked at the root node of R_Q (Node = $R_Q.root$). The procedure is recursively applied on tree nodes until a data node is reached. The function *IR2NearestNeighbor* is called repeatedly for computing the summed distance. In particular, the query entry e is pruned as soon as its current summed distance for query keywords is already not smaller than γ, where γ presents the top-*k* distance found so far. W_k is a max-heap which manages the top-*k* results. W_k and γ are updated when the last summed distance is less than γ. W_k and γ have the same meanings in the rest of this paper.

Another way to obtain the summed distance of q is to incrementally retrieve its nearest neighbors until all query keywords appear. During the output of nearest neighbors, we can get the distance of every nearest keyword through modifying the function IR2NearestNeighbor. Since the experiments show that it is more expensive, we adopt the method used in PA. PA however has a significant drawback. It requires to compute the summed distance of every query point q in Q , which in turn incurs multiple accesses to the same node of R_D and results in a large number of index and data accesses, although the distance-first IR^2-tree algorithm traverses the tree in a Best-First manner.

C. Closest Pair Based Algorithm

In [10], Papadias et al. proposed the group closest pairs (GCP) method which extends the closest pair (CP) algorithm to process the GNN queries. Assume the result of an incremental closest pair algorithm is a list of closest pairs <p_i, q_j> ($p_i \in D$, $q_j \in Q$) in ascending order by distance. GCP utilizes the upper bound to prune the subsequent pairs which can't contribute to the final result and halt the algorithm when the distance of closest pair is greater than the global threshold. Inspired by this, we propose the closest pair based (CPB) algorithm to solve our problem.

The distance of the pair <p_i, q_j> in which p_i firstly contains the keyword w is denoted as $keydist_w(q_j)$. In other words, $keydist_w(q_j)$ is the distance between q_j and its nearest keyword w. Consider that we keep a list $dislist$ <w, $keydist_w(q_j)$> for every query point q_j, where the maximum size of $dislist$ equals the number of query keywords m. If p_i of a closest pair firstly contains a keyword, the keyword and the distance of pair will be inserted into $dislist$ of q_j. When the size of $dislist$ is equal to m, the sum of distances of q_j is achieved. If this summed distance is smaller than the global distance γ found so far, q_j becomes the best point. A global map G_{map} is used to keep the $dislist$ of all query objects and the key of the map is the id of query object. The maximum size of G_{map} is the cardinality of Q. The $keycounter(q_j)$ and $currdist(q_j)$ are defined as the size and the current accumulated distance of $dislist$ respectively. Before the first best point is found, all query objects appear in the pairs are qualified and need to be kept in G_{map}. When the first best point is found, the query objects will be gradually pruned from G_{map} based on the following heuristic:

Heuristic 1: Assume that the current closest pair is <p_i,q_j> and in which p_i firstly contains the keyword w. q_j can be immediately deleted from G_{map} if:

$$(m - keycounter(q_j)) \cdot dist(p_i, q_j) + currdist(q_j) \geq \gamma$$

If q_j in G_{map} is pruned, all the subsequent pairs that include q_j will be discarded. Fig. 3 shows a possible scenario of G_{map} during the execution of CPB. Assume γ is 14. If the pair is < q_1, p_i > in which p_i contains keyword w_2 and the distance of the pair is 4, q_1 is kept since it does not satisfy heuristic 1 ($((3-1) \times 4 + 4) < 14$). We also can utilize heuristic 1 to clear G_{map}. Here q_2 and q_3 are pruned from G_{map} according to heuristic 1 ($5 + 6 + 4 > 14$ and $4 + 2 + 8 > 14$), since the distance of next pair is greater than the distance of current pair.

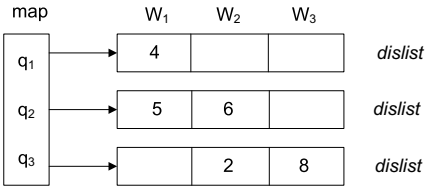


Figure 3: Example of CPB (γ = 14)

Algorithm 2: Closest Pair Based Algorithm (CPB)

```
begin
  repeat
    output next closest pair <p_i, q_j>
    /* p_i contains at least one query keyword */
    if q_j is not in map then
      if γ < ∞ then continue;
      else insert the q_j and its dislist into map;
    else
      foreach query keyword w in p_i do
        if w is not in dislist_j then
          dislist_j ← <w, dist(p_i, q_j)>;
      if size of dislist_j is m then
        if currdist(q_j) < γ then
          update W_k and γ; initialize T = 0;
          foreach point q in map do
            /*clear the map*/
            if satisfy the heuristic 1 then
              remove q from map;
            else
              calculate threshold t of q;
              T = max(t, T);
        else remove q_j from map;
      else
        if γ < ∞ then
          if satisfy the heuristic 1 then
            remove q_j from map;
          else
            calculate threshold t of q_j;
            T = max(t, T);
    until (γ < ∞) and (|p_i q_j| ≥ T or map is empty) ;
  end
```

We utilize the maximum upper bound distance T of all the subsequent pairs to halt the algorithm. For each remained q_j in G_{map}, we compute its threshold t_j = ($\gamma - currdist(q_j)$)/($m - keycounter(q_j)$). The global threshold T is the maximum value of individual thresholds t_j, which is the largest distance of the subsequent closest pair that can lead to a better result than the current one. In Fig. 3, for instance, there are three remained points in G_{map} and

the size of their corresponding $dislist$ is less than m. The threshold t_1 of q_1 is $(14-4)/(3-1)$ = 5. Similarly, threshold of q_2 and q_3 is 3 and 4 respectively. Then the global T is the maximum value 5. If the distance of the next pair is no smaller than T, the pair including all its subsequent pairs can be discarded and hence we halt the algorithm.

The pseudo-code of CPB algorithm is shown in Algorithm 2. For speeding up the output of CP, the constraint of keyword information is utilized. The pairs in which the internal and data nodes of R_D do not contain any query keyword will be pruned before output. For each pair <p_i, q_j>, the algorithm first searches the query object q_j in G_{map}. If not found and γ is found ($\gamma < \infty$), q_j will be ignored. The reason is that the distance of each subsequent pair includes q_j multiplies *m* is larger than γ. We clear G_{map} using heuristic 1 and update T when current summed distance is less than γ. If the value γ is found and G_{map} are empty or the distance of the current pair is no smaller than T, the algorithm terminates.

CPB applies an incremental CP algorithm that must keep large amounts of closest pairs in the heap. When the distance between a query point q and its nearest keyword w is far and there are too many points in D are adjacent to q, GCP must output a large number of closest pairs. The number of such pairs in the worst case equals the cardinality of the Cartesian product of the two data sets in worst case. If the size of the heap exceeds the available memory, a portion of data is moved to the disk. The heap management technique in [2] is used in this case in our implementation. As shown in Section IV, the CPU cost of CPB is very high due to the huge heap requirements when the datasets is large.

D. Hierarchical Join Algorithm

In this section, we propose a branch-and-bound algorithm called hierarchical join (HJ) algorithm, which can significantly reduce the number of query objects and data points to be examined. The key idea is to compute the upper bound of the summed distance using metric function. If the minimum possible summed distance of internal node N_q in R_Q excesses the upper bound, then there is no need to access the subtree of N_q. The function MINMINDIST [5] is chosen to compute the minimum possible distance of two internal nodes from R_D and R_Q. Based on above idea, we propose the following heuristic.

Heuristic 2: Assume N_q is an internal node in R_Q and N_p^l is a set of all internal nodes at a certain level l in R_D. Let $minkey(N_q,w_i)$ be the minimum MINMINDIST(N_q, N_{pi}^l) where $\forall\ N_{pi}^l \in N_p^l$ and N_{pi}^l contains the query keyword w_i. Then internal node N_q and its subtree can be pruned if:

$$\sum_{i=1}^{m} minkey(N_q, w_i) >= \gamma$$

In other words, all query points contained by N_q cannot produce a summed distance less than the top-*k* result γ. If N_q is pruned, there is no need to check its children nodes and the nodes of the corresponding N_p^l are pruned as well. Then the number of node accesses of R_D and R_Q are both reduced.

Fig. 4 shows an internal node N_q in R_Q and all four internal nodes N_{pi} in the same level of R_D. R_D totally includes three keywords w_1, w_2 and w_3. As shown in the figure, N_{p1}, N_{p2} and N_{p3} all contain the keyword w_1 indicated by their bitmaps. The MINMINDIST between N_{p3} and N_q is minimum, hence $minkey(N_q, w_1)$ is 2. Similarly, $minkey(N_q, w_2)$ is 3 and $minkey(N_q, w_3)$ is 7. Then the minimum possible summed distance of N_q is 2 + 3 + 7 = 12. If γ is less than 12, N_q will be pruned. Heuristic 2 also applies when the query node N_q is a point. The experiments in Section IV show heuristic 2 can prune large portion of query points.

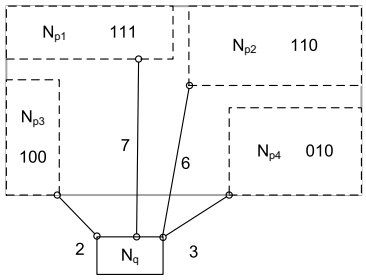


Figure 4: Example of heuristic 2

Algorithm 3 is the pseudo-code of HJ algorithm utilizing heuristic 2. The candidate entries of nodes in the same level of R_D are kept in a list EL. Initially all entries of root node of R_D is inserted into EL and $qNode$ is the root node of R_Q. The function HJ is called recursively till all computations are completed. If the entries of EL are internal node or qNode is internal node node, we compute the MINMINDIST between one entry $qent$ of $qNode$ and every entry $kent$ in EL. If the MINMINDIST is less than γ, we add all entries of the node pointed by $kent$ into a new list $entlist$. Otherwise, $kent$ will be pruned. We utilize this condition to filter portion of nodes at the same level of R_D. If the $entlist$ is empty at last, the query node pointed by $qent$ will be pruned because it cannot contribute to the final result.

The value of $minkey(kent, w_i)$ is stored in the vector $kmin$ as an element. The length of vector $kmin$ is equal to the number of query keywords m. The value of $minkey(kent, w_i)$ ($kmin[i]$) is updated when $kent$ contains w_i and the MINMINDIST is smaller. When the MINMINDIST between $qent$ and all entries in EL is computed, the $kmin$ of $qent$ is retrieved. If the sum of the distances in $kmin$ satisfies heuristic 2, $qent$ will be ignored. Then the entries of $entlist$ corresponding to $qent$ also are pruned as well.

The function HJ is called recursively following a Depth-First searching strategy. The fix-at-leaves [5] approach is chosen to treat the case when R_Q and R_D have different heights. When the algorithm is called for a data node

Algorithm 3: Hierarchical Join Algorithm (HJ)

```
HJ(qNode: node of R_Q, EL: a list of entries in R_D)
begin
  if EL( or qNode ) is internal node then
    foreach entry qent of qNode do
      initial a new list of entries entlist;
      initial a new vector kmin[m];
      foreach entry kent in EL do
        mmdist = MINMINDIST(qent, kent);
        if mmdist < γ then
          if EL is internal node then
            get the node kn pointed by kent;
            foreach entry e in kn do
              add e into entlist;
          else
            add kent into entlist;
          foreach w_i in kent do
            kmin[i]=min(mmdist, kmin[i]);
      if sum(kmin[1], ..., kmin[m]) ≥ γ then
        continue; /* pruned by heuristic 2 */
      if entlist is not empty then
        if qNode is datanode then
          HJ(qNode, entlist);
        else
          foreach child node qn of qNode do
            HJ(qn, enList);
  else
    foreach query point q in qNode do
      calculate summed distance sumdist of q ;
      if sumdist < γ then
        update W_k and γ;
end
```

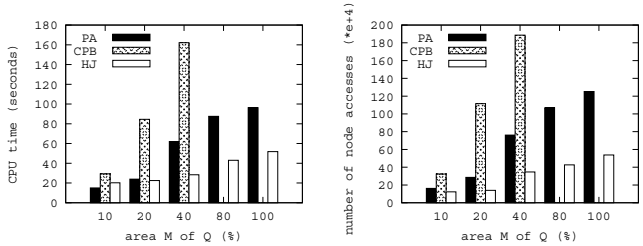
of R_Q on the one hand and an internal node in R_D on the other hand, downwards propagation stops in R_Q while propagation in R_D continues, and vice versa. When $qNode$ and the nodes pointed by entries in EL are both data node, we calculate the nearest keyword w_i in Q_w of every query point in $qNode$ and get the summed distance $sumdist$. W_k and γ will be updated when $sumdist$ is less than γ.

HJ is non-incremental, i.e., the user can not have any output until the whole algorithm terminates. The algorithm can be easily modified to an incremental algorithm, if we review one query node in R_Q and all nodes at one level of R_D as a pair like the closest pair query in [6]. But it requires a huge size heap and costs more CPU time when the cardinality of data sets is large.

IV. Experiments

In this section, we compare the performance of PA, CPB and HJ using real datasets. Historical Features (HF) and Populated Places (PP) consist of numerous geographic information and feature attributes of the entire United States (downloadable from http: //gnis.usgs.gov/domestic/download_data.htm) are used. Both datasets are plain text files where each spatial object occupies a row with multiple other attributes. We extract the FEATURE_CLASS attribute of each row in HF as the keyword of object. After cleaning and format transformation on the raw data, HF contains 110,950 data points and 17 keywords while PP contains 175,869 data points. We use HF as D indexed by IR^2-tree and PP as Q indexed by R*-tree. The testing platform is on a desktop with an AMD Turion 1.8G CPU and 1 GB main memory.

All algorithms were implemented in C++ using its standard template library. In our implementation of IR^2-tree, the page size is set to 4K and the maximum number of entries in internal nodes is set to 30. The same setting is applied to R*-tree. We study the performance of all algorithms with respect to various parameters and only one parameter varies while the others are fixed in each experiment. The average number of node accesses (NA) and CPU time are evaluated for all experiments.



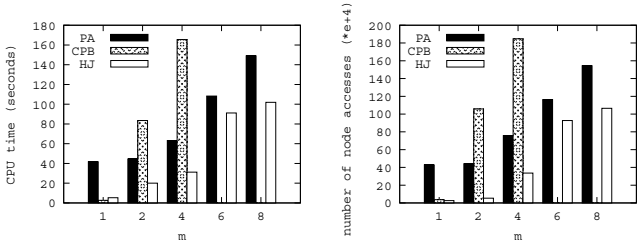
(a) Execution Time (b) Node Accesses

Figure 5: Results for varying data size of Q

The first experiment shows the effect of the cardinality of Q. We assume that the data space of both D and Q have the same centroid, but the area M (MBR of Q) varies between 10% and 100% of the data space of Q. The number k of retrieved result points is fixed at 8 and the number m of query keywords is 4. Figure 5 compares the NA and CPU cost of three algorithms. When M is 10%, PA has the best response time. But when M is over 10%, HJ becomes the best one and its I/O and CPU cost are both minimum. The number of NA and CPU time of CPB increases dramatically with M. CPB has the worst performance in all cases. Its cost is omitted from the figure when M is over 40% since the execution time is too long. This is because the heap requirement becomes too huge when M extends 40%. The I/O and CPU cost of PA and HJ both increase with M linearly. As we can see that the cost of HJ do not increase a lot compared to that of PA. The good performance of HJ is led by using heuristic 2 to prune the considerable query points. Observe that a large portion of NA of HJ are the node accesses of R_D during the experiment. HJ shows the best scalability when the size of query objects varies.

Figure 6 runs the experiment to test scalability in terms of m. M is set to 40% and k is fixed at 8. When m equals 1, the performance of CPB is the best. In this case CPB becomes similar to the performance of closest pair query.

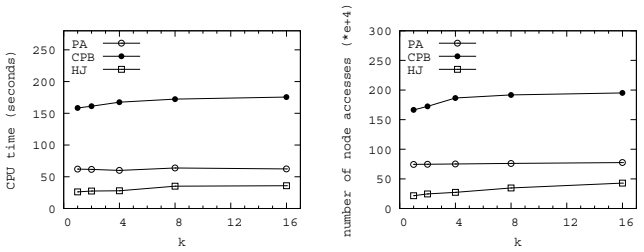
But the CPU time of CPB increases rapidly and becomes the worst one when m is larger than 1. The cost of CBP is omitted from the figures when m is larger than 4 due to the long execution time. HJ is the best algorithm when m is larger than 1 though the cost increases with m.



(a) Execution Time (b) Node Accesses

Figure 6: Results for varying m

In Figure 7, we vary the number k of retrieved points from 1 to 16 by setting m to 4 and M to 40%. The experiments show that the value of k does not influence the CPU cost of any method significantly. The relative performance of the algorithm is similar to previous diagrams and HJ is clearly the most efficient method , followed by PA. Observe that NA of CPB and HJ follow the same trend, while PA shows a different trend, because the cost of PA slightly depends on the top-k distance. The CPU time of HJ increase smoothly. This is because the large portion of the increment of NA in HJ is the node accesses of R_D, it does not influence the CPU cost significantly. In summary, the best algorithm



(a) Execution Time (b) Node Accesses

Figure 7: Results for varying k

for ANK queries depends on both the cardinality of query objects and the number of query keywords. When M is 10%, PA outperforms the other two algorithms and CPB shows its superiority when m equals 1. But when M is over 10% or m is larger than 1, HJ has the best performance and remarkable scalability.

V. Conclusion

A new type of query ANK is introduced in this paper. We propose several algorithms for processing ANK query using IR^2-tree as index structure. First we introduce the brute-force solution and the closest pair based algorithm inspired by GCP [10]. The recursive hierarchical join algorithm is proposed at last. HJ utilizes a pruning heuristic to prune the query objects and reduce the cost significantly. Our performance study on real data sets demonstrates that HJ has remarkable scalability in terms of number of query keywords and significantly outperforms the other two approaches when the cardinality of query objects is large.

References

[1] N. Roussopoulos, S. Kelley, and F. Vincent, "Nearest neighbor queries," in *SIGMOD Conference*, 1995, pp. 71–79.

[2] G. R. Hjaltason and H. Samet, "Distance browsing in spatial databases," *ACM Trans. Database Syst.*, vol. 24, no. 2, pp. 265–318, 1999.

[3] B.-U. Pagel, H.-W. Six, H. Toben, and P. Widmayer, "Towards an analysis of range query performance in spatial data structures," in *PODS*, 1993, pp. 214–221.

[4] T. Brinkhoff, H.-P. Kriegel, and B. Seeger, "Efficient processing of spatial joins using r-trees," in *SIGMOD Conference*, 1993, pp. 237–246.

[5] A. Corral, Y. Manolopoulos, Y. Theodoridis, and M. Vassilakopoulos, "Closest pair queries in spatial databases," in *SIGMOD Conference*, 2000, pp. 189–200.

[6] G. R. Hjaltason and H. Samet, "Incremental distance join algorithms for spatial databases," in *SIGMOD Conference*, 1998, pp. 237–248.

[7] I. D. Felipe, V. Hristidis, and N. Rishe, "Keyword search on spatial databases," in *ICDE*, 2008, pp. 656–665.

[8] M. L. Yiu, X. Dai, N. Mamoulis, and M. Vaitis, "Top-k spatial preference queries," in *ICDE*, 2007, pp. 1076–1085.

[9] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *SIGMOD Conference*, 1984, pp. 47–57.

[10] D. Papadias, Q. Shen, Y. Tao, and K. Mouratidis, "Group nearest neighbor queries," in *ICDE*, 2004, pp. 301 – 312.

[11] M. L. Yiu, N. Mamoulis, and D. Papadias, "Aggregate nearest neighbor queries in road networks," *IEEE Trans. Knowl. Data Eng.*, pp. 820–833, 2005.

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

[13] B. Martins, M. J. Silva, and L. Andrade, "Indexing and ranking in geo-ir systems," in *GIR '05: Proceedings of the 2005 workshop on Geographic information retrieval*, 2005, pp. 31–34.

[14] Y.-Y. Chen, T. Suel, and A. Markowetz, "Efficient query processing in geographic web search engines," in *SIGMOD Conference*, 2006, pp. 277–288.

[15] R. Hariharan, B. Hore, C. Li, and S. Mehrotra, "Processing spatial-keyword (sk) queries in geographic information retrieval (gir) systems," in *SSDBM*, 2007, p. 16.

[16] D. Zhang, Y. M. Chee, A. Mondal, A. K. H. Tung, and M. Kitsuregawa, "Keyword search in spatial databases: Towards searching by document," in *ICDE*, 2009, pp. 688–699.