A New Algorithm for m-Closest Keywords Query over Spatial Web with Grid Partitioning

Yuan Qiu*[1], Tadashi Ohmori*[2], Takahiko Shintani*[3], Hideyuki Fujita*[4]
*Graduate School of Information Systems
The University of Electro-Communications,
1-5-1 Chofugaoka,Chofu,Tokyo 182-8585
Email: [1]kyuu_genn@hol.is.uec.ac.jp, { [2]omori, [3]shintani, [4]fujita}@is.uec.ac.jp

***Abstract*—In this paper, we focus on the issue of the m-*closest keywords* (mCK) query over spatial data in the Web. The mCK query is a problem to find the optimal set of records in the sense that they are the spatially-closest records that satisfy m user-given keywords. The mCK query was proposed by Zhang et al[1]. They assumed a specialized R*-tree to store all records and proposed an Apriori-based enumeration of MBR-combinations. However, this assumption of the prepared R*-tree is not always applicable; Twitter or Flickr provides only records having position information without any prepared data-partitioning. Many services like Google Maps only provide grid partitioning at most. Thus, in this paper, we do not expect any prepared data-partitioning, but assume that we create a grid partitioning from necessary data only when an mCK query is given. Under this assumption, we propose a new search-strategy termed *Diameter Candidate Check* (DCC), and show that DCC can efficiently find a better set of grid-cells at an earlier stage of search, thereby reducing search space greatly.**

I. Introduction

Today's new web data such as Twitter or Flickr attach point-of-location information to each of their records. They are called *Spatial Web*. How to extract useful information from such spatial web is currently a hot issue [2,3,4,5,9]. Along these research trends, the m-*closest keywords* (or, mCK) query was proposed by Zhang et al[1]. This is a query to find the spatially closest set of records where those records must satisfy m keywords given as a query [1]. An individual record of spatial web data has texts and a location field, and it is denoted by an *object* from here on. According to the original definition of [1], the mCK query is defined as follows:

[mCK query]: Given a database of objects $D = \{o_1, o_2, ..., o_n\}$ and a distance $dist(o_x, o_y)$ for any two objects in D, let $O = \{o_{i1}, o_{i2}, ..., o_{il}\}(\subseteq D)$ be a set of objects, termed an *object-set*. Let also $diam(O) = max_{o_x, o_y \in O} dist(o_x, o_y)$ $(x \neq y)$ be termed a *diameter* of O. Then, the mCK query of m given keywords $Q = k_1, k_2, ..., k_m$ is a query to find the object-set $O_{opt} = \{o_{i1}, o_{i2}, ..., o_{il}\}$ $(l \leq m)$ where O_{opt} has the smallest diameter among all possible object-sets that satisfy all keywords of Q.

The mCK query is used to find the optimal object-set O_{opt} that satisfies the m keywords in D. For example, in case of Flickr, which is a photo-sharing service where each photo has a meta-data description including content-tags and a point-of-location field, O_{opt} is recognized as the best location satisfying all the keywords [1,7], where each photo's content-tags satisfy some of Q and every keyword in Q is covered by some photos of O_{opt}, and those photos in O_{opt} are closer than those of any other object-set.

To solve the mCK query problem, the study of Zhang [1] assumes a specialized R*-tree to store all records in preparation, and it proposed an Apriori-based enumeration of MBR combinations. However, this assumption of R*-tree is not applicable to all cases; Twitter or Flickr just provides only 'bare' records having location information, or, at most, some major services like Google Maps only provide grid-style partitioning. We consider that another technique is still necessary in order to compute mCK queries over such spatial web-data services on demand, without any constraint on server-side data organization.

To this end, this paper proposes a new search strategy working efficiently for mCK queries over grid-partitioned data. Furthermore we do not set any prepared data-partitioning. Instead, we assume that necessary grid partitioning is created from necessary data when an mCK query is given. Under this setting, we propose a new strategy named *Diameter Candidate Check* (DCC). DCC can efficiently find a smaller set of grid-cells at an earlier stage of search and can reduce search space.

Section II describes our assumption of executing mCK queries by creating grid-partitioning on demand, and models the search space of the mCK query over the grid. Section III describes DCC and its implementation based on a priority-based nested-loop search. Section IV describes further pruning rules. Section V shows that DCC outperforms Zhang's approach in case of skewed datasets and Flickr data.

II. Problem setting

A. Zhang's Apriori-based method on bR*-tree

Here we outline the Zhang's method[1]. It works on a specialized R*-tree, a bR*-tree, containing all objects. In a bR*-tree, MBRs are set in the same way as those in an ordinary R*-tree, but each MBR M is also given a *keyword MBR* as a side information. Zhang's method uses an Apriori-style enumeration of a set of MBRs from length-1 (one MBR) to length-m itemsets, until it finds a possible set of MBRs which must satisfy all keywords and is expected to contain an object-set of a shorter diameter. If an itemset of MBRs passes this test, these MBRs are decomposed into the set of sub-MBRs, and a recursive test continues until the optimal answer is determined.

The Zhang's method performs well especially when one MBR (or, one object) satisfies multiple keywords. In contrast,

978-1-4799-8676-7/15/$31.00 copyright 2015 IEEE
SNPD 2015, June 1-3 2015, Takamatsu, Japan

the method is weak when any set of mutually-close MBRs does not satisfy Q. In that case, the Apriori must enumerate too many itemsets of MBRs. To avoid it, the method depends on how well a bR*-tree clusters the optimal answer into one MBR of an upper level. (We test this point in Section V.)

B. Our setting

Fig.1 is our assumption of executing mCK queries over spatial web objects. When a query of m-keywords is submitted, we load objects associated with each keyword k_i from one or multiple datasets and then create a hierarchical grid partitioning G_i for each k_i. Thus m grid indexes are built.

Fig.2(a) is an example of spatial distribution of some objects, and each object is associated with one keyword among A, B, C, D. When $Q = \{A, B, C, D\}$ is given, four hierarchical grids G_A, G_B, G_C, G_D are created as shown in Fig.2(b). In a hierarchical grid partitioning of a fixed degree(4x4=16, as an example) of equi-sized partitioning at each level, each cell of the grid partitioning is uniquely denoted by an ordinal integer i. (e.g., let $i = 0$ be the root node. Then $i = 17$ is the 1st cell of the level of 2(=1 + 17 mod 16)). In the following, the symbol $A[i]$ refers to the i-th cell of the grid corresponding to the keyword A. Such $A[i]$ is termed a *node*. When Q is $\{A, B, C, D\}$, the set of nodes $\{A[i], B[j], C[k], D[l]\}$ is termed a *node-set*.

Furthermore, in the following, in each grid G_i for a keyword k_i, each cell is given an additional MBR that contains all objects stored in the grid-cell. This is equal to the *keyword-MBR* of bR*-tree. We use these keyword-MBRs for estimating distance between cells.

Next, Fig.2(c) is the *search space* for finding the optimal node-set under A, B, C, D using a naive nested loop search algorithm. Here we use δ^* to denote the current minimum diameter and it is initialized to ∞. Then this algorithm is written as follow:

Algorithm 1: Nested-Loop

Input: An m-sized node-set $curSet$, m grids $G_1, G_2, ..., G_m$.

1 If the $curSet$ is an internal node set and the distance between every two nodes $\in curSet$ is less than δ^*, we first put all child-node(s) (i.e. child grid-cells) of each internal node into a list, respectively; and then start to enumerate new child node-sets according to the nested loop of these child-node lists in the order of the ordinal integers of cells. Every time when a new child node-set is generated, we recursively invoke this Nested-Loop algorithm using the new node-set as $curSet$. When all breaches are tested, return δ^*.

2 If the $curSet$ is an m leaf node-set and the distance between every two nodes $\in curSet$ is less than δ^*, we exhaustively enumerate all the object-sets and find the minimum diameter of object-set, then update the value of δ^* to this diameter.

In the example of Fig.2(c), we start from the root node-set $\{A[0], B[0], C[0], D[0]\}$. Then the node-set $\{A[1], B[1], C[2], D[2]\}$ becomes the first child node-set generated, because A and B exist firstly in the 1st cells but C and D appear firstly in the 2nd cells. In case of a naive nested loop search over the given keyword ordering of A, B, C, D on these hierarchical grids, the search recursively proceeds in the depth-first order in the tree of the search space. δ^* will finally become the minimum diameter and be outputted as the result. Clearly, this naive approach is too expensive. Furthermore, as a disadvantage of using a grid, the grid structure is often weak in clustering correlated objects in one grid-cell (the over-splitting problem). Thus we must give a higher priority of search to a better set of grid-cells during the recursive search in the search space of Fig.2(c).

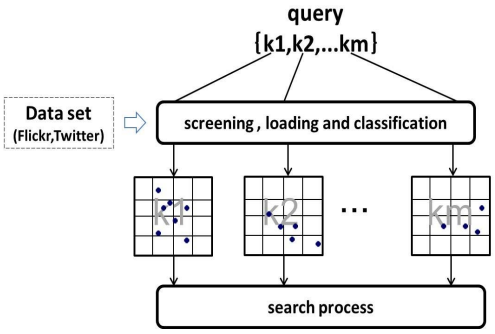
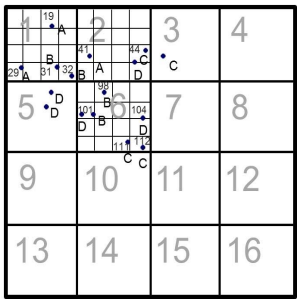
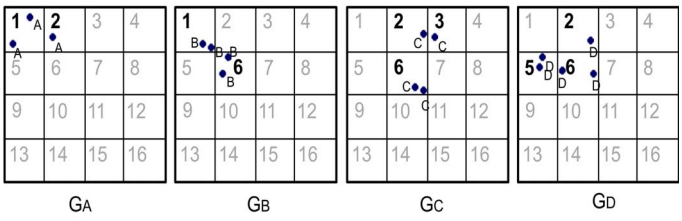


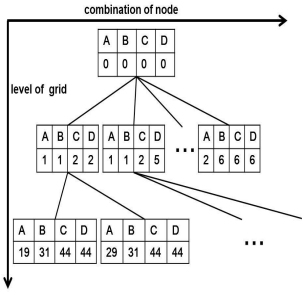
Fig. 1. On-demand creation of grid partitioning



(a) data distribution

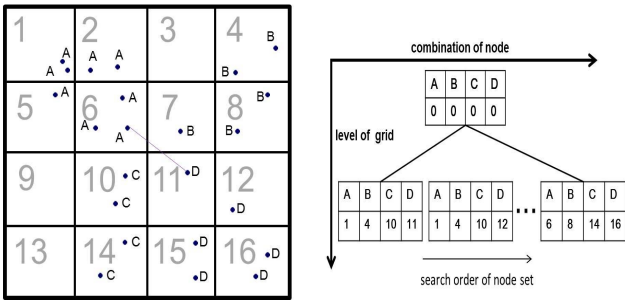


(b) m independent grids structure



(c) the search space of mCK query

Fig. 2. grid and search space of m-CK



(a) example of a data set (b) search space and order of node-sets

Fig. 3. search method of the mCK query

III. Diameter Candidate Check (DCC)

A. Basic idea of DCC

In the search space of Fig.2(c), two factors affect the efficiency of node-set enumeration.

One factor is the order of enumerating node-sets. An ideal way is to test a 'better' node-set with higher priority in the search space; a 'better' node-set is that having a smaller diameter. This leads to finding an object-set having a smaller diameter, which can prune out unnecessary node-sets. We should explore such a desirable node-set as early as possible.

As an example, let us consider a data distribution of Fig.3(a) and the corresponding search space of Fig.3(b). Actually, there are four independent grids for A, B, C, D in Fig.3(a), but we visualize these grids by one virtual grid of Fig.3(a).

Fig.3(b) shows which node-sets (by combining the cells of the first-level partitioning) are enumerated by the naive nested loop method of Algorithm 1.

In this example, the smallest diameter really exists in the node-set $\{A[6], B[7], C[10], D[11]\}$ at the first level of grid-partitioning. Clearly, we should choose this 'better' node-set firstly in Fig.3(b), from among all the node-sets of the first-level cells. Namely this 'better' node-set should be given a higher priority in the exploration of the search space. This priority-based search is not achieved either by the naive nested loop or the Zhang's method.

In case of using a nested-loop style search in the search space of Fig.3(b) or Fig.2(c), another expensive factor is the order of keywords to be tested. Let δ^* be the currently-found minimum diameter in the search process. Suppose we test a node-set $\{A[i], B[j], C[k], D[l]\}$ in the nested loop of keyword-ordering of A, B, C, D. Then, if the minimum distance between $C[k]$ and $D[l]$ is larger than δ^*, the search process will repeat expensive test of other combinations of useless node-sets. Thus a fixed global ordering of keywords must be avoided in the search process.

To overcome these factors, we propose a search strategy called Diameter Candidate Check (DCC). DCC is aimed to find a node-set having a smaller diameter as quickly as possible and reduces the search space.

The basic idea of DCC is as follows: The goal of mCK query is to find the smallest diameter, and the diameter is determined by two objects o_x, o_y. Thus, rather than enumerating the m-sized object-sets (or, node-sets) directly, we firstly enumerate all pairs made of two objects, $\langle o_x, o_y \rangle$, (or two child-nodes, $\langle n_x, n_y \rangle$,) from an inputted node-set, and sort the pairs in the ascending order of their possible diameter's lengths. Each pair is called a *diameter-candidate*. Next, in the sorted order of smaller (=better) diameter-candidates, we pick up a diameter-candidate and generate a new object-set (or, a child-level node-set) from the diameter-candidate, and recursively test the child node-sets, in a top-down manner, if necessary.

By this strategy, due to the ascending sort of diameter-candidates, a node-set having a smaller diameter is tested with a higher priority in the search space. Furthermore, the enumeration of all diameter-candidates is much less expensive than that of all m-sized object-sets.

It is a basic idea to use a pair of "closer" objects as a key to reduce the search space in various spatial keyword query problems. This idea is also used in the pairwise distance-owner finding in the MaxSum-Exact algorithm of Collective Spatial Keyword Query (CoSKQ,[9]), which is a problem to find the best disc of objects whose center is a given query-point and where the disc must also cover given m-keywords. Their algorithm depends on NN-queries from the query-point or some data-objects by using an IR-tree. In contrast, the mCK problem has no query-point, and our originality of DCC exists in the point that we explore and reduce the search space in a top-down manner without any query-point, by using DCC on dynamically-created hierarchical grid partitions.

B. Technical terms

To implement DCC, we prepare some technical terms.

Let $dist(o_1, o_2)$ be the distance between two objects o_1 and o_2. Let n_i (or, sometimes written as n_{w_i}) be a grid-cell associated with a keyword w_i. Then, $Maxdist(n_{w_1}, n_{w_2})$ is the maximum distance between two rectangles of n_{w_1} and n_{w_2}, which are MBRs in the grid-cells of w_1 and w_2. $Mindist(n_{w_1}, n_{w_2})$ is the minimum distance between the same MBRs of n_{w_1} and n_{w_2}.

When m-keywords $\{w_1, w_2, ..., w_m\}$ are given as a query, we define $MaxMaxdist$ and $MaxMindist$ of a node-set $N = \{n_{w_1}, n_{w_2}, ..., n_{w_m}\}$, as follows:

Definition 1: For a node-set $N = \{n_{w_1}, n_{w_2}, ..., n_{w_m}\}$ under m-keywords,

- $MaxMaxdist$ of N is defined as: $MaxMaxdist(N) = max_{n_{w_i}, n_{w_j} \in N} Maxdist(n_{w_i}, n_{w_j})$.
- the pair of nodes $\langle n_{w_i}, n_{w_j} \rangle$ is said to be the *diameter pair* of N if $dist(n_{w_i}, n_{w_j}) = MaxMaxDist(N)$.
- $MaxMindist$ of N is defined by $MaxMindist(N) = max_{n_{w_i}, n_{w_j} \in N} Mindist(n_{w_i}, n_{w_j})$.

$MaxMaxdist$ is an upper bound of all possible diameter's lengths that are derived from the node-set N. $MaxMindist(N)$ is a lower bound of diameter's lengths which can be found from N. Fig.4 shows their examples of $N = \{n_{w_1}, n_{w_2}, n_{w_3}\}$. The pair $\langle n_{w_1}, n_{w_3} \rangle$ is the diameter pair of N.

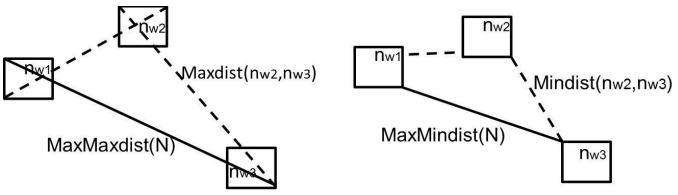


Fig. 4. MaxMaxdist and MaxMindist

C. DCC in a nested loop method

We here describe the implementation of DCC strategy in a nested loop search algorithm. This algorithm is called *DCC-NL*. DCC-NL uses a nested loop method over all keywords in order to generate and test a new node-set from a diameter-candidate.

DCC-NL has three steps in the process, as shown in Fig.5. In the following, we explain the case of four keywords A, B, C, D of Fig.2(b) as an example. It is given a node-set N_I as the input, and finally returns the minimum diameter. It starts from the root node-set $N_0 = \{A[0], B[0], C[0], D[0]\}$, where all nodes are the level-0 grid-cells:

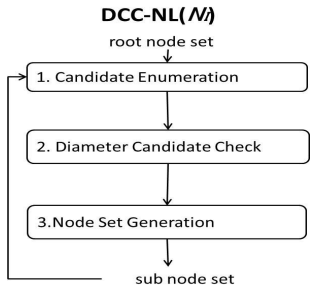


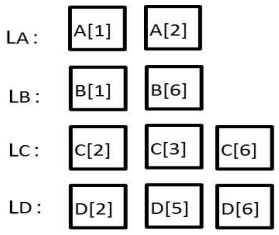
Fig. 5. workflows of DCC three steps

[step1 : Candidate Enumeration]
We assume that in the inputted node-set $N_I = \{n_{w_1}, n_{w_2}, ..., n_{w_m}\}$, all n_{w_i}'s are non-leaf nodes. (The other cases are described later.) Then, for each keyword w_i, we first find all child-nodes (i.e., = child grid-cells) which satisfy w_i in the node $n_{w_i} \in N_I$, and put these child-nodes into a corresponding list L_{w_i}. Next, from every two different lists L_{w_i} and L_{w_j}, we generate all pairs of child-nodes, by picking one from each list. These pairs are called the *diameter candidates* (denoted as DC, in short).

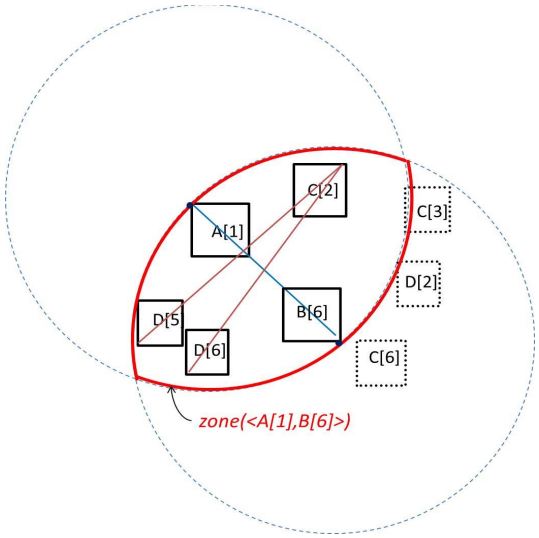
As an example, in Fig.6(a), if we use the root node-set as the input, we can get four child-node lists L_A, L_B, L_C, L_D corresponding to keyword A, B, C, D. Then four DCs, $\langle A[1], B[1]\rangle, \langle A[1], B[6]\rangle, \langle A[2], B[1]\rangle, \langle A[2], B[6]\rangle$, are generated from the list L_A and L_B. Also six DCs, $\langle B[1], C[2]\rangle, ..., \langle B[6], C[6]\rangle$, are done from L_B and L_C. Thus we finally generate 37 pairs in total.

Thereafter we sort these DCs by the ascending order of $Maxdist(DC)$. Note that if the size of each list L_{w_i} is s, the number of DCs is $({}_mC_2 \times s^2)$, which is much less than the amount of possible node-sets (= s^m).

[step2 : Diameter Candidate Check]



(a) child-node lists



(b) example of DC and Check

Fig. 6. Creating a node-set from a given DC

We pick up a $DC = \langle n_i, n_j\rangle$, from the top of the sorted list of DC's, and check if this DC cannot become a diameter pair of any child node-set of N_I. If so, we will not need to consider this DC.

There are three points to be checked: they are checked in the order of (2-1) to (2-3). If any one of the points holds, the DC is removed and we return to checking the next DC from the sorted list.

(2-1) If the DC $\langle n_i, n_j\rangle$ is such that $Mindist(n_i, n_j) \geq \delta^*$ (δ^* is the current minimum diameter), no node-set which is generated from this DC can give an object-set having a diameter $< \delta^*$. Thus we remove this DC and go to the next DC.

(note: This reduces the overhead factor of testing node-sets by the fixed global order of keywords, as described in Section III.A.)

(2-2) Next, we try to generate a node-set N_D from the DC $\langle n_i, n_j\rangle$. (N_D is a child node-set of the inputted node-set N_I in the search space.) If N_D is successfully created, the DC must be included within N_D and be the *diameter pair* of N_D. This requires that every node $n^* \in N_D$ except the two nodes of the DC must satisfy both:
$Maxdist(n^*, n_i) < Maxdist(n_i, n_j)$ and
$Maxdist(n^*, n_j) < Maxdist(n_i, n_j)$.

The above condition can be said as follows: every $n^* \in N_D$ must be a node included in the shuttle-shaped (red) zone drawn in Fig.6(b), where this zone represents the above condition.

This zone is uniquely determined by a DC, thus being denoted by $zone(DC)$. In order to compose N_D from a given DC, we need not consider any nodes which are outside the $zone(DC)$.

As an example, in Fig.6(b), when the current DC is $\langle A[1], B[6]\rangle$, then the nodes $C[3], C[6]$ in L_C and $D[2]$ in L_D are outside the $zone(\langle A[1], B[6]\rangle)$. Thus we ignore these nodes for generating N_D from the DC.

As a result, the step (2-2) is as follows: we check if there exists any keyword w_i such that all the node associated with w_i are outside $zone(DC)$. If such w_i exists, we remove the DC and go to the next DC.

(2-3) Next, let the DC be $\langle n_i, n_j\rangle$ and consider the $zone(\langle n_i, n_j\rangle)$. For every two keywords $w_x, w_y \in Q - \{w_i, w_j\}$, we check the following constraint:
there must exist some two nodes n_x, n_y where $n_x \in L_{w_x}$, $n_y \in L_{w_y}$ such that both n_x, n_y are included in $zone(DC)$ and $Maxdist(n_x, n_y) \leq Maxdist(n_i, n_j)$. If this constraint fails for some w_x, w_y, then the DC $\langle n_i, n_j\rangle$ cannot become a diameter pair any more. Thus, the DC $\langle n_i, n_j\rangle$ is not necessary to be considered, and the next DC is checked.

As an example, in Fig.6(b), both $Maxdist(C[2], D[5])$ and $Maxdist(C[2], D[6])$ are larger than $Maxdist(A[1], B[6])$. Thus the DC $\langle A[1], B[6]\rangle$ is removed, and we go to the next DC.

[step3 : Node-Set generation]
After the step 2, we generate a new child node-set N_D from the current DC. To do so, we take a combination of the child-nodes in $zone(DC)$ by using a nested loop method over all L_{w_i}'s of $(m-2)$ keywords (except the keywords of the DC). During this process, we must skip over such a node-set that some two nodes in the combination have a larger $Maxdist$ than $Maxdist(DC)$. Each time when we get an output of the combination over the $(m-2)$ keywords, the output is merged with the DC, and is used as the N_D; i.e., we recursively invoke DCC-NL(N_D).

As the last comment, if all the nodes of the input node-set are leaf-nodes in the step1, each leaf-node is decomposed into individual objects and the above procedure is used. If some node in the input is a leaf and some is not, then the decomposition of the leaf-nodes is skipped over until all nodes in N become leaf-nodes.

According to the above, we summarize the DCC-NL algorithm as Algorithm 2.

IV. Further pruning rules using MaxMindist

We can consider further pruning rules, which filter out unnecessary node-sets. Firstly, the following lemma holds:

Lemma 1 : Given a node-set $N = \{n_{w_1}, n_{w_2}, ..., n_{w_m}\}$, assume that some node $n_{w_i} \in N$ satisfies that $Maxdist(n_{w_i}, n_{w_j}) < MaxMindist(N)$ for all $n_{w_j} \in N$ $(w_j \neq w_i)$. Then n_{w_i} is not necessary to compute the diameter of N. (Thus n_{w_i} can be virtually removed from every descendent node-set of N.)

Algorithm 2: DCC-NL

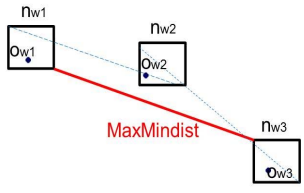
Input: An m-sized node-set $curSet$, m grids $G_1, G_2, ..., G_m$.

1 If the $curSet$ is an internal node set and the distance between every two nodes $\in curSet$ is less than δ^*, we enumerate all the node-pairs as DCs and sort them. For each DC $\langle n_i, n_j\rangle$ in these DCs do:
- 1.1 Test the step2 for the DC $\langle n_i, n_j\rangle$.
- 1.2 If the step2 judges the DC $\langle n_i, n_j\rangle$ must be skipped over, we test the next DC; otherwise, go to 1.3.
- 1.3 Generate all $(m-2)$-sized node-sets among nodes in $zone(\langle n_i, n_j\rangle)$. Every time we get an $(m-2)$-sized node-set N_{m-2} where $Maxdist$ between every two nodes $\in N_{m-2}$ is less than $Maxdist(n_i, n_j)$, we merge N_{m-2} and DC $\langle n_i, n_j\rangle$ into an m-sized node-set and use it as $curSet$, and we recursively invoke DCC-NL algorithm.

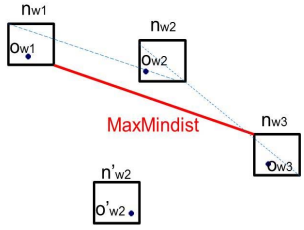
2 If the $curSet$ is an m-sized leaf node-set and the distance between every two nodes $\in curSet$ is less than δ^*, we enumerate all the object-pairs as DCs and sort them. For each DC $\langle o_i, o_j\rangle$ in these DCs :
- 2.1 Check the step2 by setting $\langle o_i, o_j\rangle$ as a DC.
- 2.2 If DC $\langle o_i, o_j\rangle$ is judged to be skipped over, we go to the next DC; otherwise, go to 2.3.
- 2.3 Generate $(m-2)$-sized object-sets. Once we get a $(m-2)$-sized object-set O_{m-2} that the distance between every two objects $\in O_{m-2}$ is less than $Dist(o_i, o_j)$, we set δ^* to $Dist(o_i, o_j)$, and return.

In Fig.7(a), for the node-set $N = \{n_{w_1}, n_{w_2}, n_{w_3}\}$, $MaxMindist(N) = Mindist(n_{w_1}, n_{w_3})$, and both $Maxdist(n_{w_2}, n_{w_1})$ and $Maxdist(n_{w_2}, n_{w_3})$ are less than $MaxMindist(N)$. Therefore, for any object-set $\{o_{w_1}, o_{w_2}, o_{w_3}\}$ generated from N, $dist(o_{w_1}, o_{w_3})$ must be the largest. Thus all o_{w_2}'s in n_{w_2} are not necessary to compute the diameter.

By the lemma 1, we consider two pruning rules:



(a) example of Lemma 1



(b) example of pruning rule 1

Fig. 7. Pruning rules

Pruning Rule 1 Given a node-set $N = \{n_{w_1}, n_{w_2}, ..., n_{w_m}\}$, assume that nodes in N can be classified into two groups $\{n_{w_a}, ..., n_{w_b}, |n_{w_c}, ..., n_{w_d}\}$, where $n_{w_c}, ..., n_{w_d}$ are the nodes which can be removed by lemma 1 and $n_{w_a}, ..., n_{w_b}$ are not removed. Then we need not test any other node-set N' where N' contains $\{n_{w_a}, ..., n_{w_b}\}$.

In Fig.7(b) where $N = \{n_{w_1}, n_{w_2}, n_{w_3}\}$, we can remove n_{w_2} from N by Lemma 1. Then, we can skip over the test of another node set $N' = \{n_{w_1}, n'_{w_2}, n_{w_3}\}$. It is because the diameter created from N' is never smaller than that computed from N.

Pruning Rule 2 : If a node-set N is generated from a DC $\langle n_{w_i}, n_{w_j} \rangle$ and all the nodes in N except n_{w_i} and n_{w_j} can be removed by Lemma 1, then any other node set which is generated from this DC can be pruned out.

Pruning Rule 2 is a special case of Pruning Rule 1. Once a node-set satisfies Pruning Rule 2, we can use only the two nodes of DC for computing the diameter. Furthermore the remaining other node-sets from this DC can be skipped over.

V. Experimental evaluation

A. Experimental set-up

Here we evaluate our algorithm over synthetic datasets and real datasets.

The synthetic datasets consist of two-dimensional data points where each point has only one keyword. We generated 1000 data points for each keyword in advance, up to 100 keywords. The x- and y- values of a data-point are in $[0, R]$, taken from a square of the side-length R. As for these synthetic datasets, we prepared a separate data-file for each keyword. When a query of m-keywords is given, we build m grids from the necessary files, as explained in Fig.1. We use three types of data-distribution:

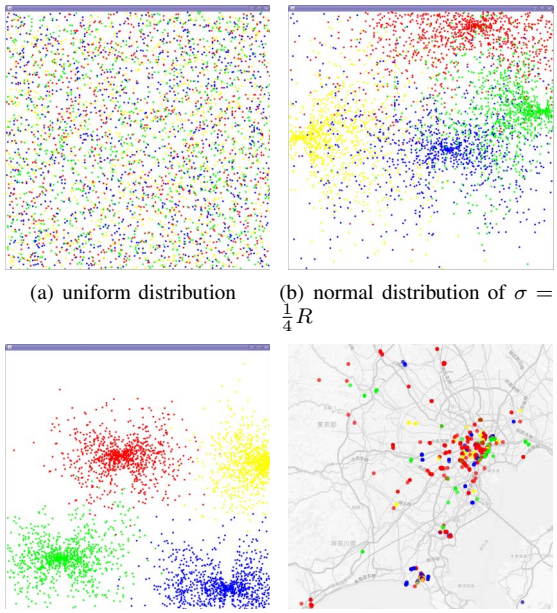
1) uniform distribution: All coordinates of data points are randomly generated (Fig.8(a)).

2) normal distribution with $\sigma = \frac{1}{4}R$: For each keyword, we randomly generate one point as the reference point, and then all the data points associated with this keyword are generated so that the distance to the reference point follows a normal distribution (Fig.8(b)). We set the standard deviation $\sigma = \frac{1}{4}R$.

3) normal distribution with $\sigma = \frac{1}{8}R$: The same as 2) except that $\sigma = \frac{1}{8}R$ (Fig.8(c)). This is the case of much higher skew than that of $\sigma = \frac{1}{4}R$.

We also employ a real dataset which collects 46,303 photo records from Flickr in Tokyo area. Each photo record contains a geographic information, which is used as the x- and y- values of the data-point. And each record is associated with 1 to 73 tags that can be viewed as keywords of data-point. As an example we choose 4 keywords $sakura$(red), $river$(green), $temple$(blue), $shrine$(yellow), and the distribution is shown in Fig.8(d). As for the Flickr data, we stored them in MongoDB, and we prepared keyword indices at first. When a query is given, we load necessary data as shown in Fig.1.

As a grid-partitioning of Section II.B, we defined that each cell is divided when the number of data points is greater than



(a) uniform distribution (b) normal distribution of $\sigma = \frac{1}{4}R$

(c) normal distribution of $\sigma = \frac{1}{8}R$ (d) Flickr photo data distribution

Fig. 8. data points distribution

100. The fan-out is set to 100 (= a square of 10 $\times$ 10). Note that the grid partitioning is created dynamically when a query is given.

We also implemented the Zhang's Apriori-based algorithm for the comparison. We execute all algorithms under our grid-partitioning setting of Section II.B. We execute the Zhang's method by making one grid structure of Fig.2(a). This is fair because our grid still keeps keyword-MBRs in each grid-cell. We implemented all the algorithm in Java with version 1.7 on a machine with an Intel(R) Xeon(R) CPU of 2.6GHz and 12GB of RAM ,running the linux vine 5.2. The performance measure is the average response time (ART). The ART includes all time of data access, grid creation and search execution. For each m, we randomly chose m keywords 10 times and takes ART.

The algorithms we test are:

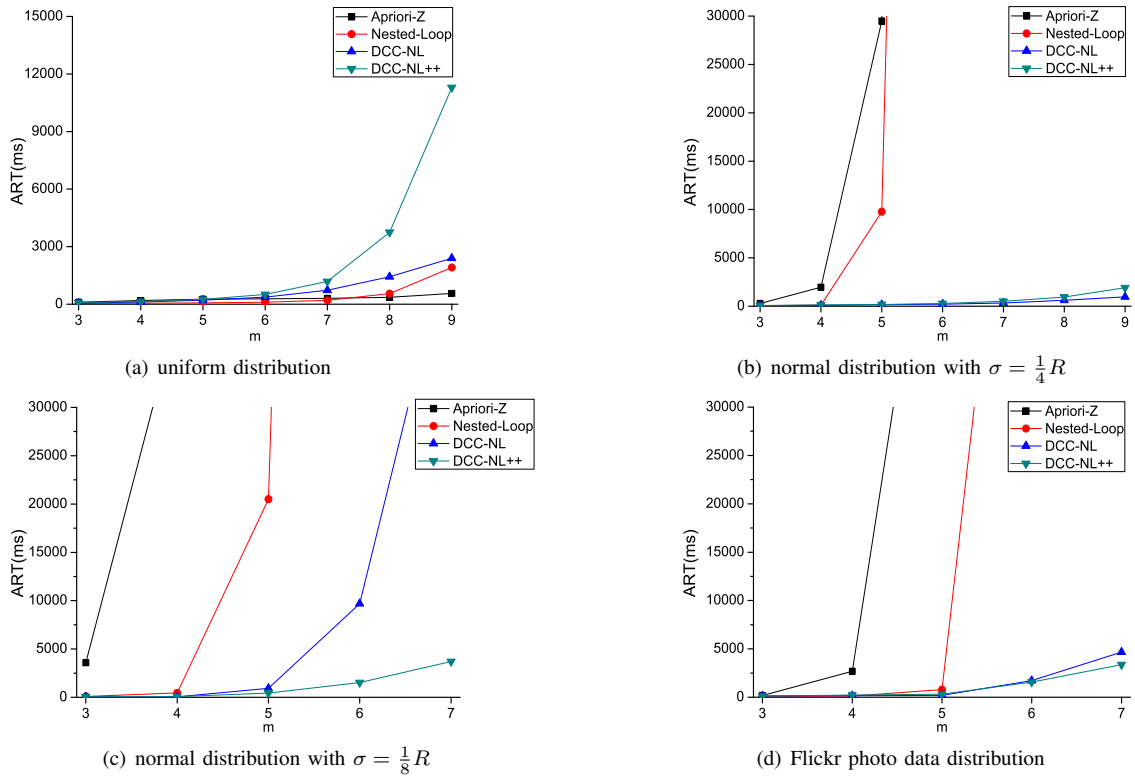
- Apriori-Z: Zhang's Apriori-based algorithm
- Nested-Loop: the naive nested loop algorithm
- DCC-NL: the DCC algorithm Algorithm2
- DCC-NL++: DCC-NL with the two pruning rules of Section IV.

B. Evaluation of Synthetic Datasets

First we tested synthetic datasets in Fig.9(a)-(c).

Fig.9(a) is ART vs. the number of keywords m in the uniform distribution.

In Fig.9(a), we can see Apriori-Z has the best performance. It keeps lower ARTs even when m increases up to 9. This is because the uniform distribution gathers necessary objects of all keywords into one smaller grid-cell. This makes Apriori-Z can firstly find a much smaller diameter in an itemset of MBRs of length-1 or -2, and it helps Apriori-Z cut down almost all node-sets at an early stage of the search.



(a) uniform distribution

(b) normal distribution with $\sigma = \frac{1}{4}R$

(c) normal distribution with $\sigma = \frac{1}{8}R$

(d) Flickr photo data distribution

Fig. 9. performance of mCK

The performances of the other algorithms are degraded when $m \geq 8$. It is because node-set enumeration using nested loop is inherently exponential to m. Due to the uniform distribution, we can easily get a relative small diameter without enumerate all node-pairs; hence Nested-Loop outperforms DCC-NL. DCC-NL++ is the worst at $m \geq 8$ because the node-sets with larger $MaxMindist$ are pruned, but we must spend CPU time to check all subsets of each node-set.

Next, Fig.9(b) is the case of the normal distribution of $\sigma = \frac{1}{4}R$. In this case, we observe that the data points with each keyword is lightly skewed, but there still exist some object-sets with small diameters, although they are not necessarily gathered in one cell. Apriori-Z and the Nested-Loop deteriorated sharply. Because all keywords may not be gathered in one cell, it is difficult to guarantee that these methods find a much smaller diameter as an initial value. In contrast, DCC-NL and DCC-NL++ worked well than the other algorithms. It means that the DCC strategy works good. DCC-NL++ is a little worse than DCC-NL, but it is because $MaxMindist$ of a node-set is not so large as to reach the conditions of the pruning rules, thereby consuming more CPU times.

Lastly, Fig.9(c) is the case of the normal distribution of $\sigma = \frac{1}{8}R$. Here the data points associated with each keyword are highly skewed. Thus any object-sets are not expected to have small diameters. As a result, Apriori-Z and Nested-Loop drop rapidly at a lower m. Even DCC-NL cannot keep good at $m = 6$. In contrast, DCC-NL++ keeps a relative slow drop in ART, because it can prune out unnecessary node-sets not only depending on a current threshold but also using the Pruning Rules 1 and 2.

In summary, DCC-NL++ is the best when $m \leq 7$ in case of the skewed datasets. Even in the uniform dataset, DCC-NL++ works well if $m \leq 7$, but DCC-NL++ spends useless CPU time at $m \geq 8$.

C. Evaluation of Flickr Datasets

Next Fig.9(d) shows the performance comparison of Flickr data. In this test, we chose m keywords randomly in the Flickr data, but we chose each keyword whose frequency $\leq 2.5\%$ of all records. We can see the average performance of Apriori-Z decreases rapidly as m increases. In our assumption of grid partitioning, due to the over-splitting problem, it is difficult to guarantee all the query keywords into a small leaf-cell. It causes the situation that a less-than-ideal initial value of diameter makes the heavy enumeration of node-sets. Furthermore an unbalanced depth of the grid also makes the enumerating cost even worse. Under this situation, the performance of Apriori-Z becomes extremely unstable, which affected the average performance. For example, we observed that when $m = 5$, in some good query which can get a small initial value, the response time of Apriori-Z only needed 0.2 seconds. However in another query, we observed that its response time dropped to 535 seconds. This unstability becomes apparent as m increases.

In contrast, in Fig.9(d) DCC-NL and DCC-NL++ worked well at $m \leq 7$. That means if there exists a small diameter, even though necessary data are not clustered in one small grid-cell, DCC strategy can find them at an earlier stage of search.

VI. SUMMARY

In this paper, we discussed a new algorithm for the mCK query, which is used to find the optimal object-set that satisfies the m keywords over spatial web objects. We assumed that there is no prepared data-partitioning for the target datasets at the data-providing servers. Hence we assumed to create grid partitioning dynamically, on the loaded necessary data when each query is given. We proposed an effective search strategy named *Diameter Candidate Check* (DCC) so as to work well under this assumption. DCC is aimed to find a better set of grid-cells at an earlier stage of search, and can reduce search space even if it is implemented in a nested loop search method, named DCC-NL. We also proposed two pruning rules adding to DCC-NL, as the method DCC-NL++, for the case of highly-skewed data.

We compared DCC algorithms with Zhang's Apriori method under our assumption. The performance under both synthetic and real datasets demonstrated that the preceding Apriori-based algorithm of Zhang is highly efficient on the uniformly distribution case, but can get worst in the other skewed data cases. Instead our DCC-based algorithm DCC-NL and DCC-NL++ keep stable and good performance in skewed data at $m \leq 7$. DCC-NL++ is good in skewed cases, but its CPU overhead is disadvantageous in a uniform dataset.

Looking at these results, DCC strategy can provide acceptable efficiency in different datasets at $m \leq 7$ even in a highly-skewed case. However DCC-NL is inherently weak when one data-point has many keywords of the mCK query, because the nested-loop search is exponential as the number of keywords increases. The idea of DCC can be applied to the original Apriori-based method of [1], and such approaches are currently under our progress.

REFERENCES

[1] D.X.Zhang, Y.M.Chee, Anirban Mondal, Anthony K.H.Tung, M. Kitsuregawa, "Keyword Search in Spatial Databases: Towards Searching by Document." IEEE ICDE, pp.688-699, 2009.

[2] Christian S. Jensen. "Data Management on Spatial Web",keynote, Proceedings of the VLDB Endowment, Vol. 5, No.12, 2012.

[3] R. Hariharan, B. Hore, C. Li, and S. Mehrotra. "Processing spatial-keyword queries in geographic information retrieval systems." SSDBM, pp. 16-25, 2007.

[4] X.Cao, G. Cong, Christian S. Jensen , and B.C.Ooi. "Collective spatial keyword querying." SIGMOD, pp. 373-384, 2011.

[5] Z. S. Li , B. H. Zhen , W. C. Lee, D. L. Lee , X. F. Wang " IR-Tree: An Efficient Index for Geographic Document Search" IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 23, NO. 4, pp. 585-599, APRIL 2011.

[6] D. Papadias, N. Mamoulis, and Y. Theodoridis. "Processing and optimization of multiway spatial joins using R-trees." Proc. PODS, pp. 44-55, 1999.

[7] D.X.Zhang, B.C.Ooi, Anthony K.H.Tung, "Locating Mapped Resources in Web2.0." IEEE ICDE, pp. 521-532, 2010.

[8] A.Corral, Y.Manolopoulos, Y.Theodoridis, M.Vassilakopoulos, "Multi-way distance join queries in spatial databases[J]." GeoInformatica, pp. 373-402, 2004, 8(4)

[9] C.Long, R.C.-W.Wong, K.Wang, A.W.-C.Fu, "Collective spatial keyword queries:a distance owner-driven approach." SIGMOD , pp. 689-700, 2013.