A Study on Top-down Search Algorithms for m-Closest Keywords Queries Problem over Spatial Web

Yuan Qiu

THE UNIVERSITY OF ELECTRO-COMMUNICATIONS

March 2017

A Study on Top-down Search Algorithms for m-Closest Keywords Queries Problem over Spatial Web

Yuan Qiu

Submitted to the Graduate School of Information Systems in partial fulfillment of the requirements for the degree of

Doctor of Engineering

THE UNIVERSITY OF ELECTRO-COMMUNICATIONS

March 2017

A Study on Top-down Search Algorithms for m-Closest Keywords Queries Problem over Spatial Web

APPROVED BY SUPERVISORY COMMITTEE:

CHAIRPERSON:PROF.Tadashi OHMORI

MEMBER:PROF.Yasuhiro MINAMI

MEMBER:PROF.Hiroyoshi MORITA

MEMBER:AP.Hisashi KOGA

MEMBER:AP.Yasuyuki TAHARA

MEMBER:AP.Takahiko SHINTANI

Abstract

This thesis addresses the problem of *m-closest keywords* queries (mCK queries) over spatial web objects that contain descriptive texts and spatial information. 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 in their texts. The mCK query can be widely used in various applications to find the place of user's interest.

Generally, top-down search techniques using tree-style data structures are appropriate for finding optimal results of queries over spatial datasets. Thus in order to solve the mCK query problem, a previous study of NUS group assumed a specialized R*-tree (called bR*-tree) to store all records and proposed a top-down approach which uses an Apriori-based node-set enumeration in top-down process. However this assumption of prepared bR*-tree is not applicable to practical spatial web datasets, and the pruning ability of Apriori-based enumeration is highly dependent on the data distribution.

In this thesis, 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), which can find a smaller node-set at an earlier stage of search so that it can reduce search space more efficiently. According to DCC search strategy, we firstly employ an implementation of DCC strategy in a nested loop search algorithm (called DCC-NL). Next, we improve the DCC-NL in a recursive way (called RDCC). RDCC can afford a more reasonable priority order of node-set enumeration. We also uses a tight lower bound to improve pruning ability in RDCC.

RDCC performs well in a wide variey of data distributions, but it has still deficiency when one data-point has many query keywords and numerous node-sets are generated. Hence in order to avoid the generation of node-sets which is an unstable factor of search efficiency, we propose another different top-down search approach called Pairwise Expansion. Finally, we discuss some optimization techniques to enhance Pairwise Expansion approach. We first discuss the index structure in the Pairwise Expansion approach, and try to use an on-the-fly

kd-tree to reduce building cost in the query process. Also a new lower bound and an upper bound are employed for more powerful pruning in Pairwise Expansion.

We evaluate these approaches by using both real datasets and synthetic datasets for different data distributions, including 1.6 million of Flickr photo data. The result shows that DCC strategy can provide more stable search performance than the Apriori-based approach. And the Pairwise Expansion approach enhanced with lower/upper bounds, has more advantages over those algorithms having node-set generation, and is applicable for real spatial web data.

Contents

Chapter 1

Introduction

1.1 Background

Nowadays massive web data are attached with geographic location information such as Twitter [14] and Flickr [15]. This type of web data associated with both textual and geographic attributes is called a *spatial web object* (or a *geo-textual web object*). For example, Figure 1.1 shows a photo data from website of Flickr [15], which is an online photo-sharing service. In this figure, we can see some other information beside the photo itself. There is a passage of descriptive text about the photo in the box area. This can be regarded as the textual attributes of this data. And it also contains a photographed location (displayed on the map) in the ellipse area, which can be regarded as the geographic attributes of it. Thus this photo data is a typical spatial web object.

To these spatial web objects, users are not only interested in the contents of them, but also increasingly consider their spatial aspects. Therefore, retrieval of geographic information by using spatial web objects (denoted as objects, in short) has been studied extensively in recent years [1, 3, 18, 19, 20, 25, 22]. The common way of retrieval is called *Spatial Keyword Queries* [23]. Spatial keyword queries usually allow users to enter several keywords, and then return object(s) that best match these keywords from both textual and spatial perspectives.

As a simple example of spatial keyword queries, *Google Map* provides a service that users can find the spatial web objects by typing in some keywords. Figure 1.2 shows a search instance when we input 'coffee' as a query keyword, then the objects which cover the user-

given keywords are returned and shown on the map based on their geographic information.

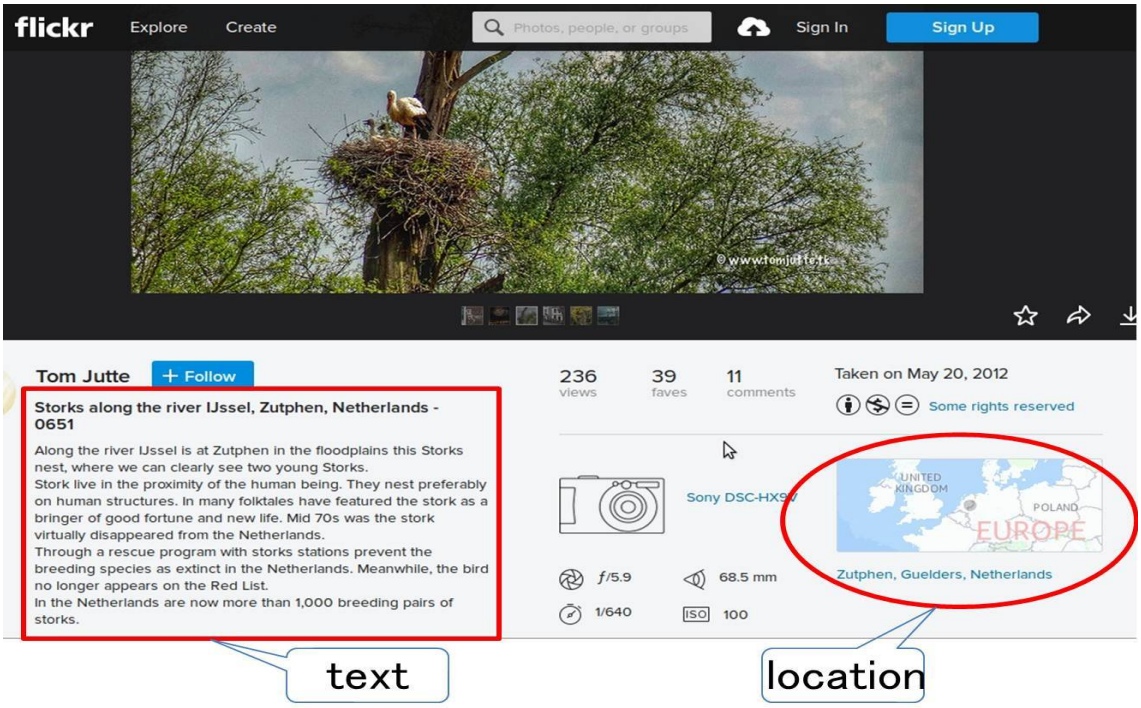


Figure 1.1: An example of Flickr data

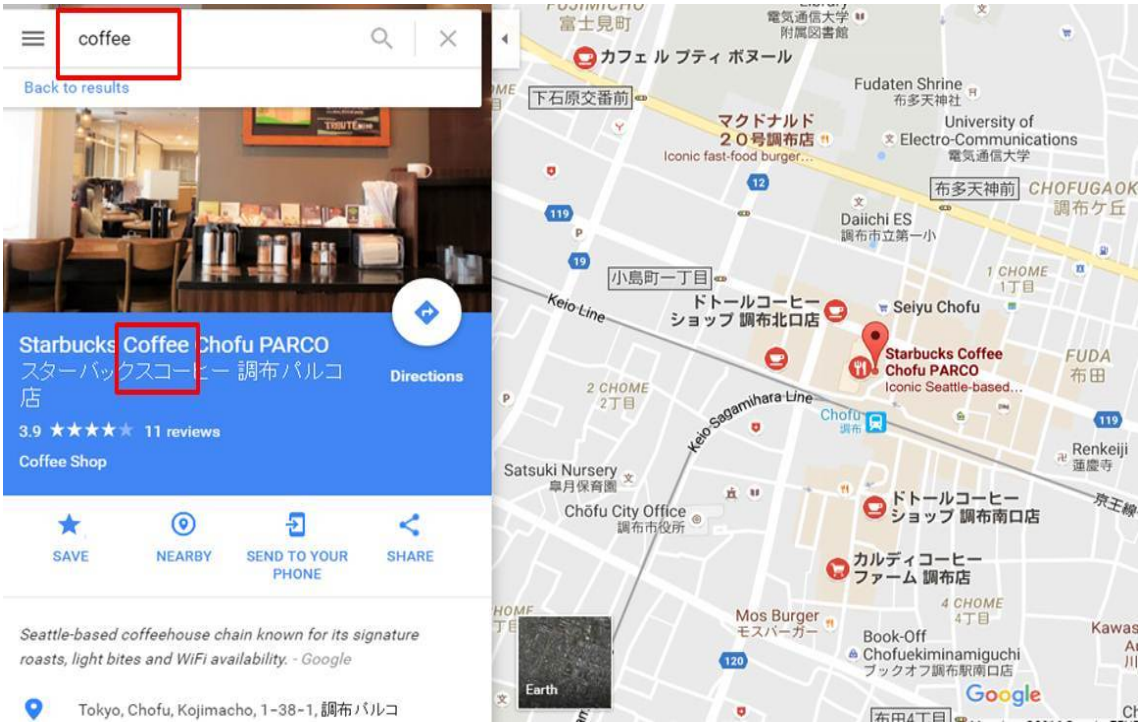


Figure 1.2: Spatial keyword search on Google map

However, in some cases, there may be no single object that can cover all the user-given keywords. In such circumstances, finding a group of objects to collectively match all query keywords had been considered in some researches of spatial keyword queries [1, 3, 7, 36, 38]. In this thesis, we focus on a typical kind of such spatial keyword queries problem, called *m-Closest Keywords*(mCK) Queries, which is proposed by Zhang et al [1] in 2009.

As an introductory example, consider a database D of spatial web objects, and suppose that a user gives m keywords as a query Q. Then, an mCK query under Q, is a query to find the 'optimal' group of objects O_{opt} from D, in the meaning that:

(i) each keyword in Q is satisfied by textual attributes of some object in O_{opt}, and

(ii)the objects in O_{opt} are, among all groups of objects that satisfy the condition (i), positioned in the spatially-closest manner.

Intuitively, the 'optimal' group of objects above is regarded as the best and smallest spatial area that satisfy all keywords of Q.

Next section will describe the mCK query in details.

1.2 Description about mCK queries problem

In the description of [1], given m keywords , an mCK query aims at finding a group of the spatially-closest objects that match these m keywords. The group of objects is called an 'object-set'.

Intuitively, the optimal object-set O_{opt} of an mCK query must satisfy two conditions about both textuality and spatiality as follow:

Condition 1: All the user-given keywords must be contained in the text collection of O_{opt}.

Condition 2: Let S be the set of all the object-sets that satisfy the Condition 1. Then the objects in O_{opt} must be placed together in the spatially closest positions, among all cases of S.

Condition 2 must be rewritten formally. That is, in order to formally measure the closeness of objects in an object-set O, Zhang et al proposed a *diameter* of O, which is defined as the

maximum distance between any two objects in O. If the diameter of O is small, then all the objects in O are positioned more closely. Therefore, the Condition 2 is formalized by saying that the diameter of O_{opt} must be the smallest among those of all object-sets in S.

We need to explain the above description by showing a typical example. As a specific example of mCK query, suppose that there are 15 objects o_1 to o_{15} in a dataset D. Figure 1.3 shows the spatial distributions of these objects on Google map. We can see these objects are located near the Chofu station, and each object is associated with one of three keywords : "coffee" (6 objects), "shopping" (5 objects) or "convenience store" (4 objects). If an user wants to find a place which contains these three keywords, then the user can use an mCK query for D by issuing $Q = \{coffee, shopping, convenience\ store\}$ as query keywords. There are $6 \times 5 \times 4 = 120$ combinational object-sets that can satisfy Q. And we can see the object-set $\{o_4, o_7, o_8\}$ has the smallest diameter among all the 120 object-sets. Thus $\{o_4, o_7, o_8\}$ will be returned to the user. The diameter of the object-set $\{o_4, o_7, o_8\}$ is the distance between o_4 and o_8.

Typically, mCK queries can be used in various location-based services like recommendation of tourist attractions or real estate to match user's interests. For instance, mCK query can be applied to the photo data of Flickr. By means of mCK queries we can find a set of photos about all keywords we are interested in, and the locations of these photos will be gathered in a point or an small area, which can be used as recommended information for tourist or others.

1.3 Objective of the thesis

To find the optimal solution of an mCK query, we need to compare all the object-sets. And each object-set is a combination of some objects from a spatial web dataset. Thus suppose there are N objects in the dataset. Then, the number of object-sets is up to $O(N^m)$. Hence the cost of comparison is expensive. Generally, a top-down search technique using tree-style data structures is appropriate for finding optimal results of queries over spatial datasets, and has been used in various spatial query problems such as nearest neighbor search [24] and multiway spatial join [25, 26], etc. Thus, for the mCK query problem, the study of Zhang et

Figure 1.3: An example of mCK query

al [1] assumed a specialized R*-tree (called bR*-tree) to store all records and proposed a top-down approach which uses an Apriori-based enumeration of node sets in top-down process. However there are still some questions to be figured out, we think.

Therefore, our main objective of this thesis is to improve the previous work of top-down search approach for mCK queries with respect to the design of data index and search strategy. We analyze the factors which restrict the search efficiency by clarifying our unique questions to existing methods, and propose four new, more efficient, top-down search methods to solve this problem.

1.4 Organization

The organization of thesis is the following.

Chapter 2 reviews related works of mCK queries problem. This chapter first introduces spatial keyword queries over geo-textual web. Then details of mCK queries problem are described, including problem setting and preceding techniques. After that we describe our motivation of this thesis.

Chapter 3 proposes a new search strategy called Diameter Candidate Check(DCC) and an on-the-fly data structure for those questions in Chapter 2. This chapter first outlines basic ideas of DCC strategy. Then a nested loop method using DCC strategy, which is called DCC-NL, is proposed.

Chapter 4 enhances the DCC strategy in a recursive way, which is called RDCC. Moreover, this chapter also introduces an optimized technique incorporated with RDCC method , which uses a new tighter lower bound to improve pruning ability,.

Chapter 5 proposes a new top-down search way Pairwise Expansion(PE). This chapter first discusses limitations of the exploration policy in the preceding chapters, which needs to generate node-sets in top-down process. Then detailed description of PE without node-set generation is given.

Chapter 6 improves PE by using a convex-hull based lower/upper bounds. And a different data structure on-the-fly kd-tree will be discussed in this chapter.

Chapter 7 concludes this thesis.

Chapter 2

Related Works and Problem Setting

2.1 Spatial web data

Spatial web data associated with both geographic location and text information can be found everywhere in our daily life. For example, a lot of online social media (or social networking services) such as Twitter and Facebook enable users to post messages with their publishing locations. Figure 2.1 shows a 'tweet' data from website of Twitter [14]. In this 'tweet' data, there is a short message with some characters as text information. In addition to this, it also contains a location information where the message is published. Besides, a 'post' data of Facebook, which is shown in Figure 2.2, also allow users to attach a geotag to the photos as well as some textual tags. In addition, the spatial data for business or PoIs (points of interests) with a name or textual description such as hotels or restaurants also increasingly appeared in web. According to official home page of Google Place API [29], Google declared that it has more than 100 million specific places with detailed information in Google's 'Places API Web Service' [29]. Therefore the spatial web data becomes a very important part in the web.

As a formal description, according to [9], a **spatial web data** is described as $o : \langle \psi, \lambda \rangle$, where $o.\psi$ is the text property of o in the form of a text message or tags, and $o.\lambda$ is the location of o by using geographic coordinates. o is denoted by an *object* from here on.

At the same time, due to these increased sources of spatial data, various web applications that satisfy both contents and spatial requirements are provided to support different services.

For example, we can search for a hotel from online hotel reservation services such as "Jalan" by using a map view (in Figure 2.3). It is more convenient for us to understand the details of the hotels including their locations and other services information. On the other hand, there is a large demand for spatial data on the Web. We can refer to the report in the article of [30] which says "50 percent of mobile users are most likely to visit shops after conducting a local search, while this number of consumers on tablets or computers is 34 percent." Thus queries with a spatial intent take up a large proportion in search engines.

Consequently, retrieval about these spatial web data brings us new issues and challenges.

Figure 2.1: An example of tweet

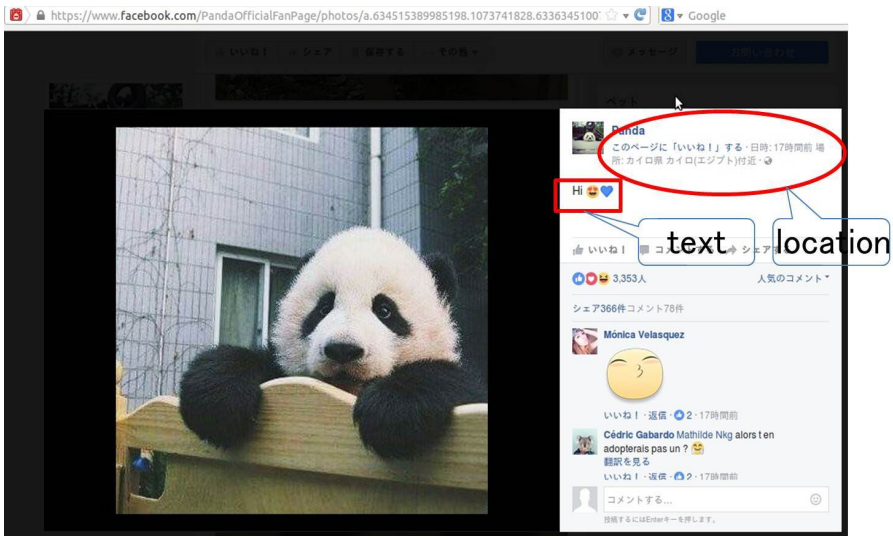


Figure 2.2: An example of Facebook data

Figure 2.3: Spatial search on Jalan

2.2 Spatial keyword queries

A spatial keyword query is a general concept that allows users to issue some keywords of interest to find the spatial object(s) which can best match their needs about textuality and spatiality.

There ara many specific spatial keyword queries for different needs of users [31, 40, 6, 41, 42, 43]. Some tutorials categorized these queries according to their targets of retrieval [2, 23, 5]. Here we introduce some typical queries to review these spatial keyword query problems based on the tutorial of [5].

Early researches of spatial keyword queries target one individual object or a list of ranked objects in a spatial web dataset [31, 32, 33, 34, 35]. They are generalized as standard queries in [5]. As an typical standard queries, Cong et al proposed a Top-k kNN query(TkQ) [32]. In the TkQ problem, given a query q with a location and a set of keywords, each object o in dataset can be evaluated by two arguments: *location proximity* and *text relevancy* [32]. The *location proximity* of o is determined by the Euclidian distance between o and location of q, and *text relevancy* of o is computed using the language model of q's keywords. Then Cong et al used a linear score function of *location proximity* and *text relevancy* to give a score to each object. Finally k objects with the highest scores are returned.

The standard queries are applicable to the cases of finding several objects, each of which can independently meet user's needs. A good example is to find some PoIs in location services by using accessible keywords, such as "comfortable hotel" or "sushi restaurant", which are easy to concentrate on one object.

However, in some other cases, the standard queries may not be appropriate. Considering the case that an user may issue the keywords "*station*, *school*, *supermarket*" to look for a real estate, these keywords are difficult to be covered by an exact place. With a view to these cases, finding a group of objects to match query keywords had been proposed in the researches of spatial keyword queries [1, 10, 9, 3, 7, 4, 36, 37, 38, 39].

One of these researches is the subject of this thesis, m-Closest Keywords(mCK) queries. An mCK query retrieves a set of objects O_{opt} which covers all the query keywords and each of them should be close to each other [1]. This query uses a *diameter* of an object-set O, which is the maximum Euclidean distance between any two objects in O to measure the closeness of O, and minimize the diameter.

Another typical query is so-called *collective spatial keyword query* that proposed by Cao et al [3]. Similar as the mCK query, a collective spatial keyword query also retrieves the optimal object-set O_{opt} that must cover all the user-given query keywords. However this query does not only consider the inner closeness of O_{opt}, but still need to take into account the closeness between O_{opt} and query location. Thus for each object-set O, Cao et al proposed a linear function of these two closenesses to calculate the 'cost' of O, and minimized this cost. Actually the collective spatial keyword query is an extension of the mCK query, whose result is the optimal group of objects with a small diameter, and near to the query location.

After that, some other queries to find optimal group(s) of objects such as *Top-k Groups Queries* [37] are proposed [4, 36, 38, 39]. Most of these queries measure closeness of object-set in the same manner as the mCK query. Thus mCK query problem is important and worthy of study.

Consequently, if the standard spatial keyword query is regarded as to find some positions of interests (PoIs), then the retrieval of object group can be considered as finding a region of interests, which may contain various individual PoIs. This can provide users with more abundant quality results in practical applications.

2.3 Spatial indices

A spatial index is a kind of organization of spatial data according to their locations. As auxiliary spatial data structures, spatial indices are used to improve the speed and efficiency of spatial queries associating with some specific query algorithms. There have been many studies about spatial indices and a number of different types of spatial indices have been proposed. Here we briefly describe two common types of them: R-tree and grid.

2.3.1 R-tree

An R-tree [48] is a height-balanced hierarchical data structure, which is an extension of B-tree in the multi-dimensional spaces. It divides spatial objects by using *minimum bounding rectangles* (MBRs). An MBR R is a 'region', that means all the spatial objects belong to R are included in the 'region' of R, and the 'region' is minimized. Each of MBR corresponds to an *node* in R-tree. There are two kinds of nodes: internal nodes and leaf nodes. An internal node of R-tree has some children nodes such that the MBRs of these children nodes must be included in the MBR of it. A leaf node contains pointers to the spatial objects in its MBR.

Figure 2.4 shows an example of R-tree. R4 to R10 are leaf nodes. Each of them corresponds with an MBR in the planar space, which tightly bounds some objects in it. R1 to R3 are internal nodes, each of which contains some leaf nodes. R0 is the root node of this R-tree. It is a specific internal node such that the MBR of R0 includes all the objects.

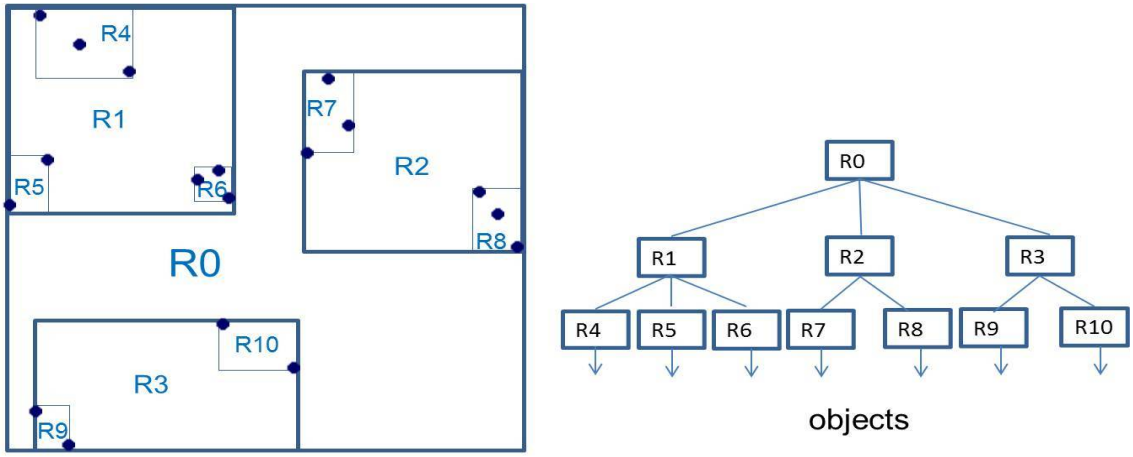


Figure 2.4: An example of R-tree

There are many variants of R-trees to improve the efficiency of R-tree such as R*-tree and R+-tree, etc. R*-trees optimized the node-splitting method in order to reduce the overlap of MBRs and number of children nodes. Thus it is one of the most widely used in various spatial queries.

2.3.2 Grid

A grid index is a simple structure that divides spatial objects into some equal-sized square or rectangular regions. Each region is called a *cell*. In some cases, spatial objects are not evenly distributed, thus the cells with dense objects are often subdivided into sub cells until the numbers of all the cells are less than a threshold (or capacity). This type of unbalanced and multi-level grid index is called a hierarchical grid partitioning. Figure 2.5 shows an example of hierarchical grid partitioning. In this figure, all the objects are assigned to $3 \times 3 = 9$ cells. If the capacity is set to 4, then the cell 1 needs to be subdivided into 9 cells again.

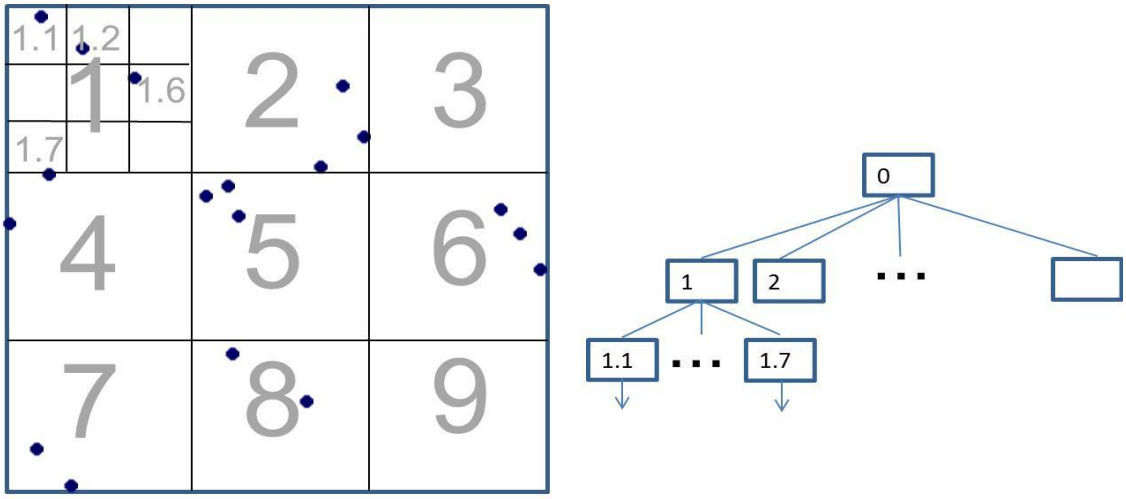


Figure 2.5: An example of grid

2.3.3 Specialized indices for spatial keyword queries

For a spatial web object, it has not only spatial but also the textual attributes. Thus, in order to efficiently search these spatial web objects, a lot of new index technologies have been proposed [46, 47, 1]. These indices often use inverted file or bitmap to index the textual

attributes of objects, then combined them with R-tree or grid by using some combination schemas for their geographic attributes. For example, Zhang et al proposed bR*-tree as index structure for mCK queries. The bR*-tree uses R*-tree as a spatial index. And in each node of R*-tree, a keyword bitmap is added to summarize the keywords in the node. Another example of IR-tree, which is widely used in spatial keyword queries, combines R-tree and inverted file in a seamless manner. It can simultaneously handle both the textual and spatial aspects of the spatial web objects, thus the efficiency of query can be improved.

2.4 mCK queries

2.4.1 Definition

According to the original definition of [1, 9], the *mCK query* (m-closest keywords query) is defined as follows:

[**mCK query**]: Given a spatial database of objects $D = \{o_1, o_2, ..., o_n\}$ and a query with m given keywords $Q = k_1, k_2, ..., k_m$, let $O = \{o_{i1}, o_{i2}, ..., o_{il}\}(\subseteq D)$ be a set of objects, termed an *object-set*. If an object-set O covers all the keywords of Q, $(\bigcup_{o \in O} o.\psi \supseteq Q)$, then we say that O 'satisfies' Q.

Let also $diam(O) = \max_{o_x, o_y \in O} dist(o_x, o_y)$ $(x \neq y)$, where $dist(o_x, o_y)$ is the distance between o_x and o_y, be termed a *diameter* of O. Then, the **mCK query** is defined as 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 object-sets O that satisfy Q.

The mCK query aims to find the object-set O_{opt} such that O_{opt} satisfies query keywords Q and the objects in O_{opt} should be closest to each other. In this definition, the *diameter* is used to measure the closeness of an object-set. As an example, for the object-set $O = \{o_1, o_2, o_3, o_4\}$ in Figure 2.6, distance between o_1 and o_4 is the *diameter* of O.

Actually an object-set O can be regarded as a set of discrete points in the multi-dimensional spaces, and we can use a convex hull of these points in O to represent the amount of space occupied by O. It is intuitive that if the convex hull of O is small, then the objects in O are

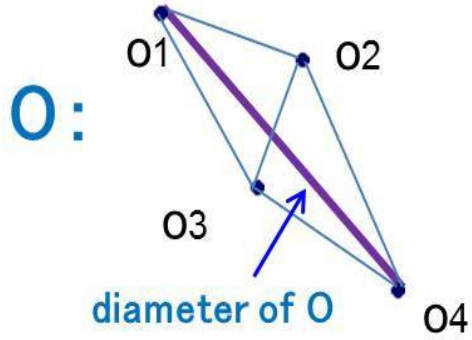


Figure 2.6: Diameter of object-set

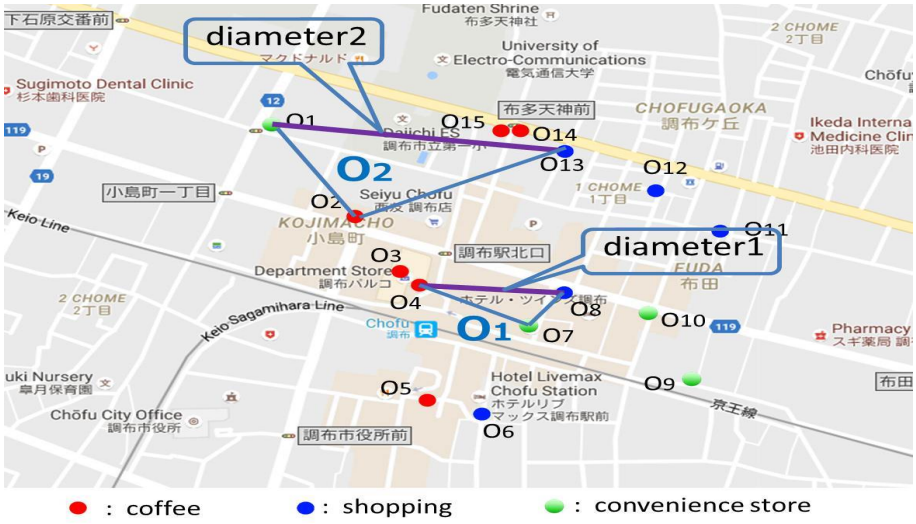


Figure 2.7: Diameter comparison

close to each other. Thus the diameter of convex hull (i.e., the diameter of object-set) is a simple way that can be good at estimating the size of its occupied space. Compared with the other measurement metrics such as area (or volume) size, using diameter can greatly simplify the calculation procedure, especially in high-dimensional space. Therefore the diameter of an object-set is well suited option to represent the its closeness, and this way is used in many other spatial keyword queries which aim for a group of objects [3, 7, 4, 36, 37]. In Figure 2.7 , because the diameter of object-set O_1 is less than object-set O_2, we can see the objects in O_1 is closer to each other than O_2.

In this thesis, the distance between two objects is the Euclidean distance.

2.4.2 Zhang's Apriori-based top-down search strategy

Generic strategy

Essentially the mCK query can be classified as the problem that find the optimal solution from possible solution space over spatial dataset. For this type of problems, a branch-and-bound technique in a top-down process based on hierarchical data structure has been successful in numerous spatial queries such as nearest neighbor query (NN query) [24], range query [27] and spatial join [25, 26].

We briefly describe a typical branch-and-bound technique of top-down strategy as follow:

Step 1 (Initialization): Use a global variable χ^* to represent the current optimal solution of the spatial query and initialize χ^* with $+\infty$ or $-\infty$.

Step 2 (Start): Start from the root node of the hierarchical data structure. Because the optimal solution must exist in the region of root node, choose this region as current solution space S_C.

Step 3 (Check and Pruning): For current solution space S_C, if it exists no better solution than χ^* in S_C, then prune the branch of S_C. Otherwise, goto Step 4.

Step 4 (Branching): If S_C is an internal-node, then divide S_C into some partial regions as candidate solution spaces by using its children nodes. Otherwise if S_C is a leaf-node, then enumerate all the solution χ in S_C and compare with χ^* for each χ: if χ is better than χ^*, then update χ^* with χ.

Step 5 (Selection of Candidate): Choose one from the candidate solution spaces as current solution space S_C, then return to Step 3.

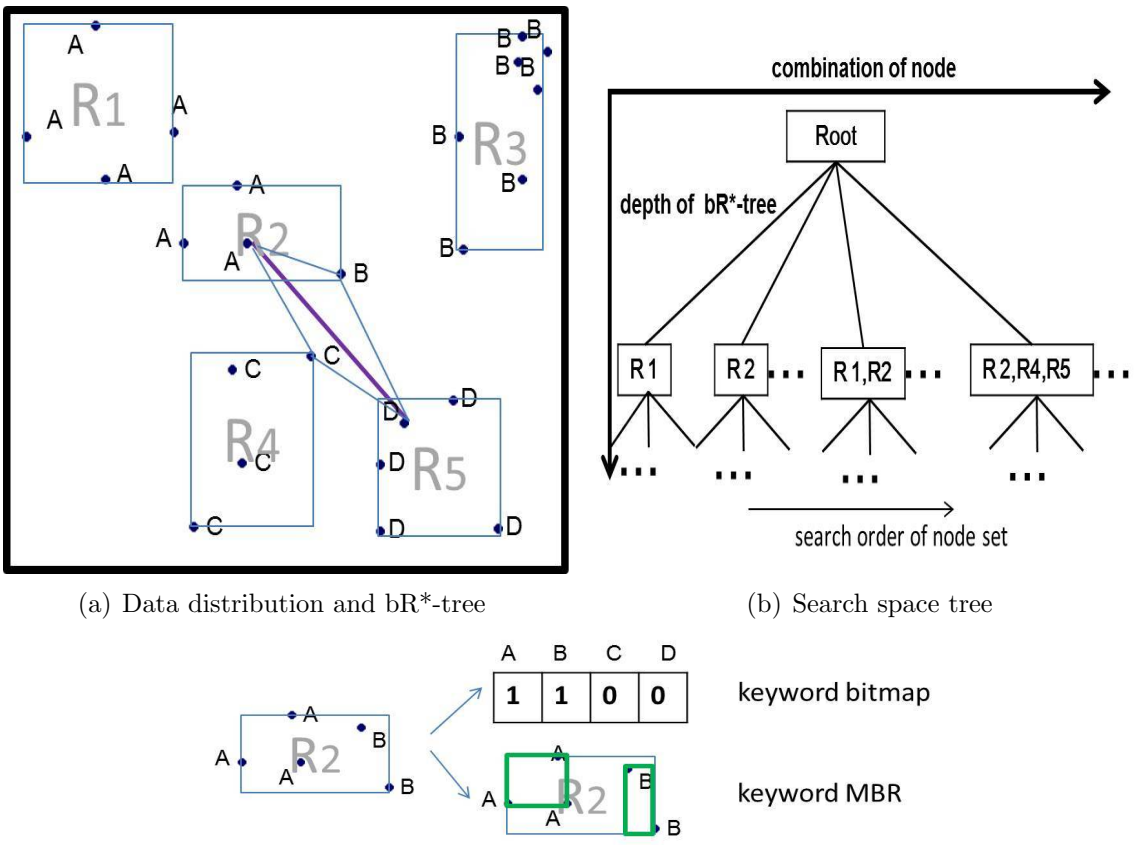
Step 6 (Termination Test): Stop if there is no candidate solution spaces. Finally χ^* is the optimal solution.

Zhang's approach

According to the above description, Zhang et al proposed a top-down exploration approach taking advantage of a special R*-tree called bR*-tree [1](2009). The bR*-tree is an extension

of the index structure of R*-tree . Beside the node MBR of R*-tree, each node N in bR*-tree is augmented with two additional information: *keyword bitmap* and *keyword MBR*.

- *keyword bitmap*: *keyword bitmap* is a bitmap that summarize the keywords in the node N. Each bit b_i shows that whether a keyword k_i exists in N. If $b_i = 1$, then there exists at least one object associated with keyword k_i in N. Otherwise, there is no object of k_i in N.
- *keyword MBR*: For each keyword k_i, the *keyword MBR* of k_i is the minimum bound rectangle of all the objects in N that are associated with k_i.



(a) Data distribution and bR*-tree

(b) Search space tree

(c) Keyword bitmap and keyword MBR

Figure 2.8: An example of Apriori-Z

Figure 2.8(a) is an example of the bR*-tree for the objects associated with four keywords A,B,C and D. In this bR*-tree, root node R_0 has five children nodes R_1 to R_5. Each node has an MBR which is tightly bound up with all objects in this node. Furthermore, *keyword*

bitmap and *keyword MBR* are also attached in each node(Figure 2.8(c)). For example of Figure 2.8(c), in node R_2 the *keyword bitmap* = 1100 denotes that R_2 contains objects only associated with keyword A and B, not C and D. Accordingly, the *keyword MBR* of A and B are the spatial bound of all the objects with keyword A and B, respectively.

Zhang et al used the bR*-tree to index all the objects and proposed a top-down search approach based on this bR*-tree. Next we describe this top-down approach.

Different from the typical spatial queries such as NN queries or range queries, the result of mCK query is a set of objects, thus the solution spaces are not confined in one node of bR*-tree. For this reason, Zhang et al used a set of nodes, termed as a node-set, as a solution space in the search process. That means for a solution (an object-set O) in a solution space (a node-set N), each object of O must belong to one node of N and each node of N must contain at least one object of O.

Therefore in the branching procedure of top-down process for the mCK query problem, the solution space S_C is divided into all the possible sub solution spaces (sub node-sets), each of which is a combination of children nodes of S_C. For example, Figure 2.8(b) is the search space tree of the bR*-tree in Figure 2.8(a). In Figure 2.8(b), we can see each branch of the root node R_0 is a subset of $\{R_1, R_2, ..., R_5\}$ such as $\{R_1, R_2, R_4\}$. Here the size of the node-set is m at most, because an object-set is compose of at most m objects. In addition, these sub node-sets are generated as an Apriori-style way which has been used for mining frequent itemsets. Thus the enumeration of these sub node-sets follows the order from length-1 (one MBR) to length-m (m MBRs). Due to this, we call Zhang's top-down approach as *Apriori-Z* approach.

Apriori-Z

In Apriori-Z, a global variable δ^* is denoted as the current smallest diameter among the object-sets explored so far. Then in the pruning procedure of top-down process, Zhang et al proposed three pruning rules for a node-set N as follow to decide whether it can be pruned.

Pruning rule 1: If N does not contain all the query keywords, then there exists no object-set that contains all the keywords in N. Thus N can be pruned directly.

Pruning rule 2: If there exists two nodes n_i, $n_j \in N$ such that the minimum distance between n_i and n_j is greater than δ^*, then N can be pruned. That is because the minimum distance between n_i and n_j is an lower bound of distances between any two objects o_i and o_j ($o_i \in n_i, o_j \in n_j$). Hence it is also the lower bound of the diameters of possible object-sets. If this lower bound is greater than δ^*, there exists no object-set with diameter than δ^* in N.

Pruning rule 3: If the distance between two keywords are greater than δ^*, which means for any object o_i associated with keyword k_i and o_j associated with keyword k_j in N, we can always find that $dist(o_i, o_j)$ is greater than δ^*, then N can be pruned. The keyword MBRs in each node of N are used to calculate the distances between two keywords.

In the selecting procedure of top-down process, Zhang et al traversed the search space tree in the depth-first order. That means if a node-set N as the current solution space cannot be pruned, then immediately access N's sub node-sets.

In consequence, we summarize Apriori-Z approach as follow:

Step 1 (Initialization): Use a global variable δ^* to represent the current smallest diameter. δ^* is initialized as follow: first find the smallest node N_I that contains all the query keywords among all the nodes of bR*-tree. If N_I is an internal-node, then initialize δ^* with the diagonal distance of N_I; if N_I is a leaf-node, then initialize δ^* with the smallest diameter in N_I by exhaustively generating all the object-sets.

Step 2 (Start): Start from the root node-set $\{root\}$ of the bR*-tree. choose it as current node-set N_C.

Step 3 (Check and Pruning): For current node-set N_C, if it can be pruned by the three pruning rules, then skip N_C and repeat to check next candidate node-set. Otherwise, goto Step 4.

Step 4 (Branching): If all the nodes in N_C are leaf-nodes, then enumerate all the object-sets in N_C and compare each diameter δ with δ^* : if $\delta < \delta^*$, then update δ^* with δ. Otherwise, create all the sub node-sets in an Apriori-style order as candidate node-sets.

Step 5 (Selection of Candidate): Choose the first sub node-set in a depth-first way, then return to Step 3.

Step 6 (Termination Test): Stop if all the candidate node-sets are checked. Finally δ^* is the optimal diameter.

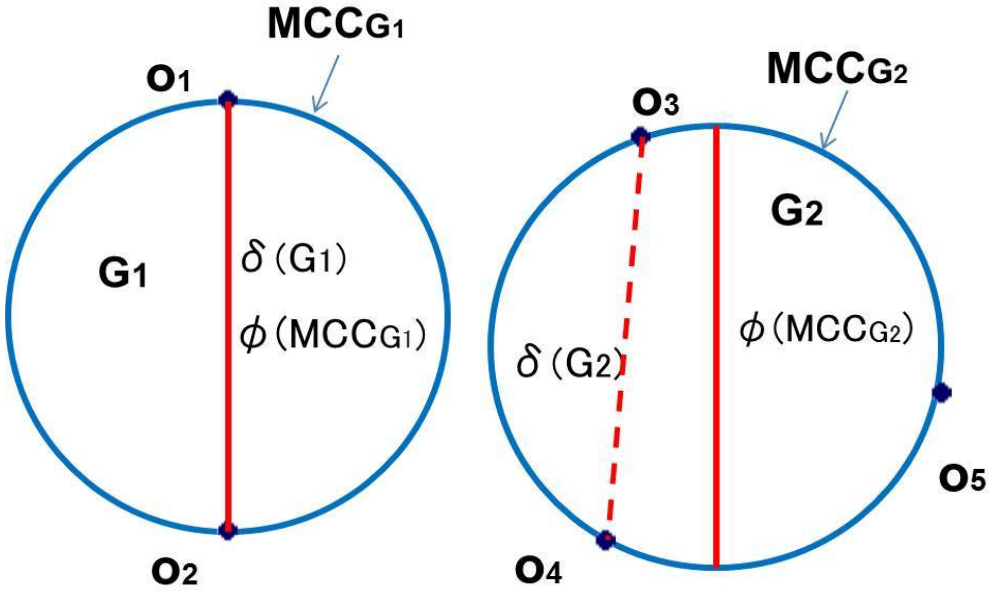
2.4.3 Guo's approach

In the study of [9] in SIGMOD 2015, Guo first theoretically proved that the problem of mCK queries is NP-hard. Then they mainly focused on several algorithms for finding the approximation solution of mCK queries problem. At the end of their study , they also presented an exact algorithm by utilizing the result of the approximation algorithm. Here we summarize their algorithms, according to the description of their paper [9].

The first proposed algorithm is called Greedy Keyword Group(GKG). Given a query $Q = \{k_1, k_2, ..., k_m\}$, make a set of object collections $\mathbb{C} = \{C_1, C_2, ..., C_m\}$ such that each C_i ($i \in \{1, ...m\}$) is the collection of objects associated with keyword k_i. Then GKG first chooses the collection having the smallest size in $\mathbb{C}$, denoted by C_{inf}. Next, for each object o in C_{inf}, GKG finds the nearest object from o, from within C_j for each keyword k_j ($k_j \in Q - o.\psi$). Then these nearest objects with o form one object-set covering all the keywords of Q. Thus, after all of these object-sets from C_{inf} are generated, GKG chooses the object-set G_{GKG} which has the smallest diameter as the answer. Guo proved that this answer is not larger than twice of the diameter of the optimal result.

Next, Guo proposed a series of algorithms which can get better approximation ratios than GKG. The basic idea of these algorithms is to construct a *Minimum Covering Circle* (MCC) for an object-set G , denoted by MCC_G. MCC_G is defined as the circle that encloses all the objects of G with the smallest diameter. Then the diameter of MCC_G is used as an approximation of G's diameter.

In [9], Guo denoted the diameter of the circle MCC_G by $\phi(MCC_G)$, and denoted the diameter of object-set G by $\delta(G)$. The relationship between $\phi(MCC_G)$ and $\delta(G)$ can be deduced by using a theorem (in [49]) that MCC_G can be determined by at most three object points in G which lie on the boundary of the circle of MCC_G. Guo states that if the circle of



(a) MCC_{G_1} is determined by two points (b) MCC_{G_2} is determined by three points

Figure 2.9: Example of two diameters (This figure is derived from Fig.2 of [9])

MCC_G is determined by only two object points, then the line segment connecting those two points must be a diameter of the circle (see Figure 2.9 (a)). If MCC_G is determined by three object points, then the triangle consisting of those three points is not obtuse (see Figure 2.9 (b)). Based on these observations, Guo derived out the following inequality relation [9].

$$\frac{\sqrt{3}}{2}\phi(MCC_G) \leq \delta(G) \leq \phi(MCC_G). \tag{2.1}$$

Accordingly, the search policy of Guo's approximate algorithms is to find the MCC having the smallest diameter such that the object-set in the MCC must cover all the query keywords of Q.

The above MCC is called *Smallest Keywords Enclosing Circle*, denoted by $SKEC_Q$. And the object-set in $SKEC_Q$ is denoted by G_{SKEC}. Then according to the inequation (2.1), Guo proved that $\delta(G_{SKEC}) \leq \phi(MCC_{SKEC}) \leq \frac{2}{\sqrt{3}}\delta(G_{opt})$ where $\delta(G_{opt})$ is the diameter of the optimal result. Thus the approximation ratio can be reduced to $\frac{2}{\sqrt{3}}$ by using $SKEC_Q$ as approximate solutions.

To find the smallest keywords enclosing circle $SKEC_Q$, Guo proposed three algorithms

which are called $SKEC$, $SKECa$ and $SKECa+$, respectively. Basically, according to the above explanations, the circle of $SKEC_Q$ is determined by two or three object points. In algorithm $SKEC$, Guo considers the set of objects that contain at least one query keyword, denoted by O', and then for each object $o \in O'$ as a seed point, o is combined with other one or two points to determine a circle and check if this circle contains all the query keywords. In $SKEC$, some objects which are combined with o can be pruned out by using the result of algorithm GKG.

Next, Guo uses algorithm $SKECa$ and $SKECa+$ to find $SKEC_Q$ approximately. $SKECa$ uses a binary search to find the approximate smallest keywords enclosing circle for each $o \in O'$. $SKECa$ first sets a circle with an upper bound D of a diameter and sets o as an object located on the boundary of the circle. Then this circle is rotated around o clockwise and tests whether or not it can cover all the query keywords at a particular position. If it can, then $SKECa$ tries to test a smaller D; otherwise, it enlarges D. This process is repeated until the error tolerance ratio of binary search is converged within a small value. After all the objects o are checked, the approximate answer of $SKEC_Q$ can be found. The other algorithm $SKECa+$ enhanced $SKECa$. In $SKECa+$, the binary search process is performed on all objects in O' together,instead of the testing on each of them separately. Thus the checking cost can be reduced.

At last, Guo also proposed an exact algorithm. This algorithm sets a circle with diameter $\frac{2}{\sqrt{3}}\phi(G_{SKECa})$ where G_{SKECa} is the answer of algorithm $SKECa+$. Then the circle is rotated around each o clockwise, and once it covers all the query keywords, then the algorithm exhaustively enumerates all the object-sets in the circle. Finally, the exact result can be found.

2.5 Our motivation based on top-down approach

In this chapter, we introduced mCK query problem and some previous researches for it. As a spatial query problem, the top-down search by using a spatial index is a kind of fundamental method. Though Guo's approach which is different from top-down style is good at finding the approximation solution of mCK query problem, when considering further requirements

of various spatial searches such as finding top-k closest object-sets, top-down search is surely an useful technique for these extensions.

However the existing Apriori-Z approach is a straightforward top-down method, which combines the node-sets of bR*-tree in an apriori way level-by-level with some pruning rules. There are some apparent questions in this approach.

- Apriori-Z decides whether or not a given node-set can be pruned out through the comparison of the N's lower bound and the current smallest diameter δ^* in pruning rule 2 and 3. Hence the δ^* is an important factor that influences the pruning efficiency. If an smaller δ^* can be found in an early stage, then more node-sets can be pruned out. Otherwise, it needs to generate enormous amount of object-sets and node-sets such that the search efficiency is poor. However, the Apriori-based enumeration of sub node-sets cannot guarantee that the smaller object-set will be enumerated firstly especially in the skewed distribution.

- Apriori-Z uses a bR*-tree to store all the objects. However the bR*-tree is not applicable to the real spatial web which are frequently updated like Flickr and Twitter data. Moreover for the practical cases of mCK query, such as that we may want the results from different datasets (like Twitter and Flickr) simultaneously, we need a more flexible index to satisfy various user's requirements.

Therefore, there remain much rooms for us to consider deeply and explore more sophisticated top-down approaches for mCK query problem. We explore these sophistication in the following chapters.

Chapter 3

DCC: A New Top-down Search Strategy with a Priority Order

3.1 Problem setting

3.1.1 Objective of this chapter

In chapter 2, we discussed some problems in the Apriori-Z approach for mCK query problem. Thus the objective in this chapter is to ameliorate these problems in two aspects:
1) Consider a new technique to organize these spatial web data for more practical situation.
2) Improve the pruning ability of search space by enumerating the node-sets in a priority order.

3.1.2 Zhang's Apriori-based method on bR*-tree

In the study of [1], Zhang et al used a specialized R*-tree, a bR*-tree, to store all records in preparation, and 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. In addition the Apriori-based enumeration method performs well especially when one MBR (or, one object) satisfies multiple keywords. In contrast, 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.

3.1.3 Our setting

In this chapter, 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.

Figure 3.1 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.

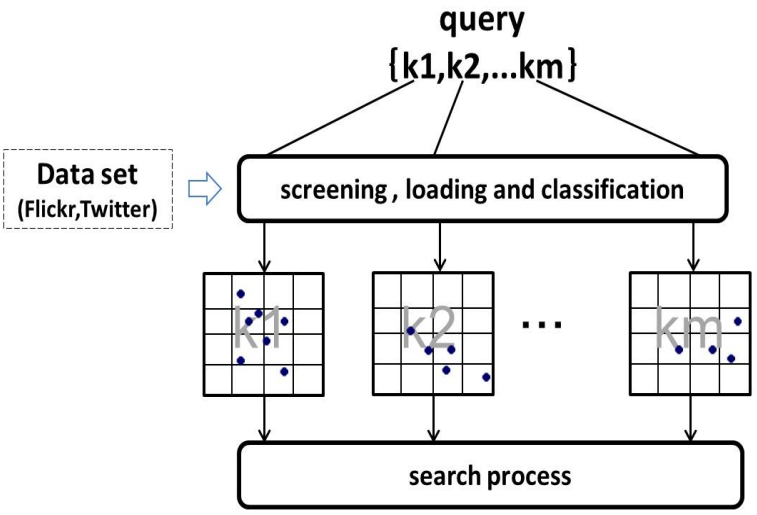


Figure 3.1: On-demand creation of grid partitioning

Figure 3.2(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 Figure 3.2(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, Figure 3.2(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:

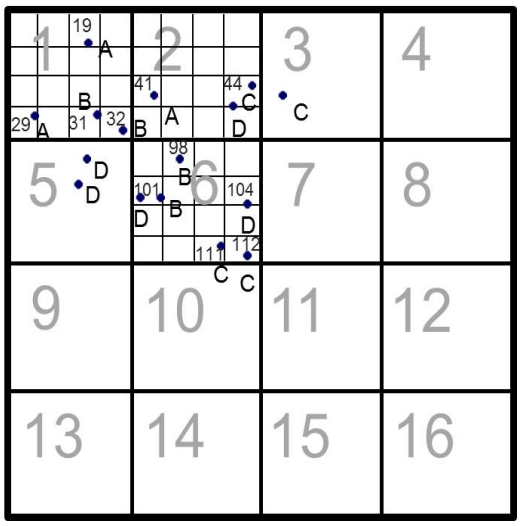
Algorithm 1: Nested-Loop(*curSet*)

curSet is an m-sized node-set each of node in *curSet* belong to a grids G_i .

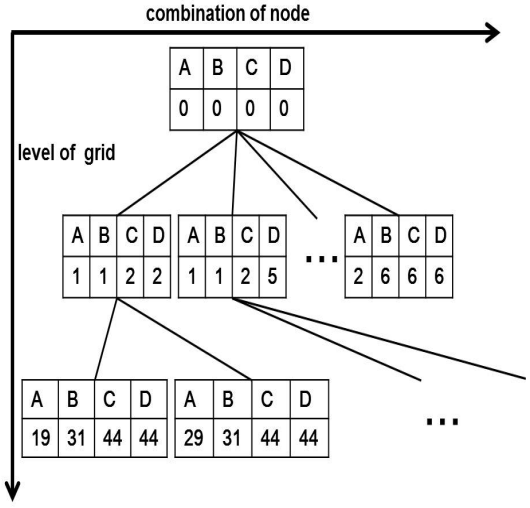
Step 1: If the *curSet* is an internal node set and the distance between every two nodes $\in$ *curSet* is less than δ^*, we first put all child-node(s) (i.e. child grid-cells) of each internal node into a list, respectively; and then start to enumerate new child node-sets according to the nested loop 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^*.

Step 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.

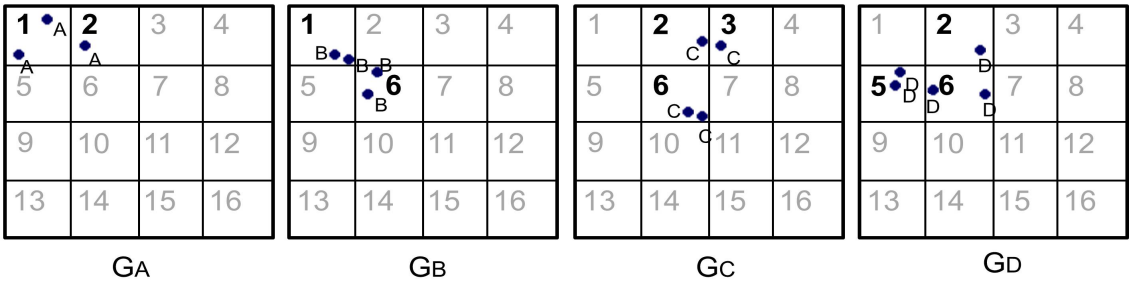
In the example of Figure 3.2(c), we start from the root node-set $\{A[0], B[0], C[0], D[0]\}$. Then the node-set $\{A[1], B[1], C[2], D[2]\}$ becomes the first child node-set generated, because A and B exist firstly in the 1st cells but C and D appear firstly in the 2nd cells. In case of a naive nested 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



(a) data distribution



(c) the search space of mCK query



(b) m independent grids structure

Figure 3.2: grid and search space of m-CK

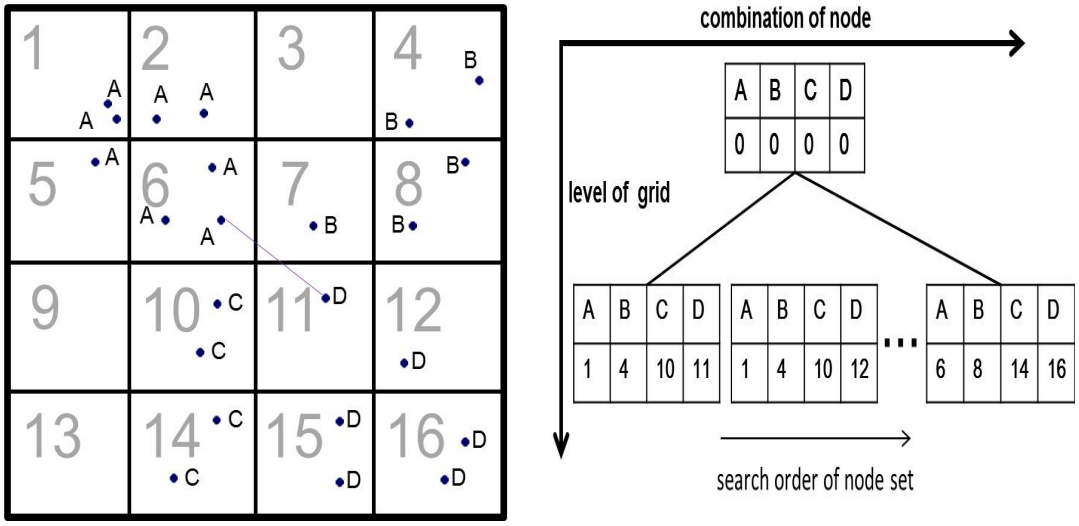
naive approach is too expensive. Furthermore, as a disadvantage of using a grid, the grid structure is often weak in clustering correlated objects in one grid-cell (the over-splitting problem). Thus we must give a higher priority of search to a better set of grid-cells during the recursive search in the search space of Figure 3.2(c).

3.2 Diameter Candidate Check (DCC)

3.2.1 Basic idea of DCC

In the search space of Figure 3.2(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



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

Figure 3.3: search method of the mCK query

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 Figure 3.3(a) and the corresponding search space of Figure 3.3(b). Actually, there are four independent grids for A, B, C, D in Figure 3.3(a), but we visualize these grids by one virtual grid of Figure 3.3(a).

Figure 3.3(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 Figure 3.3(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 Figure 3.3(b) or Figure 3.2(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,[7]), 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.

3.2.2 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
  $MaxMindist(N) = max_{n_{w_i}, n_{w_j} \in N} Mindist(n_{w_i}, n_{w_j})$.

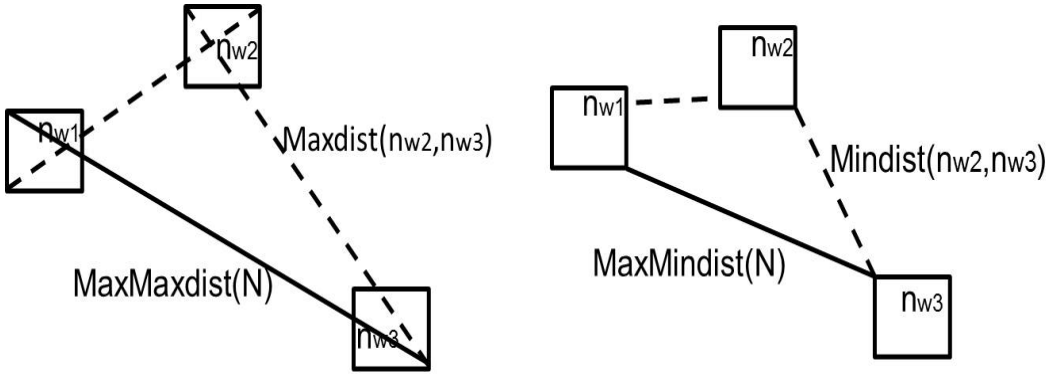


Figure 3.4: MaxMaxdist and MaxMindist

$MaxMaxdist$ 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. Figure 3.4 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.

3.2.3 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 Figure 3.5. In the following, we explain the case of four keywords A, B, C, D of Figure 3.2(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:

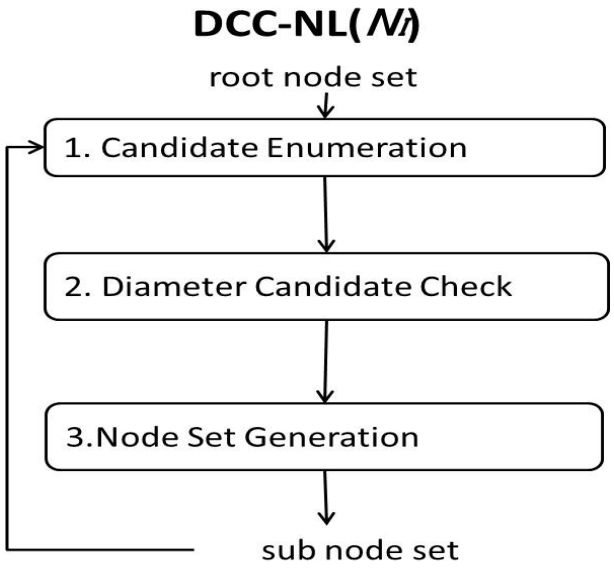
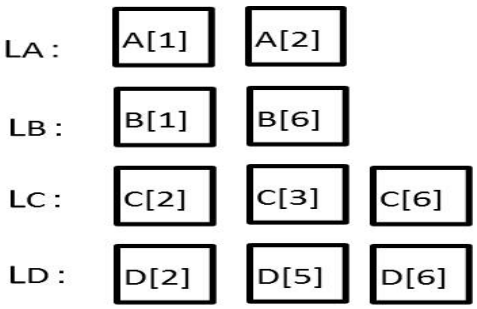


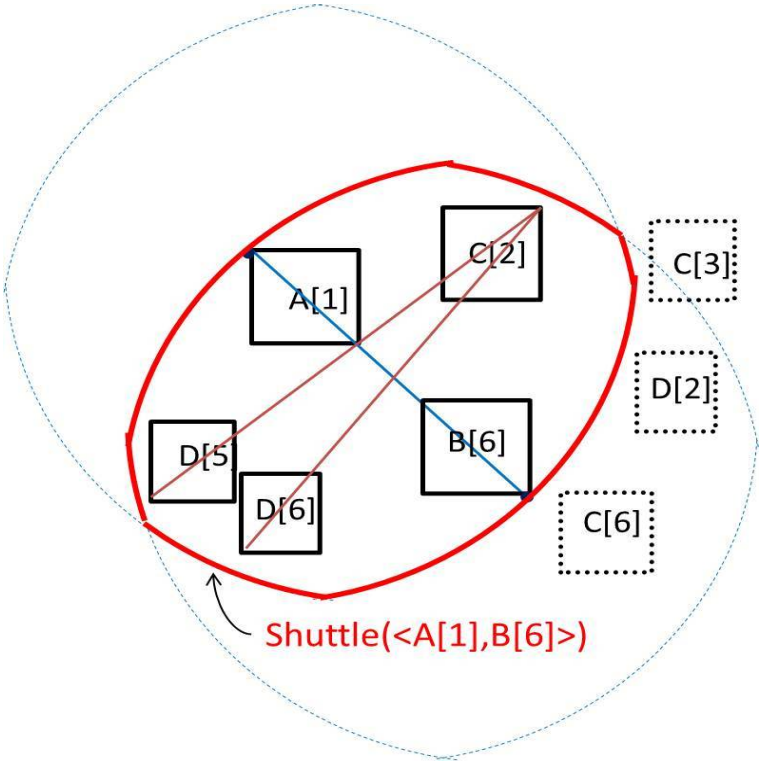
Figure 3.5: workflows of DCC three steps

[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 Figure 3.6(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,



(a) child-node lists



(b) example of *DC* and Check

Figure 3.6: Creating a node-set from a given *DC*

$\langle A[1], B[1]\rangle, \langle A[1], B[6]\rangle, \langle A[2], B[1]\rangle, \langle A[2], B[6]\rangle$, 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]

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 3.2.1.)

(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 scope (red) drawn in Figure 3.6(b), where this shuttle scope represents the above condition. This shuttle scope is uniquely determined by a DC, thus being denoted by $Shuttle(DC)$. In order to compose N_D from a given DC, we need not consider any nodes which are outside the $Shuttle(DC)$.

As an example, in Figure 3.6(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 $Shuttle(\langle A[1], B[6]\rangle)$. 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 $Shuttle(DC)$. 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 $Shuttle(\langle n_i, n_j\rangle)$. 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 $Shuttle(DC)$ 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 Figure 3.6(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 $Shuttle(DC)$ 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.

Algorithm 2: DCC-NL(*curSet*)

curSet is an m-sized node-set each of node in *curSet* belong to a grids G_i .

Step 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 *DC*s and sort them. For each *DC* $\langle n_i, n_j \rangle$ in these *DC*s 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 $Shuttle(\langle n_i, n_j \rangle)$. 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.

Step 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 *DC*s and sort them. For each *DC* $\langle o_i, o_j \rangle$ in these *DC*s :

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.

3.3 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.)

In Figure 3.7(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.

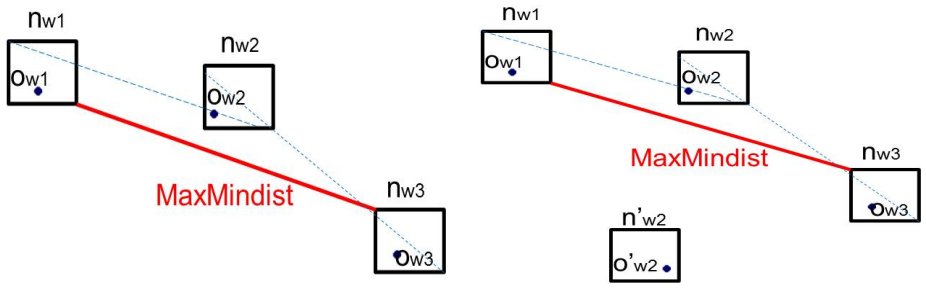
By the lemma 1, we consider two pruning rules:

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 Figure 3.7(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.



(a) example of Lemma 1 (b) example of pruning rule 1

Figure 3.7: Pruning rules

3.4 Experimental evaluation

3.4.1 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 Figure 3.1. We use three types of data-distribution:

1) uniform distribution: All coordinates of data points are randomly generated (Figure 3.8(a)).

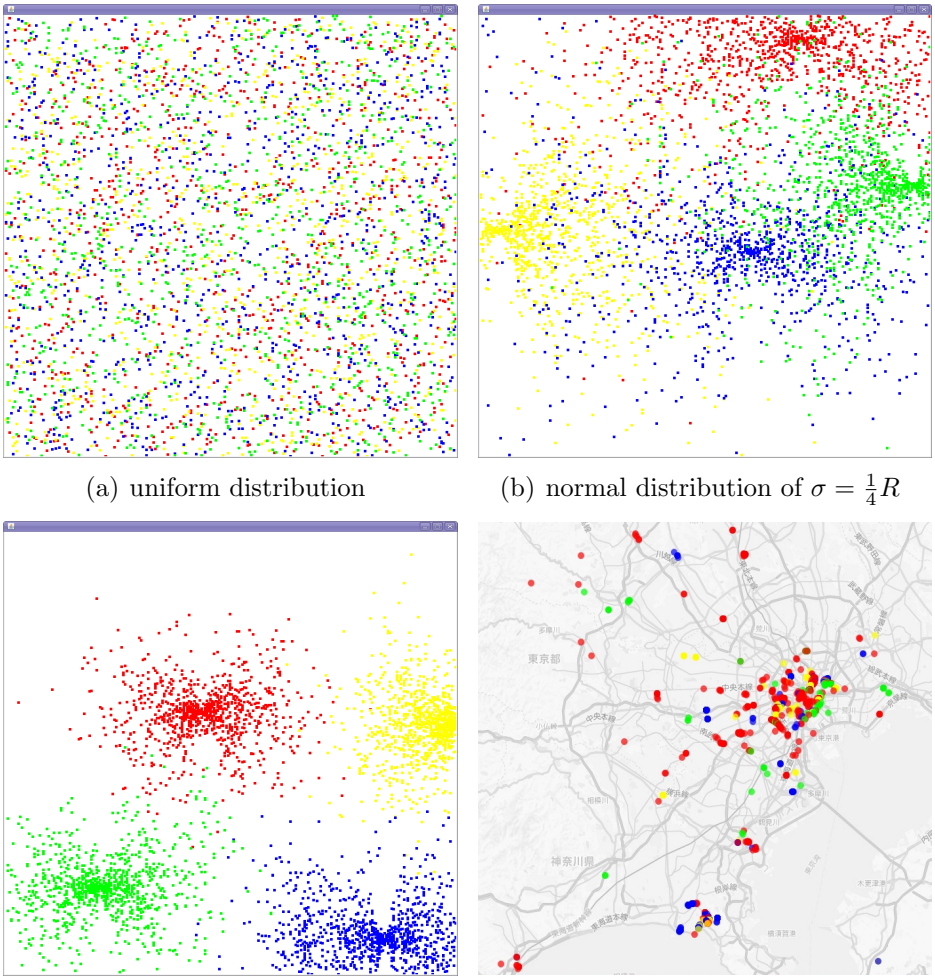
2) normal distribution with $\sigma = \frac{1}{4}R$: For each keyword, we randomly generate one point as the reference point, and then all the data points associated with this keyword are generated so that the distance to the reference point follows a normal distribution (Figure 3.8(b)). We set the standard deviation $\sigma = \frac{1}{4}R$.

3) normal distribution with $\sigma = \frac{1}{8}R$: The same as 2) except that $\sigma = \frac{1}{8}R$ (Figure 3.8(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 Figure 3.8(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 Figure 3.1.

As a grid-partitioning of Section 3.1.2, we defined that each cell is divided when the number of data points is greater than 100. The fan-out is set to 100 (= a square of 10×10). 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 3.1.2. We execute the Zhang's



(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

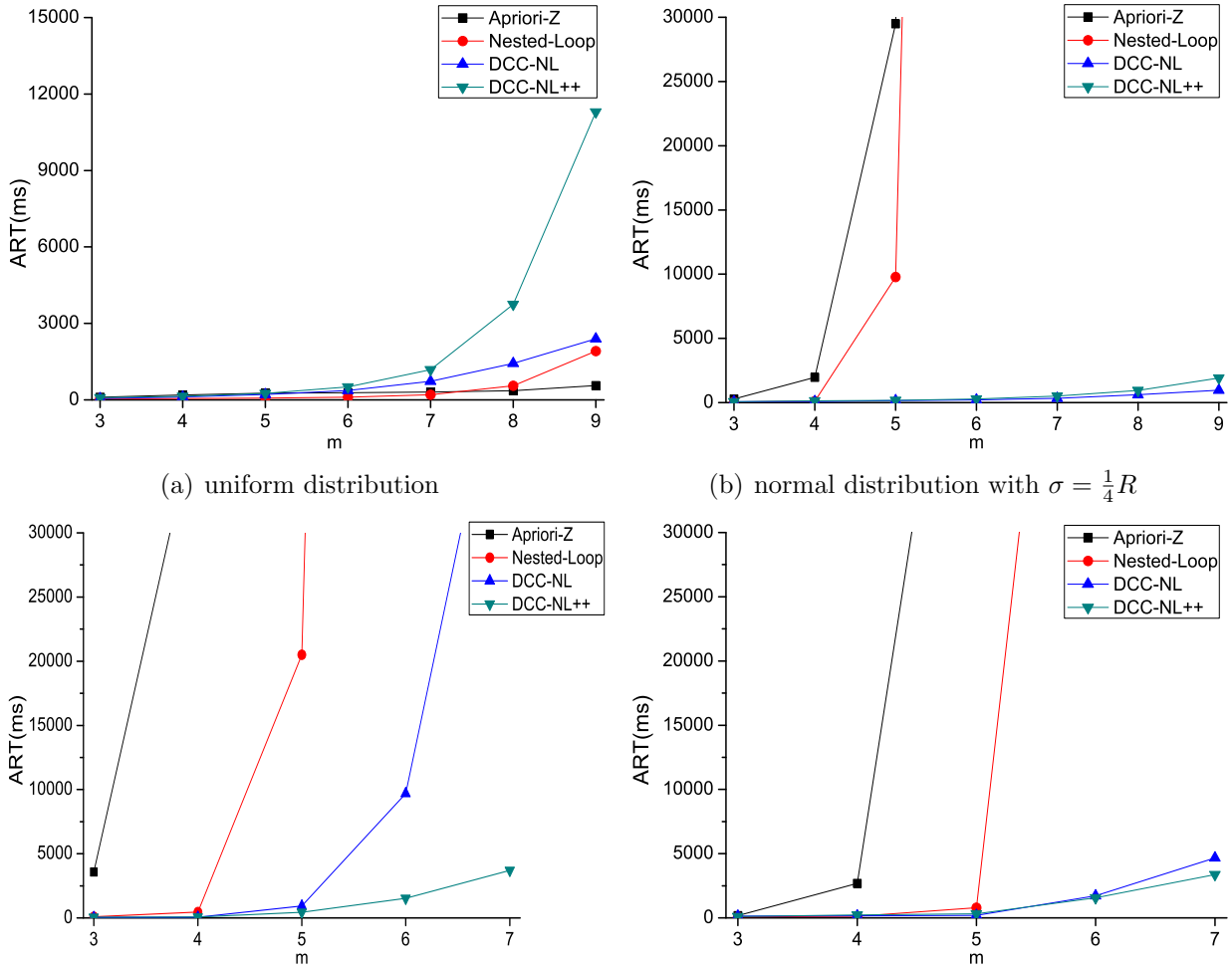
Figure 3.8: data points distribution

method by making one grid structure of Figure 3.2(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 3.3.



(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

Figure 3.9: performance of mCK

3.4.2 Evaluation of Synthetic Datasets

First we tested synthetic datasets in Figure 3.9(a)-(c).

Figure 3.9(a) is ART vs. the number of keywords m in the uniform distribution.

In Figure 3.9(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.

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, Figure 3.9(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, Figure 3.9(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 $m \geq 8$.

3.4.3 Evaluation of Flickr Datasets

Next Figure 3.9(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 Figure 3.9(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.

3.5 Summary

In this chapter, 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.

Chapter 4

Optimization of mCK Search by using Recursive DCC Strategy

4.1 Optimization of DCC-NL

4.1.1 Objective of this chapter

In this chapter, we develop the DCC strategy to further improve the search efficiency in two ways:
1)Enumerate node-set in a better priority order than DCC-NL approach in Chapter 3.
2)Strengthen the ability of pruning by using a tighter lower bound.

4.1.2 Policy to optimize DCC-NL

In the general idea about the top-down approach for a mCK query, a node-set can be pruned if the lower bound of this node-set is greater than the smallest diameter discovered so far (δ^*). Thus we can improve the pruning ability through two ways:

1. Quickly find a smaller δ^*.
2. Increase the lower bound of node-set.

In Chapter 3, in order to realize the first way above that find a smaller δ^* at an early stage, we considered a new exploration strategy called DCC, which can give a node-set with smaller

diameter a higher priority. And we proposed DCC-NL algorithm to realize DCC strategy in a nested loop method. DCC-NL first enumerates all node-pairs in an ascending order of their $Maxdist$s which are regarded as the upper bounds of diameters. Then generates node-sets for each node-pair by combining with the relevant nodes in the shuttle scope. Thus the node-set with a smaller upper bound will be given a higher priority in generation.

However the node-sets derived from same node-pair are unordered. Due to nested-loop generation, this part of node-sets will not be given a priority order. If the number of relevant nodes in the shuttle scope is small, it will not be an issue. but when the number grows larger, numerous unordered node-sets may reduce search efficiency. In this chapter, we will discuss this problem, and propose a recursive DCC strategy to solve it.

Further, we only considered the first way to improve the search efficiency. Actually, the second way is also expected. DCC-NL uses the $Mindist$ between two MBRs in a node-set N as the lower bound which is used to determine whether or not N can be pruned. In this chapter, we enlarge this lower bound by using a tighter lower bound of N to improve its pruning ability.

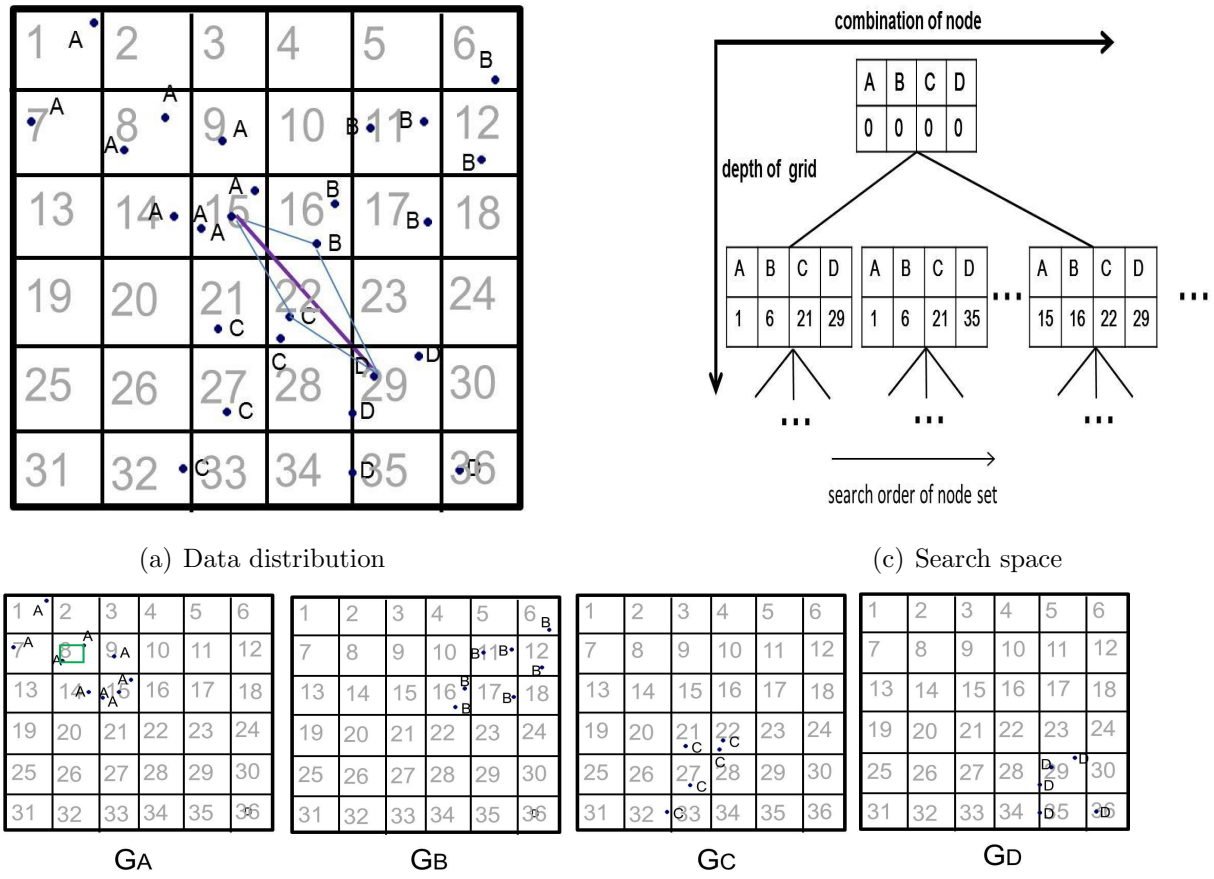
We call these technique as RDCC which contain the recursive DCC strategy for generation and tight lower bound for pruning.

4.2 Review of DCC-NL search approach

4.2.1 Description of DCC-NL

Here we describe the DCC-NL algorithm in an example of Figure 4.1 .

Similar as the description in Section 3.3, Figure 4.1(a) shows the data distribution. Once a query of four keywords $\{A, B, C, D\}$ is submitted, we create an independent grid for each keyword, thus there are four grid structures $\{G_A, G_B, G_C, G_D\}$ in Figure 4.1(b). We use a number to identify a cell of a grid. In 4.1(b) the roots are divided into 36 subcells, thus the identification numbers are from 1 to 36 in this level of grid. We also use the symbol $A[i]$ to represent the i-th cell of the grid corresponding to the keyword A. Such $A[i]$ is termed a *node* ($A[0]$ is the root node of G_A). Figure 4.1(c) shows the search space tree of mCK query. Each element of this search space tree is a node-set $\{A[i], B[j], C[k], D[l]\}$, which is composed of



(a) Data distribution

(c) Search space

(b) On-the-fly grids

Figure 4.1: An example of DCC-NL

one cell from different grid.

The algorithm $DCC\text{-}NL(N)$ (N is an variable of node-set) has the following three steps in top-down process. At first, $\delta^* = \infty$ and DCC-NL($N_0 = \{A[0], B[0], C[0], D[0]\}$) is invoked.

Step 1: For the child-nodes of N, enumerate all the node-pairs $\langle K_1[a], K_2[b]\rangle$ ($K_1, K_2 \in \{A, B, C, D\}, K1 \neq K_2$) as DCs. In the example of Figure 4.1, 107 DCs, $\langle A[1], B[6]\rangle$, $\langle A[1], B[11]\rangle, ..., \langle C[32], D[36]\rangle$, are generated from N_0. Then sort these DCs by the ascending order of $Maxdist(DC)$.

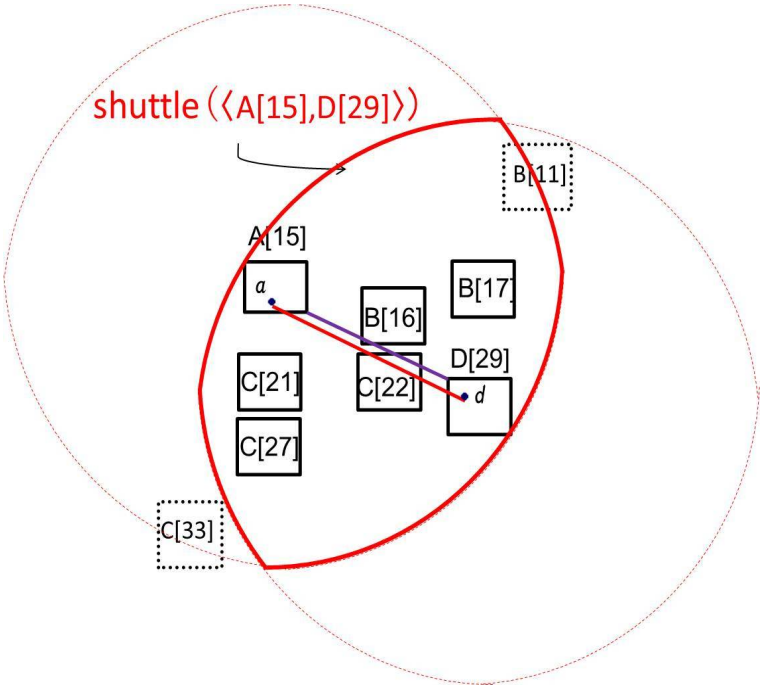
Step 2: Pick up a DC from the top of the sorted list of DCs. Then all the relevant nodes which may constitute a node-set must exist in the shuttle scope of DC. Figure 4.2(a) shows the shuttle scope of $DC = \langle A[15], D[29]\rangle$. We can see that there are five nodes in this shuttle scope, $B[16], B[17], C[21], C[22], C[27]$.

Step 3: Generate all the $(m-2)$-sized sub node-sets for the relevant nodes in the shuttle scope by using a nested loop method . These sub node-sets should cover the remaining $(m-2)$ keywords (except the keywords of the DC). Once a sub node-set is generated, merge them with the two nodes of DC to a full m-sized node-set N'. If all the nodes in N' are leaf-nodes then find the smallest diameter δ in N' and update δ^* if $\delta < \delta^*$; otherwise, recursively invoke DCC-NL(N'). In Figure 4.2(b), we generate $6(2 \times 3)$ sub node-sets for node lists of keyword B and C and merge them. Then we can see the node-set $\{A[15], B[16], C[21], D[29]\}$) will be generated firstly.

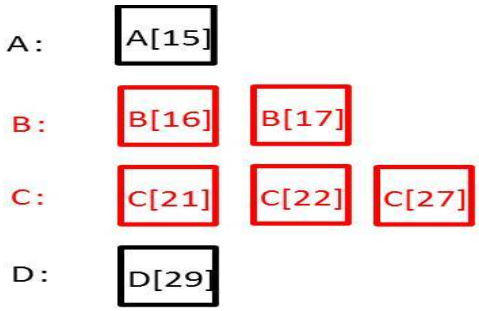
4.2.2 The problems of DCC-NL

There are two problems in DCC-NL algorithm.

Problem 1: DCC-NL can give higher priority to the part of node-sets with a smaller DC. Overall, we can get a smaller diameter at an early stage. However, for the node-sets derived from the same DC, the prioritized enumeration of these node-sets cannot be ensured. That is due to nested loop method for generating $(m-2)$-sized sub node-sets. For example, in Figure 4.2(a), among all the node-sets derived from $DC = \langle A[15], D[29]\rangle$, $\{A[15], B[16], C[22], D[29]\}$ should be given a higher priority than $\{A[15], B[17], C[27], D[29]\}$,



(a) Shuttle scope



(b) Relevant nodes

Figure 4.2: Generation of node-sets

because we can get a smaller diameter in $\{A[15], B[16], C[22], D[29]\}$. But we cannot guarantee this by using a nested loop method.

Problem 2: In DCC-NL algorithm, given a node-set $N = \{n_1, n_2, ..., n_m\}$, the lower bound (termed as LB) of a node-set N is defined by

$$LB(N) = \max_{n_a, n_b \in O} Mindist(n_a, n_b) \tag{4.1}$$

For instance, in Figure 4.3 , for the node-set $N = \{n_1, n_2, n_3\}$, the lower bound of N is determined by the $Mindist$ of n_1 and n_3, $LB(N) = Mindist(n_1, n_3)$.

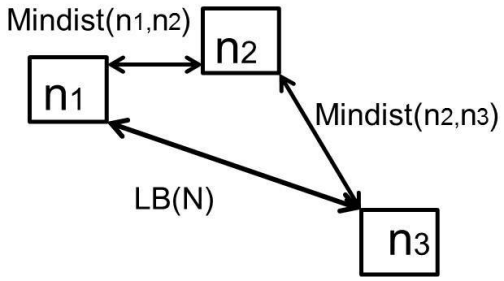


Figure 4.3: Lower bound of node-set

Next we compare the lower bound with δ^*, if $LB(N) \geq \delta^*$, then N can be pruned out. However some node-sets may not be pruned out in this way, even though there is no object-set with smaller diameter than δ^* in these node-sets.

We use the example in Figure 4.2(a) to illustrate this problem. When $DC = \langle A[15], D[29] \rangle$, suppose at first the node-set $N_1 = \{A[15], B[16], C[22], D[29]\}$ is generated. Then we find the best diameter in N_1 is $dist(a, d)$ where $a \in A[15], d \in D[29]$, and update δ^* to $dist(a, d)$. Next, we use this δ^* to determine whether the following node-set $N_2 = \{A[15], B[16], C[21], D[29]\}$ can be pruned out or not . However, at this point, the lower bound of N_2 is the $Mindist$ between $A[15]$ and $D[29]$. Due to $Mindist(A[15], D[29]) < \delta^*(dist(a, d))$, N_2 cannot meet the condition to be pruned out. That means all the node-set N^* satisfied $LB(N^*) = Mindist(A[15], D[29])$ will not be pruned out. Moreover, if the $dist(a, d)$ is the smallest distance between $A[15]$ and $D[29]$ such that it is the best diameter we can get from the node-sets with $DC = \langle A[15], D[29] \rangle$, we still need to recursively invoke DCC-NL for N^* which are useless.

We will discuss the solutions for the two problems in next section.

4.3 Recursive DCC and tight lower bound

4.3.1 Priority Search Order of Recursive DCC

DCC-NL uses DCC strategy to decide the enumeration order of child node-sets of N. We can summarize DCC strategy into the following three stages:

Stage 1: Generate all the DCs for the child-nodes of N and sort them.

Stage 2: For each DC, generate the all the $(m-2)$-sized sub node-sets for the relevant nodes in the shuttle scope of this DC.

Stage 3: Merge each such the $(m-2)$-sized sub node-sets with the two nodes of DC to a full m-sized node-set.

DCC-NL generate the $(m-2)$-sized sub node-sets using a nested loop method in Stage 2 such that it is difficult to keep the priority order for them (Problem 1). However we can also use DCC strategy in Stage 2 for generating ordered $(m-2)$-sized sub node-sets. Therefore we propose an approach called $RDCC$, which recursively uses DCC strategy in the process of generating node-sets. then we describe $RDCC$ as follow (Suppose the query $Q = \{k_1, k_2, ..., k_m\}$):

$RDCC(N)$

Step 1: For the child-nodes of N, generate all the node-pairs $\langle n_i, n_j \rangle$, where n_i is associated with keyword k_i and n_j is associated with keyword k_j $(k_i, k_j \in Q, k_i \neq k_j)$, as DCs. Then sort these DCs by the ascending order of $Maxdist(DC)$.

Step 2: Pick up a $DC = \langle n_i, n_j \rangle$, from the top of the sorted list of DCs. Then use a variable $CurSet$ as the current node-set generated and $CurSet$ is initialized with $\{n_i, n_j\}$.

Step 3: For the relevant nodes in the shuttle scope of DC, enumerate all node-pair as sub DCs in ascending order. Note the relevant nodes are not associated with the generated keywords in $CurSet$.

Step 4: For each DC in the sorted list of sub DCs, insert the two nodes of it into $CurSet$. Then repeat Step 3 and Step 4.

Step 5: For each repeat will reduce two keywords and increase two nodes in $CurSet$, thus after $\lfloor m/2 \rfloor$ times repetitions, there are only the nodes associated with $0(m$ is even) or 1 (m is odd) keyword left in the shuttle scope. Thus we insert each of these node into $CurSet$ to complete a full m-sized node-set N'. If all the nodes in N' are leaf-nodes then find the smallest diameter δ in N' and update δ^* if $\delta < \delta^*$; otherwise, recursively invoke RDCC(N').

In RDCC approach, we choose the smallest DC in each repetition, thus we can give a higher priority to the node-set with a smaller diameter than DCC-NL. Figure 4.4 shows the search order of RDCC approach for the example in Figure 4.2 . In Figure 4.4, we firstly choose DC $\langle A[15], D[29]\rangle$ and insert it into $CurSet$ such that $CurSet = \{A[15], D[29]\}$. Next we generate all the node-pairs for the nodes associated with keyword B and C in the shuttle of $\langle A[15], D[29]\rangle$. Then the smallest node-pair $\langle B[16], C[22]\rangle$ is choosen firstly and is inserted into $CurSet$ such that $CurSet = \{A[15], D[29], B[16], C[22]\}$. Therefore the node-sets with same $DC = \langle A[15], D[29]\rangle$ can be generated in a priority order. Due to $m = 4$, the m-sized node-set is completed in two repetitions.

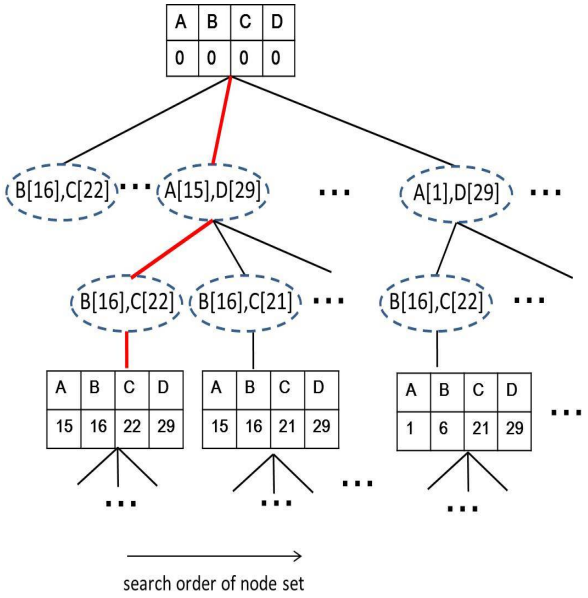


Figure 4.4: Search order of Recursive DCC

4.3.2 Tight lower bound for pruning

In Figure 4.2 (a), we can first generate the node-set $\{A[15], B[16], C[22], D[29]\}$ by RDCC and find the $\delta^* = dist(a, d)$. However the other node-sets with $DC = \langle A[15], D[29]\rangle$ cannot be pruned out by their lower bound (Problem 2). This problem still restricts the pruning efficiency. Thus we use a tighter lower bound of node-set N called $TLB(N)$ to instead of original lower bound $LB(N)$.

[**Obj-Mindist**] Giver two nodes n_1 and n_2, the *Obj-Mindist* between n_1 and n_2 is defined as

$$Obj - Mindist(n_1, n_2) = \min_{o_a \in n_1, o_b \in n_2} \{dist(o_a, o_b)\} \tag{4.2}$$

Obj-Mindist(n_1, n_2) is the minimum distance of object-pair generated from $\langle n_1, n_2\rangle$.

[**Tight Lower Bound(TLB)**] Given a node-set N=$\{n_1, n_2, ..., n_m\}$, TLB of N is defined as follow:

$$TLB(N) = \max_{n_a, n_b \in N} \{Obj - Mindist(n_a, n_b)\} \tag{4.3}$$

$TLB(N)$ is the maximum *Obj-Mindist* between any two nodes in N (Figure. 4.5).

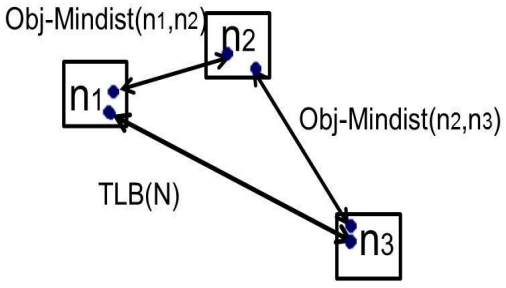


Figure 4.5: TLB of node-set

According to the definitions of LB and TLB, LB is the minimum distance between the two boundary line of two nodes, while TLB is the minimum distance between two objects in the two nodes respectively. Compared to LB , TLB is greater . Thus TLB has more efficient pruning ability than original LB. For example , in Figure 4.2 (a), we first update δ^* to $dist(a, d)$ which is the smallest diameter of node-set $N_1 = \{A[15], B[16], C[22], D[29]\}$.

Then node-set $N_2 = \{A[15], B[16], C[21], D[29]\}$ is generated, we know the TLB of N_2 is the *Obj-Mindist* between $A[15]$ and $D[29]$ $(= dist(a, d))$. Thus we get $TLB(N_2) = \delta^*$, which means there exists no smaller diameter than δ^* in N_2. Hence N_2 can be pruned out.

About the cost of computing the TLB of N, we can adopt a top-down search strategy by using grid structures for each two nodes of N. It is equivalent to the cost of mCK query with $m = 2$ which is much less than the cost when $m > 2$. Thus it can be calculated at high speed.

4.3.3 Object generation in leaf node-set

In Step 5 of RDCC approach , When all the nodes in N' are leaf-nodes, we need to enumerate object-sets for the objects in N', and find the smallest diameter in N'. Here the recursive DCC approach also can be used.

We first enumerate all the object-pairs as DCs and sort them. Then if a DC $\langle o_i, o_j \rangle$ cannot be skipped over, we check if it can be a diameter of an object-set. That means try to find an $(m-2)$-sized object-set whose diameter is less than $dist(o_i, o_j)$ in the shuttle scope of $\langle o_i, o_j \rangle$. If such an $(m-2)$-sized object-set exists , we return $dist(o_i, o_j)$ as the smallest diameter and stop search process. At the stage of finding the $(m-2)$-sized object-set for $\langle o_i, o_j \rangle$, we also recursively enumerate sub DCs and check them. Thus when $\lfloor m/2 \rfloor$-th time recursion ends, we can get an object-set.

Moreover, a property can be used to reduce the times of recursions as follow:

Property: Given an object-pair $\langle A, B \rangle$ as current DC, if it exists an sub DC $\langle C, D \rangle$ in the shuttle scope of $\langle A, B \rangle$ and $\langle C, D \rangle$ satisfies:

1. All the remaining keywords are covered in the shuttle scope of $\langle C, D \rangle$

2. $dist(C, D) \leq \frac{1}{\sqrt{3}} \times dist(A, B)$

Then $\langle A, B \rangle$ is confirmed as a diameter hence we can return $dist(A, B)$ directly (Figure 4.6).

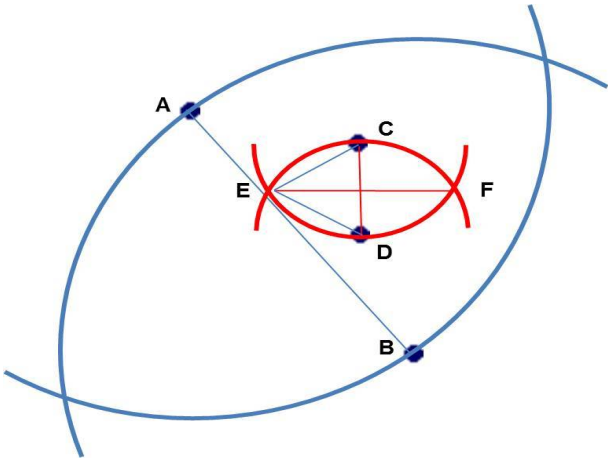


Figure 4.6: Property

4.4 Evaluation of RDCC

Our experiments were conducted over five datasets: three types of synthetic datasets and two real datasets.

The three distributions of synthetic datasets are same as Section 3.5.

1. uniform distribution
2. normal distribution of $\sigma = \frac{1}{4}R$
3. normal distribution of $\sigma = \frac{1}{8}R$

For each keyword, we generated 1000 data points and prepared a separate data-file to save them. The total keywords are up to 100, thus there are 100,000 (= 1,000 $\times$ 100) data points in all for each dataset.

In addition we use Flickr data and Twitter data as two real datasets. We collected 46,303 photo records from Flickr in Tokyo area. There are 19,425 unique tags in these photo records. Each record is associated with 1 to 73 tags that can be viewed as keywords of data-point. We also collected 164,175 tweets data as Twitter dataset and there are 300,301 tags (keywords) in these texts of tweets data.

We use two collections of MongoDB to store these data points for Flickr dataset and Twitter dataset. For each collection we prepared keyword indices at first. When a query is

given, we load necessary data and create m grids on demand. The capacity of leaf-node in each grid is 100 objects, and each internal-node has 100 child-nodes.

To evaluate the proposed approach in this chapter, we test three top-down algorithms as follow :

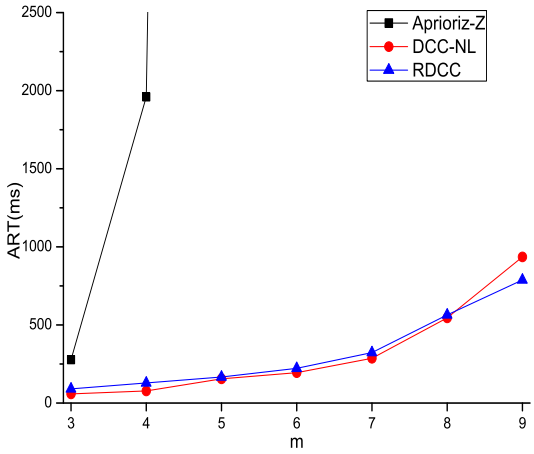
- Apriori-Z: Zhang's Apriori-based algorithm
- DCC-NL: the DCC strategy with Nested Loop method
- RDCC: the recursive DCC approach with tight lower bound

Figure 4.7 shows the ART vs. the number of keywords m in the above five datasets.

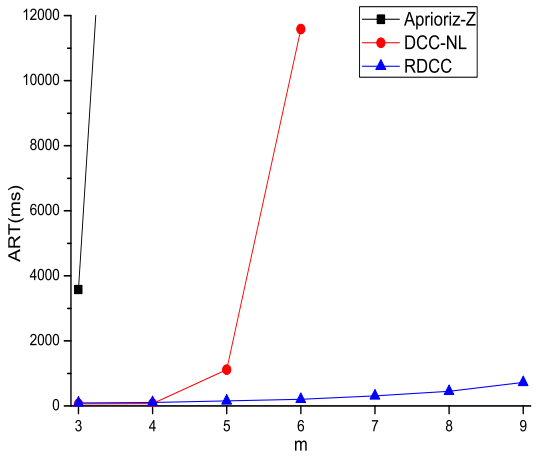
In the case of uniform distribution (Figure 4.7(a)), we can see the Apriori-Z algorithm provide a stable performance when m increase. This distribution gathers necessary objects of all keywords into one smaller grid-cell, thus Apriori-Z can easily find a small δ^* and use it to cut down almost all the node-sets. On the other hand, because the enumerated node-sets grows faster as m increases the performance of DCC-NL are degraded quickly when m gets larger. However RDCC uses TLB for pruning, thus it can cut down a lot of node-sets that could not prune in DCC-NL to curb the degradation of search performance. We can see RDCC also keeps a stable search speed close to the Apriori-Z.

Figure 4.7(b) is the case of the normal distribution of $\sigma = \frac{1}{4}R$. Though this distribution is lightly skewed , it still exists a result of object-set with small diameter δ^*_{opt} for most of queries. (when $m = 9$, the average of $\delta^*_{opt} = 0.02R$). But the result is not necessarily gathered into one cell. Therefore the generation of node-sets in Apriori-Z becomes heavy as m increases. In this case however, we can see the DCC based algorithms worked well.

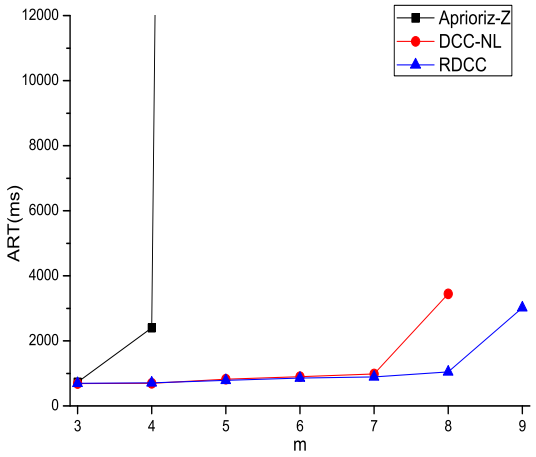
With the distribution become highly skewed. Figure 4.7(c) shows that RDCC outperforms the other two algorithms. In this case, the object-sets with small diameter may not exist. Thus Apriori-Z cannot find a smaller initial value, and have to enumerate vast amount of node-sets. DCC-NL also has more nodes in the shuttle of DCs which lead to more node-sets. Moreover, due to the problem 2 in section 4.2.2, these node-sets cannot be pruned efficiently. Hence DCC-NL also has a poor performance when m get larger. In contrast, RDCC has a better priority order than DCC-NL and solved problem 2 by using TLB, thus keeps a good performance.



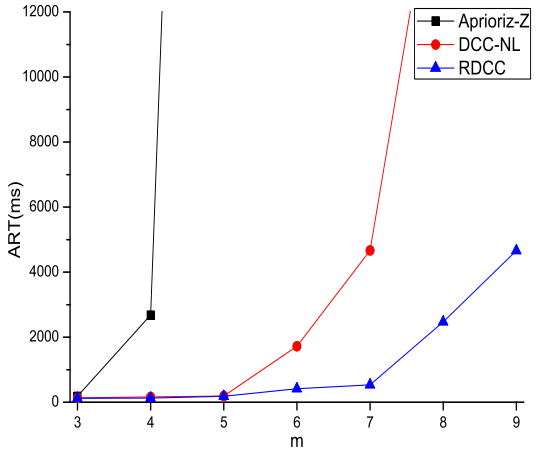
(a) Synthetic dataset of uniform distribution



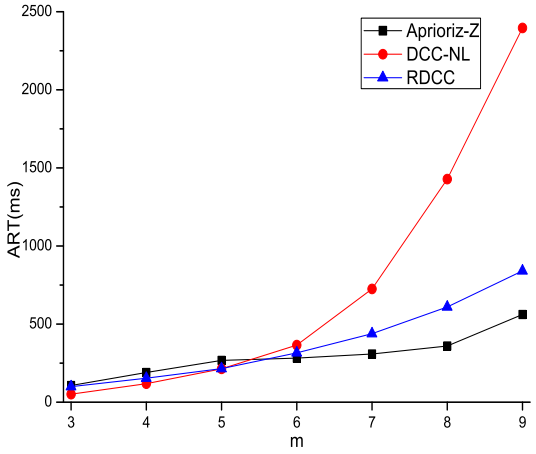
(b) Synthetic dataset of normal distribution with $\sigma = \frac{1}{4}R$



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



(d) Flickr dataset



(e) Twitter dataset

Figure 4.7: performance comparison

In the case of real datasets of Flickr(Figure 4.7(d)) and Twitter (Figure 4.7(e)), we can see the average performance of RDCC outperforms the other two algorithms. In RDCC, we can find a better δ^* at an earlier stage and increase the lower bound of node-set, thus RDCC can provide more stable search efficiency than Apriori-Z and DCC-NL.

4.5 Summary

In this chapter, we first reviewed DCC-NL algorithm for mCK query problem, and discussed two problems in DCC-NL. One problem is that the enumeration of node-sets derived from same DC cannot keep a priority order by using a nested loop search method. The other problem is the original lower bound in DCC-NL failed to prune out part of node-sets even though we know these node-sets are unnecessary enumerating. Therefore, we proposed a search approach call RDCC. RDCC improved the DCC strategy in a recursive way such that all the node-sets can be enumerated in a priority way. Moreover, in RDCC, we enhanced the original lower bound by using a tighter lower bound (TLB) of node-set which can effectively improve the pruning ability .

In our experiments, we compared RDCC algorithm with preceding DCC-NL algorithm. The results in various datasets demonstrated that RDCC can efficiently inhibit the exponential growth of CPU overhead caused by the distributions in DCC-NL. Thus we can say the RDCC significantly improved search efficiency of DCC-NL.

In these search algorithms including Apriori-Z, DCC-NL and RDCC, we need to generate both node-sets and object-sets in the top-down process. Thus the NP property exists both node-sets and object-sets enumeration. Theoretically, there exists a risk that we may generate exponential node-sets in these algorithms. Though RDCC can reduce this risk, but cannot completely avoid it. Furthermore, our assumption that create a grid partitioning for each keyword may increase the risk especially when the objects have multiple keywords.

Chapter 5

Pairwise Expansion as a new top-down search without node-set generation

5.1 New Top-down Search Strategy

5.1.1 Objective of this chapter

Up to now, the top-down search approaches we have discussed all need to enumerate node-sets such that search efficiency may become unstable. Thus in this chapter, we consider a technique that can eliminate this unstable factor of generating node-sets in the top-down process.

5.1.2 Basic idea

According to the definition of mCK queries, we know that the mCK query is essentially an combinatorial optimization problem, which is to find the optimal object-set with minimum diameter among all the object-sets relevant to query keywords. The number of these object-sets is in an exponential order of data size and this problem has been proved to be an NP-hard problem in [9]. So far all of the algorithms for mCK query problem we have discussed follow the same underlying principle:

- Use top-down exploration approaches taking advantage of hierarchical indices.

- First enumerate sets of internal/leaf-nodes of each level of a hierarchical index, then prune unnecessary node-sets to reduce generation of object-sets.
- For the leaf node-set which cannot be pruned out, exhaustively examine all possible object-sets.

However the hardness also exists in the node-set enumeration, and pruning of node-sets becomes very unstable on different data distributions.

Therefore, we propose a new top-down approach called *Pairwise Expansion*, which solves mCK problem in two stages. We know the diameter of an object-set is an object-pair (a pair of two objects). Thus in the first stage we enumerate object-pairs and give a higher priority to a closer object-pair. We also know if an object-pair is the diameter of an object-set, then the object-set must exist in a small shuttle area of the object-pair. Thus in the second stage we expand the object-pair into object-sets in the shuttle area to test if it is a diameter of any correct object-sets satisfying all query keywords. Finally we return the closest object-pair which pass the test. In the first stage of object-pair enumeration we use top-down exploration of node-pairs (a pair of two nodes) to find the closer object-pairs, thus we can avoid the risk of generating exponential node-sets and still keep the strong pruning ability. And in the second stage of object-pair check we adopt a recursive strategy like RDCC and circle check method to reduce the total number of generated object-sets.

5.2 New setting of an On-the-fly quad-tree

In chapter 3 and 4, we created a grid partitioning for each keyword. However when one object o is associated with l keywords, o will be existed in l different grids. This add to the actual amount of data retrieved.

Therefore we employ an on-the-fly quad-tree structure to compute mCK queries. When Q, as an mCK query keywords, is given, we only load necessary objects associated with Q from one or multiple datasets and then create a quad-tree for these objects. Next we start the search process from this quad-tree.

Figure 5.1 is an example of on-the-fly quad-tree when four keywords $Q = \{A, B, C, D\}$ are given. In this quad-tree each internal node (node 1) has exactly four children (node 1-1,1-2,1-

3,1-4). Furthermore, each node is given an additional MBR which is tightly bound with all objects in this node. These MBRs are used to estimate the distance between nodes instead of original cell's boundary rectangles. We also append an additional *keyword bitmap* in each node to summarize the keyword information in the same way as bR*-tree. For example in the node 4-4, the bitmap = 0111 says that this node contains objects only associated with keyword B, C, D and not A. The additional MBR and keyword bitmap are produced when we create a new node. Thus this process does not need much extra time.

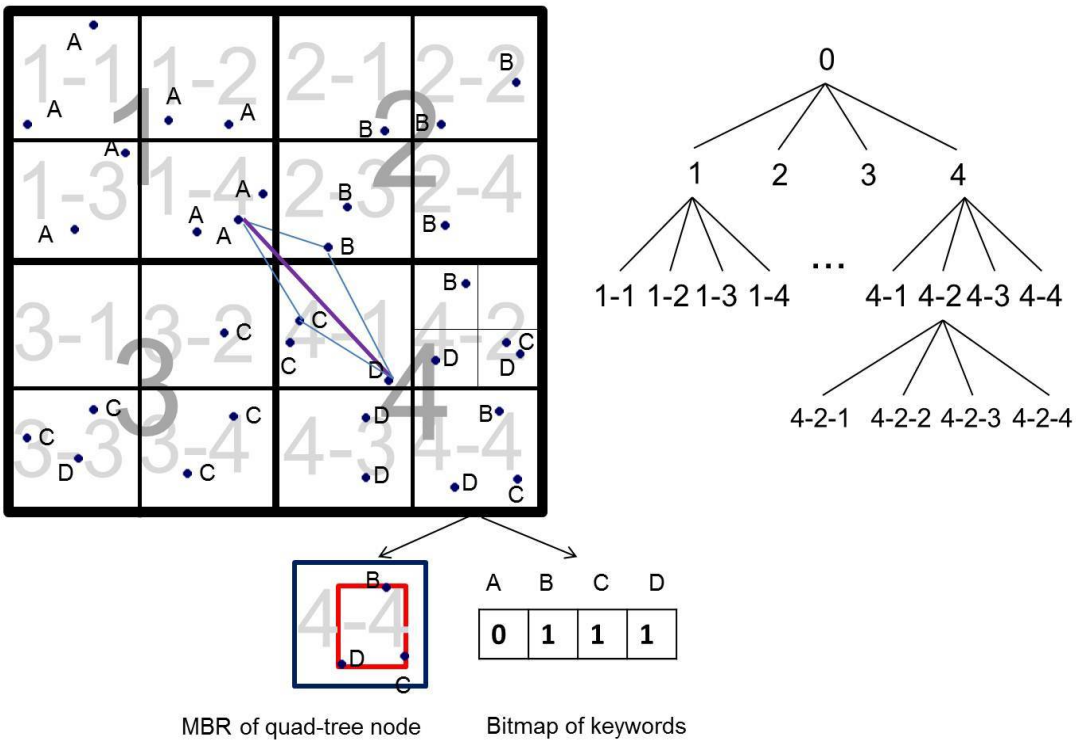


Figure 5.1: on-the-fly quad-tree

5.3 Pairwise Expansion method

5.3.1 Overview

Given a data set with n objects, the number of possible object-sets is $O(n^m)$. Because the diameter of an object-set is determined by only two objects, the number of possible diameters for all the object-sets is just $O(n^2)$. From this property it is natural to think that we must

find the closest object-pair p among all the object-pairs such that there exists an object-set O satisfying all given keywords and the diameter of O is p. Thus our idea is to accomplish the search process in two stages: First we enumerate object-pairs in ascending order of their distances. Next, we pick up each 'closer' object-pair and expand it into some object-sets and check if it can be the diameter of any of object-sets satisfying all given keywords. Here we call the object-set that satisfy all given keywords a 'correct' object-set. This approach is called *Pairwise Expansion*.

5.3.2 Stage1: Top-down Generation of Object-Pair

In this stage we need to generate object-pairs for all objects in the dataset. In order to give a higher priority to a closer object-pair, we employ an algorithm according to the method to Closest Pair Query (CPQ) problem [11] . We depict the algorithm of CPQ problem first by using our quad-tree structure.

The algorithm of CPQ problem in [11] uses a branch-and-bound strategy of top-down search. There are two global variables in the algorithm :

1. δ^*: minimum distance found so far as an upper bound of the true minimum value;

2. *Queue*: a queue holds node-pairs in ascending order according to $Mindist$ of these node-pairs.

The algorithm of [11] can be written as follows:

Algorithm: CPQ

Step 1: Initialize δ^* with ∞, and add the root node-pair ($\langle root, root \rangle$) into the *Queue*.

Step 2: Pull out the head node-pair $\langle n_i, n_j \rangle$ of *Queue*. If $Mindist(n_i, n_j) \geq \delta^*$, then return δ^*.

Step 3: If both n_i and n_j are leaves, calculate the distance of each object-pair $\langle o_i, o_j \rangle$ ($o_i \in n_i, o_j \in n_j$). If $dist(o_i, o_j) < \delta^*$, then update δ^* by $dist(o_i, o_j)$.

Step 4: If either n_i or n_j is an internal node, add all the child-node pairs into *Queue*. If $Maxdist(n_i, n_j) < \delta^*$, then update δ^* by $Maxdist(n_i, n_j)$.

Step 5: If *Queue* is empty then return δ^*; otherwise go to Step 2.

In CPQ algorithm the result is the simply minimum 'distance' of object-pair. However the result of mCK is the minimum 'diameter'. Thus, even if we get an object-pair whose distance is smaller than δ^*, we cannot use it to update δ^* directly. It is because we do not know if it can be a diameter of a 'correct' object-set. Furthermore, in Step 4, $Maxdist(n_i, n_j)$ is not an upper bound of a diameter. It's because a 'correct' object-set O including $\langle o_x, o_y \rangle (o_x \in n_i, o_y \in n_j)$ may have a diameter greater than $dist(o_x, o_y)$. That means δ^* will not be updated until down to leaf node-pairs. Thus the size of *Queue* may be too large and be very costly to maintain it. We will make this algorithm to be suitable for mCK problem.

Given an object-pair $\langle o_1, o_2 \rangle$, if it is the diameter of an object-set O, then $dist(o_1, o_2)$ is larger than any distance of object-pair $\langle o_a, o_b \rangle (\forall o_a, o_b \in O)$. Thus all the objects of O must exist in the area of $Shuttle(\langle o_1, o_2 \rangle)$ (Figure 5.2(a)).

Here $Shuttle(\langle o_1, o_2 \rangle)$ is defined as the set of all objects o' such that o' must satisfy that $dist(o', o_1) \leq dist(o_1, o_2)$ and $dist(o', o_2) \leq dist(o_1, o_2)$.

Therefore we have the property that if the objects in $Shuttle(\langle o_1, o_2 \rangle)$ do not satisfy all given keywords, then $\langle o_1, o_2 \rangle$ will never be a diameter of any 'correct' object-set, and we can skip this object-pair.

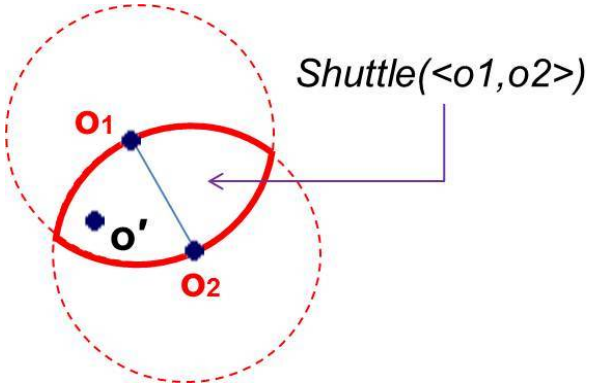
Similarly, given a node-pair $\langle n_1, n_2 \rangle$, if we know that no object-pair belonging to $\langle n_1, n_2 \rangle$ can be a diameter of a 'correct' object-set, then we can skip $\langle n_1, n_2 \rangle$. Hence we define the area of $Shuttle(\langle n_1, n_2 \rangle)$ for node-pair $\langle n_1, n_2 \rangle$, as follows (Figure 5.2 (b)).

$Shuttle(\langle n_1, n_2 \rangle)$ is defined to be the set of all possible objects o' such that o' must satisfy that $Mindist(o', n_1) \leq Maxdist(n_1, n_2)$ and $Mindist(o', n_2) \leq Maxdist(n_1, n_2)$.

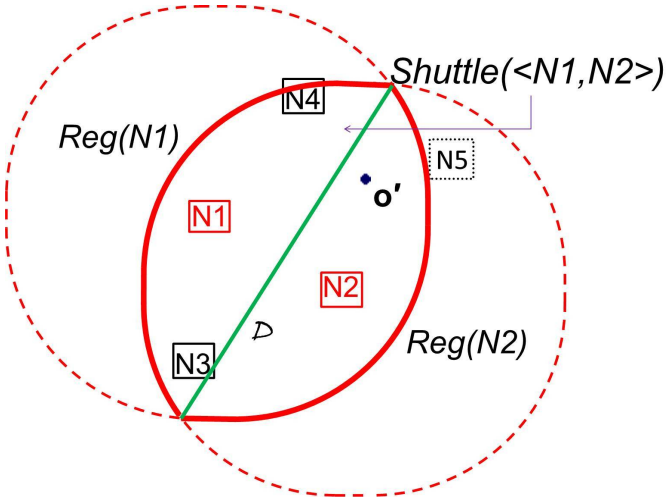
Here $Mindist(o', n_1)$ is the minimum distance between object o' and MBR of n_1. In Fig 5.2 (b) , we can see that the area of $Shuttle(\langle n_1, n_2 \rangle)$ is formed by the intersection of two regions by definition. We denote these two regions by $Reg(n_1)$ and $Reg(n_2)$, respectively.

According to the definition, we know that $Shuttle(\langle n_1, n_2 \rangle)$ covers all the shuttle areas of object-pairs $\langle o_i, o_j \rangle$ such that$\forall o_i \in n_1, \forall o_j \in n_2$. Thus if $Shuttle(\langle n_1, n_2 \rangle)$ does not satisfy all the query keywords Q, then $Shuttle(\langle o_i, o_j \rangle)$ for each object-pair $\langle o_i, o_j \rangle$ cannot satisfy Q, and thus we can skip $\langle n_1, n_2 \rangle$.

On the other hand, if the $Shuttle(\langle n_1, n_2 \rangle)$ contains all the keywords Q, then there exists



(a) shuttle area of object-pair



(b) shuttle area of node-pair

Figure 5.2: shuttle area

at least one 'correct' object-set O in the shuttle area such that the diameter of $O \leq$ the maximum distance between any two points on the boundary of the shuttle area. The maximum distance of shuttle area is difficult to calculate directly, but we can know its upper bound $\mathcal{D}$.

In Fig 5.2 (b) , we can see $\mathcal{D}$ is no larger than the maximum distance existing in $Reg(N_1)$ and that in$Reg(N_2)$. We know that the maximum distance in $Reg(N_1)$ is $Diag(N_1) + 2 \times Maxdist(N_1, N_2)$ ($Diag(N_1)$ is the diagonal distance of MBR of N_1), the maximum distance in $Reg(N_2)$ is $Diag(N_2) + 2 \times Maxdist(N_1, N_2)$. Thus we can set $\mathcal{D}$ as follow:
$\mathcal{D} = Min(Diag(N_1), Diag(N_2)) + 2 \times Maxdist(N_1, N_2)$.
$\mathcal{D}$ is also an upper bound of a diameter. Thus we can use it to update δ^*.

To determine the keywords contained in $Shuttle(\langle n_1, n_2 \rangle)$, we employ range search of [13] . The deterministic process is recursive. Starting from the root node of quad-tree, we check

the positional relation between a current node n^* and $Shuttle(\langle n_1, n_2 \rangle)$. If n^* is an internal node of the shuttle (e.g., N_3 in Figure 5.2 (b)), then we add all the keywords of n^* by using the bitmap of n^*; if n^* is an external node of the shuttle (e.g., N_5 in Figure 5.2 (b)), then n^* is skipped; if n^* is crossed by the boundary of the shuttle (e.g., N_4 in Figure 5.2 (b)), then we recursively check the children nodes or objects in n^*. Lastly, when all the nodes and objects in the shuttle area are checked, we can determine the keywords in $Shuttle(\langle n_1, n_2 \rangle)$.

In summary, our algorithm of PE is written based on the algorithm of CPQ problem as follows:

Algorithm: Pairwise Expansion (PE)

// Input: *root*: root node of quad-tree; Q: m given keywords.

// Output: δ^*_{opt}: the minimal diameter.

// Variables: *Queue*: a sorted list of node-pair $\langle n_i, n_j \rangle$, sorted in the ascending order of $Mindist(n_i, n_j)$; δ^*: current minimum diameter.

Step 1: Set the node-pair $\langle root, root \rangle$ into *Queue*. Initialize δ^* with ∞.

// This is a (do-while) loop from Step 2 to Step 5, in order to repeat pulling out the current smallest node-pair from *Queue*.

Step 2: Dequeue node-pair $\langle n_i, n_j \rangle$ from *Queue*. Do:

2-1: If the $Mindist(n_i, n_j) \geq \delta^*$, then return δ^*.

2-2: If $Shuttle(\langle n_i, n_j \rangle)$ does not contain all the keywords of Q, then goto Step 5.

2-3: Calculate the upper bound $\mathcal{D}$ of $Shuttle(\langle n_i, n_j \rangle)$. If $\mathcal{D} < \delta^*$, update δ^* by $\mathcal{D}$.

2-4: If either n_i or n_j is not a leaf-node, goto Step 4. Else goto Step 3.

Step 3: // (both n_i and n_j are leaf-nodes.)

For each object-pair $\langle o_a, o_b \rangle$ in $\langle n_i, n_j \rangle$ ($o_a \in n_i, o_b \in n_j$) in the ascending order of $dist(o_a, o_b)$, do:

3-1: If $dist(o_a, o_b) \geq \delta^*$, then break this for-loop and goto Step 5.

3-2: Check if $\langle o_a, o_b \rangle$ is a diameter of a 'correct' object-set which exists in $Shuttle(\langle o_a, o_b \rangle)$.

3-3: If the check is true , update δ^* by $dist(o_a, o_b)$ and goto Step 5.

3-4: If the check is false , continue this for-loop.

(//the next $\langle o_a, o_b \rangle$ is tested.)

Step 4: If either n_i or n_j is an internal node, generate all the child-node pairs into the *Queue*.

Step 5: Return δ^* if *Queue* is empty. Otherwise goto Step 2.

5.3.3 Stage2: Check of Object-Pair

Next, we must formalize the 'check' action of Step 3. To do so, we firstly propose RecursiveCheck(o_a,o_b). After that, we propose SophisticatedCheck(o_a,o_b) as the final version.

RecursiveCheck

At (3-2) of PE, to check if an object-pair $\langle o_1, o_2 \rangle$ (denoted as $\langle o_a, o_b \rangle$ in PE's (3-2)) can be a diameter of a 'correct' object-set, we will expand $\langle o_1, o_2 \rangle$ into object-sets by using the objects contained in $Shuttle(\langle o_1, o_2 \rangle)$. Once we find at least one 'correct' object-set whose diameter is $dist(o_1, o_2)$, then the check of (3-2) succeeds, and we goto (3-3).

Firstly we use a recursive way to do the checking process. In order to check if $\langle o_1, o_2 \rangle$ is the diameter of a 'correct' object-set which covers all the query keywords Q, we need to confirm whether or not there exists an object-set O' in $Shuttle(\langle o_1, o_2 \rangle)$ such that the diameter of $O' \leq dist(o_1, o_2)$ and O' covers all the rest of keywords of $\{Q - o_1.\psi - o_2.\psi\}$. This is because, if such O' exists, then $\langle o_1, o_2 \rangle$ can be determined as the diameter of the 'correct' object-set which is formed by the combination of $\langle o_1, o_2 \rangle$ and O'. Otherwise, $\langle o_1, o_2 \rangle$ cannot be the diameter of any 'correct' object-set. Here, O' can be viewed as a sub object-set of a 'correct' object-set, and the diameter of O' is an object-pair in $Shuttle(\langle o_1, o_2 \rangle)$. Therefore, in order to confirm or deny the existence of O', it is sufficient to test each object-pair $\langle o_x, o_y \rangle$ ($dist(o_x, o_y) \leq dist(o_1, o_2)$) in $Shuttle(\langle o_1, o_2 \rangle)$ to see whether or not $\langle o_x, o_y \rangle$ can be the diameter of a sub object-set which covers the rest of keywords of $\{Q - o_1.\psi - o_2.\psi\}$. Thus we repeat this test recursively. We perform this recursive process until one O' is found

(succeed, $\langle o_1, o_2 \rangle$ is a diameter) or all the object-pairs are tested and no such O' exists (fail, $\langle o_1, o_2 \rangle$ is not a diameter).

We explain our idea in an example of Figure 5.3. In Figure 5.3, given a query $Q = \{A, B, C, D, E, F, G, H\}$, consider that at the step 3 of PE, we check that $\langle o_1, o_2 \rangle$ is the diameter of some 'correct' object-set. If we can find a sub object-set O' of O (suppose that $O' = \{o_3, o_4, o_5\}$ in Figure 5.3) in $Shuttle(\langle o_1, o_2 \rangle)$, such that O' must cover all the rest of keywords of $\{Q - o_1.\psi - o_2.\psi\}(= \{D, E, F, G, H\})$ and the diameter of $O' = dist(o_3, o_4) \leq dist(o_1, o_2)$, then we can determine that $\langle o_1, o_2 \rangle$ is the diameter of a 'correct' object-set $O = \{o_1, o_2, o_3, o_4, o_5\}$ by combining $\langle o_1, o_2 \rangle$ with O'. To examine O', we can pick up $\langle o_3, o_4 \rangle$ and test to see that $\langle o_3, o_4 \rangle$ is the diameter of O' with a sub object-set $\{o_5\}$ of O' such that $\{o_5\}$ covers the rest of keywords ($\{G, H\}$) which are not included in o_3 and o_4. Here $\{o_5\}$ only has one object, thus the recursive process is ended and the test of $\langle o_1, o_2 \rangle$ succeeded.

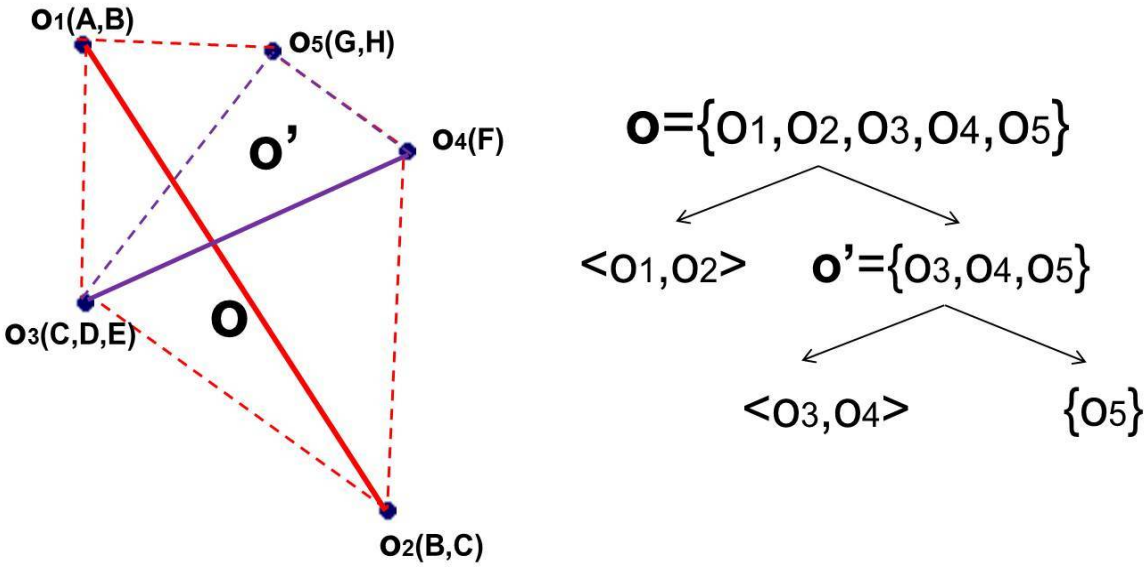


Figure 5.3: division of object-set

We summarize this strategy as algorithm $RecursiveCheck(o_a, o_b)$. $RecursiveCheck(o_a, o_b)$ is recursive and has two return-values: true ($\langle o_a, o_b \rangle$ is diameter) or false(otherwise). In the algorithm, let q be the set of the keywords ($\subseteq Q$) whose tests have been finished. q is initialized to $\emptyset$. The algorithm is described as follows:

Algorithm: RecursiveCheck(o_a,o_b)

// Input: Q: m given keywords.

// Output: true or false.

// Variables: q: the set of keywords ($\subseteq Q$) which has been tested.

// according to q, we can know the rest of keywords ($Q - q$) that a sub object-set O' must contain.

Step 1: Add the keywords of o_a and o_b to q. If $q = Q$ or there exists one object o^* in $Shuttle(\langle o_a, o_b \rangle)$ such that $q \cup o^*.\psi = Q$, then return true.

Step 2: Check if $Shuttle(\langle o_a, o_b \rangle)$ satisfies keywords $Q - q$. If the check fails, return false.

Step 3: Generate all the object-pairs $\langle o_i, o_j \rangle$ such that $o_i, o_j \in Shuttle(\langle o_a, o_b \rangle)$ and $dist(o_i, o_j) \leq dist(o_a, o_b)$. Then sort them in ascending order of $dist(o_i, o_j)$ into d.

Step 4: For each object-pair $\langle o_i, o_j \rangle$ in d in the sorted order, do:

4-1: Invoke $RecursiveCheck(o_i, o_j)$.

4-2: If $RecursiveCheck(o_i, o_j) = true$, then return true;

4-3: If $RecursiveCheck(o_i, o_j) = false$, continue this loop.

(//the next object-pair is picked up in d.)

Step 5: When all the object-pairs in d are checked, then no object-pair can be the diameter of any 'correct' object-set, return false.

In this algorithm, we generate an object-set by repeated iteration of a set of object-pairs. And we skip all object-pairs if their shuttle areas lack necessary keywords.

SophisticatedCheck

Next we propose a new check procedure called SophisticatedCheck(o_a,o_b).

In section 2.4.3, we briefly introduced Guo's approach that uses the minimum covering circle MCC of an object-set to find an approximate solution. Our idea is considered along the proof process of the approximate property using MCC in [9]. Here we use Figure 5.4 to illustrate this idea.

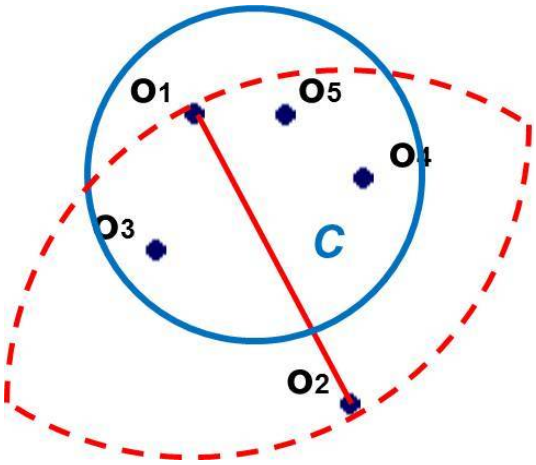


Figure 5.4: boundary circle

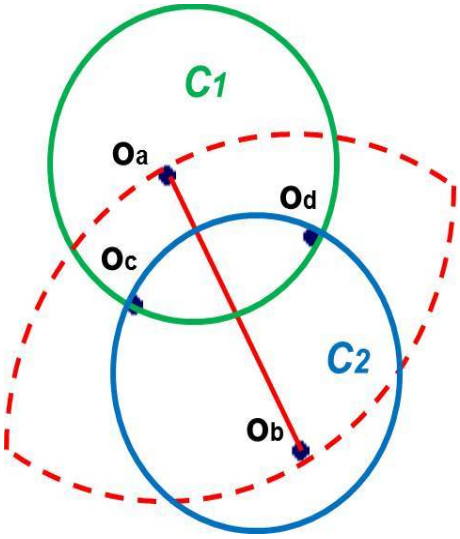


Figure 5.5: boundary circle check

When we check the object-pair $\langle o_1, o_2 \rangle$ in Figure 5.4, if there is a set of objects O' (such as $O' = \{o_3, o_4, o_5\}$) in $Shuttle(\langle o_1, o_2 \rangle)$ such that O' covers all the rest of keywords $(Q - o_1.\psi - o_2.\psi)$ and O' can be enclosed in a circle C whose diameter $= dist(o_1, o_2)$, then the diameter of $O' \leq dist(o_1, o_2)$. Thus we can generate a 'correct' object-set by a combination of $\langle o_1, o_2 \rangle$ and O' and the diameter of the 'correct' object-set is $dist(o_1, o_2)$.

Based on this idea, in the step 3 of RecursiveCheck(o_a,o_b), when an object-pair $\langle o_c, o_d \rangle$ is recursively generated, we can make two circles C_1 and C_2 that the diameters of both C_1 and C_2 are $dist(o_a, o_b)$, and that o_c and o_d are on the boundary of these two circles. This situation is shown in Figure 5.5. Then we test each circle to examine whether or not it contains the rest keywords $Q - q$. This test is called 'the circle test'. If a circle C is determined to contain the rest keywords, then we say that the circle test of C succeeds; otherwise, this circle test fails. Apparently, the following rule 1 holds:

Rule 1: If either C_1 or C_2 of an object-pair succeeds in the circle test, we can safely stop recursion and return true. It is because there exists an sub object-set O' in C_1 or C_2 which covers all the rest of keywords and the diameter of $O' \leq dist(o_a, o_b)$.

Next, assume the case where we have examined the circle tests for all object-pairs in $Shuttle(\langle o_a, o_b \rangle)$ and where we cannot find any successful circles. Under this situation, the following rule 2 holds:

Rule 2: If the circle tests for all object-pairs fail, then in Step 4 of algorithm RecursiveCheck, we have no need to recursively check any object-pair $\langle o_i, o_j \rangle$ which satisfies $dist(o_i, o_j) \leq \frac{\sqrt{3}}{2} dist(o_a, o_b)$.

This is because $dist(o_i, o_j)$ can not be the diameter of any sub object-set O' covering all the rest of keywords. We use the following lemma to illustrate it.

lemma: The rule 2 never misses the answer.

proof: We use proof by contradiction. Let the hypothesis be that, (1) there is no object-pair that succeeds in the circle test of rule 1 and that (2) there exists a sub object-set O' whose diameter $\delta(O') = dist(o_x, o_y)$ such that O' covers all the rest of query keywords and $dist(o_x, o_y) \leq \frac{\sqrt{3}}{2} dist(o_a, o_b)$. Then, we can know the following inequation, according to Guo's theorem 4 of [9] which is described as inequation (2.1) in section 2.4.3.

$$\phi(MCC_{O'}) \leq \frac{2}{\sqrt{3}} \delta(O'). \tag{5.1}$$

Due to our assumption of (2), we also know that

$$\delta(O') = dist(o_x, o_y) \leq \frac{\sqrt{3}}{2} dist(o_a, o_b). \tag{5.2}$$

Thus we can get

$$\phi(MCC_{O'}) \leq \frac{2}{\sqrt{3}} \times dist(o_x, o_y) \leq \frac{2}{\sqrt{3}} \times \frac{\sqrt{3}}{2} \times dist(o_a, o_b) = dist(o_a, o_b) \tag{5.3}$$

The relationship (5.3) says that because of $\phi(MCC_{O'}) \leq dist(o_a, o_b)$, O' can be enclosed by a larger circle whose diameter is $dist(o_a, o_b)$. Thus there must be two objects o_m and o_n in O' such that the circle test of $\langle o_m, o_n \rangle$ succeeds in rule 1. This is a contradiction. Therefore, any object-pair $\langle o_i, o_j \rangle$ which can be the diameter of such a sub object-set O' must satisfy $dist(o_i, o_j) > \frac{\sqrt{3}}{2} dist(o_a, o_b)$. ■

As an example of Figure 5.6, if there exists $O' = \{o_3, o_4, o_5, o_6\}$ whose diameter is $dist(o_5, o_6)$ in $Shuttle(\langle o_1, o_2 \rangle)$ and $dist(o_5, o_6) \leq \frac{\sqrt{3}}{2} dist(o_1, o_2)$, then the diameter of $MCC_{O'}$ is less than $dist(o_1, o_2)$. Thus O' should be successfully checked in the circle C of object-pair

$\langle o_3, o_6 \rangle$.

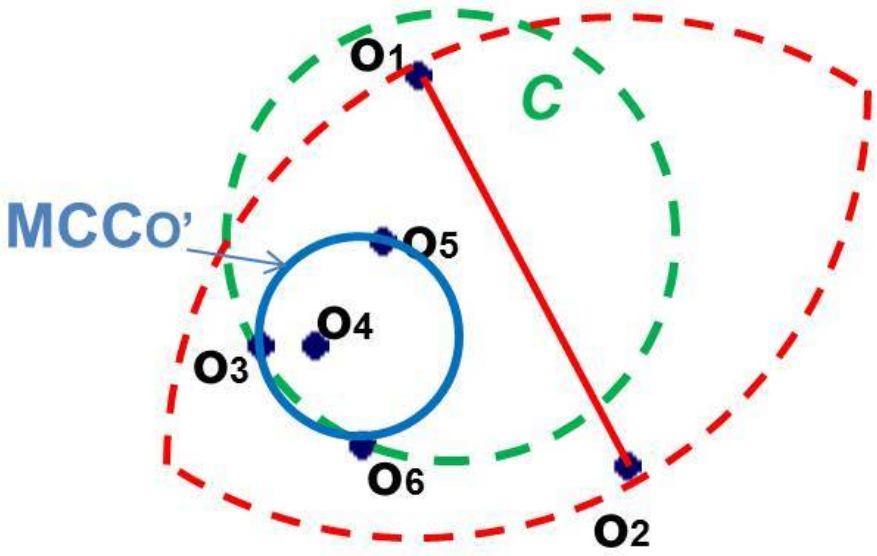


Figure 5.6: Successful example of circle test

The circle test can speed up the process of object-pair check. If it succeeds, then we can return true directly due to the rule 1. Even if it fails, a part of object-pairs for recursion can be cut down due to the rule 2.

In summary, we add the process of circle test into RecursiveCheck algorithm and rewrite it as SophisticatedCheck as follow.

Algorithm: SophisticatedCheck(o_a,o_b)

// Input: Q: m given keywords.

// Output: true or false.

// Variables: q: the set of keywords ($\subseteq Q$) which has been tested.

// according to q, we can know the rest of keywords ($Q - q$) that a sub object-set O' must contain.

Step 1: Add the keywords of o_a and o_b to q. If $q = Q$ or there exists one object o^* in $Shuttle(\langle o_a, o_b \rangle)$ such that $q \cup o^*.\psi = Q$, then return true.

Step 2: Check if $Shuttle(\langle o_a, o_b \rangle)$ satisfies keywords $Q - q$. If the check fails, return false.

Step 3: Generate all the object-pairs in $Shuttle(\langle o_a, o_b \rangle)$, and examine the two circles C_1 and C_2 of rule 1 for every object-pair. If there exists a circle satisfying keywords $Q - q$, then return true; otherwise we choose all object-pairs $\langle o_i, o_j \rangle$ such that $\frac{\sqrt{3}}{2} \times dist(o_a, o_b) < dist(o_i, o_j) \leq dist(o_a, o_b)$ and sort them into d.

Step 4: For each object-pair $\langle o_i, o_j \rangle$ in d in the sorted order, do:

4-1: Invoke $SophisticatedCheck(o_i, o_j)$.

4-2: If $SophisticatedCheck(o_i, o_j) = true$, then return true;

4-3: If $SophisticatedCheck(o_i, o_j) = false$, continue this loop.

(//the next object-pair is picked up in d).

Step 5: When all the object-pairs in d are checked, then no object-pair can be the diameter of any 'correct' object-set, return false.

5.4 Preliminary evaluation

5.4.1 Experimental Set-up

In this section we also evaluate our algorithm over a synthetic dataset and real datasets.

As the uniform dataset in chapter 3 and 4, 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, thus the total data is 100,000. These data points are randomly generated and stored in a data file. When a query of m-keywords is given, we build a quad-tree only for the necessary data points after reading the data-file.

We also employ a real dataset which collects 399,754 photo records from Flickr in Tokyo area. Each record is associated with from 1 to 75 tags that can be viewed as keywords of data-point. There are 89,277 unique tags as a whole. We stored these records in MongoDB, and we built index of tags for them in advance. When a query is given, we load necessary data by using query statements of MongoDB.

As a quad-tree, we defined that each node is divided when the number of data points is greater than 30.

As a comparison, we test four top-down algorithms as follow :

- Apriori-Z: Zhang's Apriori-based algorithm
- DCC-NL: the DCC strategy with Nested Loop method in chapter 3
- RDCC: the recursive DCC approach with tight lower bound in chapter 4
- PE: Pairwise Expansion method in this chapter

We implemented the Zhang's Apriori-based algorithm denoted by Apriori-Z under our quad-tree. And for each of nodes we also prepared bitmap and keyword MBR information like bR*-tree, and the other set is the same. the other algorithms are DCC-NL in chapter 3 and RDCC in chapter 4. In its implementation we use m pieces of hierarchical grid-partitionings for m given keywords. The maximum capacity and fan-out of each grid-cell are both 100. Note that both quad-tree and grid-partitioning are also created dynamically when a query is given.

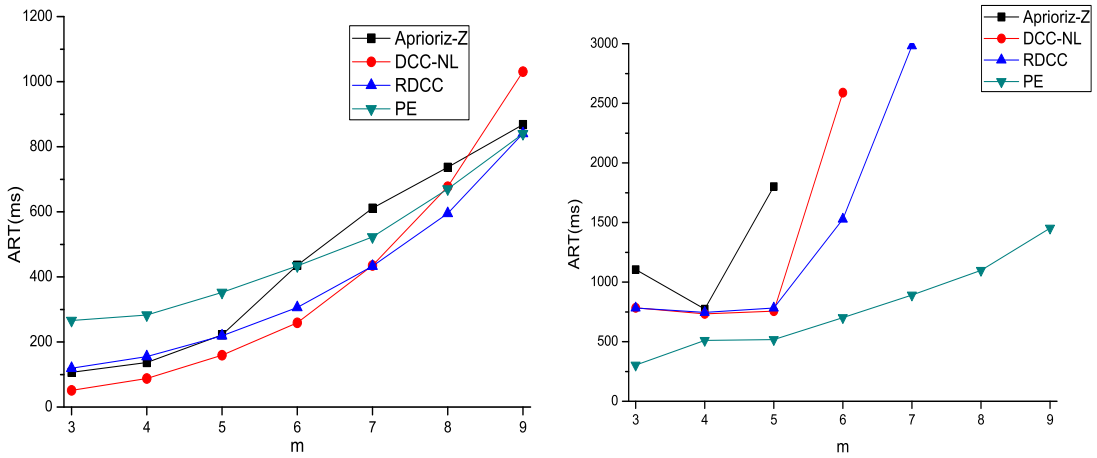
All the algorithms are implemented in Java with version 1.7 on a machine with an Intel(R) Xeon(R) CPU of 2.6GHz and 12GB of RAM. The performance measure is the average response time (ART). The ART includes all time of data access, tree creation and search execution. For each m, we use 50 sets of query keywords with size m and take ART over them.

5.4.2 Experimental evaluation

Efficiency

Figure 5.7(a) is ART vs. the number of keywords m in the synthetic dataset.

In Figure 5.7(a), we vary the number of keywords and evaluate the performance of the three algorithms. We can see all the algorithms have good performance when m is small, because the uniform distribution gathers necessary objects of all keywords into a small area. Thus it can easily get a small diameter as a threshold, which has high pruning efficiency of node-set. However with the increase of m, the diameter becomes larger and pruning efficiency drops down. Thus the enumeration of node-sets is the major overhead affecting



(a) synthetic dataset (b) Flickr dataset

Figure 5.7: performance comparison with respect to average response time (ART) for four algorithms: Apriori-Z, DCC-NL, RDCC and PE varying m (3-9)

the performance. Pairwise Expansion does not need to enumerate node-set. Hence the ART just increases slowly as the amount of data increases.

Next Figure 5.7(b) shows the performance comparison of Flickr data. We test 50 different sets of query keywords for each m. For each test, we declare that this test is invalid if the response time is longer than 30,000 milliseconds. Thus in Figure 5.7(b) there are no value for Apriori-Z when $m > 5$ and DCC-NL when $m > 6$ and RDCC when $m > 7$.

For the mCK query problem, the enumeration of node-sets is an unstable factor for performance. On some data distributions, the number of enumerated node-sets may be too large such that the response time become too long. Therefore we can see the ARTs of Apriori-Z and DCC-NL and RDCC decrease rapidly as m increases for the real dataset. Under this situation, Pairwise Expansion worked well, as a result of no node-set enumeration.

Scalability

We evaluate the scalability of Pairwise Expansion algorithm. We prepare four data sets of Flickr data which collect 0.4M, 0.8M, 1.2M and 1.6M photo records, respectively. In Figure 5.8 we can see the ARTs of each m in the four different data sets. We tested 50 different sets of queries for each dataset. In order to clearly demonstrate the search performance of our algorithm, we show the ART in three parts of time: *Load time* , *Building time of quad-tree*

and *CPU time*.

- *Load time*: the time accessing necessary data from MongoDB
- *Building time of quad-tree*: the time of construction for a quad-tree
- *CPU time*: the time of executing the algorithm and returning result

The performance of Figure 5.8 shows that *CPU time* increases with the data size. It is because we need to enumerate more object-pairs when the data size increases. But compared with the exponential enumeration of node-sets, Pairwise Expansion has an acceptable rate as the data scales up in our experiment. In addition, we can also see that *Load time* grows linearly as data size increases, while the *Building time of quad-tree* grows faster than linearly.

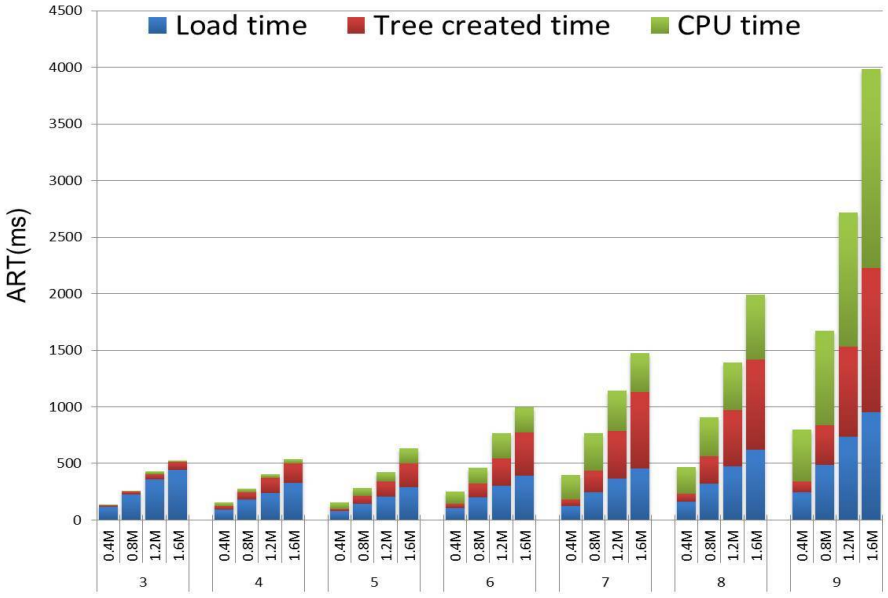


Figure 5.8: comparison of the performance for PE algorithm in four different sizes of datasets: 0.4M, 0.8M, 1.2M and 1.6M

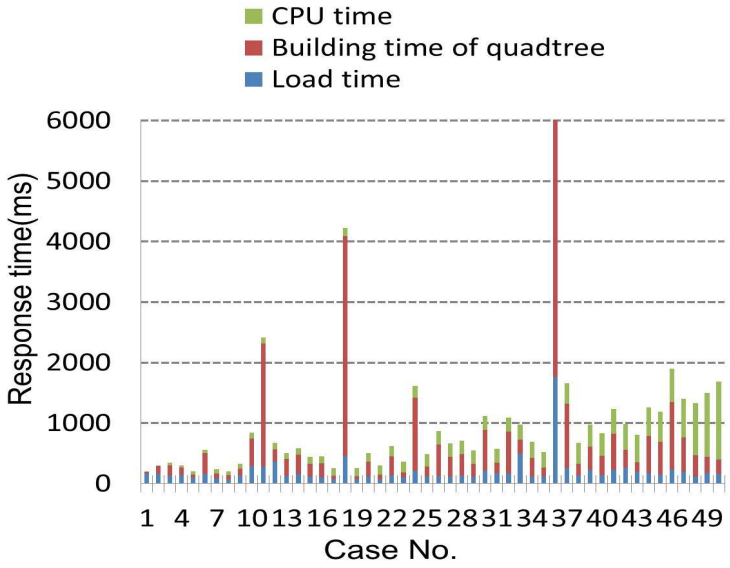


Figure 5.9: performance for each test case for data size=1.6M and $m = 7$

5.4.3 Further tests

In order to know more details about the performance, Figure 5.9 shows the respective performances for 50 test cases when $m = 7$ in the Flickr dataset with 1.6M records. We can see that in most of test cases, the *Building time of quad-tree* takes a higher proportion of the overall response time.

Furthermore, we observe the impact about number of relevant objects for each case in Figure 5.10, we can see the *Load time* grows at the same rate with relevant objects increase in Figure 5.10(a). In contrast, the *Building time of quad-tree* grows obviously faster than *Load time*, and the growth is not stable (Figure 5.10(b)). It is because that the quad-tree is an unbalanced tree, thus when the distribution of data is skewed, we spend more additional cost for the division of cell. And these additional costs will be much greater with the data size increases.

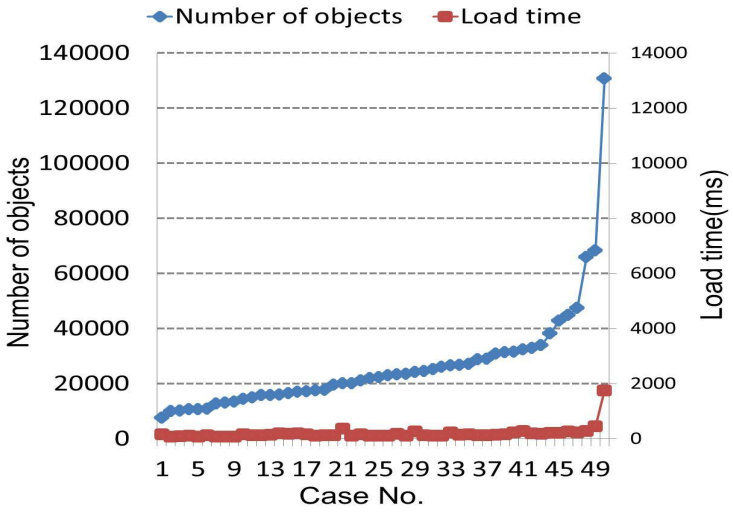
5.5 Summary

In this work, we proposed *Pairwise Expansion* approach for the issue of mCK query. We accomplished the search process in two stages. In the stage of object-pair enumeration we

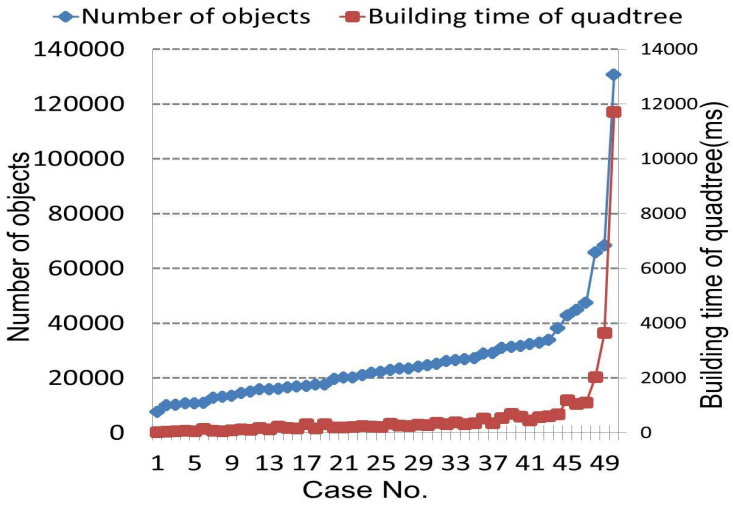
generated the closer object-pair in the first place by the top-down enumeration of node-pairs. And we checked the keywords in the shuttle area of each node-pair to prune unnecessary object-pairs. In the stage of object-pair check, we checked if an object-pair is a diameter of any object-set by expanding it into some object-sets. We used repeated iteration of a set of object-pairs to generate the object-sets. And we adopted the circle check strategy to accelerate the object-pair check process in the stage. The search process is executed under the assumption that there is no prepared data structure. Thus we create a simple quad-tree dynamically by using the necessary data loaded when each query is given.

Compared with the existing top-down search methods, *Pairwise Expansion* can avoid the unstable factor of enumeration of node-sets. The performance of synthetic and real datasets demonstrated that our approach keeps stable and the well *CPU time* for different distribution of data. This is efficient and applicable for real spatial web data.

However, the cost of construction of our quad-tree takes a higher proportion of the total search cost such that the overall performance is reduced. We will discuss the improvement method in next chapter.



(a) number of objects vs. Load time



(b) number of objects vs.Building time of quad-tree

Figure 5.10: Impact about number of objects

Chapter 6

PE Enhanced with Convex-hull based Lower/Upper Bounds

6.1 Remaining issues of the naive PE method

In Chapter 5, we proposed *Pairwise Expansion* approach for the mCK query problem. Different from the existing top-down search methods, *Pairwise Expansion* does not expand all the node-sets at each level of a hierarchical index in top-down process. Instead, it just enumerates node-pairs to find the closer object-pairs, and expands only in the shuttle scopes. Thus *Pairwise Expansion* can avoid the unstable factor of enumeration of node-sets, and remain a high pruning efficiency of top-down search strategy. However though the naive *Pairwise Expansion* method keeps a well performance for different data distributions compared with other existing top-down methods, it still has some deficiencies to be improved and perfected.

Based on the performance result of Figure in Section 5.5, we can see that when data size is small, it takes tiny time to build the quad-tree. However as the size of related data set increases, the construction of the on-the-fly quad-tree will become very time consuming, and take a higher proportion of the overall response time. That is not inconformity with our original intention that quickly build a data index on demand for real spatial web data. Thus this problem needs to be addressed. We will discuss the data structure in detail later in this chapter.

Furthermore, in the naive *Pairwise Expansion* approach, we noticed a trend that when the diameter of final result is large, the search time will correspondingly grows very quickly. In our approach, we need to generate and check the object-pairs whose distances are less than the diameter of final result. Hence there are too many object-pairs to be enumerated under a large optimal diameter. Though we can prune parts of them in the case that the shuttle area of a node-pair/object-pair does not cover all the keywords, this is not sufficient enough for this situation. Thus we need some other way to improve the pruning efficiency. In this chapter, we will discuss a convex-hull based lower and upper bounds for this problem.

6.2 Discussion about data structure

6.2.1 Review of on-the-fly quad-tree

Up to now, the spatial indexing methods employed here, including grid partitioning in DCC and quad-tree in naive PE, are all based on a fixed grid scheme which means recursively divides nodes(cells) into fixed numbers of equal-sized subnodes(subcells). It is difficult to decide which index is the most suitable in mCK queries problem. Hence we chose a fixed grid because it lends itself to relatively simple scheme that does not spend much effort in seeking optimal split way, thus it should be quickly built for the loaded data on demand. However the performance result of Flickr data set brings to our attention that we ought to reconsider this spatial indexing method. Here we will figure out the problem of our quad-tree.

First of all, we describe how to build an on-the-fly quad-tree in naive PE. Suppose we want to create a quad-tree for the related data set $O = \{o_1, o_2, ...o_l\}$ in the sense that each object in O contains at least one of query keywords. And we use C to denote the *Capacity* of the quad-tree, which means the maximum number of objects in a leaf node. The algorithm $CreateQuadtree(O)$ is described as follow:

Algorithm: CreateQuadtree(O)

Step 1: Create a node N for O, then scan all the objects in O, and determine the Minimum Boundary Rectangle(MBR) for N.

Step 2: If the size l of O is larger than C, then divide the region of N into four equal-sized region and N is marked as an internal-node. Accordingly, objects in O are also divided into four sub data sets O_1, O_2, O_3 and O_4. Next, for each sub data set $O_k(k = 1, 2, 3, 4)$, we recursively invoke $CreateQuadtree(O_k)$ to create sub nodes.

Step 3: If the size of O is not larger than C, then N is marked as leaf-node.

According to the description of algorithm $CreateQuadtree(O)$, it needs a full scan for the objects in O when the node N is created. Thus the cost of building a quad-tree is determined by the times of object scanned. However because the quad-tree divides node into equal-sized regions, the number of objects in each region is unbalanced. That may create too much repeated scan for the same objects and make the cost expensive. As an extreme case, when all the objects gathered together in a small area, we have to repeatedly scan these objects for many times before reach to this small area. There are quite a few cases of data distributions like this in the real spatial data sets, especially the objects associated with a toponym. For example, the objects associated with keyword 'choufu' are mainly gathered around the choufu station. Therefore, it is difficult to quickly built a quad-tree for the large number of such skew distributional data.

Moreover, the capacity C in the quad-tree is an important parameter for the search performance including both build time of quad-tree and CPU time for execution of search algorithm. In order to make clear the relationship between C and search performance, we compare various performance metrics for different numbers of C. Beside the performance metrics of 'CPU time', 'Load time' and 'Building time of quad-tree' in Section 5.5, we also measure the search performance based on the following three performance metrics to compare the different aspects of CPU time.

- **Number of enumerated node-pairs:** in our naive *Pairwise Expansion* method, we create a queue to enumerate the node-pairs in top-down process. For each node-pair $\langle n_a, n_b \rangle$ in queue, it is necessary to check the keywords in shuttle scope of $\langle n_a, n_b \rangle$ to decide if the sub node-pairs of $\langle n_a, n_b \rangle$ will be enumerated. Hence the number of enumerated node-pairs represents the pruning ability of node-pair side.

- **Number of generated object-pairs:** compared with the node-pair enumeration, the cost in object-pair side constitutes a more significant portion of the CPU time. Here we use two numbers (of generated object-pairs and checked object-pairs) to represent the total cost of object-pairs. If the two nodes of the dequeued node-pair are both leaf-nodes, we will generate all the object-pairs for this node-pair. After an object-pairs is generated, we need to compute its distance and compare with threshold for pruning. Thus the total number of generated object-pairs is an indication of absolute computational cost.

- **Number of checked object-pairs:** some of generated object-pairs can be pruned directly when the keywords of two objects are same or the distance between them is larger than δ^*. The rest of object-pairs which cannot be pruned directly are called checked object-pairs. For each checked object-pair, it is necessary to check that if it can be the diameter of a 'correct' object-set. Since the check cost may be large, we hope less object-pairs to be checked. Thus the number of checked object-pairs reflects the pruning ability of object-pair side.

We proceed with the experiment by using the same set-up of naive PE, and fix the number of query keywords $m = 6$. We randomly choose 6 keywords 50 times and take average values for each performance metrics. Then we choose the following three different distributions of data sets.

- **uniform distribution:** in this distribution, the quad-tree will be relatively balanced, and each leaf-node objects will has nearly same number of objects. When $m = 6$, the average number of total objects associated with the query keywords is 5991.

- **normal distribution with $\sigma = \frac{1}{8}R$:** In this data set, data will be gathered in one or some small area, thus there may be big different numbers of objects among leaf nodes. And the average number of total objects is also 5991.

- **Flickr data set of 1.6M:** In the Flickr data set, data may be extremely skew and the height of quad-tree may become very high. The average number of objects is 22968.

Table 6.1: Performance comparison of quad-tree on different capacity under uniform distribution

Capacity	10	30	100
CPU time(ms)	116	177	421
Load time(ms)	252	252	262
Building time of quad-tree(ms)	14	12	11
Number of generated object-pairs	91521	467818	1544567
Number of checked object-pairs	8798	9438	9553
Number of enumerated node-pairs	5027	1785	632

Table 6.2: Performance comparison of quad-tree on different capacity under normal distribution with $\sigma = \frac{1}{8}R$

Capacity	10	30	100
CPU time(ms)	417	630	1515
Load time(ms)	257	250	262
Building time of quad-tree(ms)	17	14	12
Number of generated object-pairs	31135	163180	763692
Number of checked object-pairs	8589	29728	95373
Number of enumerated node-pairs	2414	1389	611

Table 6.3: Performance comparison of quad-tree on different capacity under Flickr dataset

Capacity	10	30	100
CPU time(ms)	88	230	1073
Load time(ms)	157	143	147
Building time of quad-tree(ms)	403	387	361
Number of generated object-pairs	11706	96604	718597
Number of checked object-pairs	2125	9425	38224
Number of enumerated node-pairs	4227	2660	1335

Table 6.1, 6.2 and 6.3 show the comparisons under three data sets, respectively. Each table compares the six performance metrics on different capacity $C = 10, 30$ and 100.

From these tables, we can observe that the building times of quad-tree get shorter as C increases. That is because large capacity of node can reduce the times of node split, accordingly the cost of scanning objects becomes smaller.

In contrast, the CPU times become longer as C increases. We observe that if we increase the value of C, the numbers of enumerated node-pairs are small. It can be explained that the total numbers of nodes in quad-trees are reduced, thus less node-pairs are enumerated. We also observe that the number of generated object-pairs increase. That is because more objects in leaf-node will produce more object-pairs. Considering that if a leaf node-pair $\langle n_a, n_b \rangle$ cannot be pruned, then all the object-pairs of $\langle n_a, n_b \rangle$ need to be generated. If both the objects number of n_a and n_b are 30, then the number of generated object-pairs is $30 \times 30 = 900$; and if the objects number increase to 100, then $100 \times 100 = 10,000$ object-pairs will be generated. Thus we can see a rapid growing number of generated object-pairs in these tables. In addition, a larger capacity also slow down the convergent rate of threshold δ^* especially in the case of skewed distributions such that the numbers of checked object-pairs increase inevitably.

In consequence, as C increases, though the enumeration of node-pairs can be reduced, the costs in object-pair side increase more significantly. Thus the total CPU times increase.

6.2.2 Balance tree: on-the-fly kd-tree

According to the discussion of Section 6.2.1, we knew that since quad-tree is an unbalanced tree, it is difficult to limit the depth of the tree with a small capacity such that we may have to take much time for building this data index. Therefore we apply the solution of a kd-tree[Jon Bentley 1975] to decrease the cost in construction of data index.

The kd-tree is known as a special case of binary space partitioning tree. In a kd-tree, each internal-node has two subnodes which divided the space into two parts. The two subnodes are determined by a specific dimension, which means all the objects in the internal-node are splited into two groups equally according to their coordinate values on the specific dimension. For instance, if we use 'X' axis as the specific dimension, and M is the median value of the

coordinate values of all objects on the 'X' axis, then the objects with smaller X-coordinates than M will be in one group and the rest of objects will be in another group. Thus the object numbers of two subnodes are equal (when objects number is even) or almost equal (when objects number is odd). Hence the whole tree is balanced. There are some strategies to decide the division dimension:

1. The division dimension at each branching level is chosen in a round-robin fashion. For instance, in a 2-D space, we alternatively use the X and Y axis to split objects, which means if a node is divided by X axis, then its two subnodes should be divided by Y axis.

2. The division dimension at each node is chosen as the one with the widest spread. The spread of an axis is the difference between the maximum value and minimum value on an axis. For instance, the spread of X axis can be calculated by X.max -X.min in which X.max/X.min is the largest/smallest X-coordinate among all the objects in this node. Here we use this strategy to construct our kd-tree.

3. At each node, calculate the variance of all coordinate values on each dimension and the largest one will be chosen as the division dimension.

Based on the above description about kd-tree, the algorithm of creating a kd-tree can be written like *CreateQuadtree* as follows (here we also use C to denote the capacity in the kd-tree):

Algorithm: CreateKdtree(O)

Step 1: Create a node N for O, then scan all the objects in O, and determine the Minimum Boundary Rectangle(MBR) for N.

Step 2: If the object number of O is greater than C, then invoke Split(O) to divide O into two subsets O_l and O_r, and N is marked as an internal-node. Next, for O_l and O_r, we recursively invoke CreateKdtree(O_l) and CreateKdtree(O_r) to create subnodes.

Step 3: If the size of O is not greater than C, then N is marked as a leaf-node.

Then we depict the division algorithm of O as follows:

Algorithm: Split(O)

Step 1: Calculate the spread in X axis and Y axis by the MBR of N. Then choose the widest one as division dimension denoted as D axis.

Step 2: Find the median value on D axis among all the objects and any object whose D-coordinate is smaller than the median will be put into O_l; otherwise it will be put into O_r. Then output O_l and O_r.

Due to the balance of kd-tree, the depth of kd-tree is $O(log\frac{N}{C})$. The object scan and the calculation of median value can be accomplished in linear time. Thus it can guarantee an $O(Nlog\frac{N}{C})$ time to create a kd-tree, which should be better than a quad-tree.

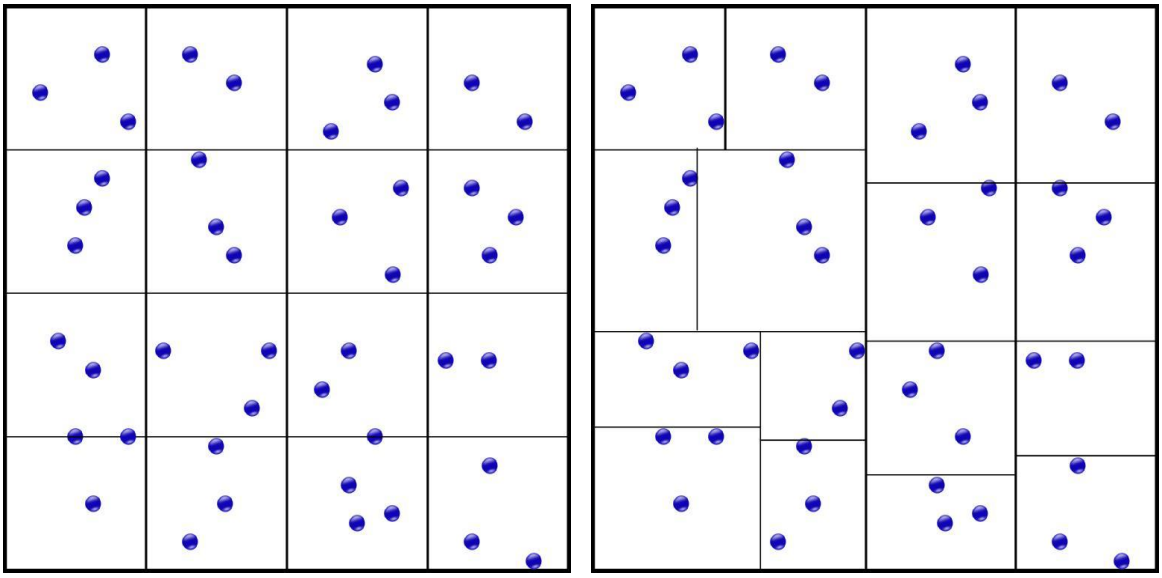
The above kd-tree is actually a binary tree in which each internal-node has only two subnodes. However the quad-tree divide an internal-node into four subnodes. In view of comparing the two indices fairly, we will modify the above $CreateKdtree$ algorithm to change the fanout of the 'kd-tree' into four as follow.

Step 2: If the object number of O is greater than C, then invoke Split(O) to divide O into two subsets O_l and O_r. Next we invoke Split(O_l) again and divide O_l into two subsets O_{l_l} and O_{l_r}. In the same way O_r is also divided into two subsets O_{r_l} and O_{r_r}. Thus there are four subsets of O, then we recursively invoke CreateKdtree(O_k) ($k = l_l, l_r, r_l, r_r$) to create subnodes for them.

In this way, each internal-node in this particular 'kd-tree' has four subnodes. We call such 'kd-tree' a Quad-Split kd-tree, an QSkd-tree for short. In addition, the complexity of creating a QSkd-tree is the same as standard kd-tree.

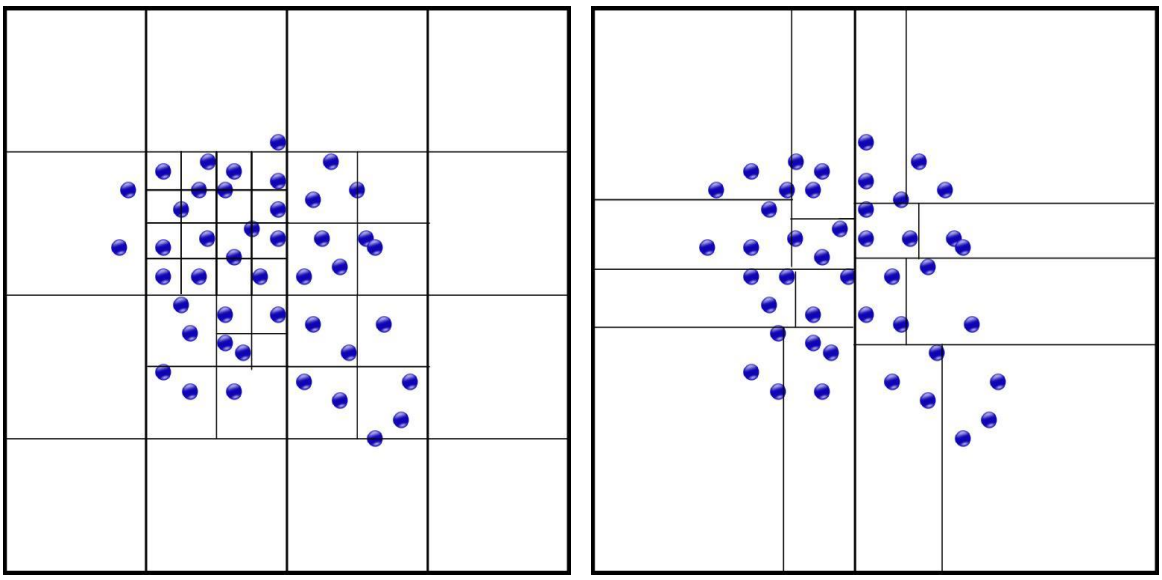
Comparing the difference between a quad-tree and a QSkd-tree, the region of an internal-node in the quad-tree are divided into four smaller regions with same region size; while in the QSkd-tree the region are divided into four smaller regions with same number of objects in order to keep balanced. Thus in the case of uniform distribution, these two partition structures will be nearly same. However in the case of skewed distribution, quad-trees will

become unbalanced in depth. In contrast QSkd-trees can keep balanced in depth, but the region size of each node will become extremely unbalanced. Figure 6.1 and Figure 6.2 show the two partition structures in uniform distribution and skewed distribution respectively.



(a) quad-tree (b) QSkd-tree

Figure 6.1: Comparison between quad-tree and QSkd-tree under uniform distribution



(a) quad-tree (b) QSkd-tree

Figure 6.2: Comparison between quad-tree and QSkd-tree under skewed distribution

Same as the quad-tree, we also evaluate the search performance by varying the number of capacity C. Here we also fix the query keywords number $m = 6$ and use the same test cases to compare the six performance metrics in Section 6.2.1. Since the data structures and performances between quad-tree and QSkd-tree are almost same, We only evaluate the case of skewed distribution by using Flickr data set of 1.6M.

Table 6.4 shows the average values of these performance metrics when capacities C of QSkd-tree are $10, 30$ and 100. The result demonstrates that as C increases, the building time of QSkd-tree become shorter but the CPU time get longer. The reason is same, although larger capacity can reduce the number of enumerated node-pairs, the cost generated object-pairs and checked object-pairs increases more significantly, thus the whole CPU time increased.

Table 6.4: Performance comparison of QSkd-tree on different capacity under Flickr dataset

Capacity	10	30	100
CPU time(ms)	156	333	2119
Load time(ms)	164	156	157
Building time of quad-tree(ms)	189	157	129
Number of generated object-pairs	39895	275919	1573849
Number of checked object-pairs	3647	12957	52739
Number of enumerated node-pairs	8297	4051	1533

We can compare the performances between quad-tree and QSkd-tree by Table 6.3 and Table 6.4. First, QSkd-tree performs consistently better in terms of building time than quad-tree. Note that the QSkd-tree is a balanced tree and it can guarantee an $O(Nlog\frac{N}{C})$ time in construction. In contrast, quad-tree may be too deep which lead to higher object scanned cost in skewed distributions. In order to prove it, we randomly chose 10 test cases when $m = 6$ and compare the depth of the two structures in Table 6.5. Then we found that the depths of the quad-tree are much higher than the QSkd-tree. Consequently QSkd-tree is able to exhibit a good performance in the building time.

We also observe that QSkd-tree performs worse than quad-tree in terms of CPU time. We thought there are two reasons:

1. Since the nodes of quad-tree are divided into fixed region size, the number of objects in a leaf node may be relatively small. It is possible that a leaf node has only one object. In contrast, in order to keep balance, QSkd-tree ensure each leaf node with nearly same object number. A node has at least $\frac{C}{4}$ objects. Thus most of the time, the number of objects in a node of quad-tree is smaller than QSkd-tree. This can be proved in Table 6.5 which compares the average object numbers of leaf nodes for the two trees. This is one of the reasons that more object-pairs are generated from a leaf node-pair of QSkd-tree.

2. Due to more object number in a node, the size of the node's MBR of QSkd-tree is bigger than the quad-tree. Table 6.5 shows the comparison of the average sizes of leaf nodes for the two trees. Here the size of a leaf node is the diagonal distance of node's MBR. For this reason, QSkd-tree may has the bigger shuttle area of node-pair such that it is easy to contain all the keywords in the shuttle area. That will lead to worse pruning efficiency. Thus the numbers of enumerated node-pairs and checked object-pairs are larger than quad-tree.

Consequently QSkd-tree spends more CPU time than quad-tree.

Table 6.5: Node comparison between quad-tree and QSkd-tree under Flickr dataset

	depth of tree		average object number		average size(km)	
case number	quad-tree	QSkd-tree	quad-tree	QSkd-tree	quad-tree	QSkd-tree
case1	15	5	10	11	3.18	5.98
case2	17	6	10	8	1.44	1.91
case3	16	5	10	12	1.82	4.18
case4	18	6	10	11	0.74	1.55
case5	15	5	11	29	1.95	7.08
case6	14	5	10	10	2.86	5.36
case7	16	5	10	11	2.84	5.61
case8	16	5	10	18	2.02	6.65
case9	18	6	10	14	1.02	2.36
case10	16	5	10	21	2.33	7.09

6.3 Convex-hull as new lower/upper bounds in Pairwise Expansion

6.3.1 Motivation

According to the naive *Pairwise Expansion* approach, a node-pair or an object-pair can be pruned out if the shuttle scope of it does not contain all the query keywords. However the pruning efficiency will reduce as the distance between the two nodes or two objects get larger. Therefore we consider a further way to enhance the pruning ability by using convex-hull based lower and upper bounds. This approach is called *EnhancedPE*.

6.3.2 Preparation

To illustrate it, we prepare some technical terms.

Firstly, we define as the $Dist(\langle o_a, o_b\rangle, o)$ (Figure 6.3) is the 'distance' between an object-pair $\langle o_a, o_b\rangle$ and an object o as follow:

Definition 1 $Dist(\langle o_a, o_b\rangle, o)$: Given an object-pair $\langle o_a, o_b\rangle$ and an object o, the distance between them is defined by:

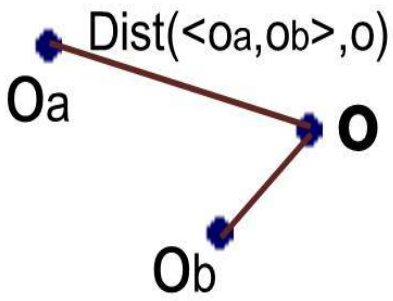


Figure 6.3: $Dist(\langle n_a, n_b\rangle, o)$

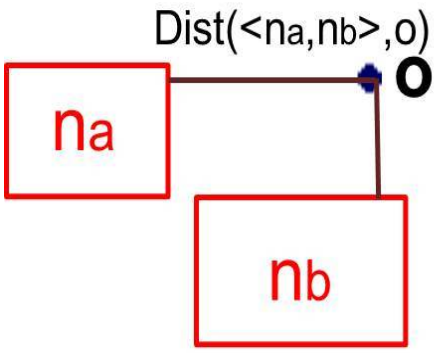


Figure 6.4: $Dist(\langle n_a, n_b\rangle, o)$

$$Dist(\langle o_a, o_b\rangle, o) = \max\{dist(o_a, o), dist(o_b, o)\} \tag{6.1}$$

$Dist(\langle o_a, o_b\rangle, o)$ represents the relation between object o and the shuttle scope of $\langle o_a, o_b\rangle$. If this $Dist$ is larger than $dist(o_a, o_b)$, then o is outside of shuttle scope of $\langle o_a, o_b\rangle$; otherwise, o is in it.

Similar as Definition 1, $Dist(\langle n_a, n_b\rangle, o)$ (Figure 6.4)is defined as the 'distance' between a node-pair $\langle n_a, n_b\rangle$ and an object o as follow:

Definition 2 $Dist(\langle n_a, n_b\rangle, o)$: Given a node-pair $\langle o_a, o_b\rangle$ and an object o, the distance between them is defined by:

$$Dist(\langle n_a, n_b\rangle, o) = \max\{Mindist(n_a, o), Mindist(n_b, o)\} \tag{6.2}$$

For each keyword in the shuttle scope of a node-pair, we have the follow definition:

Definition 3 $Min\_Dist_{k_i}(\langle n_a, n_b\rangle)$: Given a node-pair $\langle n_a, n_b\rangle$, if the shuttle scope of $\langle n_a, n_b\rangle$ contains all the keywords, then for any one of query keywords $k_i (k_i \notin n_a, k_i \notin n_b)$, $Min\_Dist_{k_i}(\langle n_a, n_b\rangle)$ is defined as the minimum $Dist$ between $\langle n_a, n_b\rangle$ and any object o associated with k_i:

$$Min\_Dist_{k_i}(\langle n_a, n_b\rangle)) = \min_{o \in O} Dist(\langle n_a, n_b\rangle, o),$$

where O is the object collection of all the objects associated with k_i in the shuttle scope of $\langle n_a, n_b\rangle$.

According to Definition 2, the following lemma holds:

Lemma If an object-pair $\langle o_1, o_2\rangle$ generated from $\langle n_a, n_b\rangle$ satisfies that $dist(o_1, o_2) < Dist(\langle n_a, n_b\rangle, o)$, then o is outside of shuttle scope of $\langle o_1, o_2\rangle$.

In Figure 6.5 , we know the $Dist(\langle n_a, n_b\rangle, o) = Mindist(n_a, o) = a$, thus $dist(o_1, o)$ must be larger than a. Due to $dist(o_1, o_2) < a$, we can infer $dist(o_1, o) < dist(o_1, o_2)$, then o is outside of $\langle o_1, o_2\rangle$.

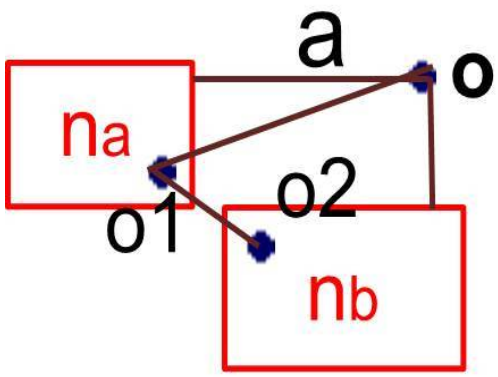


Figure 6.5: Lemma

6.3.3 New lower bound

By this lemma, we consider the following pruning rules:

Pruning Rule Given an object-pair $\langle o_a, o_b \rangle$ generated from node-pair $\langle n_a, n_b \rangle$, if the distance of $\langle o_a, o_b \rangle$ is less than the $Min\_Dist_{k_i}(\langle n_a, n_b \rangle)$ for any keyword k_i, then $\langle o_a, o_b \rangle$ can be pruned out directly.

In Figure 6.6 there are two objects o_1 and o_2 associated with keyword E. We can see that Min_Dist_E is $Dist(\langle n_a, n_b \rangle, o_2)$, denoted by d_E. Then d_E can be a lower bound of the distance for the object-pairs generated from $\langle n_a, n_b \rangle$. That means any object-pair o_m, o_n that satisfies $dist(o_m, o_n) < d_E$ cannot be a diameter of any 'correct' object-set, because the shuttle scope of $\langle o_m, o_n \rangle$ cannot contain E.

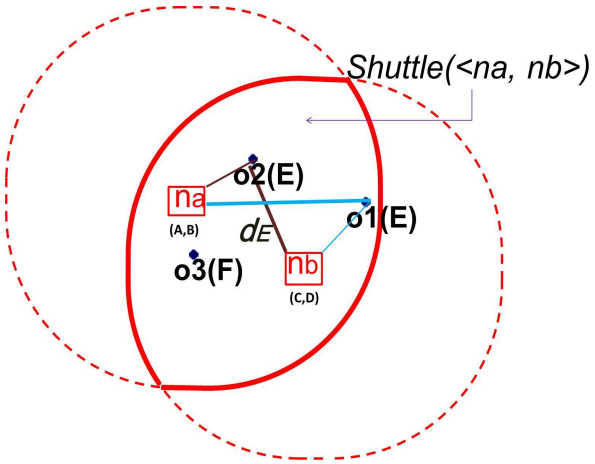


Figure 6.6: rule

We can use this pruning rule in the naive *Pairwise Expansion* algorithm. Given a node-pair $\langle n_a, n_b \rangle$, if the shuttle scope of $\langle n_a, n_b \rangle$ contains all the keywords, then for each keyword $k_i (k_i \notin n_a, k_i \notin n_b)$, we can get $Min\_Dist_{k_i}$ for it ,denoted by d_{k_i}. Thus $\max\limits_{k_i \in Q} d_{k_i}$ is a lower bound LB of $\langle n_a, n_b \rangle$.

We rewrite the PE algorithm in chapter 5 by adding this lower bound as follow:

Algorithm: PEwithLB

// Input: *root*: root node of spatial index; Q: m given keywords.

// Output: δ^*_{opt}: the minimal diameter.

// Variables: *Queue*: a sorted list of node-pair $\langle n_i, n_j \rangle$, sorted in the ascending order of $Mindist(n_i, n_j)$; δ^*: current minimum diameter.

Step 1: Set the node-pair $\langle root, root \rangle$ into $Queue$. Initialize δ^* with ∞.

// This is a (do-while) loop from Step 2 to Step 5, in order to repeat pulling out the current smallest node-pair from $Queue$.

Step 2: Dequeue node-pair $\langle n_i, n_j \rangle$ from $Queue$. Do:

2-1: If the $Mindist(n_i, n_j) \geq \delta^*$, then return δ^*.

2-2: If $Shuttle(\langle n_i, n_j \rangle)$ does not contain all the keywords of Q, then goto Step 5.

2-3: Calculate d_{k_l} for each keyword $k_l \in Q$, and set $LB = \max_{k_l \in Q} d_{k_l}$.

2-4: Calculate the upper bound $\mathcal{D}$ of $Shuttle(\langle n_i, n_j \rangle)$. If $\mathcal{D} < \delta^*$, update δ^* with $\mathcal{D}$.

2-5: If either n_i or n_j is not a leaf-node, goto Step 4. Else goto Step 3.

Step 3: // (both n_i and n_j are leaf-nodes.)

For each object-pair $\langle o_a, o_b \rangle$ which satisfies $dist(o_a, o_b) \geq LB$ in $\langle n_i, n_j \rangle$ ($o_a \in n_i, o_b \in n_j$), in the ascending order of $dist(o_a, o_b)$, do:

3-1: If $dist(o_a, o_b) \geq \delta^*$, then break this for-loop and goto Step 5.

3-2: Check if $\langle o_a, o_b \rangle$ is a diameter of a 'correct' object-set which exists in $Shuttle(\langle o_a, o_b \rangle)$.

3-3: If the check is true , update δ^* by $dist(o_a, o_b)$ and goto Step 5.

3-4: If the check is false , continue this for-loop.

(//the next $\langle o_a, o_b \rangle$ is tested.)

Step 4: If either n_i or n_j is an internal node, add the child-node pair $\langle n_a, n_b \rangle$ which satisfies $Maxdist(n_a, n_b) \geq LB$ into the $Queue$.

Step 5: Return δ^* if $Queue$ is empty. Else goto Step 2.

In addition, in order to calculate $Min\_Dist_{k_i}$ for each keyword k_i, we adopt a top-down method which is similar to find the nearest neighbour (NN) object. Thus we can apply the branch-and-bound solution for NN search [24] in a top-down way.

6.3.4 New upper bound

Furthermore, since all the 'closest' objects $o_{k_{i_{min}}}$ for each keyword k_i were found, these 'closest' objects and the MBRs of n_a and n_b can be enclosed in a convex-hull. Thus convex-hull contains all the keywords and we can use the diameter of it as an upper bound of result.

In Figure 6.7, given a node-pair $\langle n_a, n_b \rangle$ in which node n_a contains keywords $\{A, B\}$ and node n_b contains keywords $\{C, D\}$. Then suppose the query keywords $Q = \{A, B, C, D, E, F\}$ and the o_2 and o_3 are the 'closest' objects of E and F respectively. Then we can find a convex-hull H such that n_a, n_b, o_2, o_3 are in it. Thus H can contain all the query keywords of Q, and there exists a 'correct' object-set in it. Consequently, the diameter of H must be an upper bound of distances of object-pairs generated from $\langle n_a, n_b \rangle$.

Once $\langle n_a, n_b \rangle$ computed H, consider we test another node-pair $\langle n_x, n_y \rangle$, then if any object-pair $\langle o_x, o_y \rangle \in \langle n_x, n_y \rangle$ satisfies that $dist(o_x, o_y) >$ H's diameter, such $\langle o_x, o_y \rangle$ can be pruned out. Furthermore, if any child node-pair $\langle n'_x, n'_y \rangle \in \langle n_x, n_y \rangle$ satisfies that $Mindist(n'_x, n'_y) >$ H's diameter, then $\langle n'_x, n'_y \rangle$ can be pruned out.

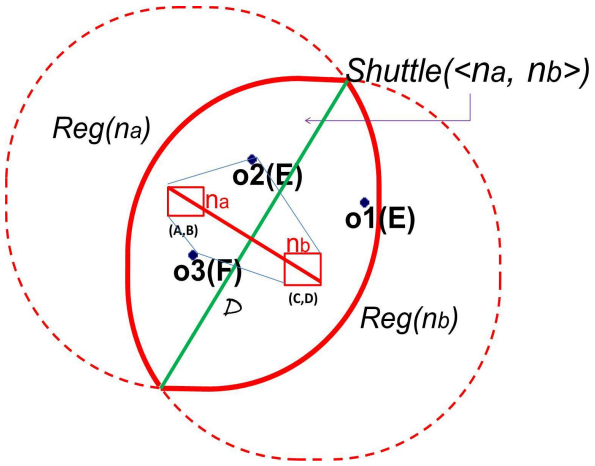


Figure 6.7: Upper bound of node-pair

In the *Pairwise Expansion* approach in chapter 5, we used $\mathcal{D}$ which is the maximum distance between any two points on the boundary of the shuttle area as the upper bound of result. As the Figure 6.7 shows, we can observe that the diameter of H is further smaller than $\mathcal{D}$. Thus this is more efficient for pruning . We replace $\mathcal{D}$ with the diameter of H in the algorithm PEwithLB and revise the Step 2 as follow.

Algorithm: PEwithLBandUB (EnhancedPE)

// Input: $root$: root node of spatial index; Q: m given keywords.

// Output: δ^*_{opt}: the minimal diameter.

// Variables: $Queue$: a sorted list of node-pair $\langle n_i, n_j \rangle$, sorted in the ascending order of $Mindist(n_i, n_j)$; δ^*: current minimum diameter.

Step 1: Set the node-pair $\langle root, root \rangle$ into $Queue$. Initialize δ^* with ∞.

// This is a (do-while) loop from Step 2 to Step 5, in order to repeat pulling out the current smallest node-pair from $Queue$.

Step 2: Dequeue node-pair $\langle n_i, n_j \rangle$ from $Queue$. Do:

2-1: If the $Mindist(n_i, n_j) \geq \delta^*$, then return δ^*.

2-2: If $Shuttle(\langle n_i, n_j \rangle)$ does not contain all the keywords of Q, then goto Step 5.

2-3: Calculate d_{k_l} for each keyword $k_l \in Q$ and find the 'closest' object for each keyword $k_l \in Q$. Set $LB = \max_{k_l \in Q} d_{k_l}$.

2-4: Create a convex hull H by using these 'closest' objects from 2-3 and two MBRs of n_i and n_j. If the diameter of $H < \delta^*$, then update δ^* with it.

2-5: If either n_i or n_j is not a leaf-node, goto Step 4. Else goto Step 3.

Step 3: // (both n_i and n_j are leaf-nodes.)

For each object-pair $\langle o_a, o_b \rangle$ which satisfies $dist(o_a, o_b) \geq LB$ in $\langle n_i, n_j \rangle$ ($o_a \in n_i, o_b \in n_j$), in the ascending order of $dist(o_a, o_b)$, do:

3-1: If $dist(o_a, o_b) \geq \delta^*$, then break this for-loop and goto Step 5.

3-2: Check if $\langle o_a, o_b \rangle$ is a diameter of a 'correct' object-set which exists in $Shuttle(\langle o_a, o_b \rangle)$.

3-3: If the check is true , update δ^* by $dist(o_a, o_b)$ and goto Step 5.

3-4: If the check is false , continue this for-loop.

(//the next $\langle o_a, o_b \rangle$ is tested.)

Step 4: If either n_i or n_j is an internal node, add the child-node pair $\langle n_a, n_b \rangle$ which satisfies $Maxdist(n_a, n_b) \geq LB$ into the $Queue$.

Step 5: Return δ^* if *Queue* is empty. Else goto Step 2.

6.4 Evaluation

The *EnhancedPE* approach utilizes two techniques to improve the efficiency of search process. First, QSkd-tree is used as the on-the-fly index structure, instead of the quad-tree, to reduce the building time of index. Next, the basic *Pairwise Expansion* (PE) method is enhanced with convex-hull based lower/upper bounds, in order to improve the pruning ability. This section evaluates performances of these two techniques. The experimental set-up is the same as that of section 5.4.1. We use the same Flickr data set with 1.6 million photo records for all experiments.

6.4.1 Performance comparison between quad-tree and QSkd-tree

Here, we compare the performances of two index structures, i.e., QSkd-tree and quad-tree, by proceeding with both PE and EnhancedPE methods. The capacity C is set to 30 in both quad-tree and QSkd-tree.

Figure 6.8 shows the comparison of average response times(ARTs) of PE method under both quad-tree and QSkd-tree when we vary the number of query keywords m. Similar as Section 5.4.2, ART is divided into three parts: 'CPU time', 'Building time of tree' and 'Load time' to show more details of performance information.

In Figure 6.8, we can see the Building time of QSkd-tree is much smaller than that of quad-tree. That is because the QSkd-tree takes less cost on the construction of data partitioning. However, due to the large size of node, CPU time of PE method under QSkd-tree is much larger than that under quad-tree, especially when m is large. This slows down the overall performance, thus the ART of QSkd-tree is larger than that of quad-tree when $m > 7$.

Figure 6.9 shows the comparison between QSkd-tree and quad-tree with EnhancedPE method. In this figure, we can see that the QSkd-tree can keep lower ARTs than the quad-tree. Although QSkd-tree with EnhancedPE method also takes more CPU time than the quad-tree, the EnhancedPE method can efficiently improve the pruning ability, which makes the difference of CPU time becomes smaller. And the building time of QSkd-tree is much

smaller than that of quad-tree. Thus the overall performance of QSkd-tree with EnhancedPE is better than quad-tree.

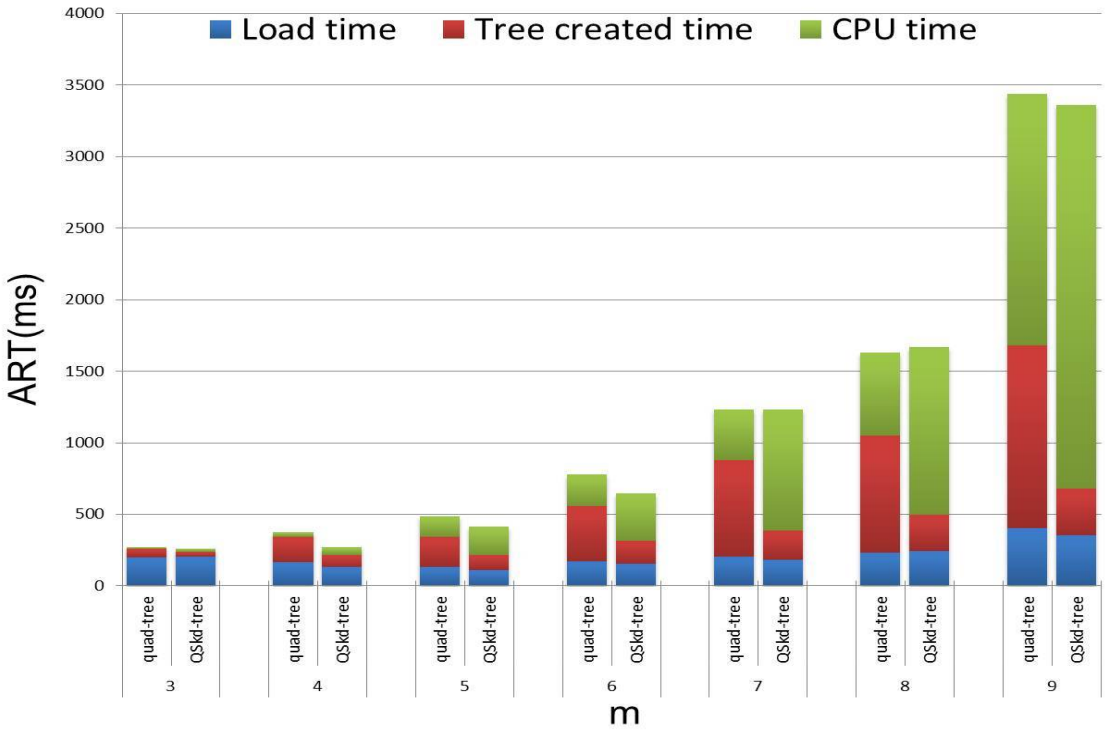


Figure 6.8: m vs. ARTs between quad-tree and QSkd-tree with basic PE method

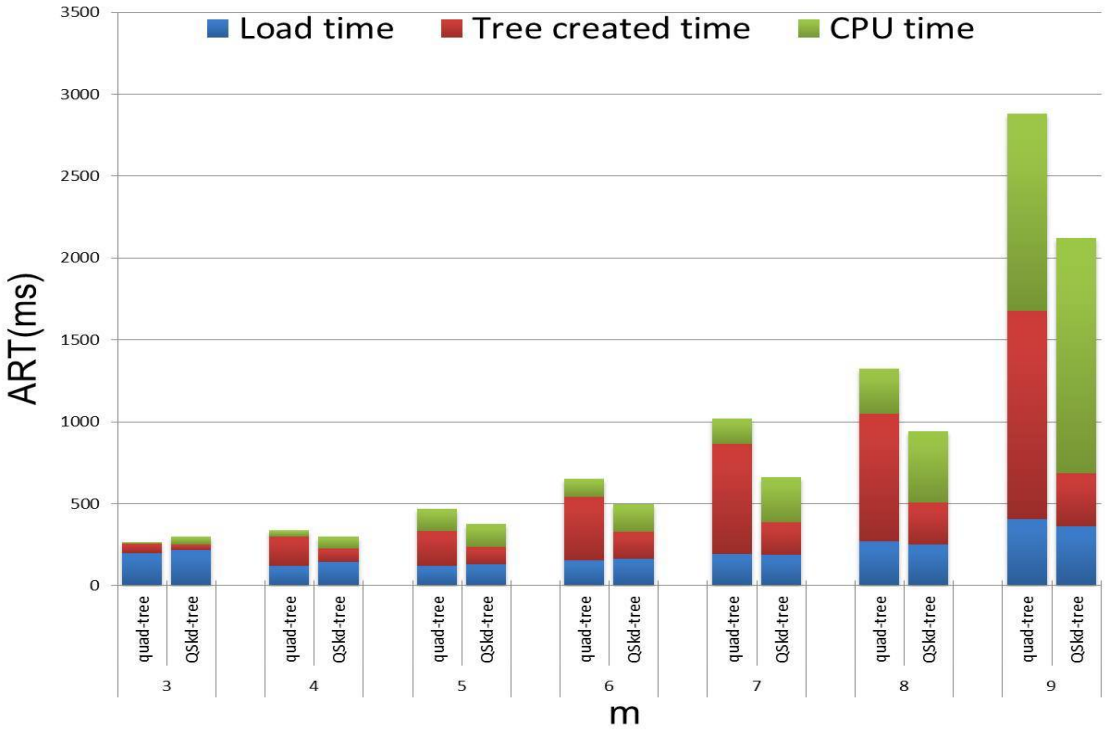


Figure 6.9: m vs. ARTs between quad-tree and QSkd-tree with EnhancedPE method

6.4.2 Performance comparison between PE and EnhancedPE

There are two aspects mainly to analyse and compare the performance of EnhancedPE method with basic PE method. Firstly we focus on elapsed time (ARTs) and their gradient components.

First, we use QSkd-tree as index structure and compare the ARTs of two methods in different numbers of query keywords m. The result is shown in Figure 6.10. According to Figure 6.10, we can observe that the CPU time of EnhancedPE is much smaller than PE, thus the overall performance of EnhancedPE is better than original PE method.

Second, in order to know more details about the performances of using lower bound and upper bound under different index structures, we use the following six test approaches for comparison:

- **quad:** The *Pairwise Expansion* in chapter 5 under quad-tree structure.
- **quad+LB:** The PEwithLB algorithm under quad-tree structure.

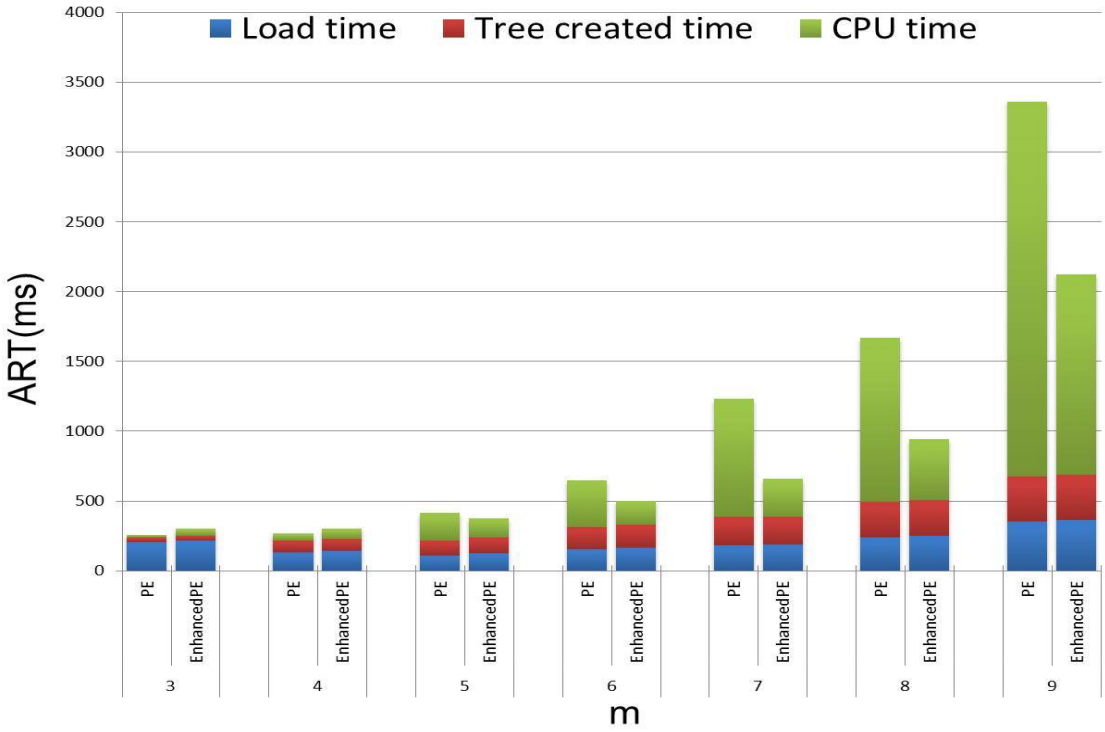


Figure 6.10: m vs. ARTs between PE and EnhancedPE under QSkd-tree

- **quad+LB+UB:** The PEwithLBandUB(EnhancedPE) algorithm under quad-tree structure.
- **QSkd:** The *Pairwise Expansion* in chapter 5 under QSkd-tree structure.
- **QSkd+LB:** The PEwithLB algorithm under QSkd-tree structure.
- **QSkd+LB+UB:** The PEwithLBandUB(EnhancedPE) algorithm under QSkd-tree structure.

We fix the numbers of query keywords $m = 6$ and $m = 8$ respectively, then test the six performance metrics in Section 6.2.1 to evaluate these approaches. The results are shown in Table 6.5 and 6.6.

According to the results of Table 6.5 and 6.6 , we can observe that using lower bound can significantly cut down the checked object-pairs and enumerated node-pairs, accordingly the CPU times are reduced. That demonstrates that it can work well for pruning both in quad-tree and QSkd-tree. We can also observe that when we use the upper bound, the number of

checked object-pairs and enumerated node-pairs are further decreased, hence this can further reduce the CPU time when the number of keywords is large ($m = 8$).

Table 6.6: Performance comparison for all kinds of PE based methods when $m = 6$ under Flickr dataset (data size =1.6M). ("quad" : basic PE under quad-tree; "quad+LB+UB" : EnhancedPE under quad-tree; "QSkd" : basic PE under QSkd-tree; "QSkd+LB+UB" : EnhancedPE under QSkd-tree)

Method	quad	quad+LB	quad+LB+UB	QSkd	QSkd+LB	QSkd+LB+UB
Total response time(ms)	760	651	654	646	489	498
CPU time(ms)	230	105	114	333	165	169
Load time(ms)	143	156	155	156	162	165
Building time of tree(ms)	387	390	385	157	162	164
# of generated object-pairs	96604	96604	96604	275919	275919	275919
# of checked object-pairs	9425	2064	1665	12957	2337	1957
# of enumerated node-pairs	2660	1972	1859	4051	2889	2705

Table 6.7: Performance comparison for all kinds of PE based methods when $m = 8$ under Flickr dataset (data size =1.6M). ("quad" : basic PE under quad-tree; "quad+LB+UB" : EnhancedPE under quad-tree; "QSkd" : basic PE under QSkd-tree; "QSkd+LB+UB" : EnhancedPE under QSkd-tree)

Method	quad	quad+LB	quad+LB+UB	QSkd	QSkd+LB	QSkd+LB+UB
Total response time(ms)	1640	1353	1344	1667	927	910
CPU time(ms)	595	285	277	1171	428	409
Load time(ms)	247	269	268	240	240	241
Building time of tree(ms)	798	799	799	256	259	260
# of generated object-pairs	137251	137251	137251	333427	333427	333427
# of checked object-pairs	20954	3562	2785	35771	4743	4053
# of enumerated node-pairs	3972	2856	2508	7370	4298	3791

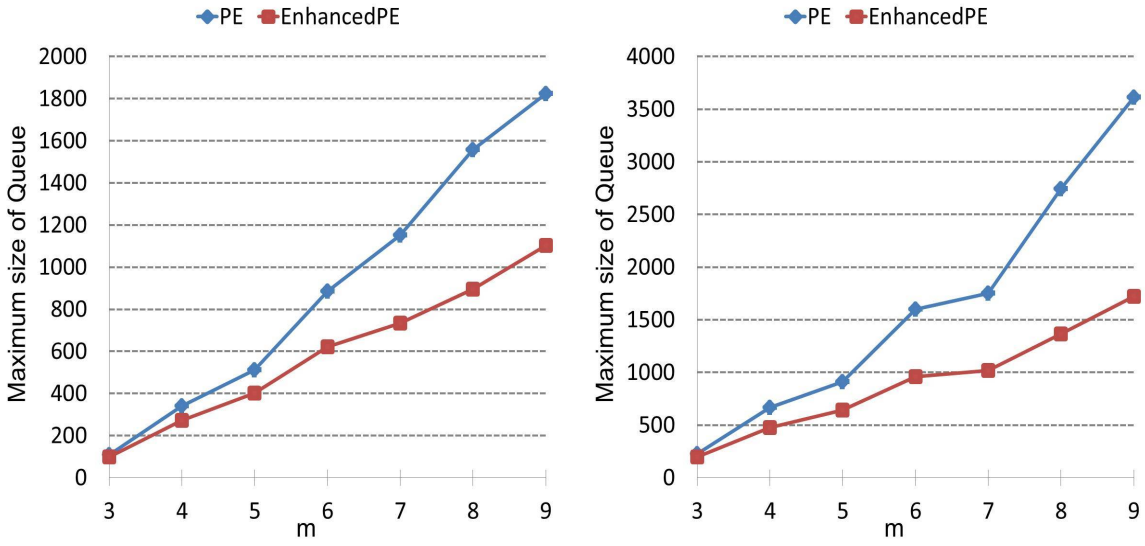
6.4.3 Memory consumption test

Beside the search time as the competitive performance, we also discuss the memory consumption of the two methods. Different from the other methods (Apriori-Z, DCC-NL and RDCC), all of which used the depth-first search strategies, PE and EnhancedPE need to hold a sorted queue of node-pairs in memory. Thus the memory consumption of them should be considered.

We measure the maximum size of the global queue for each test case and take the average values for each m. Figure 6.11(a) shows comparison of maximum size of queue between PE and EnhancedPE under the quad-tree index structure. According to the result, we can see the maximum size of queue of EnhancedPE is smaller than that of PE. That is because

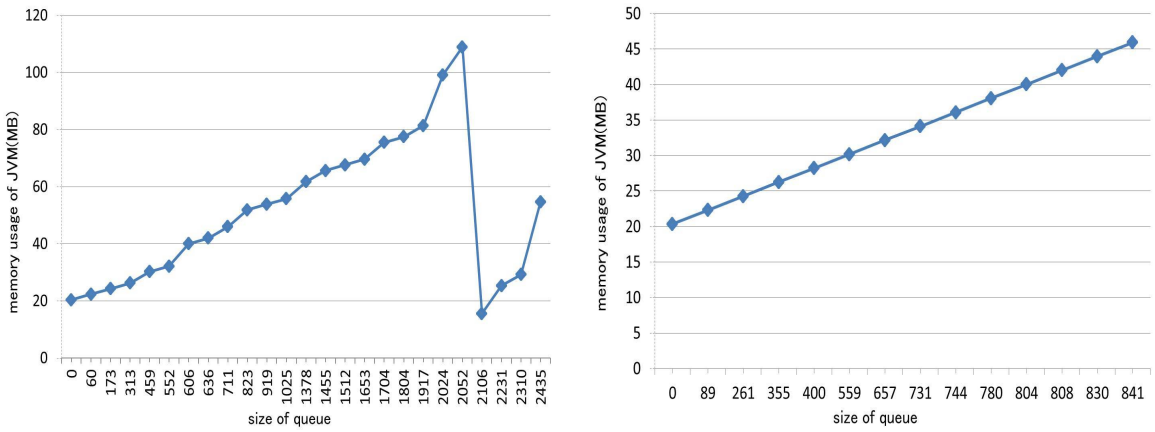
EnhancedPE excludes the node-pair $\langle n_i, n_j \rangle$ which satisfies $Maxdist(n_i, n_j) < LB$ (LB: lower bound) from the queue. Furthermore, EnhancedPE also uses a smaller upper bound to prevent the size of queue from getting too large before arriving at leaf node-pairs, thus the maximum size of queue is effectively limited. That can demonstrate that the memory consumption of EnhancedPE is smaller than that of PE. Figure 6.11(b) shows the same comparison under the QSkd-tree index structure. We can see that the maximum sizes of queue in EnhancedPE are only around a half of sizes of queue in PE when the $m > 6$.

In addition, we use the runtime class of Java to track the memory usage of JVM for the search instance of query keywords $Q = \{winter, hotel, flower, temple, garden, ski, snow\}$. We use QSkd-trees as index structure, and respectively execute PE and EnhancedPE for this Q. During the executions of both PE and EnhancedPE, we continuously monitor the memory usage of JVM at all the points of queue operations, including enqueue and dequeue. That means once the enqueue or dequeue operation performs, we record the two values: (1)the size of queue and (2) memory usage, at the time. Thus we get a list of these records.



(a) quad-tree (b) QSkd-tree

Figure 6.11: m vs. maximum size of queue between PE and EnhancedPE



(a) PE (b) EnhancedPE

Figure 6.12: comparison of memory usage of JVM between PE and EnhancedPE. $m = 7$, $Q = \{winter, hotel, flower, temple, garden, ski, snow\}$

We observed that the value of memory usage does not constantly change with the size of queue changes. Hence we only choose the records at which the value of memory usage changed from this list to show the relationship between size of queue and memory usage in Figure 6.12. According to Figure 6.12, we can observe that the memory usage of EnhancedPE is much smaller than that of PE overall. That can demonstrate that EnhancedPE can effectively reduce the memory consumption by pruning more node-pairs from the global queue. In Figure 6.12(a), we observed that the memory usage suddenly fell when the size of queue is 2024. It is because the memory allocated for the dequeued node-pairs is freed by the garbage collection system of Java.

In consequence, these results show that EnhancedPE is more advantageous than PE in the performances of both search time and memory consumption.

6.5 Discussion

In Section 2.3.4, we introduced Guo'a approach in [9]. Based on the description, we can know that Guo's approach does not use the top-down search technique. Instead, this approach finds both approximation solutions and exact solution by traversing through all the objects O' associated with query keywords in data set D. To find the exact solution, Guo uses a rotated circle whose diameter is determined by the approximation solution to restrict the potential

solutions in some small areas around each object $o \in O'$, then exhaustively enumerates the object-sets in these areas. At this point, our EnhancedPE method also restricts the potential solutions in some small shuttle areas and finds the exact solution by exhaustive enumeration of object-sets in them. However, EnhancedPE uses a top-down way to find these shuttle areas instead of the direct manner.

In addition, there are some differences between these two approaches in the details of object-set enumeration.

1. In Guo's exact algorithm, the aim of object-set enumeration is to find the smallest object-set. Thus it is necessary to enumerate and compare all the object-sets which cannot be pruned out in each circle area. On the other hand, our EnhancedPE aims to check whether the object-pair can be the diameter of a 'correct' object-set in Section 5.3. Thus once we find such 'correct' object-set, we can stop the enumeration. That means EnhancedPE is more probable to reduce the risk of exponential object-set enumeration.

2. Guo's exact algorithm uses an iterative way to generate object-sets. In each iteration, it selects one object from candidate objects in the circle area. In this way , an object-set can be generated after (m-1) iterations. When the number of candidate objects is large, this iterative way is easy to generate exponential object-sets. Our EnhancedPE method uses recursive DCC strategy to generate size-(m-2) sub object-sets in shuttle area. In each recursion, EnhancedPE uses circle check to reduce the number of sub DCs. Thus this can ease the generation of exponential object-sets to some extent.

Moreover, in the aspect of search policy, Guo's approach uses a direct manner that enumerates all the related objects O' and finds approximation solutions and exact solution for each objects $o \in O'$ instead of the traditional top-down way. Although this policy of Guo's approach is beneficial to find good approximate solutions of mCK query problem, we consider that our top-down approach of EnhancedPE also has some advantages in both mCK query problem and its extension.

1. Guo proved that the mCK query is an NP-hard problem. In the previous study of top-down approach of Apriori-Z, it is necessary to generate node-sets in the top-down process. However the number of node-sets also can be exponential, which may be

the major problem affecting the search efficiency. Guo's approach directly enumerates the objects, thus it can avoid this problem. Our EnhancedPE also can overcome this problem. It uses the generation of node-pairs instead of node-sets, thus it can rapidly find the restricted areas in a quadratic complexity (in worst case) and its shuttle areas is much smaller than the circle area of Guo's approach.

2. If we expand the mCK queries problem to find top-k smallest object-sets, or if we consider using some other parameters to calculate the score of an object-set and find the top-1 or top-k results under the score, the top-down method is a natural way to flexibly deal with these possible situations. However it is difficult to expand Guo's approach to these extensions.

3. mCK queries use the diameter to measure the closeness of an object-set. Essentially the diameter is an 'line segment' of two objects. Guo's approach uses a circle (MCC) as a key of search. This can be viewed as using a 'volume' of MCC as the approximate solution of the 'line segment'. This can work well in two-dimensional data. But when mCK queries are used in higher dimensional data, the circle (MCC) will become a hypersphere, and the 'volume' of this hypersphere may become more complicated and the approximate ratio will be declined. In this case, our EnhancedPE keeps the diameter of the line segment (maximum $dist(o_x, o_y) \in O$) as a key of search, and recursively enumerates object-pairs (line segment) to generate an object-set. Thus it can be easier to expand to higher dimensional data than Guo's approach. (Note we use RecursiveCheck instead of SophisticatedCheck algorithm in Section 5.3.3).

6.6 Summary

In this chapter, we proposed the EnhancedPE method. It solves two points of basic *Pairwise Expansion* approach in chapter 5.

One is the building time of the data index. As our original intention, we expect a data index which can be quickly constructed for the loaded object on demand. However the on-the-fly quad-tree might create too much repeated scan for the objects which would slow down

the set-up speed when all the objects gathered together in a small area. To overcome this problem, we adopted a kd-tree based data structure called QSkd-tree to reduce the cost of building time of index. The performance of Flickr dataset demonstrated that the QSkd-tree could be constructed faster than the quad-tree.

Another point is that in the *Pairwise Expansion* approach in chapter 5, we pruned out a node-pair or an object-pair only if the shuttle scope of it does not contain all the query keywords. However the pruning efficiency is not enough especially when the diameter of final result is large. Thus we proposed convex-hull based lower bound and upper bound to improve the pruning ability. This technique can significantly decrease the numbers of enumerated node-pairs and checked object-pairs, hence the CPU time is reduced.

Finally, we compared this enhanced approach under quad-tree and QSkd-tree, and found that though quad-tree took less CPU time than QSkd-tree, the cost of building a quad-tree is far more than the QSkd-tree. In conclusion, the EnhancedPE is the most reasonable in all the top-down methods for the mCK query problem.

Chapter 7

Conclusions and Future Work

Recently, spatial keyword query problem becomes a hot topic in the field of spatial database. Quite a number of spatial keyword queries focus on finding a set of objects which combine to satisfy user's requirements about both textuality and spatiality. As an earlier study, mCK query first employed a 'diameter' to measure the closeness of a set of objects. After that, this manner is used in a lot of other spatial keyword query problems targeted to a set of objects such as collective spatial keyword queries and top-k group queries. Thus efficient processing of mCK queries is of great significance in this type of problems.

As a general idea, a top-down search schema by using hierarchical data structures is well suited to solve this type of query problems. Zhang et al proposed a top-down exploration approach (Apriori-Z) using a special R*-tree call bR*-tree. This approach enumerates some sets of nodes (node-sets) level-by-level in the bR*-tree and prune unnecessary ones to improve search efficiency. However due to the Apriori-based enumeration of node-sets, this approach may enumerate too many node-sets especially when the optimal object-set does not gather in one small node. Thus in this thesis, the main contribution of our work is to learn about restricted factors in top-down search approach of mCK query problem, and to propose four top-down search methods to improve search efficiency.

In addition , the assumption that store all the objects by using a bR*-tree in preparation is not applicable to all cases of real spatial web. Thus, we adopted an on-the-fly way to build index for these spatial web data. When an mCK query is given, we create grid partitionings or kd-trees from necessary data. Under this assumption, we list our proposed methods as

follow:

- **DCC-NL** When a node-set is generated in top-down process, whether or not it can be pruned depends on the comparison between the lower bound of this node-set and the smallest diameter discovered so far (δ^*). Thus we considered that if we can find a smaller δ^* at an early stage, then the pruning ability of node-sets can be improved. For this reason, we proposed our first algorithm DCC-NL. Because the diameter of an object-set is determined by only two objects, DCC-NL first enumerates DCs (the pairs of two nodes) in an ascending order. Then the node-sets are generated for each DC in a nested loop method. Thus a priority order of generation is given. The evaluation for DCC-NL results to the fact that it is efficient to find a smaller diameter at an early stage.

- **RDCC** After that, we improved the DCC strategy in a recursive way. This strategy can enumerate node-sets in a more reasonable priority order so that a smaller diameter can be found earlier than DCC-NL. Furthermore, we employed a tighter lower bound (TLB) of node-set to ultimately improve the pruning ability. This method is called RDCC. Though the search performance of RDCC is better than DCC-NL and Apriori-Z, the enumeration of node-sets is still an unstable factor that limits the search efficiency.

- **Pairwise Expansion** To solve this problem, we proposed Pairwise Expansion (PE) method, which can skip the intermediate process of enumerating node-sets. PE first enumerates object-pairs in an ascending order by using a top-down exploration of node-pairs according to the method to Closest Pair Query (CPQ). Then it expands each object-pair into object-sets in the shuttle area, in order to test if the object-pair becomes a diameter of any object-sets satisfying all query keywords. Once an object-pair passed the test, it is the result of the smallest diameter. PE can work well in a larger scale of real spatial web dataset, though it has some weak points about the construction of data index.

- **EnhancedPE** Finally, we improved the PE by using two techniques. First we adopted a kd-tree based data structure called QSkd-tree to reduce the cost of building time of

index. Then we proposed convex-hull based lower bound and upper bound to improve the pruning ability. These two techniques can significantly reduce the cost of building time of index and decrease the enumerated node-pairs and checked object-pairs. Hence the overall search performance can be improved. To the best of the ours knowledge, this method is the most efficiency top-down approach for mCK query problem.

As future works,

- Our top-down search approach may also be employed in other queries to find one or top-k set(s) of objects. We learned that in the collective spatial keyword queries [3], Cao et al also used enumeration of node-sets in top-down search process to find the exact solution. It is possible to use a top-down exploration of node-pairs for these queries , and the more appropriate pruning techniques may be generated.

- In our search approach, we enumerated object-pairs in an ascending order by using a top-down node-pair exploration. Thus it can be naturally extended to find top-k smallest diameter. In addition, the diversity of these top-k results may be considered.

Bibliography

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

[4] X. Cao, G. Cong, T. Guo, C. S. Jensen, and B. C. Ooi. "Efficient processing of spatial group keyword queries." ACM Trans. Database Syst., 40(2):13, 2015.

[5] G. Cong, Christian S. Jensen. "Querying Geo-Textual Data: Spatial Keyword Queries and Beyond." SIGMOD, pp. 2207-2212 , 2016.

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

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

[8] 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, APRIL 2011, pp. 585-599

[9] T.Guo, X.Cao, G.Cong, "Efficient Algorithms for Answering the m-Closest Keywords Query. " ACM SIGMOD, pp. 405-418, 2015.

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

[11] A.Corral, Y.Manolopoulos, Y.Theodoridis, M.Vassilakopoulos"Closest pair queries in spatial databases. " ACM SIGMOD, pp. 189-200, 2000.

[12] Y.Qiu, T.Ohmori, T.Shintani, H.Fujita, "A New Algorithm for m-Closest Keywords Query over Spatial Web with Grid Partitioning. " IEEE/ACIS SNPD, pp. 507-514, 2015.

[13] P.K. Agarwal, and J. Erickson, "Geometric range searching and its relatives. "In Discrete and Computational Geometry: Ten Years Latter., (B. Chazelle, E. Goodman, and R. Pollack eds.), American Math. Society, Providence, 1998.

[14] https://twitter.com/

[15] https://www.flickr.com/

[16] I.D.Felipe, V.Hristidis, and N.Rishe. "Keyword searchon spatial databases. " IEEE ICDE, pp. 656—665, 2008.

[17] D. Wu, M. L. Yiu, and C. S. Jensen. "Moving spatial keyword queries: Formulation, methods, and analysis. " ACM Trans. Database Syst., 38(1):7, 2013.

[18] D. Wu, M. L. Yiu, C. S. Jensen, and G. Cong. "Efficient continuously moving top-k spatial keyword query processing." IEEE ICDE, pp. 541—552, 2011.

[19] L. Chen, X. Lin, H. Hu, C. S. Jensen, and J. Xu. "Answering why-not questions on spatial keyword top-k queries." IEEE ICDE, pp. 279—290, 2015.

[20] J. Lu, Y. Lu, and G. Cong "Reverse spatial and textual k nearest neighbor search." ACM SIGMOD, pp. 349-360, 2011.

[21] J. Fan, G. Li, L. Zhou, S. Chen, and J. Hu. "SEAL:spatio-textual similarity search." PVLDB, 5(9):824—835, 2012.

[22] J. B. Rocha-Junior and K. Nrv ag. "Top-k spatial keyword queries on road networks." EDBT, 168—179, 2012.

[23] X. Cao, L. Chen, G. Cong, C. S. Jensen, Q. Qu, A. Skovsgaard, D. Wu, and M. L. Yiu. "Spatial keyword querying." ER, pp. 16—29, 2012.

[24] N. Roussopoulos, S. Kelley, and F. Vincent. "Nearest neighbor queries." ACM SIGMOD, pp. 71—79, 1995.

[25] A. Corral, Y. Manolopoulos, Y. Theodoridis and M. Vassilakopoulos. "Multi-Way Distance Join Queries in Spatial Databases." GeoInformatica, vol. 8, no. 4, pp. 373-402, 2004.

[26] A. Corral, Y. Manolopoulos, Y. Theodoridis and M. Vassilakopoulos. " Distance Join Queries of Multiple Inputs in Spatial Databases." ADBIS, LNCS, vol. 2798, pp. 323—338, 2003 .

[27] C.-T. Ho , R. Agrawal , N. Megiddo , R. Srikant. "Range queries in OLAP data cubes. " ACM SIGMOD, pp. 73-88, 1997.

[28] https://www.flickr.com/groups/geotagging/

[29] https://developers.google.com/places/web-service/

[30] https://searchenginewatch.com/sew/study/2343577/google-local-searches-lead-50-of-mobile-users-to-visit-stores-study

[31] A. Cary, O. Wolfson, and N. Rishe. " Efficient and scalable method for processing top-k spatial Boolean queries." SSDBM, pp. 87—95, 2010.

[32] G. Cong, C. S.Jensen, and D. Wu. "Efficient retrieval of the top-k most relevant spatial web objects." PVLDB, pp. 337—348, 2009.

[33] J. B. Rocha-Junior, O. Gkorgkas, S. Jonassen, and K. Nrv ag. " Efficient processing of top-k spatial keyword queries." SSDT, pp. 205—222, 2011.

[34] S. Vaid, C. B. Jones, H. Joho, and M. Sanderson. " Spatio-textual indexing for geographical search on the web." SSDT, pp. 218—235, 2005.

[35] M. Christoforaki, J. He, C. Dimopoulos, A. Markowetz, and T. Suel. " Text vs. space: efficient geo-search query processing." CIKM, pp. 423—432, 2011.

[36] K. S. Bgh, A. Skovsgaard, and C. S. Jensen. "GroupFinder: A new approach to top-k point-of-interest group retrieval. " PVLDB, 6(12):1226—1229, 2013.

[37] A. Skovsgaard and C. S. Jensen. "Finding top-k relevant groups of spatial web objects. " VLDB J., 24(4):537—555, 2015.

[38] X. Cao, L. Chen, G. Cong, and X. Xiao. "Keyword-aware optimal route search. " PVLDB, 5(11):1136—1147, 2012.

[39] B. Yao, M. Tang, and F. Li. "Multi-approximate-keyword routing in GIS data. " SIGSPATIAL, pp. 201—210, 2011.

[40] G. Li, J. Feng, and J. Xu. "Desks: Direction-aware spatial keyword search. " IEEE ICDE, pp. 474—485, 2012.

[41] D. Wu, M. L. Yiu, C. S. Jensen, and G. Cong. "Efficient continuously moving top-k spatial keyword query processing. " IEEE ICDE, pp. 541—552, 2011.

[42] W. Huang, G. Li, K.-L. Tan, and J. Feng. "Efficient safe-region construction for moving top-k spatial keyword queries. " CIKM, pp. 932—941, 2012.

[43] D. Wu, M. L. Yiu, and C. S. Jensen. "Moving spatial keyword queries: Formulation, methods, and analysis." ACM Trans. Database Syst., 38(1):7, 2013.

[44] J. Lu, Y. Lu, and G. Cong. "Reverse spatial and textual k nearest neighbor search. " ACM SIGMOD, pp. 349—360, 2011.

[45] F. Choudhury, J. S. Culpepper, T. Sellis, and X. Cao. "Maximizing bichromatic reverse spatial and textual k nearest neighbor queries. " PVLDB, 9(6):456—467, 2016.

[46] Z. Li, K. C. K. Lee, B. Zheng, W.-C. Lee, D. L. Lee, and X. Wang. "Ir-tree: An efficient index for geographic document search." IEEE TKDE, 23(4):585—599, 2011.

[47] A. Khodaei, C. Shahabi, and C. Li. Hybrid indexing and seamless "ranking of spatial and textual features of web documents." In DEXA, pp. 450—466, 2010.

[48] A. Guttman. "R-trees: a dynamic index structure for spatial searching." In SIGMOD, pp. 47—57, 1984.

[49] J.Elzinga, D.W.Hearn. "Geometrical solutions for some minimax location problems. " Transportation Science,6[4], pp. 379—394, 1972.

Acknowledgements

The study of PhD has been a truly memorable experience for me and it would not have been possible to do without the support and guidance that I received from many people.

First and foremost, I would like to thank my supervisor, Prof. Tadashi Ohmori, for the patient guidance, encouragement and advice he has provided throughout my graduate career. I have been extremely lucky to have a supervisor who cared and supported so much about my work. I would also like to thank all the members of staff at university who helped me in all aspects of my life. In particular I would like to thank Assoc.Prof. Takahiko Shintani and Assist.Prof. Hideyuki Fujita for their help and suggestions in my PhD study.

I would like to thank the rest of my supervisory committee: Prof. Yasuhiro Minami, Prof. Hiroyoshi Morita, Assoc.Prof. Hisashi Koga, and Assoc.Prof. Yasuyuki Tahara for their encouragement and insightful comments from various perspectives.

I must express my gratitude to my wife and my son, for their continued support and encouragement. I warmly thank and appreciate my parents and my mother and father-in-law for their material and spiritual support.

List of Publications Related to the Dissertation

Journal Paper

- 邱 原, 大森 匡, 新谷 隆彦, 藤田 秀之“ m-最近接空間キーワード検索における探索優先順制御とタイトな下界値を用いた高速化手法の提案,” 電子情報通信学会論文誌 D, VOL.J99-D No.7 pp.638-651, 2016 年 6 月.

International Conference Papers

- Y. Qiu, T. Ohmori, T. Shintani, H. Fujita ”A New Algorithm for m-Closest Keywords Query over Spatial Web with Grid Partitioning,” Proceedings of 16th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD 2015), pp.507-514, June 2015.

- Y. Qiu, T. Ohmori, T. Shintani, H. Fujita ” Pairwise Expansion: A New Topdown Search for mCK Queries Problem over Spatial Web,” Proceedings of 18th Asia Pacific Web Conference (APWeb2016), short paper(LNCS 9932 Volume 2), pp.459-463, September 2016.

International Conference Poster Presentation

- Y. Qiu, H. Zhai, D. H. Anh, T. Ohmori, H. Fujita and T. Shintani ”A new search strategy for m-closest keywords query by using dynamically-created grid partitioning over spatial datasets,” CIU2015, p2, March 2015.

Domestic Convention Papers

- 邱 原, 大森 匡, 新谷 隆彦,“ 空間データにおける 2^n 分割木を用いた m-最近傍キーワード検索,” DEIM 2013, A9-5, 2013 年 3 月.

- 邱 原, 大森 匡, 新谷 隆彦, 藤田 秀之, ”空間データベースにおける m-最近接キーワード検索の一方式,” 76 回情報処理全国大会 5M-1, 2014.(学生奨励賞)

- D.H.Anh, 邱 原, 大森 匡, 藤田 秀之, 新谷 隆彦, ”Flickr データを用いた m-最近傍キーワード検索の評価,” 第 76 回情報処理全国大会 5M-2, 2014.
- 邱 原, 大森 匡, 新谷 隆彦, 藤田 秀之 ”空間 Web データにおける m-最近接キーワード検索方式 DCC の性能評価,” FIT2014, D-005, 2014.
- 邱 原, 大森 匡, 新谷 隆彦, 藤田 秀之 ”再帰的な DCC 戦略による mCK 検索の高速化,” FIT2015, D-030, 2015.(FIT 奨励賞)
- 杜 翔, 大森 匡, 藤田 秀之, 新谷 隆彦, 邱 原 ”データ取得制限のある Deep Web からのサンプルデータ収集方式,” FIT2015, D-028, 2015.

A Study on Top-down Search Algorithms for m-Closest Keywords Queries Problem over Spatial Web Yuan Qiu 2017