JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 35, 903-921 (2019)
DOI: 10.6688/JISE.201907_35(4).0012

Efficient Spatial Keyword Search Methods for Reflecting Multiple Keyword Domains

BUMJOON JO, JUNHONG AHN AND SUNGWON JUNG
Department of Computer Science and Engineering
Sogang University
Seoul, 04107 Korea
E-mail: {bumjoonjo; junhongahn; jungsung}@sogang.ac.kr

In this paper, we propose multiple keyword domain-based spatial keyword search queries, called the Multiple Keyword Domain based range (MKDR) query and *k*-Nearest Neighbor (MKDkNN) query, and their query processing algorithms. The proposed queries retrieve objects that satisfy the searching conditions for the given environmental conditions of object as well as their spatial and textual relevance. The proposed queries consist of two sub-queries. The first sub-query, called the primary query, identifies a group of geo-textual objects that satisfy the requirements for spatial and textual relevance of the query. The second sub-query, called the refining range query, identifies the geo-textual objects that satisfy the requirement for environmental conditions of objects. Because the existing methods for spatial keyword queries cannot efficiently handle the proposed queries, we first categorize the data according to their domains of keywords and simultaneously search multiple indexes constructed for objects in each domain. Since our methods prune the nodes that cannot satisfy environmental conditions in the earlier stage of searching, they reduce the number of refining range queries for MKDR and MKDkNN. Our experimental performance analyses show that our proposed query processing algorithms significantly reduce the query response times of MKDR and MKDkNN.

Keywords: spatial keyword query, geo-textual object, multiple keyword domain based range (MKDR) query, multiple keyword domain based kNN (MKDkNN) query, IR-trees collaboration algorithm

1. INTRODUCTION

With the growing popularity of micro blogs in recent years, various services that use geo-textual data have received significant attention. Geo-textual data refers to data that contain both geographical information and textual contents. For example, tweets, reviews, news comments, and other social media contents are tagged with their localities and the keywords of their context. Unlike the spatial data retrieval and the document retrieval, geo-textual data retrieval has a unique characteristic that requires spatial proximity and textual relevance simultaneously. In order to satisfy various demands on geo-textual data retrieval, a significant amount of research has been done to develop various kinds of spatial keyword queries and their query processing methods.

However, in our observation, there is a lack of queries to search for objects satisfying requirements for the environmental conditions of the object. The environmental condition refers to which other interested objects are located in the near the object. For example, the existing spatial keyword query, Boolean range query, can be given as the following statement: “Find the hotels that provide ‘free Wi-Fi’ and ‘twin bed room’ within

Received January 30, 2018; revised May 15, 2018; accepted July 27, 2018.
Communicated by Jianliang Xu.

1km from the bus station." In this example, '1km' is a requirement for spatial relevance, and the keywords 'Wi-Fi' and 'twin bed room' are requirements for textual relevance. However, sometimes we need the additional filtering conditions for surrounding environment, such as "There should be a Chinese restaurant selling 'Beijing roast duck' nearby the hotel." Although the requirements for these search conditions are often used in our daily life, the queries for filtering objects by their environmental conditions are not yet developed.

In order to deal with the requirements for these search conditions, we define two types of novel spatial keyword queries, called the Multiple Keyword Domain based range (MKDR) query and the Multiple Keyword Domain based *k*-Nearest Neighbor (MKDkNN) query by extending the existing spatial keyword queries. The proposed queries filter objects by environmental conditions of the object as well as query range and query keywords. In the proposed queries, the search conditions for the environmental conditions of an object are given as the sub query, in the form of the Boolean range query. This sub query, named refining range query, searches around the object to identify whether the objects of the desired type are located nearby.

In order to effectively process our proposed queries, we present a query processing method. Since the proposed queries should access various types of objects to evaluate the environmental conditions of the target objects, the existing methods using a single spatial keyword index are not efficient to process our queries. Suppose that we construct a single IR-tree on a geo-textual database that covers all types of the objects used to evaluate environmental conditions for these objects. Then the geo-textual objects which contain the keywords of the query can be grouped together with unrelated objects, and we might need to access a significant amount of node to find the objects of interest. Moreover, increasing the size of index and inverted file makes the search process more inefficient.

In our query processing methods, we first divide the geo-textual database according to keyword domain of objects and construct IR-trees [1] in each domain to construct small sized index and grouping textual related objects. Based on this multiple keyword domain based indexes, our query processing algorithms cooperatively search multiple indexes to prune nodes that cannot satisfy the environmental conditions in the earlier stage of search. The advantages of our methods are as follows. First, they can process more efficiently because the size of indexes and inverted files of index are relatively small. Second, they can prune the node that cannot satisfy the environmental condition in the earlier stage of search. Finally, the proposed algorithms minimize the repeated search for the same node of the indexes because multiple indexes share the intermediate result for cooperative searching process.

The rest of this paper is organized as follows. In Section 2, we discuss the related work. In Section 3, we describe formal definitions for the MKDR query and MKDkNN query. In Section 4, we propose a collaboration algorithm of multiple IR-trees for efficient processing of the MKDR and MKDkNN queries. In Section 5, we evaluate the performance of our method through experiments. Finally, Section 6 concludes this paper.

2. RELATED WORK

2.1 Spatial Keyword Queries

The query for geo-textual data is characterized by the spatial proximity used in the

spatial database and the textual relevance used in the information retrieval. In this viewpoint, Gao Cong [2] defines standard queries of spatial keyword queries by combination of information retrieval queries – such as Boolean queries and ranking based queries – and spatial queries – such as range query and kNN query –. The name of each standard spatial keyword query and their definitions are as follows.

- **Boolean range query**
  A Boolean range query returns all objects that are located in the query region and that contain all the query keywords.
- **Boolean kNN query**
  A Boolean kNN query returns up to ***k***-nearest objects. Each of which contains all the query keywords.
- **Top-*k* range query**
  A top-*k* range query returns up to ***k*** objects with the most textual relevance. The result objects must also be located within the query region.
- **Top-*k* kNN query**
  A top-*k* kNN query returns ***k*** objects that have the highest textual and locational relevance.

As the definition of each query, the keywords and location of objects, which are explicitly stored as attribute values of individual objects, are directly compared to the query point and query keywords for query processing. However, the ***m***-closest keywords (mCK) query [3] and collective queries [4, 5] having implicit conditions as search conditions, it becomes important what kinds of objects are located around the object. The mCK query retrieves a set of the closest objects whose combined textual descriptions cover ***m*** query keywords. In the case of mCK query, the similarity between individual object and query keywords are less important than what objects are located in nearby object. The collective keyword query is a query that combines the mCK query and spatial range query. It retrieves a set of objects that their combined textual descriptions collectively cover the query keywords, and are located closely each other as well as all of them are located within the query region.

As the mCK query and collective spatial keyword queries, the relation between objects can be used as a third filtering condition together with spatial proximity and textual relevance. However, in our observation, there is no query that simultaneously examines all of these three conditions to search individual object. To satisfy this requirement, we develop a new type of query that defines the environmental condition of the object as a third search condition in section 3.

2.2 Indexing Techniques for Spatial Keyword Data

Most indexing techniques for geo-textual data are also designed to meet the requirements of standard queries. The key idea of these methods is to allow objects to be filtered by using keywords and spatial information simultaneously in the search process. To achieve this goal, various techniques for combining inverted files with spatial indexing techniques have been developed [6-9]. For example, IR-tree [1] and other R-tree based methods construct an R-tree using spatial information of the geo-textual data, and

combine each node with an inverted file and a document summary that describe the textual information for the objects in their sub-tree. The inverted file indicates which keyword is associated with which child node, and the document summary provides a summary of textual information of the documents in the sub-tree of the node. The limitation of these methods using inverted indexes is that they require a lot of resources to maintain and retrieve large inverted files. Moreover, as the number of keywords increases, the textual relevance of the data which are grouped together should decrease because there is no consideration to preserve the textual relevance of grouped objects.

S2I [10] and I3 [11] pointed out these problems and proposed methods for decomposing objects by keyword and managing them individually. In these methods, the geo-textual objects are decomposed into multiple key-value pairs, and indexes are only provided for objects containing frequently appearance keyword. These methods show superior performance for single keyword queries because the sizes of separate indexes are relatively smaller than a single-domain-based database, and the textual relevance of grouped objects are maximized. However, these methods are not suitable for processing our proposed queries because they require refining process for duplicate searching for the decomposed object.

3. PROBLEM DEFINITION AND MULTIPLE KEYWORD DOMAINS

3.1 Multiple Keyword Domains based Spatial Keyword Queries

Multiple Keyword Domains based Range Query

Geo-textual data can be viewed as a spatial object that contains a set of keywords. Let O be a database consisting of a set of geo-textual objects. Each geo-textual object $o \in O$ is defined as a pair $(o.\rho, o.\Psi)$, where $o.\rho$ is a two-dimensional geographical point location and $o.\Psi$ is a set of keywords. The MKDR query retrieves geo-textual objects that satisfy three conditions: the objects are located within query range, contain a set of query keywords, and must pass an environmental evaluation. Two sub-queries, called primary range query and refining range query, are used to examine these conditions. The primary range query takes three arguments, a set of keywords, query range, and query point to retrieve the geo-textual objects that are located within query range and contain the query keywords. The refining range query examines the environmental conditions of objects. It takes a keyword-range pairs as arguments, and investigates each result of the primary range query to identify whether there is an object with a refining keyword within the given refining range. Note that an MKDR query may take multiple refining range queries. We formally define the MKDR query and its sub-queries as follows.

Definition 1: Multiple Keyword Domains based Range (MKDR) query An MKDR query $q_{MKDR} = \langle p, r_P, w_p, \{(w_{R1}, r_{R1}), (w_{R2}, r_{R2}), \ldots, (w_{Rm}, r_{Rm})\}\rangle$ takes four arguments; query point p, query range r_P, a set w_P of keywords for primary range query, and a set of keywords-range pairs (w_{Ri}, r_{Ri}) as refinement conditions for refining range queries. The result of an MKDR query, $q_{MKDR}(O)$, is then the intersections of the results of primary range query and the results of all the refining range queries corresponding to the given refinement conditions.

Definition 2: Primary Range Query (PRQ) The primary range query $q_{PRQ} = \langle p, r_P, w_P \rangle$ takes three arguments, where p is a query point, r_P is a range of query and w_P is a set of keywords. The result of PRQ, $q_{PRQ}(O)$, is a set of objects such that $\forall o \in q_{PRQ}(O)(dist(p, o.\rho) < r_p \wedge w_p \subseteq o.\Psi)$.

Definition 3: Refining Range Query (RRQ) A refining range query $q_{RRQ} = \langle w_R, r_R \rangle$ accepts a set w_R of keywords and range r_R pair as arguments. The result of an RRQ, $q_{RRQ}(O)$, is a subset of O containing objects such that $\forall o \in q_{PRQ}(O)((\exists o' \in O \backslash q_{RRQ}(O))(dist(o'.\rho, o.\rho) < r_R) \wedge w_R \subseteq o'.\Psi)$.

A running example of an MKDR query is shown in Fig. 1. For simplicity of example, each keyword set for sub queries contains only a single keyword. The objects have a one of three types of keyword, w_1, w_2, and w_3. Suppose that the MKDR query $q_{MKDR} = \langle q, r_1, w_1, \{(r_2, w_2)\} \rangle$ is issued. In this example, the number of refining queries is one (i.e., $m = 1$). For the primary range query, objects d_1 and d_9 are returned because they are located within range r_1 and contain the keyword w_1. Then, the refining range query is processed by using this result set. Object d_1 satisfies the condition because an object d_7 of keyword w_2 exists within the range r_2. However, the object d_9 does not satisfy the condition of the refining query because there is no object within the query range r_2. Therefore, object d_1 is returned as the result of the MKDR query.

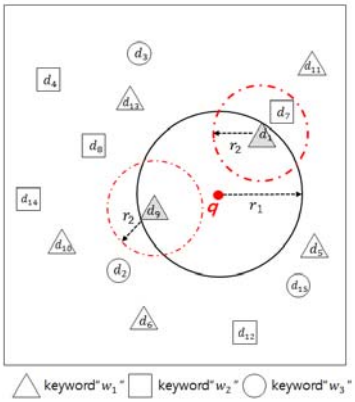


Fig. 1. A running example of an MKDR query.

Multiple Keyword Domains based *k*-Nearest Neighbor Query

The MKDkNN query retrieves the k geo-textual objects nearest from the user's current location. The objects in the result set of MKDkNN query must contain query keywords and also satisfy the environmental conditions. The MKDkNN query can be divided into two sub-queries – primary kNN query and refining range query. The primary kNN query takes three arguments, a set of keywords, the number of nearest neighbors, and the query point. It finds the top-k closest geo-textual objects that contain the query keywords. The refining range query, same as described in Definition 3, is also used to examine the environmental evaluation of objects for the MKDkNN query. Formal definitions of the MKDkNN query and its sub-queries are as follows.

Definition 4: Multiple Keyword Domain based *k*-Nearest Neighbor(MKDkNN) query An MKDR query $q = \langle p, k, w_p, \{(w_{R1}, r_{R1}), (w_{R2}, r_{R2}), \ldots, (w_{Rm}, r_{Rm})\}\rangle$ takes four arguments; query point p, the number of neighbors k, a set w_p of keywords for primary kNN query, and a set of refinement conditions (w_{Ri}, r_{Ri}) for refining range queries. The result of an MKDkNN query, $q(O)$, is then the intersections of the result of primary kNN query and the results of all the refining range queries corresponding to the given refinement conditions.

Definition 5: Primary kNN Query (PkNNQ) The primary kNN query PkNNQ $q_{kNNQ} = \langle p, k, w_P\rangle$ takes three arguments, where p is a query point, k is the number of neighbors and w_p is a set of keywords. The result of PkNNQ, $q_{kNNQ}(O)$, is a set of k objects, each of which covers all the keywords in w_p. The objects are ranked according to their distances from p. Formally, $\forall o \in q_{kNNQ}(O)((\nexists o' \in O \backslash q_{kNNQ}(O))(dist(p, o'.\rho) \leq dist(p, o.\rho)) \wedge w_p \subseteq o'.\Psi)$.

A running example of an MKDkNN query is shown in Fig. 2. Similar to Fig. 1, there are three types of keyword, w_1, w_2, and w_3. For a given MKDkNN query $q_{kNNQ} = \langle p, 2, w_1, \{r_1, w_2\}\rangle$, the 2-nearest objects of primary kNN Query are d_4 and d_8. However, considering with environmental condition of refining query, d_4 cannot be the result of query because there is no corresponding object nearby of it. Therefore, the object d_8 and d_7, that are located far away from q than d_4, is selected result set of the query.

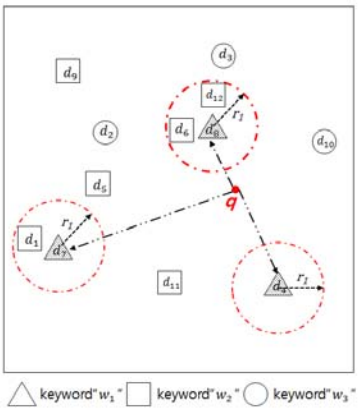


Fig. 2. A running example of an MKDkNN query.

3.2 Multiple Keyword Domains based Spatial Keyword Data Indexing

The most time-consuming process of the multiple keyword domain based queries is the refining process. If we first process a primary query, then a large number of refining range queries would be processed to refine a result set of the primary query. In order to reduce the search space for each sub queries, we categorize objects according to their keyword relevance, and construct small-sized individual indexes for each group of objects. We call the keywords of each group as a keyword domain. If the domain of data is

ambiguous, the database is required to be artificially split into several domains. The existing keyword clustering techniques can be used to split domains. There are various methods of keyword clustering, but we do not discuss each clustering method in this paper because that is out of the focus of this paper.

Fig. 3 shows the overall structure of our multiple domain based spatial keyword indexes. On top of the structure, keyword domain classification layer is implemented to find the appropriate domains for query keywords. The keyword domain classification layer contains a keyword index (such as B^+ tree) for each keyword and a pointer to the root node of the spatial keyword index of the corresponding domain. We used the IR-tree as an index for the individual domains. When processing the query, the keyword classifier first finds the domains of query keywords through the keyword index, and explores the IR-trees of the corresponding domains. If the same keyword appears in multiple domains at the same time, objects containing the keyword are replicated to each domain.

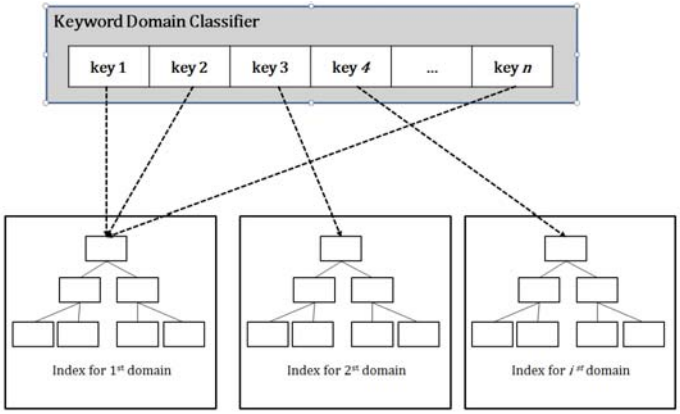


Fig. 3. The overall structure for multiple keyword domains based index.

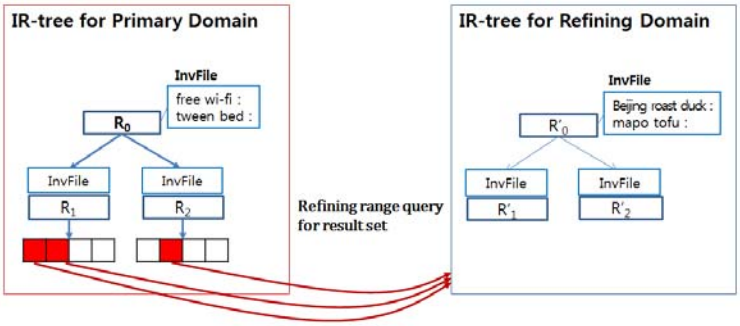


Fig. 4. A naïve algorithm for multiple keyword domains based query.

Fig. 4 shows the naïve algorithm for processing MKD and MKDkNN queries based our multiple keyword domain based index structure. The algorithm first finds the domain of the primary query (abbreviated as DP) through the keyword domain classifier and searches the index of the DP for the processing primary query. When the primary query

process is finished, the refining domain (abbreviated as DR) is searched in the same way, and each object of the result set is used as a query point to process the refining.

4. MULTIPLE KEYWORD DOMAIN BASED SPATIAL KEYWORD QUERY PROCESSING TECHNIQUES

4.1 Approximate Refinement for Multiple Domain based IR-trees

The multiple IR-trees not only reduce the search space of the database required to process the primary and refining queries, but also give us additional pruning chance during index searching. Notice that the number of refining range queries in the naïve algorithm is equal to the size of the result set for the primary query. If the nodes or objects that will be removed by the refining range query are filtered during primary query processing, the number of refining range queries can be reduced. Based on this approach, we define an approximate refinement range for cooperation among multiple IR-trees. The basic idea of the approximate refinement range is to identify nodes that are not likely to satisfy environmental conditions at an earlier stage of the search.

Fig. 5 illustrates an example of an expanded range for the approximate refinement. Based on the size and position of the current node ϵ to be traversed in DP, the expanded range is derived from expanding the refinement range R_i to the MBR. If there is an overlapped node of IR-tree in the DR within the expanded range, the current traversed node is classified as a node that is likely to satisfy the condition. If there is no object or node within the expanded range in the DR, the current traversed node is removed from the searching plan without waiting for the results of the primary range query. Fig. 6 illustrates an example of an approximate refinement for DR. In this example, MBR N_2 is considered to satisfy the environmental conditions because there are nodes of the refinement domain overlapping with the extended range.

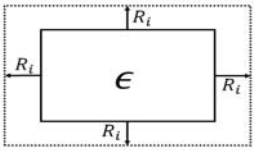


Fig. 5. Expanded range of the node in DP.

(a) MBRs of IR-tree in the DP. (b) Extended Range ER based on N_2 for approximate refinement.

Fig. 6. Example of an approximate refinement.

Even if the approximate refinement provides additional pruning opportunities, performing an approximate refinement whenever we visit a node is still costly. The reason is that, when we try to perform refining query on adjacent objects, we should perform a large number of repetitive search on the same node, for example, the root of the refining domain. In order to eliminate this inefficiency, we simultaneously traverse multiple IR-trees and record intermediate result for approximate refinement for each node. Because the geo-textual objects are already grouped by their spatial proximity, adjacent objects that have the same parent node can share intermediate results of the refining process. Therefore, we store the nodes that are searched in the DP in the stack with the list of DR nodes overlapped the approximate refining range. This allows the refining query for the child node to start searching from the node list stored in the parent node instead of searching for a common ancestor node. Fig. 7 illustrates the node stack for search primary query and a list of associated nodes in DR when we search node R_5 in depth first search order. In next iteration of the example, the refining process of objects contained in node R_5 is started from R'_6 of DR directly.

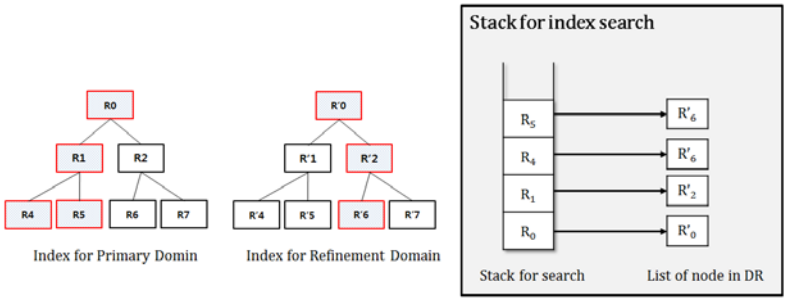


Fig. 7. An example for stack to search the node R_5.

4.2 Multiple IR-trees Collaboration Algorithm for the MKDR

An effective strategy for searching the IR-trees for the primary and refining range queries is through the cooperation of multiple IR-trees using an approximate refinement. To simplify the problem, we assume that we already know what domains are required for the refining query. Algorithm 1 shows the collaboration technique-based MKDR query processing algorithm. The stack $\boldsymbol{S}$ contains candidate nodes derived from the primary range query and nodes in DR overlap with the approximate refining range. $\boldsymbol{Q}$ is a set of node lists. Each node list $\boldsymbol{Q_i} \in \boldsymbol{Q}$ contains the intermediate result for approximate refining query in ith domain. Set $\boldsymbol{V}$ is a final result of MKDR query.

Algorithm 1: Generate a set of Multiple Keyword Domain based Range Query result

Input: Query point q, primary query range r_P, primary query keywords k_P, set K of keywords-range pairs $<k_{Ri}, r_{Ri}>$ for ith refining domain .

Output: A set of geo-textual objects

$V = \varnothing$

$Q = \varnothing$

for each domain i

```
    Add a root node of IR tree for DR_i to Q_i
// The element of S is a pair <node, Set of node list for refining query>
S.push(root node of IR tree for DP, Q)
while S is not empty do
    <node n, Set of node list Q> = S.pop()
    if n is not a leaf node then
        for each child node c of n
            if (dist(c, q) < r && ContainsKeywords(c, k)) then
                for each domain i
                    window range W = expand(c.MBR, r_i)
                    Q_i = (ApproximateRefining(W, k_i, Q_i))
                if (all of Q_i is not empty) then
                    S.push(c, Q)
    else if n is a leaf node then
        for each object o in n
            if(dist(o, q) < r && ContainsKeywords(o, k))
                for each domain i
                    if(RefiningRangeQuery(o, r_i, k_i, Q_i) returns true) then
                        continue process to next domain
            if all of refining range queries return true
                V = V∪{o}
return V
```

In initial stage of algorithm, ***S*** stores the root node of IR tree in primary domain, and set of the root nodes for multiple DR. If the candidate node in ***S*** is not a leaf node, the approximate refinement is processed by expanding the range of the MBR. A function *ApproximateRefining*() represented in Algorithm 2 traverses the IR-trees for DR, and returns a set of nodes overlapped with the expanded window range. If there is no node or object which overlapped by the expanded range, the algorithm returns an empty list and current node is not inserted into search plan.

If the candidate node is a leaf node, the algorithm examines the objects by using the function *RefiningRangeQuery*() shown in Algorithm 3. This function evaluates the environmental condition of object and returns a result as a Boolean value. After processing refining range queries, the algorithm inserts the object that passed refining range query into the result set ***V***. As a result, the result set of MKDR query contains elements within query range ***r***, contains the keywords of ***k***, and satisfies the conditions of the refining range queries.

Algorithm 2: Approximate Refining

Input: Query window *W*, keywords *k* of *i*th domain, list *L* of node for IR tree
Output: A node list *S* contains nodes overlapped with the query window for each node *n* in *L*

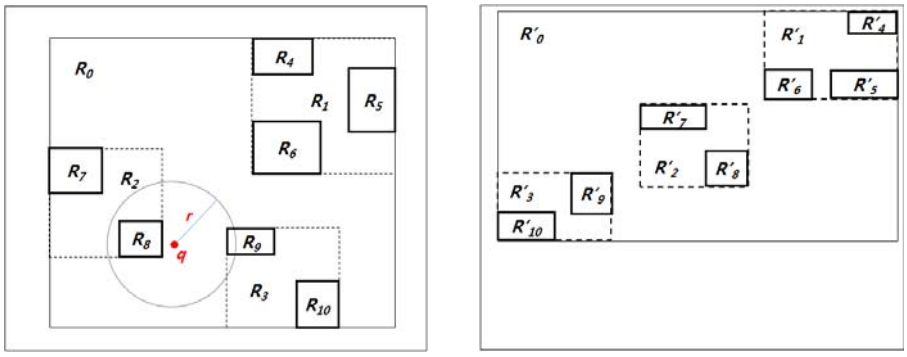
```
    if n is not a leaf node then
        for each child node c of n
            if W overlap with c.MBR && ContainsKeywords(c, k) then
                add c to node list S
```

```
    else if n is a leaf node
        add n to node list S
return S
```

Algorithm 3: Refining Range Query

```
Input: Geo-textual object o, query range r, keywords-range pairs
       <k_Ri, r_Ri>for ith refining domain, a node lists Q_i
Output: Boolean value for each node n in Q_i
    if n is not a leaf node then
        delete n in Q_i
        for each child c of n do
            if (dist(o, c) <= r && ContainsKeywords(c, k)) then
                Add c to Q_i
        else if n is a leaf node then
            for each object b of n
                if (dist(o, b) <= r && ContainsKeywords(b, k)) then
                    return true
return false
```

Fig. 8 illustrates an example of an MKDR query. The algorithm starts by inserting the root node of the IR-tree constructed in the DP. There are two children, R_2 and R_3, which overlap the query range. Therefore, the algorithm expands the range of the MBR for R_2 and R_3, and processes the approximate refinement. Because the expanded range of R_2 overlaps that of R'_3 in the DR, the children of R_2 are inserted into the searching queue. However, node R_3 is removed from the searching queue because there are no nodes overlapping with R_3 in the DR. In the next iteration, the children of R_2, R_7 and R_8 are examined using the primary query range and keywords. R_7 is located outside of the range. Therefore, only node R_8 is evaluated through the refining range query. As a result, the result of refining range query on R_8 is inserted into the result set, and the algorithm is terminated because there are no more nodes to traverse.



(a) IR-tree and primary range query in the DP. (b) IR-tree in the DR.

Fig. 8. Example of an MKDR query.

4.3 Multiple IR-trees Collaboration Algorithm for the MKDkNN

The algorithm for MKDkNN query processing is similar to the MKDR query processing algorithm. For effective querying, we search IR-trees in the order of best-first search (BFS) and maintain two priority queues: *P*, *R*. First, *P* stores nodes in ascending order of the minimum distance from the query point to their MBRs and *R* stores the candidate set of kNN in descending order by distance. The algorithm explores multiple IR-trees simultaneously for filtering nodes using an approximate refinement. Algorithm 4 shows the collaboration technique-based MKDkNN query processing algorithm. In each iteration, the algorithm explores the closest node among the nodes stored in *P*. If the candidate node is not a leaf node, the approximate refinement is processed by expanding the range of the MBR. The *ApproximateRefining*() function represented in Algorithm 2 also used in MKDkNN query processing algorithm. In this process, if the window query does not satisfy the approximate refinement, the child node is removed from *P*. On the other hand, when it satisfies the approximate refinement, the algorithm inserts its child nodes into *P*.

Algorithm 4: Generate a set of Multiple Keyword Domain based kNN Query result

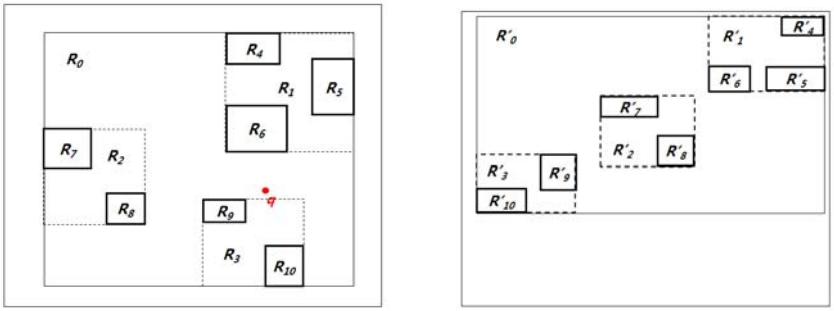
Input: Query point q, number of nearest neighbor K, primary query keywords k, keywords k_i of ith domain, query range r_i for ith domain
Output: A set of geo-textual objects

```
Q = ∅    // Q is a set of list of nodes. Each element Q_i contains intermediate result for approximate refining query in ith domain
P←CreatePriorityQueue()
R←CreatePriorityQueue() /*capacity of queue is k*/
for each domain i
    Q_i = Q_i∪{root node of IR tree for DR_i}
    P = P∪{root node of IR tree for DP, Q}
r = MAX_VALUE
while P is not empty do
    node n = Dequeue(P)
        if (dist(n, q) > r) then
            continue
        if n is not a leaf node then
            for each child node c of n
                if (dist(c, q) < r && ContainsKeywords(c, k)) then
                    for each domain i
                        window range W = expand(c.MBR, r_i)
                        Q_i = ApproximateRefining(W, k_i, Q_i))
                        if(Q_i is not empty) then
                            Enqueue(P, c)
        else if n is a leaf node then
            for each object o in n
                if (dist(o, q) <= r && ContainsKeywords(o, k)) then
                    for each domain i
                        if (RefiningRangeQuery(o, r_i, k_i, Q_i) returns true) then
```

Enqueue(*R*, *o*)
if (size of *R* > *K*)
Dequeue(*R*) until size of *R* is equal to *k*
Update *r* to farthest distance of candidates
return *R*

If the candidate node is a leaf node, the algorithm inserts the object into the candidate set *R* if it satisfies the conditions of the refining queries. If the number of current candidate objects exceeds *k*, the farthest candidates are dropped from the queue. The query range *r* is updated to the distance of the farthest candidate whenever each insertion *R* occurs. If there is no node or object that is closer than current range *r*, the algorithm is terminated and current candidates are returned to answer.

Fig. 9 illustrates an example of an MKDkNN query. We will find a single nearest neighbor in this example. The algorithm starts with inserting the root node of the IR-tree constructed in the DP. The algorithm examines the child nodes, R_1, R_2 and R_3, and inserts R_1 and R_2 into the priority queue *P* with the corresponding refinement nodes R'_1 and R'_3. Because the minimum distance of R_1 is smaller than R_2, R_1 is explored first. The nodes R_5 and R_6, the child of R_1, are evaluated by approximate refinement queries, and inserted into the queue with their overlapped node in DR. The priority queue *P* has three nodes at this iteration, in order of R_6, R_5 and R_2. Now the objects containing in R_6 are evaluated by refining query from R'_6 and an object is inserted into a result set. After inserting the candidate object into a result set, the range *r* is updated to the distance of candidate result, and search the remaining node stored in *P*. However, because the minimum distance for remaining nodes R_5 and R_2 are farther than current candidate result, the algorithm is terminated and returns the result set.



(a) IR-tree and primary query point in the DP. (b) IR-tree in the DR.

Fig. 9. Example of an MKDkNN query.

5. PERFORMANCE EVALUATION

In this chapter, we describe the experimental evaluation of the proposed algorithm. We compare two different methods by measuring the average response time for 100 randomly generated queries. First, our baseline algorithm, tagged *IRTreeSeq* in the figures, refers to the naïve algorithm presented in section 3.2. It first processes primary

queries and then processes the refining range query for refining result set of primary query. The second method, tagged *CMIRTree* in the figures, refers to our collaboration algorithm for multiple IR-trees. The experiments were performed on a physical machine consisting of a 3.60 GHz quad-core processor, 16 GB RAM, and a 1 TB HDD that operates on a 64-bit Linux operating system. Three of datasets are used in our experiments. The first dataset is synthetically generated one million objects with uniform distribution. It divided into 10 domains, and contained 100,000 geo-textual objects per domain. The second dataset, named *Euro dataset*, is a real dataset that contains points of interest in Europe. The last dataset, named *Korean building dataset* contains a list of buildings classified according to building purpose. This dataset has 10,000,000 objects for buildings in the Republic of Korea, and objects are categorized according to 270 kinds of domains. The summarization of properties for each dataset is described in Table 1.

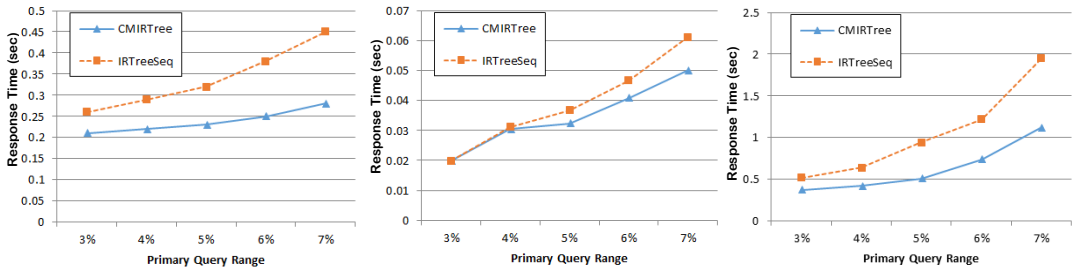
Table 1. Dataset properties.

Dataset Name	Type	#Domain	#Object
Synthetic	Synthetic, Uniform	10	1,000,000
Euro	Real	29	184,612
Korean Building	Real	270	10,000,000

5.1 Performance Studies for MKDR Query

Effect of the Primary Query Range

To evaluate the effect of the range of the primary range query, we increased the range from 3% to 7% of the space. In this experiment, the refining query range is fixed to 20% of the primary query range. In Fig. 10, the experimental results of *IRTreeSeq* and *CMIRTree* are plotted in terms of the average response time. As shown in the figure, the collaboration algorithm not only demonstrated better performance, but was also less affected by the primary query range. This is because when the primary query range is increased, the number of refining query for false positives also increases in the sequential algorithm. However, our collaboration algorithm prunes the false positives that do not satisfy the environmental conditions in earlier stage of the algorithm. The reduction in the number of refining queries reduces the overall processing time for querying.

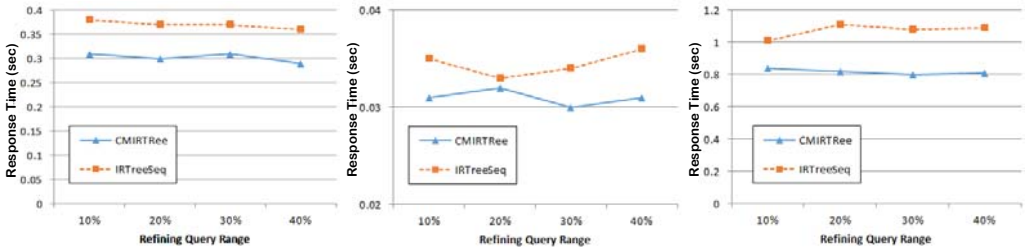


(a) Result for a synthetic dataset (b) Result for a Euro dataset (c) Result for a Korean Building dataset

Fig. 10. Effect of the primary query range on the query execution time.

Effect of the Refining Query Range

To evaluate the effect of the refining query range, we increased the range from 10% to 40% of primary query range. In this experiment, the primary query range is fixed to 5% of the space. In Fig. 11, the experimental results of IRTreeSeq and CMIRTree are plotted in terms of the average response time. As the range of the refining query increases, each refining queries can terminate in earlier stage of searching because the refining query range can easily covers the node of IR-tree in DR. Therefore, the performance of both algorithms show slightly better performance when extend the refining query range. However, our algorithm still performs better for the overall condition because the efficiency of search for the refining query.

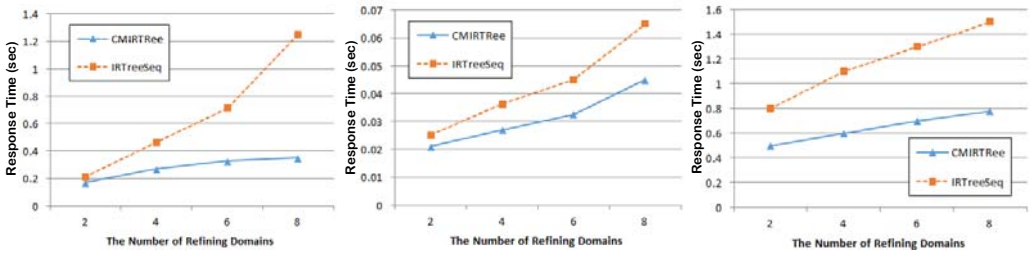


(a) Result for a synthetic dataset (b) Result for a Euro dataset (c) Result for a Korean Building dataset

Fig. 11. Effect of the refining query range on the query execution time.

Effect of the Number of the Domain for MKDR Query

To evaluate the effect of the number of domains used for the refining query, we increased the number of domains from two to eight. In this experiment, the primary query range is fixed to 5% of the space, and refining query range is fixed to 1% of the space. In Fig. 12, the experimental results of *IRTreeSeq* and *CMIRTree* are plotted in terms of the average response time for the given parameters. When the refining conditions become more complex, our collaboration algorithm has more chance for pruning nodes during approximate refining query. Therefore, the gap in the performances increases when the number of domains for the refining queries is increased.



(a) Result for a synthetic dataset (b) Result for a Euro dataset (c) Result for a Korean Building dataset

Fig. 12. Effect of the number of the DR on the query execution time.

Effect of the Database Size

To evaluate the effect of the size of the data, we increased the number of data from 20,000 to 100,000 per domain. The number of domains used in the query was fixed at

four, and primary query range is 5% of the space, and refining query range is 1% of the space. In Fig. 13, the experimental results of the effect of the number of data on query execution time are shown. The performance gap is not clear in the case of a small database. Because the IR-trees in small and sparse databases have relatively low depth and large sized MBR, our approximate refining process has fewer chances for filtering nodes. Therefore, because the number of refining queries is nearly the same, both algorithms yield identical performance. However, as the size of data increases, the proposed algorithm exhibits better performance because of the increased number of data located within the fixed primary query range and the number of visited nodes for refining process is also increased.

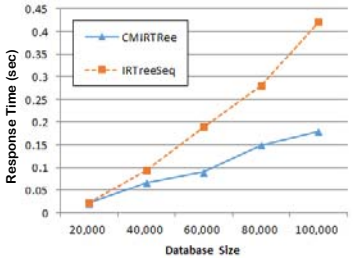
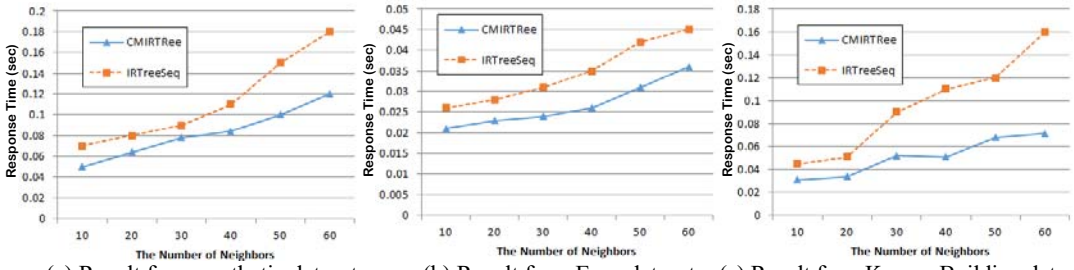


Fig. 13. Effect of the number of data on query execution time.

5.2 Performance Studies for MKDkNN Query

Effect of the Number of Nearest Neighbor

To evaluate the effect of the number of nearest neighbor of the primary kNN query in MKDkNN, we increased the number of neighbor from 10 to 60. The refining query range is fixed to 5% of the space, and the number of refining domain is fixed to three. In Fig. 14, the experimental results of *IRTreeSeq* and *CMIRTree* are plotted in terms of the average response time. Unlike the MKDR query, the difference in performance is relatively constant even if the k value slightly increases. Comparing to MKDR query, kNN has a relatively narrow search range. Therefore, even if the k value slightly increases, the number of visited nodes is not significantly different. But our algorithm still shows better performance because it can avoid the searching for unnecessary nodes in DP and reduce the search for redundant nodes in DR.

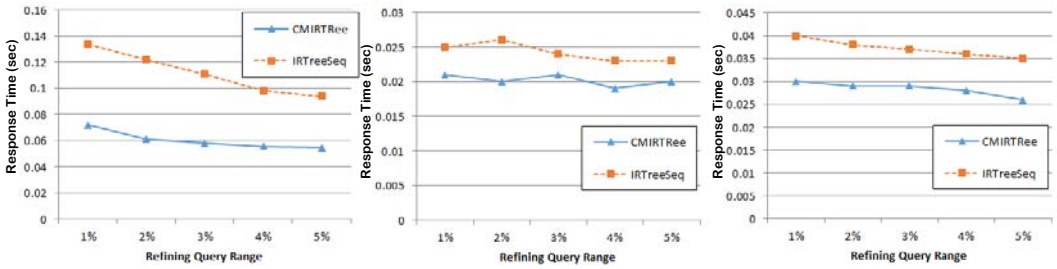


(a) Result for a synthetic dataset (b) Result for a Euro dataset (c) Result for a Korean Building dataset

Fig. 14. Effect of the number of neighbors on the query execution time.

Effect of the Refining Query Range

To evaluate the effect of the range of refining range query in MKDkNN, we increased the refining range from 1% to 5% of the space. In Fig. 15, the experimental results of *IRTreeSeq* and *CMIRTree* are plotted in terms of the average response time. The number of nearest neighbor k is fixed to 10, and the number of domains for refining query is fixed to three. When the refining query range is increased, the performance gap between the two methods decreases. This is attributed to the fact that as the refining query range is increased, the filtering power of our approximate refining query weakens owing to almost all nodes passing the environmental evaluations.

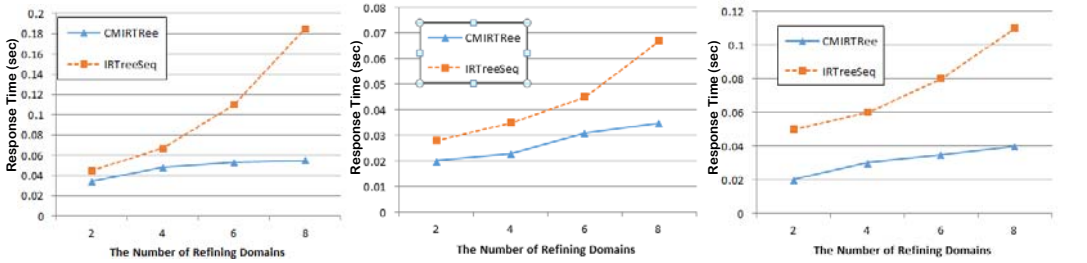


(a) Result for a synthetic dataset (b) Result for a Euro dataset (c) Result for a Korean Building dataset

Fig. 15. Effect of the refining query range on the query execution time.

Effect of the Number of the DR

To evaluate the effect of the number of domains used for the refining query in MKDkNN, we increased the number of domains from 2 to 8. The number of nearest neighbor k is fixed to 10, and the refining query range is fixed to 5% of the space. Fig. 16 shows the experimental results of IRTreeSeq and *CMIRTree* in terms of the average response time. Similar to MKDR query results, we can observe that the filtering power of our collaboration algorithm increases if the refining conditions become more complex.



(a) Result for a synthetic dataset (b) Result for a Euro dataset (c) Result for a Korean Building dataset

Fig. 16. Effect of the number of the DR on the query execution time.

Effect of the Database Size

To evaluate the effect of the size of the data in MKDkNN, we increased the number of data from 20,000 to 100,000 per domain. The number of domains used in the query is fixed at four, and the number of nearest neighbor k is fixed to 10, and the refining query range is fixed to 5,000. In Fig. 17, the experimental results of IRTreeSeq and *CMIRTree*

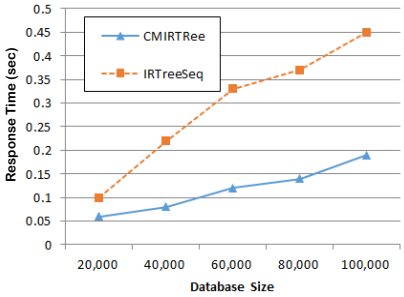


Fig. 17. Effect of the number of data on query execution time.

in terms of the average response time are shown. The result obtained is similar those of the above experiments. The performance of the approximate refining query can be confirmed not only by the query argument but also by the structure of the IR-tree. The small IR-tree resulting from small and sparse databases degrades the filtering power of approximate refining.

6. CONCLUSION

In this paper, we propose the MKDR and the MKDkNN queries. The proposed queries additionally filter the geo-textual objects by considering requirements for environmental conditions of the objects. Therefore, these queries realize a more selective search than the existing spatial keyword queries. To explore objects belonging to different domains efficiently during search processing, we propose algorithms that use the collaboration of multiple IR trees. The proposed scheme simultaneously traverses multiple IR-trees constructed in each domain. Our collaboration algorithm allows additional pruning during the processing of the primary spatial keyword query. The proposed algorithms significantly reduce the query execution times by reducing the number of refining queries and the number of nodes to be searched in refining processing. Experimental results demonstrate that the proposed algorithms outperform the conventionally used sequentially processing algorithm.

REFERENCES

1. Z. Li, K. C. K Lee, B. Zheng, W. Lee, D. Lee, and X. Wang, "IR-tree: An efficient index for geographic document search," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 23, 2011, pp. 585-599.
2. G. Cong and C. S. Jensen, "Querying geo-textual data," in *Proceedings of International Conference on Management of Data*, 2016, pp. 2207-2212.
3. T. Guo, X. Cao, and G. Cong, "Efficient algorithms for answering the *m*-closest keywords query," in *Proceedings of International Conference on Management of Data*, 2015, pp. 405-418.
4. X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi, "Collective spatial keyword querying," in *Proceedings of International Conference on Management of Data*, 2011, pp. 373-384.
5. X. Cao, G. Cong, T. Guo, C. S. Jensen, and B. C. Ooi, "Efficient processing of spatial group keyword queries," *ACM Transactions on Database Systems*, Vol. 40, 2015, pp. 1-48.

6. G. Cong, C. S. Jensen, and D. Wu, "Efficient retrieval of the top-*k* most relevant spatial web objects," in *Proceedings of the VLDB Endowment*, Vol. 2, 2009, pp. 337-348.
7. R. Hariharan, B. Hore, C. Li, and S. Mehrotra, "Processing spatial-keyword (SK) queries in geographic information retrieval (GIR) systems," in *Proceedings of the 19th International Conference on Scientific and Statistical Database Management*, 2007, p. 16.
8. I. D. Felipe, V. Hristidis, and N. Rishe, "Keyword search on spatial databases," in *Proceedings of International Conference on Data Engineering*, 2008, pp. 656-665.
9. S. Vaid, C. B. Jones, H. Joho, and M. Sanderson, "Spatio-textual indexing for geographical search on the web," in *Proceedings of the 9th International Symposium on Spatial and Temporal Databases*, 2005, pp. 218-235.
10. J. B. Rocha-Junior, O. Gkorgkas, S. Jonassen, and K. Nørvåg, "Efficient processing of top-*k* spatial keyword queries." in *Proceedings of International Symposium on Spatial and Temporal Databases*, 2011, pp. 205-222.
11. D. Zhang, K.-L. Tan, and A. K. H. Tung, "Scalable top-*k* spatial keyword search," in *Proceedings of the 16th International Conference on Extending Database Technology*, 2013, pp. 359-370.

Bumjoon Jo (趙範俊) is a Ph.D. candidate in the Computer Science at Sogang University. He received the BS and MS degrees in Computer Science from Sogang University in 2010 and 2012, respectively. His research interests include spatial databases, LBS and NoSQL.

Junhong Ahn (安埈弘) received the BS and MS degrees in Computer Science from Sogang University in 2014 and 2017, respectively. His research interests include spatial databases and spatial keyword search.

Sungwon Jung (鄭盛元) received the BS degree in computer science from Sogang University, Seoul, Korea in 1988. He received the M.S. and Ph.D. degrees in Computer Science from Michigan State, in University, East Lansing, Michigan in 1990 and 1995, respectively. He is currently a Professor in the Computer Science and Engineering Department at Sogang University. His research interests include spatial and mobile databases, data mining, and blockchain technology.