An Efficient I/O Cost Model for Novel Indexing algorithms in a Joint Top K Spatial Keyword Query Processing

T. Bairavi¹, R. Priya², A. Evangelin³

M.E. (CSE), Jayaram College of Engineering and Technology, Trichy, Tamilnadu, India¹
M.E. (SE), Jayaram College of Engineering and Technology, Trichy, Tamilnadu, India²,³

ABSTRACT: A Top-k spatial keyword query returns the k best objects ranked in terms of both distance to the query location and textual relevance to the query keywords. The Web users and Web content are increasingly being geo-positioned such as GPS and WIFI. It also focuses to convert local content in response to web queries. It takes into an account of both the locations and textual descriptions of content. In order to process such queries efficiently, spatial-textual indexes combining R-trees and inverted files are employed. The studies deal with an efficient joint processing of multiple top-k spatial keyword queries with cost model for Novel Algorithms. The joint processing deals high query loads and also occurs when multiple queries are used to obfuscate a user’s true query. The Novel algorithm with an index structure for the joint processing of top-k spatial keyword queries are proposed with an efficient cost model.

Index Terms: Query Processing, Spatial keyword queries, Spatial Database and Textual Database

I. OVERVIEW

The range of technologies combines to afford the web and its users a geographical dimension. Geo-positioning technologies such as GPS and Wi-Fi and cellular geo-location services, e.g., as offered by Skyhook, Google, yahoo and Spotigo, are being used increasingly; and different geo-coding technologies enable the tagging of web content with positions. The percentage is likely to be higher for mobile users. This renders so-called spatial keyword queries important. Such queries take a location and a set of keywords as Arguments and return the content that best matches these arguments. The Google Maps and a yellow pages are the best Examples for providing a local search service in spatial keyword queries, where they enable search for, e.g., restaurants annotated with a text), a query location (latitude and longitude), and a set of query keywords, a top-k spatial keyword query on road networks returns the k best objects in terms of both 1) shortest path to the query location, and 2) textual relevance to the query keywords.
For example, Figure 1 illustrates the road networks and spatial textual objects in a tourist area of Norway. The circles represent spatial-textual objects p with a textual description, and the cross mark q.l represents the query location. Assume a tourist in q.l with a GPS-enabled mobile phone. The tourist poses a top-k spatial keyword query looking for “hotel” (his spatial location is automatically sent by the mobile phone). If a traditional query (Euclidean distance) is considered, the top-1 hotel is p9 on the left side of the figure. However, when road networks are considered, the top-1 hotel is p4 on the right side of the figure. In top-k spatial keyword queries on road networks both shortest path and textual relevance are considered. For example, for the query “bar café” posed in q.l, the spatial-textual object p6 may appear better ranked than p7 because the description of p6 (“Egon Solsiden bar & café”) is more textually relevant to the query keywords than the description of p7 (“Choco café”), and p6 is only slightly more distant to q.l than p7. The top-1 object, however, is p10 because it is very near to q.l and is also relevant to the query keywords. Note that p11 is not returned as a result of this query, since none of the terms in the description of p11 appear in the query keywords. The joint processing is motivated by main applications a. Multiple Query Optimizations and b. Privacy-Aware Querying Support.

II. NOVEL ALGORITHMS

The existing IR-tree and then proceed to develop a basic and an advanced algorithm for processing joint top k spatial keyword queries. The algorithms are generic and are not tied to a particular index.

2.1 IR-Tree

The IR-tree [2], which we use as a baseline, is essentially an R-tree [5] extended with inverted files [32]. The IR-tree’s leaf nodes contain entries of the form (p, p.λ, p.di), where p refers to an object in dataset D, p.λ is the bounding rectangle of p, and p.di is the identifier of the description of p. Each leaf node also contains a pointer to an inverted file with the text descriptions of the objects stored in the node. An inverted file index has two main components.

- A vocabulary of all distinct words appearing in the description of an object.
- A posting list for each word t that is a sequence of identifiers of the objects whose descriptions contain t.

Each non-leaf node R in the IR-tree contains a number of entries of the form (cp, rect, cp.di) where cp is the address of a child node of R, rect is the MBR of all rectangles in entries of the child node, and cp.di is the identifier of a pseudo text description that is the union of all text descriptions in the entries of the child node. As an example, Figure 2a contains 8 spatial objects p1, p2, . . . , p8, and Figure 2b shows the words appearing in the description of each object. Figure 3a illustrates the corresponding IR-tree, and Figure 3b shows the contents of the inverted files associated with the nodes.
The LOOPING algorithm for computing the joint top-k spatial keyword query is adapted from an existing algorithm [2] that considers a single query. Recall that a joint top-k spatial keyword query $Q$ consists of a set of sub queries $q_i$. The LOOPING algorithm computes the top-k results for each sub query separately. The arguments are a joint query $Q$, the root of an index $root$, and the number of results $k$ for each sub query. When processing a sub query $q_i \in Q$, the algorithm maintains a priority queue $U$ on the nodes to be visited. The key of an element $e \in U$ is the minimum distance $mindist(q_i, e)$ between the query $q_i$ and the element $e$. The algorithm utilizes the keyword information to prune the search space. It only loads the posting lists of the words in $q_i$. A non-leaf entry is algorithm returns $k$ elements that have the smallest Euclidean distance to the query and contain the query keywords.

**Example 1:** Consider the joint query $Q = \{q_1, q_2, q_3\}$ in Figure 2, where $q_1.\psi = \{a, b\}$, $q_2.\psi = \{b, c\}$, $q_3.\psi = \{a, c\}$, and all sub queries (and $Q$) have the same location $\lambda$. Table 1 shows the minimum distances between $Q$ and each object and bounding rectangle in the tree. We want to find the top-1 object. For each sub query $q_i$, LOOPING thus computes the top-1 result. Subquery $q_1 = \lambda, \{a, b\}$ first visits the root and loads the posting lists of words $a$ and $b$ in $InvFile-root$. Since entries $R5$ and $R6$ both contain $a$ and $b$, both entries are inserted into the priority queue with their distances to $q_1$. The next dequeued entry is $R5$, and the posting lists of words $a$ and $b$ in $InvFile-R5$ are loaded. Since only $R1$ contains $a$ and $b$, $R1$ is inserted into the queue, while $R2$ is pruned. Now $R6$ and $R1$ are in the queue, and $R6$ is dequeued. After loading the posting lists of words $a$ and $b$ in $InvFile-R6$, $R3$ is inserted into the queue, while $R4$ is pruned. Now $R1$ and $R3$ are in the queue, and $R1$ is dequeued. Its child node is loaded, and the top-1 object $p1$ is found, since the distance of $p1$ is smaller than that of the first entry ($R3$) in the queue ($2 < 4$). Similarly, the result of sub query $\lambda, \{b, c\}$ is empty and the result of sub query is $\lambda, \{a, c\}$ is $P2$. The disadvantage of the LOOPING Algorithm is that it may visit a tree node multiple times, leading to high I/O cost.

The JOINT Algorithm: The JOINT algorithm aims to process all sub queries of $Q$ concurrently by employing a shared priority queue $U$ to organize the visits to the tree nodes that can contribute to closer results (for some sub query). Unlike LOOPING, JOINT guarantees that each node in the tree is accessed at most once during query processing.
Pruning Strategies: The algorithm uses three pruning rules. Let be an entry in a non-leaf tree node. We utilize the MBR and keyword set of to decide whether its sub tree may contain only objects that are farther from or irrelevant to all sub queries of Q.

- Cardinality-Based Pruning
- MBR-Based Pruning
- Individual Pruning

III. NOVEL INDEXES AND ADAPTATIONS

We present the TB-IR-tree that organizes data objects according to both location and keywords. We discuss the processing of the joint top-k query using existing indexes.

3.1 The Text Based-IR-Tree (TBIR TREE)

Text Based Partitioning: As a first step in presenting the index structure, we consider the partitioning of a dataset according to keywords. We hypothesize that a keyword query will often contain a frequent word (say w). This inspires us to partition the dataset D into the subset D+ whose objects contain w and the subset D− whose objects do not contain w and that we need not examine when processing a query containing w. We aim at partitioning D into multiple groups of objects, such that the groups share as few keywords as possible. However, this problem is equivalent to, e.g., the clustering problem and is NP-hard. Hence, we propose a heuristic to partition the objects. Let the list W of keywords of objects sorted in descending order of their frequencies be; w1, w2, . . . ,wm, where m is the number of words in D. Frequent words are handled before infrequent words. We start by partitioning the objects into two groups using word w1: the group whose objects contain w1, and the group whose objects do not. We then partition each of these two groups by word w2. This way, the dataset can be partitioned into at most 2, 4, . . . ,2m groups. By construction, the word overlap among groups is small, which will tend to reduce the number of groups accessed when processing a query. Algorithm 3 recursively applies the above partitioning to construct a list of tree nodes L. To avoid underflow and overflow, each node in L must contain between B/2 and B objects, where B is the node capacity. In the algorithm, D isthe dataset being examined, and W is the corresponding wordlist sorted in the descending order of frequency. When the number of objects in D (i.e., |D|) is between B/2and B, D is added as a node to the result list L (lines 1–2). If |D| < B/2 then D is returned to the parent algorithm call (lines 3–4) for further processing. If |D| > B (line 5) then we partition D (lines 6–19). In case W is empty (line 6), all the objects in D must have the same set of words and cannot be partitioned further by words. We hence use a main-memory R-tree with fanout (node capacity) B to partition D according to location and add the leaf nodes of the R-tree to the result list L (lines 7–8). When W is non-empty, we take the most frequent (first) word in W (lines 9–10). The objects in D are partitioned into groups D+ and D− based on whether or not they contain w (lines 11–12). Next, we recursively partition D+ and D− (lines 13–14). The remaining objects from these recursive calls (i.e., the sets T+ and T−) are then merged into the set T (line 15). If T has enough objects, it is added as a node to L (lines 16–17). Otherwise, set T is returned to the parent algorithm call (lines 18–19). If the initial call of the algorithm returns a group with less than B/2 objects, it is added as a node to the result list L, since no more objects are left to be merged.

Algorithm The TBIR TREE Dataset D, Sorted list of words W, Integer B, List of tree nodes L
1: if B/2 ≤ |D| ≤ B then
2: add D as a node to L;
3: else if |D| < B/2 then
4: return D;
5: else _ partitioning phase
6: if W is empty then _ partitioning by location
7: insert D into a main-memory R-tree with fanout B;
8: add the leaf nodes of the main-memory R-tree to L;
9: else _ partitioning by words
3.2 The TB-IBR-Tree and Variants

Inverted Bitmap Optimization: Each node in the TBIR TREE contains a pointer to its corresponding inverted file. By replacing each such inverted file by an inverted bitmap, we can reduce the storage space of the TBIR TREE and also save I/O during query processing. We call the resulting tree the TB-IBR-tree.

Updates: A deletion in the TBIR TREE is done as in the R-tree, by finding the leaf node containing the object and then removing the object. An insertion selects a branch such that the insertion of the object leads to the smallest “enlargement” of the keyword set, meaning that the number of distinct words included in the branch increases the least if the object is inserted there.
The inverted bitmap optimization technique is applicable to the original IR-tree [2], yielding the IBR-tree. The bitmap optimization technique is also applicable to the CDIR-tree [2], yielding the CD-IBR-tree.

### 4.1 Estimation of Keyword Probability and \( k \)NN Distance

#### 4.1.1 Estimation of Keyword Probability

Let \( F(q, \lambda) \) be the probability of having \( q, \lambda \) as the keyword set of an object of \( D \). Let \( z \) be the number of words in each object (and also in the query). Let an arbitrary list (i.e., sequence) of \( z \) words \( wq_1, wq_2, \ldots, wq_z \). Due to the “without replacement” rule, when we draw the \( j \)-th word of an object, any previously drawn word \( (wq_1, wq_2, \ldots, wq_{j-1}) \) cannot be drawn again. Thus, the probability of the \( j \)-th drawn word being \( wq_j \) is:

\[
F(wq_j) = \frac{1}{\sum_{i=1}^{n} \frac{1}{wq_i^s}},
\]

where \( n \) is the total number of words and \( s \) is the value of the exponent characterizing the distribution (skew). Let a query be \( q = \lambda q \). Suppose that each object and the query contain \( z \) keywords. We assume that the words of each object are drawn without replacement based on the occurrence probabilities of the words. Let \( dknn \) denote the \( k \)NN distance of \( q \), i.e., the distance to the \( k \)-th nearest neighbor in \( D \). Let \( e \) be any non-leaf entry that points to a leaf node. We need to access \( e \)'s leaf node when:

1) the keyword set of \( e \) contains \( q, \lambda \), and
2) the minimum distance from \( q \) to \( e \) is within \( dknn \).

Let the probability of the above two events be the keyword containment probability \( Pr(e, q \supseteq q) \) and the spatial intersection probability \( Pr(\text{mindist}(q, e, \lambda) \leq dknn) \), respectively. Thus, the access probability of the child node of \( e \) is:

\[
Pr(\text{access } e) = Pr(e, q \supseteq q) \cdot Pr(\text{mindist}(q, e, \lambda) \leq dknn).
\]

Estimate the total number of accessed leaf nodes as:

\[
\text{COST} = NL \cdot Pr(\text{access } e).
\]

We proceed to derive the probability of an object matching the query keywords and the \( k \)NN distance \( dknn \). We then study the probabilities \( Pr(e, q \supseteq q, \psi) \) and \( Pr(\text{mindist}(q, e, \lambda) \leq dknn) \) for the IR-tree and the W-IR-tree, respectively. Finally, we compare the two trees using the cost models.

#### 4.1.2 Estimation of \( k \)NN Distance

Probability of an Object Matching the Query Keywords:

Let \( F(q, \lambda) \) be the probability of having \( q, \lambda \) as the keyword set of an object of \( D \). Let \( z \) be the number of words in each object (and also in the query). Let an arbitrary list (i.e., sequence) of \( q \)-words \( wq_1, wq_2, \ldots, wq_z \). Due to the “without replacement” rule, when we draw the \( j \)-th word of an object, any previously drawn word \( (wq_1, wq_2, \ldots, wq_{j-1}) \) cannot be drawn again. Thus, the probability of the \( j \)-th drawn word being \( wq_j \) is:
We obtain the spatial intersection probability by substituting the number of leaf nodes that contain the query word.

During the construction of the TBIR-tree, the objects are first partitioned based on their keywords. There are objects in the leaf node pointed to by \( e \). Note that in the IR-tree, the keywords of the objects are partitioned into leaf nodes based on their keywords. Thus, the keyword containment probability is

\[
Prir(e, \psi) = 1 - (1 - F(q, \psi))B.
\]

We obtain the spatial intersection probability by substituting into Equation 3. Observe that out of NL leaf nodes, max\( NL \cdot F(q, \psi) \), 1 nodes contain the query keyword \( q, \psi \). Thus, the keyword containment probability is

\[
Prir(e, \psi, \exists q, \psi) = \max \left\{ F(q, \psi) \cdot \frac{1}{N_I} \right\}
\]
The joint top-k spatial keyword query and presents efficient means of computing the query. Our solution consists of: (i) the TB-IBR-tree that exploits keyword partitioning and inverted bitmaps for indexing spatial keyword data, and (ii) the JOINT algorithm that processes multiple queries jointly. In addition, we describe how to adapt the solution to existing index structures for spatial keyword data. Studies with combinations of two algorithms and a range of indexes demonstrate that the JOINT algorithm on the TB-IBR-tree is the most efficient combination for processing joint top-k spatial keyword queries. It is straightforward to extend our solution to process top-k spatial keyword queries in spatial networks. We take advantage of Euclidean distance being a lower bound on network distance. While reporting the top-k objects incrementally, if the current object is farther away from the query in terms of Euclidean distance than is the kth candidate object in terms of network distance, the algorithm stops and the top-k result objects in the spatial network are found. The network distance from each object to a query can be easily computed using an existing, efficient approach [20]. An interesting research direction is to study the processing of joint moving user.

REFERENCES