Finding Relevant Search using Keywords

Akshay Mahajan¹, Mohit Chaurasia², Priyanka Kamthe³, Anuj Koul⁴ Mrs. Archana Kadam⁵

Students of Computer Department, PCCOE ¹,²,³,⁴
Assistant Professor in Computer Department, PCCOE ⁵

Abstract: Old spatial queries, like range search, nearest neighbour retrieval, involved only conditions on objects’ geometric properties. Gradually many new applications call for novel forms of queries which aim to find objects which satisfy both a spatial predicate, as well as a predicate on their associated texts. Consider an example: instead of considering all the hotels, a nearest neighbour query would just ask for the hotel that lies closest among the ones whose menu cards contain “rolls, burger, caramel custard” all at the same moment. At this time, the best possible solution to such type of queries is based on the ir²-tree, which, as shown here, has some deficiencies that seriously would impact its efficiency. Inspired by this, we are developing a new method called the spatial inverted index which extends the conventional inverted index to cope with multidimensional data, and produces algorithms which will answer nearest neighbor queries with keywords in real time.

Keywords: ir² tree, spatial inverted index, r-tree.

INTRODUCTION

A spatial database manages more than one dimensional objects (such as points, rectangles, etc.), and provides faster access to those objects based on different selection criteria. The importance of spatial databases is reflected by the convenience of modelling entities of reality in a thematic manner of geometry. Say, for example, the locations of banks, hotels, ATM machines and so on are often illustrated as points in a map, while larger extents like gardens, rivers, and landscapes as a combination of rectangles. Many functions of a spatial database are useful in various ways in contexts specifically. Consider, for example, in a geography information system, range search can be implemented to find all restaurants in a specific area, whereas the nearest neighbor retrieval may discover the restaurant nearest to a given address. Today, the use of search engines has made it more real to write spatial queries in a whole new way. Normally, queries are based on objects’ geometric properties. We have seen some new applications that call for the ability to select objects based on both of their geometric coordinates and their related texts. For example, it would be fairly useful if a search engine can be used to find the nearest restaurant that offers “burger, pizza, and desert” all at the same time. Note that this is not the “globally” nearest restaurant (which would have been returned by a traditional nearest neighbor query), but it is the nearest restaurant within only those providing all the demanded item. There are easy ways to support queries that combine spatial and features of text. Say, for example, in the above query, we would first fetch all the hotels whose menu contains the set of keywords {dessert, burger, pizza}, and then from the retrieved hotels, shortlist the nearest one. In the similar way, one could also do it oppositely by targeting first the spatial conditions – browse all the restaurants in ascending order of their distances to the query point until encountering one whose menu has all the keywords. The major drawback of these straight forward approaches is that they will fail to provide real time answers on strange hard inputs. An example is that the real nearest neighbor lies much far away from query point, whereas the closer neighbors are missing at least one of the query keywords.

Spatial queries with keywords have not been majorly explored. In the past years, the community has sparked enthusiasm in studying keyword search in RDBMS. It is considered recently that the attention was diverted to multidimensional data [1], [2], [3]. The best method to date for nearest neighbor search with keywords is due to Felipe et al. [2]. They nicely integrate two well-known concepts: R-tree [2], a popular spatial index, and signature file [4], an effective method for keyword-based retrieval of document. By doing this they can develop a structure called the IR²-tree [5], which has the strengths of both R-trees and signature files. Like R-trees, the IR²-tree preserves objects’ spatial proximity, which is the key to solving spatial queries with efficiency. Whereas, like signature files, the IR²-tree can filter a amount of portion of the objects that will not contain all the query keywords, thus reducing the number of objects to be examined significantly.

The IR²-tree, also has a drawback of signature files; false hits. A signature file, because of its conservative nature, may direct the search to some basic objects, even if they don’t have all the

RELATED WORK

It provides the information retrieval R-tree (IR²-tree) [1], which is the used to answer the nearest neighbor queries explained. It explains an alternative solution based on the inverted index

The IR²-tree As mentioned earlier, the IR²-tree [2] combines the R-tree with signature files. Secondly, will review what is a signature file before explaining IR²-trees. Our discussion assumes the information of R-trees and best-first algorithm [4] for NN search, both of which are well-known methods in spatial databases. Signature file generally refers to a hashing-based framework, whose instantiation in [6] is known as superimposed coding (SC), it is shown to be more effective than other instantiations
[4]. It is designed to do membership tests: check whether a query word w exists in a set W of words. SC is reserved, in the sense that if it says “no”, then w is definitely not in W. If, on the other hand, SC returns “yes”, the true answer can be either way, in which case the whole W should be neglected to prevent false hit. In context of [7], SC works in same way as classic technique of bloom filter. In preprocessing, it builds a bit signature of length 1 from W by hashing each word in W to a string of 1 bits, and then making the separation of all bit strings. To illustrate, denote by h(w) the bit string of a word w. Firstly, all 1 bits of h(w) are made 0. Then, SC repeats the following m times: randomly choose a bit and set it to 1. Most importantly, randomization must use w as its seed to make sure that same w always ends up with a similar h(w). Further, the m choices are mutually independent, and may even happen to be the same bit. The concrete values of l and m affect the space cost and false hit probability. The 2nd figure gives an example to illustrate the above process, assuming l = 5 and m = 2. Say for example, in the bit string h(a) = 00100, the 3rd and 5th (counting from left) are set to 1, and h(b) = 01011, the 3rd and 4th (counting from left) are set to 1. Given a query keyword w, SC performs the member-ship test in W by testing whether all the 1’s of h(w) appear at the same places in the signature of W. If not, it is guaranteed w cannot belong to W. Else, the test cannot be resolved using only signature, and a scan of W follows. A false hit occurs if the scan reveals that W actually does not contain w. For instance, assume that we want to test whether word c is a member of set {a, b} using only the set’s signature 01101. Since the 4th bit of h(c) = 00011 is 1 but that of 01110 is 0, SC suddenly reports “no”. Like another instance, assume the membership test of c in {b, d} whose signature is 01111. This time, SC returns “yes” because 01111 has 1’s at all the bits where h(c) is set to 1; as a result, a full scan of set is needed to verify that this is a false hit. This IR-tree is a R-tree where every (leaf or nonleaf) entry E is augmented with a signature that summarizes the union of the texts of the objects in the subtree of E. The 2nd figure shows an example based on the dataset of Figure 1 and hash values in Figure 2. The string 01111 in the leaf entry p6, for instance, is the signature of Wp6 = {b, d} (which is the document of p6; see Figure 1b). The string 11111 in the nonleaf entry E3, is the signature of Wp2 U Wp6, namely, the set of all words describing p2 and p6. Notice, in general, the signature of nonleaf entry E can be conveniently obtained simply as disjunction of all signatures in the leaf node of E. A nonleaf signature may allow a query algorithm to realize that certain word cannot exist in the subtree. For instance, as the 2nd bit of h(b) is 1, we know that no object in the subtrees of E2 and E6 can have word b in its texts—notice that signatures of E2 and E6 have 0 as their 2nd bits. Normally, the signatures in an IR-tree may have different lengths at various levels. For R-trees, the best-first algorithm [10] is a well-known solution to NN search. It is straightforward to adapt it to IR-trees. Specially, given a query point q and a keyword set Wq, the used algorithm accesses entries of an IR-tree in ascending order of distances of their MBRs to q (the MBR of a leaf entry is just the point itself), pruning the entries whose signatures indicate the absence of at least one word of Wq in their sub trees. Whenever a leaf entry, say of point p, cannot be pruned, a random I/O is performed to retrieve its text description Wp. If Wq is a subset of Wp, the algorithm terminates with p as the answer; otherwise, it continues until no more entry remains to be processed. In Figure 3, assume that the query point q has a keyword set Wq = {c, d}. It can be verified that the algorithm must read all the nodes of the tree, and fetch the documents of p2, p4 and p6 (in this order). The final answer is p6, while p2 and p4 are false hits.

**PROBLEM DEFINITION**

Let P be a set of multidimensional points. As our intention is to combine keyword search with the initial location finding services on facilities such as restaurants, hotels and banks etc. we will focus on dimensionality 2, but our method can be expanded to arbitrary dimensionalities with no technical hindrance. We will assume that the points in P have integer coordinates, such that each coordinate ranges in [0, t], where t is a large integer. This is not as it may seem, because even if one would like to insist on actual-valued coordinates, the set of different coordinates representable under a space limit is still finite and numerous; therefore, we could convert all things to integers with proper scaling. As with [6], each point p ∈ P is related with a group of words, which is represented as WP and defines the document of p. For example, if p stands for a hotel, WP can be its services, or if p is a bank, WP can be the description of its services and facilities, or if p is a blood bank, WP can be the list of its in stock blood groups specialities. It is clear that WP may potentially contain numerous words. Traditional nearest neighbor search returns the data point nearest to a query point. After [6], we extend the problem to include predicates on objects’ texts. Formally, in our context, a nearest neighbor (NN) query specifies a point q and a set Wq of keywords (we refer to Wq as the document in the query). It returns the point in Pq that is the closest to q, where Pq is defined as the words of P whose documents contain all the keywords in Wq. In the case where Pq is empty, the query returns null. The problem

\[ P_q = \{ p \in P | W_p \subseteq W_q \} \]  

(1)

definition can be generalized to k nearest neighbor search, which searches the k points in Pq closest to q; if Pq has less than k points, the entire Pq should be returned. For example, assume that P consists of 8 points whose locations are as shown in Figure 1a (the black dots), and their documents are given in Figure 1b. Consider a query point q at the white dot of Figure 1a with the set of keywords \( W_q = \{ c, d \} \). Nearest neighbor search finds \( p_6 \), noticing that all points closer to q than \( p_6 \) are missing either the query keyword c or d. If \( k = 2 \) nearest neighbors are wanted, \( p_9 \) is also returned in addition. The result is still \( \{ p_6, p_9 \} \) even if \( k \) increases to 3 or higher, because only 2 objects have the keywords c and d at the same time. We assume that the dataset will not fix in memory, and needs to be indexed by efficient access methods in order to minimize the number of input outputs in answering a query.
PSEUDO CODE
Pseudo code for managing Hotel Data in Spatial Index Tree:
1. Admin Adds a Hotel ‘L’ having (Location L, and L2), set of Keywords(k)
Location information of hotel in list is converted to 1D form
2. Sorting is required in order to apply the compression method…
3. Location Data is sorted using 1D list l(l(d),n)
4. Repeat until list is sorted
5. Put lin i
6. Repeat while i<n
7. If l(d)[i] > l(d)[i+1]
8. Then swap l(d)[i],l(d)[i+1]
9. End if
10. Increment i
11. End repeat
12. End repeat
13. End sort

Compression scheme is applied to the id’s as to minimize
the size of inverted index by it. The difference between
the id’s is calculated and only is stored. This is used as it is an
useful approach rather than storing the precise data. The
distances can be re-calculated back.(Algorithm contd.)

Apply Compression Scheme
Record gaps Between id’s for list l
1. Put id in l(j)
2. Repeat till i<n
3. Calculate the difference in (diff in l(j) – l(j+1))
4. Store difference diff in list l at index i
5. End repeat
6. Assign new id to Hotel (h)
7. Put id of n-1 in newID
8. Increment newID
9. Assign newIDto new element l(n) in list

To insert an item, the tree is traversed repeatedly from the
head node. At each step, all rectangles in the current
directory node are tested, and a candidate is picked using a
heuristic such as picking the rectangle which requires least
space. The search then descends into this page, until
reaching a leaf node. If the leaf node is full, it must be split
before the insertion is made. Again, as an exhaustive
search is too costly, a heuristic is employed to split the
node into two. Adding the newly created node to the
previous level, this level can again overflow, and these
overflows can propagate up to the root node; when this
node also overflows, a new root node is created and the
tree has increased in height

Algorithm contd.(Search)
1. Generate R-Tree
2. While node n in list l
3. Traverse until leaf node l
4. If first element

REFERENCES
pages 373–384, 2011.
In Proc. of ACM Management of Data (SIGMOD), 2009.