ISSN:0975-9646

Spatial Index Keyword Search in Multi dimensional Database

[1]C. Usha Rani , [2]N.Munisankar

[1] *Assistant Professor, Department of CSE, Annamacharya Institute of Technology and Sciences, JNTUA, Tirupati, Chittor, Andhra Pradesh, India*
[2] *Assistant professor, Department of CSE, Sri venaktesaperumal College of Engineering and Technology , JNTUA, Puttor , Chittor, Andhra Pradesh, India*

Abstract: - An efficient algorithm for spatial databases, various approaches are delivered by the various researchers for finding the result based on the keywords, and usually spatial query is a combination of a location and set of features. in this paper, has a few deficiencies that seriously impact its efficiency. Motivated by this, we develop a new access method called *the spatial inverted index* that extends the conventional inverted index to cope with multidimensional data, and comes with algorithms that can answer nearest neighbor queries with keywords in real time. As verified by experiments, the proposed techniques outperform the IR2-tree in query response time significantly, often by a factor of orders of magnitude. Spatial databases are mainly focus on multi dimensional database. In our approach we are handling the spatial queries jointly and returns the only user specified number of optimal results, we implemented a cache based approach for efficient results.

Index Terms: —Nearest Neighbor Search, Keyword Search, and Spatial Index.

1. INTRODUCTION

The World-Wide Web has reached a size where it is becoming increasingly challenging to satisfy certain information needs. While search engines are still able to index a reasonable subset of the (surface) web, the pages a user is really looking for are often buried under hundreds of thousands of less interesting results. Thus, search engine users are in danger of drowning in information. Adding additional terms to standard keyword searches often fails to narrow down results in the desired direction. A natural approach is to add advanced features that allow users to express other constraints or preferences in an intuitive manner, resulting in the desired documents to be returned among the first results. In fact, search engines have added a variety of such features, often under a special *advanced search* interface, but mostly limited to fairly simple conditions on domain, link structure, or modification date.

A spatial keyword query consists of a query area and a set of keywords shown in below figure. The answer is a list of objects ranked according to a combination of their distance to the query area and the relevance of their text description to the query keywords. A simple yet popular variant, which is used in our running example, is the distance-first spatial keyword query, where objects are ranked by distance and keywords are applied as a conjunctive filter to eliminate objects that do not contain them.

Unfortunately there is no efficient support for top-k spatial keyword queries, where a prefix of the results list is required. Instead, current systems use ad-hoc combinations of nearest neighbor (NN) and keyword search techniques to tackle the problem. For instance, an R-Tree is used to find the nearest neighbors and for each neighbor an inverted index is used to check if the query keywords are contained. We show that such two-phase approaches are inefficient.

Today, the widespread use of search engines has made it realistic to write spatial queries in a brand new way. Conventionally, queries focus on objects' geometric properties only, such as whether a point is in a rectangle, or how close two points are from each other. We have seen some modern applications that call for the ability to select objects based on *both* of their geometric coordinates and their associated texts. For example, it would be fairly useful if a search engine can be used to find the nearest restaurant that offers "steak, spaghetti, and brandy" 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 the nearest restaurant among *only* those providing allthe demanded foods and drinks.

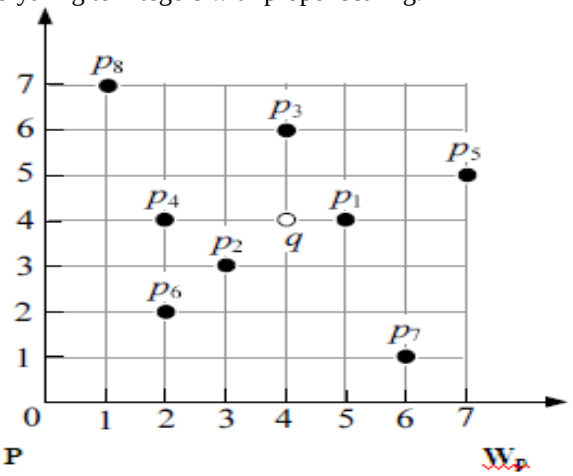
2. RELATED WORK

Inverted indexes (I-index) have proved to be an effective access method for keyword-based document retrieval. In the spatial context, nothing prevents us from treating the text description W_p of a point p as a document, and then, building an I-index. Note that the list of each word maintains a sorted order of point ids, which provides considerable convenience in query processing by allowing an efficient merge step. For example, assume that we want to find the points that have words c and d. This is essentially to compute the intersection of the two words' inverted lists. As both lists are sorted in the same order, we can do so by merging them, whose I/O and CPU times are both linear to the total length of the lists.

3. PROBLEM DEFINITION

Let P be a set of multidimensional points. As our goal is to combine keyword search with the existing location-finding services on facilities such as hospitals, restaurants, hotels, etc., we will focus on dimensionality 2, but our technique can be extended to arbitrary dimensionalities with no technical obstacle. 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

restrictive as it may seem, because even if one would like to insist on real-valued coordinates, the set of different coordinates represent able under a space limit is still finite and enumerable; therefore, we could as well convert everything to integers with proper scaling.



P	W_p
P1	{a,b}
P2	{b,d}
P3	{d}
P4	{a,e}
P5	{c,e}
P6	{c,d,e}
P7	{b,e}
P8	{c,d}

Fig.1. (a) shows the locations of points (b) gives their associated texts.

4. ALGORITHMS USED

Spatial inverted index Algorithm:

Input: Query, Cache Queries
Output: Result set generated for query
Procedure:
If Query available in cache
Result related to query: =
ForwardToTreeprocess (Query)
Else
Result related to query: = GeocodingtreeProcess (Query)
Geocoding process(Query):
Parameters

Qi—Input Spatial Query
Qj (j=1…n) ---Set of Queries contains same Location
Dist[j] (j=1…..n)-----Array for set of distances
Procedure:
(xi,yi)---Geocodings of Qi
(xj,yj)--- Geocodings of all queries with respect to location
Dist[i]=Euclidean distance between the geocodes
While not leafnode
Read nodes from tree For Q.features
If Q.features[i]==Q.features[j]
Add to list
End while
Sort list by feature and distance
Return list.

ForwardToTreeprocess ()

1. Build an empty list
2 .Make a root node
3. if Qi in cache and status=false
For j=0 to n
Compare features(Qi,Qj) status=true;
For Each child in tree
If(status==true)
Getnodebyfeature (Qi);
Getnodebyfeature (Qj);
End
Else
Empty list ()
End For Each
4.Add nodes to list
5.Return list

Inverted index:

Inverted indexes (I-index) have proved to be an effective access method for keyword-based document retrieval. In the spatial context, nothing prevents us from treating the text description W_p of a point p as a document, and then, building an I-index. Figure 4 illustrates the index for the dataset of Figure 1. Each word in the vocabulary has an inverted list, enumerating the ids of the points that have the word in their documents.

Note that the list of each word maintains a sorted order of point ids, which provides considerable convenience in query processing by allowing an efficient merge step. For example, assume that we want to find the points that have words c and d. This is essentially to compute the intersection of the two words' inverted lists. As both lists are sorted in the same order, we can do so by merging them, whose I/O and CPU times are both linear to the total length of the lists.

Recall that, in NN processing with IR2-tree, a point retrieved from the index must be *verified* (i.e., having its text description loaded and checked). Verification is also necessary with I-index, but for exactly the opposite reason. For IR2-tree, verification is because we do not have the detailed texts of a point, while for I-index, it is because we do not have the coordinates. Specifically, given an NN query q with keyword set Wq, the query algorithm of I-index first retrieves (by merging) the set Pq of all points that have all the keywords of Wq, and then, performs |Pq| random I/Os to get the coordinates of each point in Pq in order to evaluate its distance to q.

According to the experiments, when Wq has only a single word, the performance of I-index is very bad, which is expected because *everything* in the inverted list of that word must be verified. Interestingly, as the size of Wq increases, the performance gap between Iindex and IR2-tree keeps narrowing such that I-index even starts to outperform IR2-tree at |Wq| = 4. This is not as surprising as

it may seem. As |Wq| grows large, not many objects need to be verified because the number of objects carrying all the query keywords drops rapidly.

On the other hand, at this point an advantage of Iindex starts to pay off. That is, scanning an inverted list is relatively cheap because it involves only sequential I/Os1, as opposed to the random nature of accessing the nodes of an IR2-tree.

Given the texts

T [0] = "it is what it is"
T [1] = "what is it"
T [2] = "it is a banana"

We have the following inverted file index (where the integers in the set notation brackets refer to the indexes (or keys) of the text symbols, T[0], T[1] etc.):

"a" : {2}
"banana" : {2}
"is" : {0, 1, 2}
"it" : {0, 1, 2}
"what" : {0, 1}

A term search for the terms "what", "is" and "it" would give the set

{0, 1} ∩ {0, 1, 2} ∩ {0, 1, 2} = {0, 1}

With the same texts, we get the following full inverted index, where the pairs are document numbers and local word numbers. Like the document numbers, local word numbers also begin with zero. So, "banana": {(2, 3)} means the word "banana" is in the third document (T[2]), and it is the fourth word in that document (position 3).

"a" : {(2, 2)}
"banana" : {(2, 3)}
"is" : {(0, 1), (0, 4), (1, 1), (2, 1)}
"it" : {(0, 0), (0, 3), (1, 2), (2, 0)}
"what" : {(0, 2), (1, 0)}

If we run a phrase search for "what is it" we get hits for all the words in both document 0 and 1. But the terms occur consecutively only in document 1.

Word	*Inverted Index*
a	p_1 p_4
b	p_1 p_2 p_7
c	p_5 p_6 p_8
d	p_2 p^3 p_6 p_8
e	p_4 p_5 p_6 p_7

Fig. 2. Example of an inverted index

Signature file in general refers to a hashing-based framework, whose instantiation is known as *superimposed coding* (SC), which is shown to be more effective than other instantiations. It is designed to perform *membership tests*: determine whether a query word w exists in a set W of words. SC is conservative, 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 must be scanned to avoid a false hit. In the context, SC works in the same way as the classic technique of *bloom filter*. In preprocessing, it builds a bit signature of length l from W by hashing each word in W to a string of l bits, and then taking the disjunction of all bit strings. To illustrate, denote by h(w) the bit string of a word w. First, all the l bits of h(w) are initialized to 0. Then, SC repeats the following m times: randomly choose a bit and set it to 1. Very importantly, randomization must use w as its seed to ensure that the same w always ends up with an identical h(w).

A spatial keyword query consists of a query area and a set of keywords shown in below figure. The answer is a list of objects ranked according to a combination of their distance to the query area and the relevance of their text description to the query keywords. A simple yet popular variant, which is used in our running example, is the distance-first spatial keyword query, where objects are ranked by distance and keywords are applied as a conjunctive filter to eliminate objects that do not contain them.

Furthermore, 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, as will be discussed later. Gives an example to illustrate the above process, assuming l = 5 and m = 2. For example, in the bit string h (a) of a, the 3rd and 5th (counting from left) bits are set to 1. As mentioned earlier, the bit signature of a set W of words simply ORs the bit strings of all the members of W. For instance, the signature of a set {a, b} equals 01101, while that of {b, d} equals 01111.

Word	*hashed bit string*
a	00101
b	01001
c	00011
d	00110
e	10010

Fig. 3. Example of bit string computation with l = 5 and m = 2

5. CONCLUSION

Finally we proposed an efficient a novel search implementation on spatial databases with simple implementation than the complex tree constructions like R trees, in both cache based and non cache based(with geo-codlings),our algorithms shows an optimal results than the traditional approaches. Multi dimensional databases with key word searches. Search for nearest neighbor locations and keywords. In this paper, we have remedied the situation by developing an access method called *the spatial inverted index* (SI-index). Not only that the SI-index is fairly space economical, but also it has the ability to perform keyword-augmented nearest neighbor search in time that is at the order of dozens of milliseconds. Furthermore, as the SI-index is based on the conventional technology of inverted index, it is readily incorporable in a commercial search engine that applies massive parallelism, implying its immediate industrial merits.

REFERENCES

[1] R. Hariharan, B. Hore, C. Li, and S. Mehrotra. Processing spatialkeyword (SK) queries in geographic information retrieval (GIR) systems. In *Proc. of Scientific and Statistical Database Management (SSDBM)*, 2007.

[2] S. Agrawal, S. Chaudhuri, and G. Das. Dbxplorer: A system for keyword-based search over relational databases. In *Proc. Of International Conference on Data Engineering (ICDE)*, pages 5–16,2002.

[3] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using banks. In *Proc. of International Conference on Data Engineering (ICDE)*, pages 431–440, 2002.

[4] X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi. Collective spatial keyword querying. In *Proc. of ACM Management of Data (SIG-MOD)*, pages 373–384, 2011.

[5] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proc. of ACM Management of Data (SIGMOD)*, pages 322–331, 1990.

[6] X. Cao, L. Chen, G. Cong, C. S. Jensen, Q. Qu, A. Skovsgaard, D. Wu, and M. L. Yiu. Spatial keyword querying. In *ER*, pages 16–29, 2012.

[7] X. Cao, G. Cong, and C. S. Jensen. Retrieving top-k prestige-based relevant spatial web objects. *PVLDB*, 3(1):373–384, 2010.

[8] Y.-Y. Chen, T. Suel, and A. Markowetz. Efficient query processing in geographic web search engines. In *Proc. of ACM Management of Data (SIGMOD)*, pages 277–288, 2006.

[9] C. Faloutsos and S. Christodoulakis. Signature files: An access method for documents and its analytical performance evaluation. *ACM Transactions on Information Systems (TOIS)*, 2(4):267–288, 1984.

[10] G. Cong, C. S. Jensen, and D. Wu. Efficient retrieval of the top-k most relevant spatial web objects. *PVLDB*, 2(1):337–348, 2009.

[11] B. Chazelle, J. Kilian, R. Rubinfeld, and A. Tal. The bloomier filter: an efficient data structure for static support lookup tables. In *Proc. of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 30–39, 2004.

[12] E. Chu, A. Baid, X. Chai, A. Doan, and J. Naughton. Combining keyword search and forms for ad hoc querying of databases. In *Proc. of ACM Management of Data (SIGMOD)*, 2009.

[13] I. D. Felipe, V. Hristidis, and N. Rishe. Keyword search on spatial databases. In *Proc. of International Conference on Data Engineering (ICDE)*, pages 656–665, 2008.