Swift Immediate Neighbor Hunt with Keywords

Sherifa. G[1],PG Student, Rajarajan. A[2], Asst. Professor,
Department Of Computer Science and Engineering,
[1]PG Student, Parisutham Institute of Technology and Science, Thanjavur, Tamilnadu, India.
[2]Asst. Professor, Parisutham Institute of Technology and Science, Thanjavur, Tamilnadu, India.

***Abstract*—Conventional spatial queries, such as range search and nearest neighbor retrieval, involve only conditions on objects' geometric properties. Today, many modern applications call for novel forms of queries that aim to find objects satisfying both a spatial predicate, and a predicate on their associated texts. For example, instead of considering all the restaurants, a nearest neighbor query would instead ask for the restaurant that is the closest among those whose menus contain "steak, spaghetti, brandy" all at the same time. Currently, the best solution to such queries is based on the IR^2-tree, which, as shown 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 IR^2-tree in query response time significantly, often by a factor of orders of magnitude.**

***Index Terms*—Nearest neighbor search, keyword search, spatial index**

I. INTRODUCTION

A spatial database manages multidimensional objects (such as points, rectangles, etc.), and provides fast access to those objects based on different selection criteria. The importance of spatial databases is reflected by the convenience of modeling entities of reality in a geometric manner. For example, locations of restaurants, hotels, hospitals and so on are often represented as points in a map, while larger extents such as parks, lakes, and landscapes often as a combination of rectangles. Many functionalities of a spatial database are useful in various ways in specific contexts. For instance, in a geography information system, range search can be deployed to find all restaurants in a certain area, while nearest neighbor retrieval can discover the restaurant closest to a given address.

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.

There are easy ways to support queries that combine spatial and text features. For example, for the above query, we could first fetch all the restaurants whose menus contain the set of keywords {steak, spaghetti, brandy}, and then from the retrieved restaurants, find the nearest one. Similarly, one could also do it reversely 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 key-words. The major drawback of these straightforward approaches is that they will fail to provide real time answers on difficult inputs. A typical example is that the real nearest neighbor lies quite far away from the query point, while all the closer neighbors are missing at least one of the query keywords. Spatial queries with keywords have not been extensively explored. In the past years, the community has sparked enthusiasm in studying keyword search in relational databases.

It is until recently that attention was diverted to multidimensional data [12], [13], [21]. The best method to date for nearest neighbor search with key-words is due to Felipe et al. [12]. They nicely integrate two well-known concepts: R-tree [2], a popular spatial index, and signature file [11], an effective method for key-word-based document retrieval. By doing so they develop a structure called the IR^2-tree [12], which has the strengths of both R-trees and signature files. Like R-trees, the IR^2-tree preserves objects' spatial proximity, which is the key to solving spatial queries efficiently. On the other hand, like signature files, the IR^2-tree is able to filter a consider-able portion of the objects that do not contain all the query keywords, thus significantly reducing the number of objects to be examined.

The IR^2-tree, however, also inherits a drawback of sig-nature files: false hits. That is, a signature file, due to its conservative nature, may still direct the search to some objects, even though they do not have all the keywords. The penalty thus caused is the need to verify an object whose satisfying a query or not cannot be resolved using only its signature, but requires loading its full text description, which is expensive due to the resulting random accesses. It is noteworthy that the false hit problem is not specific only to signature files, but also exists in other methods for approximate set membership tests with compact storage (see [7] and the references therein). Therefore, the problem cannot be remedied by simply replacing signature file with any of those methods.

In this paper, we design a variant of inverted index that is optimized for multidimensional points, and is thus named the spatial inverted index (SI-index). This access method successfully incorporates point coordinates into a conventional inverted index with small extra space, owing to a delicate compact storage scheme. Meanwhile, an SI-index preserves the spatial locality of data points, and comes with an R-tree built on every inverted list at little space overhead. As a result, it offers two competing ways for query processing. We can (sequentially) merge multiple lists very much like merging traditional inverted lists by ids.

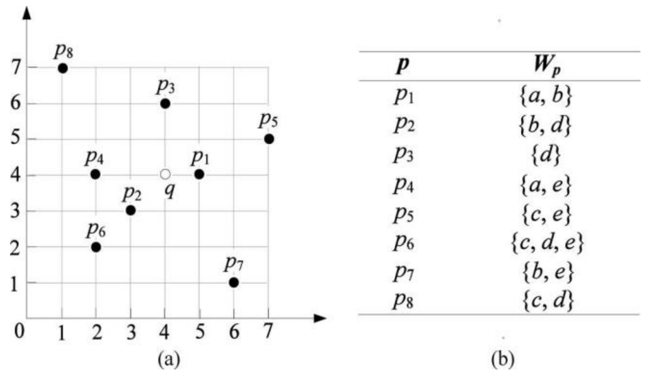
The rest of the paper is organized as follows. Section 2 defines the problem studied in this paper formally. Section 3 surveys the previous work related to ours. Section 4 gives an analysis that reveals the drawbacks of the IR-tree. Section 5 presents a distance browsing algorithm for performing keyword-based nearest neighbor search. Section 6 proposes the SI index, and establishes its theoretical properties. Section 7 evaluates our techniques with extensive experiments. Section 8 concludes the paper with a summary of our findings.

II. PROBLEM DEFINITIONS

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 ½0; 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 representable under a space limit is still finite and enumerable; therefore, we could as well convert everything to integers with proper scaling.

As with [12], each point p 2 P is associated with a set of words, which is denoted as W_p and termed the document of p. For example, if p stands for a restaurant, W_p can be its menu, or if p is a hotel, W_p can be the description of its services and facilities, or if p is a hospital, W_p can be the list of its out-patient specialities. It is clear that W_p may potentially contain numerous words.

Traditional nearest neighbor search returns the data point closest to a query point. Following [12], we extend the problem to include predicates on objects' texts. Formally, in our context, a nearest neighbor (NN) query specifies a point q



p	W_p
p_1	$\{a, b\}$
p_2	$\{b, d\}$
p_3	$\{d\}$
p_4	$\{a, e\}$
p_5	$\{c, e\}$
p_6	$\{c, d, e\}$
p_7	$\{b, e\}$
p_8	$\{c, d\}$

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

and a set W_q of keywords (we refer to W_q as the document of the query). It returns the point in P_q that is the nearest to q, where P_q is defined as

$$Pq = \{ p€P \mid Wq \leq Wp \} \qquad (1)$$

In other words, P_q is the set of objects in P whose documents contain all the keywords in W_q. In the case where P_q is empty, the query returns nothing. The problem definition can be generalized to k nearest neighbor (kNN) search, which finds the k points in P_q closest to q; if P_q has less than k points, the entire P_q should be returned.

For example, assume that P consists of eight points whose locations are as shown in Fig. 1a (the black dots), and their documents are given in Fig. 1b. Consider a query point q at the white dot of Fig. 1a with the set of keywords W_q = fc; dg. 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_8 is also returned in addition. The result is still $\{p_{6,}p_8\}$ even if k increases to 3 or higher, because only two objects have the keywords c and d at the same time.

III. RELATED WORK

Section 3.1 reviews the information retrieval R-tree (IR^2-tree) [12], which is the state of the art for answering the nearest neighbor queries defined in Section 2. Section 3.2 explains an alternative solution based on the inverted index. Finally, Section 3.3 discusses other relevant work in spatial keyword search.

1.The IR^2-Tree

As mentioned before, the IR^2-tree [12] combines the R-tree with signature files. Next, we will review what is a signature file before explaining the details of IR^2-trees. Our discussion assumes the knowledge of R-trees and the best-first algorithm [14] for NN search, both of which are well-known techniques in spatial databases.

Signature file in general refers to a hashing-based frame-work, whose instantiation in [12] is known as superimposed coding (SC), which is shown to be more effective than other
instantiations [11]. 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 of [12], 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, the SC repeats the following m times: randomly choose a bit and set it to 1. Very importantly, the randomization must use was its seed to ensure that the same framework always ends up with an identical h(w).

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

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

Fig. 2 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 third and fifth (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.

Given a query keyword w, SC performs the membership test in W by checking whether all the 1s of h(w) appear at the same positions in the signature of W . If not, it is guaranteed that w cannot belong to W . Otherwise, the test cannot be resolved using only the signature, and a scan of W follows. A false hit occurs if the scan reveals that W actually does not contain w. For example, assume that we want to test whether word c is a member of set a{a,b} using only the set’s signature 01101. Since the fourth bit of h© = 00011 is 1 but that of 01101 is 0, SC immediately reports “no”. As another example, consider the membership test of c in {b,d} whose signature is 01111. This time, SC returns “yes” because 01111 has 1s at all the bits where h(c) is set to 1; as a result, a full scan of the set is required to verify that this is a false hit.

2

The IR -tree is an R-tree where each (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. Fig. 3 demonstrates an example based on the data set of Fig. 1 and the hash values in Fig. 2. The string 01111 in the leaf entry p_2, for example, is the signature of W_{p2} = {b,d} (which is the document of p_2; see Fig. 1b).

On conventional R-trees, the best-first algorithm [14] is a well-known solution to NN search. It is straightforward to adapt it to IR^2-trees. Specifically, given a query point q and a keyword set W_q, the adapted algorithm accesses the entries of an IR^2-tree in ascending order of the distances of their MBRs to q (the MBR of a leaf entry is just the point itself), pruning those entries whose signatures indicate the absence of at least one word of W_q in their subtrees. If W_q is a subset of W_p, the algorithm terminates with p as the answer; otherwise, it continues until no more entry remains to be processed. In Fig. 3, assume that the query point q has a keyword set W_q = fc; dg. It can be verified that the algorithm must read all the nodes of the tree, and fetch the documents of p_2, p_4, and p_6 (in this order). The final answer is p_6, while p_2 and p_4 are false hits.

2.Solutions Based on Inverted Indexes

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. Fig. 4 illustrates the index for the data set of Fig. 1. Each word in the vocabulary has an inverted list, enumerating the ids of the points that have the word in their documents.

According to the experiments of [12], when W_q 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 W_q increases, the performance gap between I-index and IR^2-tree keeps narrowing such that I-index even starts to outper-form IR^2-tree at W_q = 4. This is not as surprising as it may seem. As W_q 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 I-index starts to pay off. That is, scanning an inverted list is relatively cheap because it involves only sequential I/Os, as opposed to the random nature of accessing the nodes of an IR^2-tree.

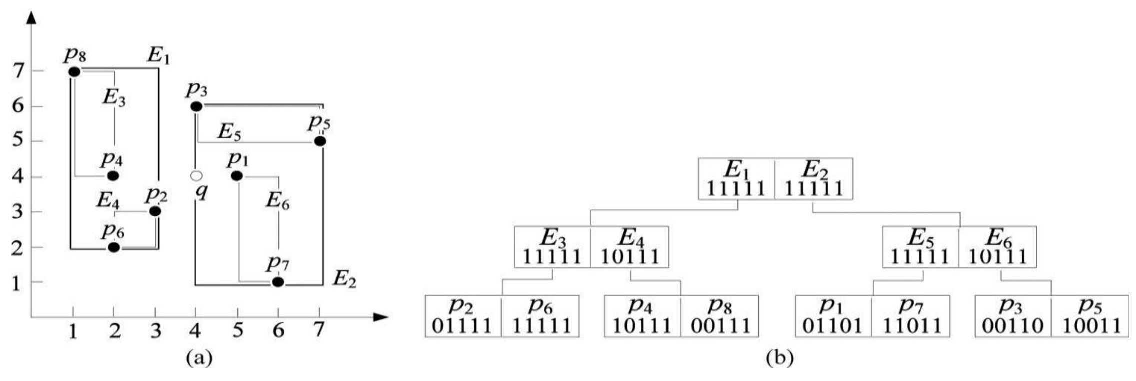


Fig. 3. Example of an IR^2-tree. (a) shows the MBRs of the underlying R-tree and (b) gives the signatures of the entries.

word	*inverted list*
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. 4. Example of an inverted 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.

IV. MERGING AND DISTANCE BROWSING

Since verification is the performance bottleneck, we should try to avoid it. There is a simple way to do so in an I-index: one only needs to store the coordinates of each point together with each of its appearances in the inverted lists. The presence of coordinates in the inverted lists naturally motivates the creation of an R-tree on each list indexing the points therein (a structure reminiscent of the one in [21]). Next, we discuss how to perform keyword-based nearest neighbor search with such a combined structure.

The R-trees allow us to remedy an awkwardness in the way NN queries are processed with an I-index. Recall that, to answer a query, currently we have to first get all the points carrying all the query words in W_q by merging several lists (one for each word in W_q). This appears to be unreasonable if the point, say p, of the final result lies fairly close to the query point q. It would be great if we could discover p very soon in all the relevant lists so that the algorithm can terminate right away. This would become a reality if we could browse the lists synchronously by distances as opposed to by ids. In particular, as long as we could access the points of all lists in ascending order of their distances to q (breaking ties by ids), such a p would be easily discovered as its copies in all the lists would definitely emerge consecutively in our access order. So all we have to do is to keep counting how many copies of the same point have popped up continuously, and terminate by reporting the point once the count reaches $|W_q|$. At any moment, it is enough to remember only one count, because whenever a new point emerges, it is safe to forget about the previous one.

Distance browsing is easy with R-trees. In fact, the best-first algorithm is exactly designed to output data points in ascending order of their distances to q. How-ever, we must coordinate the execution of best-first on $|W_q|$ R-trees to obtain a global access order. This can be easily achieved by, for example, at each step taking a "peek" at the next point to be returned from each tree, and output the one that should come next globally. This algorithm is expected to work well if the query keyword set W_q is small. For sizable W_q, the large number of random accesses it performs may overwhelm all the gains over the sequential algorithm with merging.

A serious drawback of the R-tree approach is its space cost. Notice that a point needs to be duplicated once for every word in its text description, resulting in very expensive space consumption. In the next section, we will over-come the problem by designing a variant of the inverted index that supports compressed coordinate embedding.

V. SPATIAL INVERTED LIST

The spatial inverted list (SI-index) is essentially a compressed version of an I-index with embedded coordinates as described in Section 5. Query processing with an SI-index can be done either by merging, or together with R-trees in a distance browsing manner. Furthermore, the compression eliminates the defect of a conventional I-index such that an SI-index consumes much less space.

1.The Compression Scheme

Compression is already widely used to reduce the size of an inverted index in the conventional context where each inverted list contains only ids. In that case, an effective approach is to record the gaps between consecutive ids, as opposed to the precise ids. For example, given a set S of integers {2,3,6,8}, the gap-keeping approach will store {2,1,3,2} instead, where the ith value i>=2 is the difference between the I th and i=1 values in the original S. As the original S can be precisely reconstructed, no information is lost. The only overhead is that decompression incurs extra computation cost, but such cost is negligible compared to the overhead of I/Os. Note that gap-keeping will be much less beneficial if the integers of S are not in a sorted order.

Compressing an SI-index is less straightforward. The difference here is that each element of a list, a.k.a. a point p, is a triplet $(id_p,x_{p,}y_{p})$ including both the id and coordinates of p. As gap-keeping requires a sorted order, it can be applied on only one attribute of the triplet. For example, if we decide to sort the list by ids, gap-keeping on ids may lead to good space saving, but its application on the x- and y-coordinates would not have much effect.

To attack this problem, let us first leave out the ids and focus on the coordinates. Even though each point has two coordinates, we can convert them into only one so that gap-keeping can be applied effectively. The tool needed is a space filling curve (SFC) such as Hilbert- or Z-curve. SFC converts a multidimensional point to a 1D value such that if two points are close in the original space, their 1D values also tend to be similar. As dimensionality has been brought to 1, gap-keeping works nicely after sorting the (converted) 1D values.

p_6	p_2	p_8	p_4	p_7	p_1	p_3	p_5
12	15	23	24	41	50	52	59

Fig. 5. Converted values of the points in Fig. 1a based on Z-curve.

For example, based on the Z-curve,[2] the resulting values, called Z-values, of the points in Fig. 1a are demonstrated in Fig. 5 in ascending order. With gap-keeping, we will store these 8 points as the sequence 12,3,1, 7, 9, 2, 7. Note that as the Z-values of all points can be accurately restored, the exact coordinates can be restored as well.

Let us put the ids back into consideration. Now that we have successfully dealt with the two coordinates with a 2D SFC, it would be natural to think about using a 3D SFC to cope with ids too. As far as space reduction is concerned, this 3D approach may not a bad solution. The problem is that it will destroy the locality of the points in their original space. Specifically, the converted values would no longer preserve the spatial proximity of the points, because ids in general have nothing to do with coordinates.

If one thinks about the purposes of having an id, it will be clear that it essentially provides a token for us to retrieve (typically, from a hash table) the details of an object, e.g., the text description and/or other attribute values. Further-more, in answering a query, the ids also provide the base for merging. Therefore, nothing prevents us from using a pseudo-id internally. Specifically, let us forget about the “real” ids, and instead, assign to each point a pseudo-id that equals its sequence number in the ordering of Z-values. For example, according to Fig. 5, p_6 gets a pseudo-id 0, p_2 gets a 1, and so on. Obviously, these pseudo-ids can co-exist with the “real” ids, which can still be kept along with objects’ details.

The benefit we get from pseudo-ids is that sorting them gives the same ordering as sorting the Z-values of the points. This means that gap-keeping will work at the same time on both the pseudo-ids and Z-values. As an example that gives the full picture, consider the inverted list of word d in Fig. 4 that contains p_2; p_3; p_6; p_8, whose Z-values are 15, 52, 12, 23 respectively, with pseudo-ids being 1; 6; 0; 2, respectively. Sorting the Z-values automatically also puts the pseudo-ids in ascending order. With gap-keeping, the Z-values are recorded as 12, 3, 8, 29 and the pseudo-ids as 0, 1, 1, 4. So we can precisely capture the four points with four pairs: {(0,12),(1, 3);,(1, 8),(4, 29)}.

$$O(\log u_1+\log u_2+\ldots\ldots\ldots+\log u_r)=o(\log(\pi u_{i)})$$

Since SFC applies to any dimensionality, it is straightforward to extend our compression scheme to any dimensional space. As a remark, we are aware that the ideas of space filling curves and internal ids have also been mentioned in [8] (but not for the purpose of compression). In what follows, we will analyze the space of the SI-index and discuss how to build a good R-tree on each inverted list. None of these issues is addressedin[8].

2.Building R-Trees

Remember that an SI-index is no more than a com-pressed version of an ordinary inverted index with coordinates embedded, and hence, can be queried in the same way as described in Section 3.2, i.e., by merging several inverted lists. In the sequel, we will explore the option of indexing each inverted list with an R-tree. As explained in Section 3.2, these trees allow us to process a query by distance browsing, which is efficient when the query keyword set W_q is small.

Our goal is to let each block of an inverted list be directly a leaf node in the R-tree. This is in contrast to the alternative approach of building an R-tree that shares nothing with the inverted list, which wastes space by duplicating each point in the inverted list. Furthermore, our goal is to offer two search strategies simultaneously: merging (Section 3.2) and distance browsing (Section 5).

As before, merging demands that points of all lists should be ordered following the same principle. This is not a problem because our design in the previous section has laid down such a principle: ascending order of Z-values. Moreover, this ordering has a crucial property that conventional id-based ordering lacks: preservation of spatial proximity. The property makes it possible to build good R-trees without destroying the Z-value ordering of any list. Specifically, we can (carefully) group consecutive points of a list into MBRs, and incorporate all MBRs into an R-tree. The proximity-preserving nature of the Z-curve will ensure that the MBRs are reasonably small when the dimensionality is low. For example, assume that an inverted list includes all the points in Fig. 5, sorted in the order shown. To build an R-tree, we may cut the list into 4 blocks {(p_6,p_2), (p_8, p_4), (p_7, p_1), and (p_3, p_5). Treating each block as a leaf node results in an R-tree identical to the one in Fig. 3a. Linking all blocks from left to right preserves the ascending order of the points' Z-values.

Creating an R-tree from a space filling curve has been considered by Kamel and Faloutsos [16]. Different from their work, we will look at the problem in a more rigorous manner, and attempt to obtain the optimal solution. Formally, the underlying problem is as follows. There is an inverted list L with, say, r points p_1, p_2; . . . ; p_r, sorted in ascending order of Z-values. We want to divide L into a number of disjoint blocks such that (i) the number of points in each block is between B and 2B -1, where B is the block size, and (ii) the points of a block must be consecutive in the original ordering of L. The goal is to make the resulting MBRs of the blocks as small as possible.

How "small" an MBR is can be quantified in a number of ways. For example, we can take its area, perimeter, or a function of both. Our solution, presented below, can be applied to any quantifying metric, but our discussion will use area as an example. The cost of a dividing scheme of L is thus defined as the sum of the areas of all MBRs. For notational convenience, given any 1 <= i <= j <= r, we will use C[i,j] to denote the cost of the optimal division of the subsequence p_i, p_{i+1}; . . . ; p_j. The aim of the above problem is thus to find C[1,r]. We also denote by A½i; j& the area of the MBR enclosing p_i, p_{i+1}; . . . ; p_j.

We have finished explaining how to build the leaf nodes of an R-tree on an inverted list. As each leaf is a block, all the leaves can be stored in a blocked SI-index as described in Section 6.1. Building the non leaf levels is trivial, because they are invisible to the merging-based query algorithms, and hence, do not need to preserve any common ordering. We are free to apply any of the existing R-tree construction algorithms. It is noteworthy that the non leaf levels add only a small amount to the overall space o verhead because, in an R-tree, the number of non leaf nodes is by far lower than that of leaf nodes.

VI. EXPERIMENTS

In the sequel, we will experimentally evaluate the practical efficiency of our solutions to NN search with keywords, and compare them against the existing methods.

Competitors. The proposed SI-index comes with two query algorithms based on merging and distance browsing respectively. Our evaluation also covers the state-of-the-art IR^2-tree; in particular, our IR ^{2}-tree implementation is the fast variant developed in [12], which uses longer signatures for higher levels of tree. Furthermore, we also include the method, named index file R-tree (IFR) henceforth, which, as discussed in Section 5, indexes each inverted list (with coordinates embedded) using an R-tree, and applies distance browsing for query processing.

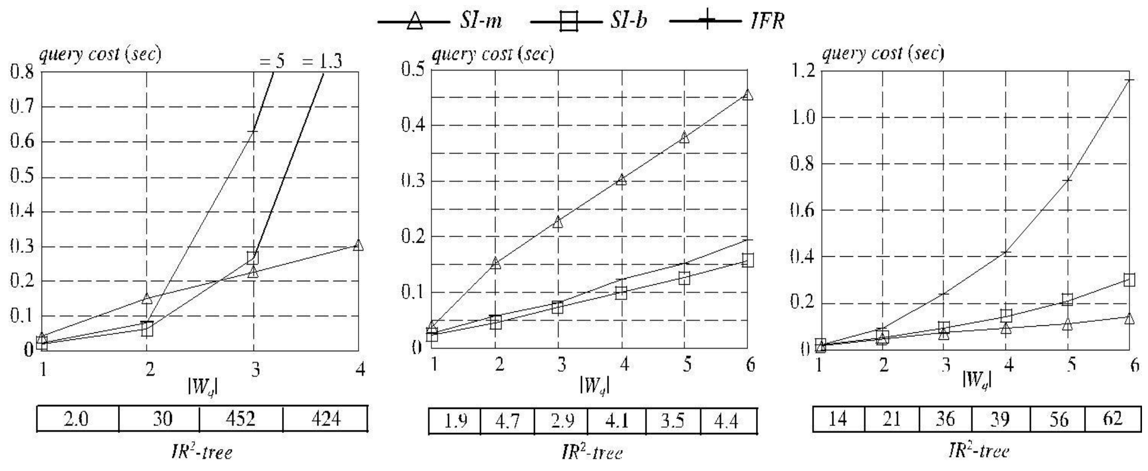


Fig. 6. Query time versus the number of keywords $|W_q|$ (a)Data set Uniform, (b) Skew, (c) Census.

IFR can be regarded as an uncompressed version of SI-b. The page size is always 4;096 bytes. All the SI-indexes have a block size of 200 (see Section 6.1 for the meaning of a block). The parameters of IR^2-tree are set in exactly the same way as in [12]. Specifically, the tree on Uniform has 3 levels, whose signatures (from leaves to the root) have respectively 48, 768, and 840 bits each. The corresponding lengths for Skew are 48, 856, and 864. The tree on Census has two levels, whose lengths are 2; 000 and 47; 608, respectively.

As in [12], we consider NN search with the AND semantic. There are two query parameters: (i) the number k of neighbors requested, and (ii) the number $|W_q|$ of keywords. Each workload has 100 queries that have the same parameters, and are generated independently as follows. First, the query location is uniformly distributed in the data space. Second, the set W_q of key-words is a random subset (with the designated size $|W_q|$ of the text description of a point randomly sampled .

VII. CONCLUSION

We have seen plenty of applications calling for a search engine that is able to efficiently support novel forms of spatial queries that are integrated with keyword search. The existing solutions to such queries either incur prohibitive space consumption or are unable to give real time answers. 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 per-form 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 parallel-ism, implying its immediate industrial merits.

VIII. ACKNOWLEDGMENT

My sincere thanks to my guide Mr.A.Rajarajan Asst.Professor, Department of Computer Science and Engineering, Parisutham Institute of Technology and Science, Thanjavur for his help and guidance.

IX. REFERENCES

[1] I.D. Felipe, V. Hristidis, and N. Rishe, **"Keyword Search on Spatial Databases,"** Proc. Int'l Conf. Data Eng. (ICDE), pp. 656-665, 2008.

[2] R. Hariharan, B. Hore, C. Li, and S. Mehrotra, **"Processing Spatial-Keyword (SK) Queries in Geographic Information Retrieval (GIR) Systems,"** Proc. Scientific and Statistical Database Management (SSDBM), 2007.

[3] G.R. Hjaltason and H. Samet, **"Distance Browsing in Spatial Data-bases,"** ACM Trans. Database Systems, vol. 24, no. 2, pp. 265-318, 1999.

[4] V. Hristidis and Y. Papakonstantinou, **"Discover: Keyword Search in Relational Databases,"** Proc. Very Large Data Bases (VLDB), 670-681, 2002.

[5] I. Kamel and C. Faloutsos**, "Hilbert R-Tree: An Improved R-Tree Using Fractals,"** Proc. Very Large Data Bases (VLDB), pp. 500-509, 1994.

[6] J. Lu, Y. Lu, and G. Cong, **"Reverse Spatial and Textual k Nearest Neighbor Search,"** Proc. ACM SIGMOD Int'l Conf. Management of Data, pp. 349-360, 2011.

[7] S. Stiassny, **"Mathematical Analysis of Various Superimposed Coding Methods,"** Am. Doc., vol. 11, no. 2, pp. 155-169, 1960.

[8] J.S. Vitter, **"Algorithms and Data Structures for External Memo-ry,"** Foundation and Trends in Theoretical Computer Science, vol. 2, no. 4, pp. 305-474, 2006.

[9] D. Zhang, Y.M. Chee, A. Mondal, A.K.H. Tung, and M. Kit-suregawa, **"Keyword Search in Spatial Databases: Towards Searching by Document,"** Proc. Int'l Conf. Data Eng. (ICDE), 688-699, 2009.

[10] Y. Zhou, X. Xie, C. Wang, Y. Gong, and W.-Y. Ma**, "Hybrid Index Structures for Location-Based Web Search,"** Proc. Conf. Information and Knowledge Management (CIKM), pp. 155-162, 20

[11] S. Agrawal, S. Chaudhuri, and G. Das, **"Dbxplorer: A System for Keyword-Based Search over Relational Databases,"** Proc. Int'l Conf. Data Eng. (ICDE), pp. 5-16, 2002.

[12] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger**, "The R - tree: An Efficient and Robust Access Method for Points and Rec-tangles,"** Proc. ACM SIGMOD Int'l Conf. Management of Data, pp. 322-331, 1990.

[13] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan**, "Keyword Searching and Browsing in Databases Using Banks,"** Proc. Int'l Conf. Data Eng. (ICDE), pp. 431-440, 2002.

[14] X. Cao, L. Chen, G. Cong, C.S. Jensen, Q. Qu, A. Skovsgaard, D. Wu, and M.L. Yiu, **"Spatial Keyword Querying,"** Proc. 31st Int'l Conf. Conceptual Modeling (ER), pp. 16-29, 2012.

X. AUTHOR DETAILS

G.sherifa received B.E (Computer Science And Engineering) from Periyar Maniammai University, Thanjavur in 2013. I am currently pursuing M.E (Computer Science And Engineering) in Parisutham Institute of Technology and Science, Thanjavur.

A.Rajarajan received M.C.A from Panimalar Engineering College, Chennai in 2009, M.E (Computer Science and Engineering) from Mount Zion College of Engineering and Technology, Pudukkottai in 2013 and currently working as an Assistant Professor in Parisutham Institute of Technology and Science, Thanjavur.