ISSN: 2321-7782 (Online)
Volume 2, Issue 11, November 2014
International Journal of Advance Research in Computer Science and Management Studies
Research Article / Survey Paper / Case Study
Available online at: www.ijarcsms.com
Special Issue: International Conference on Advanced Research Methodology Held by The International Association for Scientific Research & Engineering Technology, India

SI-index: A New Access Method for Conventional Spatial Queries

Gali Dhanuja[1]
M.Tech Student
Department of Computer Science and Engineering,
Siddharth Institute of Engineering and Technology

S. Hrushikesava Raju[2]
Associate Professor
Department of Computer Science and Engineering,
Siddharth Institute of Engineering and Technology

P. Nirupama[3]
Professor & Head
Department of Computer Science and Engineering,
Siddharth Institute of Engineering and Technology,

Abstract: ***Spatial databases 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 modelling 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 neighbour retrieval can discover the restaurant closest to a given address. Conventional spatial queries, such as range search and nearest neighbour 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.***

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. Alternatively, we can also leverage the R-trees to browse the points of all relevant lists in ascending order of their distances to the query point.

Keywords: Spatial databases, Spatial quiries, SI-index

I. INTRODUCTION

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 neighbour 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. 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 neighbour query), but the nearest restaurant among only those providing all the demanded foods and drinks.

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 keywords. 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 neighbour lies quite far away from the query point, while all the closer neighbours are missing at least one of the query keywords.

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. Alternatively, we can also leverage the R-trees to browse the points of all relevant lists in ascending order of their distances to the query point. As demonstrated by experiments, the SI-index significantly outperforms the IR2-tree in query efficiency, often by a factor of orders of magnitude.

The rest of the paper is organized as follows. Section 2 Surveys the previous work related to ours. Section 3 defines the problem studied in this paper formally Section 4 presents a distance browsing algorithm for performing keyword-based nearest neighbour search. Section 5 proposes the SI-idnex, and establishes its theoretical properties. Section 6 evaluates our techniques with extensive experiments. Section 7 concludes the paper with a summary of our findings.

II. Related Work

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 [1], [2],. The best method to date for nearest neighbour search with keywords is due to Felipe et al. [1]. They nicely integrate two well-known concepts: R-tree [4], a popular spatial index, and signature file [3], an effective method for keyword based document retrieval. By doing so they develop a structure called the IR2-tree [1], 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 efficiently. On the other hand, like signature files, the IR2-tree is able to filter a considerable portion of the objects that do not contain all the query keywords, thus significantly reducing the number of objects to be examined.

The IR2-tree, however, also inherits a drawback of signature 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 [5] 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. Alternatively, we can also leverage the R-trees to browse the points of all relevant lists in ascending order of their distances to the query point. As demonstrated by experiments, the SI-index significantly outperforms the IR2-tree in query efficiency, often by a factor of orders of magnitude.

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

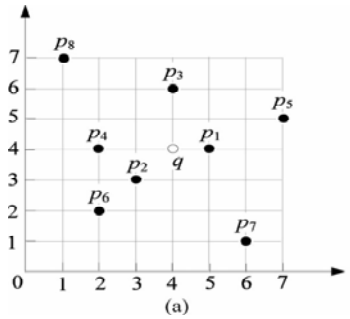
As with [1], each point p € P is associated with a set of words, which is denoted as Wp and termed the document of

p. For example, if p stands for a restaurant, Wp can be its menu, or if p is a hotel, Wp can be the description of its services and facilities, or if p is a hospital, Wp can be the list of its out-patient specialities. It is clear that Wp may potentially contain numerous words.

Traditional nearest neighbour search returns the data point closest to a query point. Following [1], we extend the problem to include predicates on objects' texts. Formally, in our context, a nearest neighbour (NN) query specifies a point q and a set Wq of keywords (we refer to Wq as the document of the query). It returns the point in Pq that is the nearest to q, where Pq is defined as

$$P_q = \{p \in P \mid W_q \subseteq W_p\}.$$

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



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}

(b)

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

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 Wq = c,d. Nearest neighbour search finds p6, noticing that all points closer to q than p6 are missing either the query keyword c or d. If k = 2 nearest neighbours are wanted, p8 is also returned in addition. The result is still {p6; p8} even if k increases to 3 or higher,

because only two objects have the keywords c and d at the same time. We consider that the data set does not fit in memory, and needs to be indexed by efficient access methods in order to minimize the number of I/Os in answering a query.

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). Next, we discuss how to perform keyword-based nearest neighbour 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 Wq by merging several lists (one for each word in Wq). 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 |Wq| 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.

As an example, assume that we want to perform NN search whose query point q is as shown in Fig. 1, and whose Wq equals fc; dg. Hence, we will be using the lists of words c and d in Fig. 4. Instead of expanding these lists by ids, the new access order is by distance to q, namely, p2; p3; p6; p6; p5; p8; p8. The processing finishes as soon as the second p6 comes out, without reading the remaining points. Apparently, if k nearest neighbours are wanted, termination happens after having reported k points in the same fashion. 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. However, we must coordinate the execution of best-first on |Wq|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 Wq is small. For sizable Wq, 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 overcome 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 4. 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 f2; 3; 6; 8g, the gap-keeping approach will store f2; 1; 3; 2g instead, where the ith

value (i-2) is the difference between the ith and (i-1)Þth 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. This is because the space saving comes from the hope that gaps would be much smaller (than the original values) and hence could be represented with fewer bits. This would not be true had S not been sorted. 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 (idp; xp; yp) 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 gapkeeping 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 value.

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. 2. 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. 2 in ascending order. With gap-keeping, we will store these 8 points as the sequence 12, 3, 8, 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. Furthermore, in answering a query, the ids also provide the base for merging. Therefore, nothing prevents us from using a pseudo-id internally.

For example, according to Fig. 2, p6 gets a pseudo-id 0, p2 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. As an example that gives the full picture, consider the inverted list of word d in Fig. 2 that contains p2, p3, p6, p8, 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)}. Since SFC applies to any dimensionality, it is straightforward to extend our compression scheme to any dimensional space.

Let us assume that the whole data set has n>1 points and r of them appear in L. To make our analysis general, we also take the dimensionality d into account. Also, recall that each coordinate ranges from 0 to t, where t is a large integer. Naively, each pseudo-id can be represented with O(log n) bits, and each coordinate with O(log t) bits. Therefore, without any compression, we can represent the whole L in O(r(log n + d log t)) bits.

2) Building R-Trees

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 and distance browsing (Section 4).

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 {p6; p2}, {p8; p4g}, {p7; p1} and {p3; p5}. 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 p1, p2; . . . ; pr, 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. The total number of choices may be less than B-1 because care must be taken to make sure that the number of those remaining points is at least B. In any case, C[i; j] equals the lowest cost of all the permissible choices, or formally:

$$C[i,j] = \min_{k=i+B-1}^{\min\{i+2B-2, j+1-B\}} (A[i,k] + C[k+1,j]).$$

The equation indicates the existence of solutions based on dynamic programming. One can easily design an algorithm that runs in $O(Br^2)$time: it suffices to derive C[I, j] in ascending order of the value of j- i, namely, starting with those with j- i=2B, followed by those with j-i=2B-1, and so on until finishing at j - i = r -1. We can significantly improve the computation time to O(Br), by the observation that j can be fixed to r throughout the computation in order to obtain C[1, r] eventually.

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 5 A. Building the nonleaf 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 nonleaf levels add only a small amount to the overall space overhead because, in an R-tree, the number of nonleaf nodes is by far lower than that of leaf nodes.

	number of points	vocabulary size	average number of objects per word	average number of words per object
Uniform	1 million	200	50k	10
Skew	1 million	200	50k	10
Census	20847	292255	33	461

Table 1. Data set

VI. EXPERIMENTS

Data. Our experiments are based on both synthetic and real data. The dimensionality is always 2, with each axis consisting of integers from 0 to 16; 383. The synthetic category has two data sets: Uniform and Skew, which differ in the distribution of data points, and in whether there is a correlation between the spatial distribution and objects' text documents. Specifically, each data set has 1 million points.

We use the name of the subdivision to search for its page at Wikipedia, and collect the words there as the text description of the corresponding data point. All the points, as well as their text documents, constitute the data set Census. The main statistics of all of our data sets are summarized in Table 1.

Results on space consumption. We will complete our experiments by reporting the space cost of each method on each data set. While four methods are examined in the experiments on query time, there are only three as far as space is concerned. Remember that SI-m and SI-b actually deploy the same SI-index and hence, have the same space cost. In the following, we will refer to them collectively as SI-index. Fig.3 gives the space consumption of IR2-tree, SI-index, and IFR on data sets Uniform, Skew, and Census, respectively.

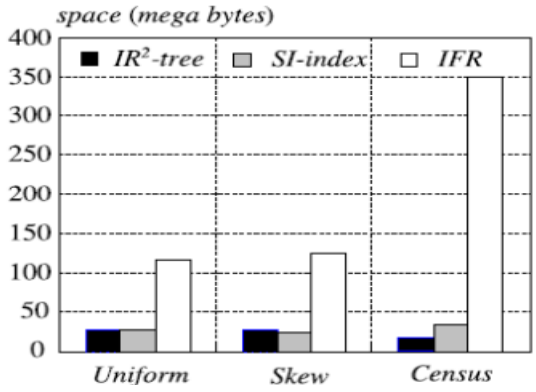


Fig 3 comparison of space consumption

As expected, IFR incurs prohibitively large space cost, because it needs to duplicate the coordinates of a data point p as many times as the number of distinct words in the text description of p. As for the other methods, IR2-tree appears to be slightly more space efficient, although such an advantage does not justify its expensive query time, as shown in the earlier experiments. Summary. The SI-index, accompanied by the proposed query algorithms, has presented itself as an excellent tradeoff between space and query efficiency. Compared to IFR, it consumes significantly less space, and yet, answers queries

VII. CONCLUSION

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. Alternatively, we can also leverage the R-trees to browse the points of all relevant lists in ascending order of their distances to the query point.

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. C. Faloutsos and S. Christodoulakis, "Signature Files: An Access Method for Documents and Its Analytical Performance Evaluation," ACM Trans. Information Systems, vol. 2, no. 4, pp. 267-288 1984.

4. N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger, "The Rtree: An Efficient and Robust Access Method for Points and Rectangles," Proc. ACM SIGMOD Int'l Conf. Management of Data, pp. 322-331, 1990.

5. B. Chazelle, J. Kilian, R. Rubinfeld, and A. Tal, "The Bloomier Filter: An Efficient Data Structure for Static Support Lookup Tables," Proc. Ann. ACM-SIAM Symp. Discrete Algorithms (SODA), pp. 30-39, 2004.

