Improved IR2-Tree Using SI-Index For Spatial Search

Sridevi.P 1, Loganathan.B 2
1Student , 2Associate Professor
Government arts college, Coimbatore, India

Abstract—Many applications require finding objects nearest neighbor to a specified location that contains a set of keywords. This gives prominence to spatial web data management. Specifically, a spatial keyword query takes a user location and user-supplied keywords as arguments and returns web objects that are spatially and textually relevant to these arguments. This paper aims to achieve spatial keyword querying functionality that is easy to use, relevant to users, and can be supported efficiently. To do so, we currently use an indexing structure called IR2-Tree (Information Retrieval R-Tree) which combines an R-Tree with superimposed text signatures which has a few deficiencies that seriously impact its efficiency. We present spatial Inverted Index, an efficient method to answer top-k spatial keyword queries. Our algorithms are experimentally compared to current methods and are shown to have superior performance and excellent scalability.

Keywords- IR2-Tree, R-tree, inverted index (SI-index), Meanwhile, spatial proximity, spectrophotometer.

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.

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. The best method to date for nearest neighbor search with keywords is due to Felipe et al. They nicely integrate two well-known concepts: R-tree, a popular spatial index, and signature file, an effective method for keyword-based document retrieval. By doing so they develop a structure called the IR2-tree 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 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. 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.

- The main objective of this system is to find nearest node and result should be accurate.
- To find nearest hotels based on user queries.

PROBLEM DEFINITION

Today, the widespread use of search engines has made it realistic to write spatial queries in a brand-new way. Conventionally, spatial queries such as range search and nearest neighbor retrieval 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. But the major drawback of these straightforward approaches is that they will fail to provide real time answers on difficult inputs. The another method for nearest neighbor search with keywords is based on IR2-tree. The IR2-tree preserves objects’ spatial proximity, which is the key to solving spatial queries efficiently and also it can 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 but it still direct the search to some objects, even though they do not have all the keywords so it is considered as one of the drawback of this approach. 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. 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.

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 to gather 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 Wq by merging several lists (one for each word in Wq).

Fig. 1. The locations of points and (b) their associated texts.

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.

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.

**MERGING AND DISTANCE BROWSING**

**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.

**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 1Pth 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 triple 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

<table>
<thead>
<tr>
<th>p6</th>
<th>p2</th>
<th>p8</th>
<th>p4</th>
<th>p7</th>
<th>p1</th>
<th>p3</th>
<th>p5</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>15</td>
<td>23</td>
<td>24</td>
<td>41</td>
<td>50</td>
<td>52</td>
<td>59</td>
</tr>
</tbody>
</table>

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

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.

For example, based on the Z-curve, 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; 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 be 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. 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, 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.

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. 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: f0; 10f; 12f; 10f; 3f; 3f; 8f; 3f; 29P. 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

**Building R-Trees**
Remember that an SI-index is no more than a compressed 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 Wq 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: p6; p2g, p8; p4g, p7; p1g, and p3; p5g. 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 blocksize, 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. constitute a smaller set on which the division problem needs to be solved recursively. 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, Cj = the lowest cost of all the permissible choices, or formally

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 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.

**EXISTING SYSTEM**

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. They nicely integrate two well-known concepts: R-tree, a popular spatial index, and signature file, an effective method for keyword-based document retrieval. By doing so they develop a structure called the IR2-tree, 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.

- Less accuracy
- Nearest node cannot be found

**PROPOSED SYSTEM**

Nearest neighbor search is an optimization problem for finding closest points of the objects current location. In this system a variant of inverted index is used, 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.

A spatial keyword consists of a query and a set of keywords. The answer is a list of objects ranked according to a combination of their distance to the query keywords. Objects are ranked by distance and keywords are applied as to eliminate objects that do not contain them. Search result can be viewed as per the distance from user’s location. In this user search location has been depicted in map. User can also get the information like the food festival, offers in particular restaurant or shop nearby his current location.

- More accurate nearest neighborhood search.
- Compute the distance and sort all the training data at each prediction.
- Greatly improved search performance

**Fast Keyword Knn Search System**

Location-based services (LBS) have become more and more popular nowadays since people equipped with smart phones. Existing keyword k-nearest neighbor (kNN) search methods focus more on the keywords; therefore, they use direct distance of two points, also known as, Euclidean distance as spatial constraints. However, the nearest point-of-interest (POI) returned by these services may not be the nearest on the road networks. For some services that consider the road networks, they use road expansion methods to solve this problem. The speed limitation for a large road network and index structures for millions POI may be the bottlenecks for these services. To address those problems, we develop a fast keyword kNN search system in road networks, called Saturn. Instead of using road expansion methods, we introduce a grid-based shortest path computation method, a filter-and-verification framework to search fast in road networks, and we also devise an improvement of the grid index to further improve the performance. We conduct extensive experiments on real data sets, and the experimental results show that our method is efficient and scalable to large data sets, significantly outperforming state-of-the-art methods.

Conservative spatial queries, such as range search and nearest neighbor reclamation, involve only conditions on objects’ numerical 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 InformationRetrieval2-tree, which has a few deficiencies that seriously impact its efficiency.

Motivated by this, there is a development of 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 InformationRetrieval2-tree in query response time significantly, often by a factor of orders of magnitude. Currently the best solution to such queries is based on the InformationRetrieval2-tree, which has a few deficiencies that seriously impact its efficiency. Motivated by this, there is a development of 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 InformationRetrieval2-tree in query response time significantly, often by a factor of orders of magnitude.

**High Dimensional Nearest Neighbor Search**

Nearest neighbor (NN) search in high dimensional space is an important problem in many applications. Ideally, a practical solution (i) should be implementable in a relational database, and (ii) its query cost should grow sub linearly with the dataset size, regardless of the data and query distributions. Despite the bulk of NN literature, no solution fulfills both requirements, except locality sensitive hashing (LSH).

The existing LSH implementations are either rigorous or adhoc. Rigorous-LSH ensures good quality of query results, but requires expensive space and query cost. Although adhoc-LSH is more efficient, it abandons quality control, i.e., the neighbor it outputs can be arbitrarily bad. As a result, currently no method is able to ensure both quality and efficiency simultaneously in practice.

Motivated by this, we propose a new access method called the locality sensitive B-tree (LSB-tree) that enables fast high-dimensional NN search with excellent quality. The combination of several LSB-trees leads to a structure called the LSB-forest that ensures the same result quality as rigorous-LSH, but reduces its space and query cost dramatically.

The LSB-forest also outperforms adhoc-LSH, even though the latter has no quality guarantee. Besides its appealing theoretical properties, the LSB-tree itself also serves as an effective index that consumes linear space, and supports efficient updates. Our extensive experiments confirm that the LSB-tree is faster than (i) the state of the art of exact NN search by two orders of magnitude, and (ii) the best (linear-space) method of approximate retrieval by an order of magnitude, and at the same time, returns neighbors with much better quality.
The algorithm builds a nearest neighbor graph in an offline phase and when queried with a new point, performs hill-climbing starting from a randomly sampled node of the graph. We provide theoretical guarantees for the accuracy and the computational complexity and empirically show the effectiveness of this algorithm.

**K-Nearest Neighbor Search In Spatial Databases**

With the ever-increasing number of spatiotextual objects, many applications require to find objects close to a given query point in spatial databases. In this paper, we study the problem of keyword-based k-nearest neighbor search in spatial databases, which, given a query point and a set of keywords, finds k-nearest neighbors of the query point that contain all query keywords.

To efficiently answer such queries, we propose a new indexing framework by integrating a spatial component and a textual component, which can efficiently prune search space in terms of both spatial information and textual descriptions. We develop effective index structures and pruning techniques to improve query performance. Experimental results show that our approach significantly outperforms state-of-the-art methods.

Conventional abstraction queries, like vary search and nearest neighbor retrieval, involve solely conditions on objects’ geometric properties. Today, several trendy applications involve novel kinds of queries that aim to seek out objects satisfying each a abstraction predicate, and a predicate on their associated texts. As an example, rather than considering all the restaurants, a nearest neighbor question would instead elicit the eating house that’s the highest among those whose menus contain “steak, spaghetti, brandy” all at an equivalent time. Presently the most effective resolution to such queries is predicated on the IR2-tree, which, as shown during this paper, features a few deficiencies that seriously impact its potency.

Impelled by this, we tend to develop a replacement access methodology known as the abstraction inverted index that extends the standard inverted index to address flat knowledge, and comes with algorithms that may answer nearest neighbor queries with keywords in real time. As verified by experiments, the projected techniques outgo the IR2-tree in question latent period considerably, typically by an element of orders of magnitude.

**Multidimensional Nearest Neighbor Search**

The nearest neighbor search in high-dimensional spaces is an interesting and important problem that is relevant for a wide variety of applications, including multimedia information retrieval, data mining, and pattern recognition. For such applications, the curse of high dimensionality tends to be a major obstacle in the development of efficient search methods.

This paper addresses the problem of designing a new and efficient algorithm for high-dimensional nearest neighbor search based on ellipsoid distance. The proposed algorithm uses Cholesky decomposition to perform data conversion beforehand so that calculation by ellipsoid distance function can be replaced with calculation by Euclidean distance, and it improves efficiency by omitting an unnecessary operation. Experimental results indicate that our scheme scales well even for a very large number of dimensions.

Many applications require finding objects closest to a specified location that contains a set of keywords. For example, online yellow pages allow users to specify an address and a set of keywords. In return, the user obtains a list of businesses whose description contains these keywords, ordered by their distance from the specified address. The problems of nearest neighbor search on spatial data and keyword search on text data have been extensively studied separately. The algorithm builds a nearest neighbor graph in an offline phase and when queried with a new point, performs hill-climbing starting from a randomly sampled node of the graph. We provide theoretical guarantees for the accuracy and the computational complexity and empirically show the effectiveness of this algorithm.

However, to the best of our knowledge there is no efficient method to answer spatial keyword queries, that is, queries that specify both a location and a set of keywords. In this work, we present an efficient method to answer top-k spatial keyword queries. To do so, we introduce an indexing structure called IR2-Tree (Information Retrieval R-Tree) which combines an R-Tree with superimposed text signatures. We present algorithms that construct and maintain an IR2-Tree, and use it to answer top-k spatial keyword queries. Our algorithms are experimentally compared to current methods and are shown to have superior performance and excellent scalability.

**CONCLUSIONS**

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 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. Google Map search is also added with this system. It assists user find the nearest neighbor efficiently. Multilingual support can be provided so that it can be understandable by the person of any language.

- More graphics can be added to make it more user-friendly and understandable.
- The future enhancement is we develop the application through website and useful manner than the present one.
- Automatically e-mail communication to Customer about food details and new schemes and discounting.

REFERENCES