International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Impact Factor (2012): 3.358

Developing an Effective System for Keyword Based Query Processing

Ashwini P. Kshirsagar, G.M.Bhandari

Abstract: ***Range search and nearest neighbor retrieval, involve only conditions on objects which come under conventional spatial queries. 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. 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 milli-seconds. 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. So the results are more relevant and fast, 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.***

Keywords: Spatial Index, Ranking, Inverted Index

1. Introduction

On the one hand, to meet the effective data retrieval need, the large amount of documents demand the cloud server to perform result relevance ranking, instead of returning undifferentiated results. Such ranked search system enables data users to find the most relevant information quickly, rather than burdensomely sorting through every match in the content collection. Ranked search can also elegantly eliminate unnecessary network traffic by sending back only the most relevant data, which is highly desirable in the "pay-as-you-use" cloud paradigm.

We are going to develop a new access method called the spatial inverted index that extends the conventional inverted index to cope with multidimensional data, and comes with algorithms that can answer nearest neighbor queries with keywords in real time. As verified by experiments, the proposed techniques outperform the IR2-tree in query response time, often by a factor of orders of magnitude.

Spatial web objects are gaining in prevalence, and numerous works on geographical retrieval study the problem of extracting geographic information from web pages (e.g., [1, 11, 13]), which yields spatial web objects that can subsequently be queried. Commercial services such as Google and Yahoo! offer local search functionality. Given a spatial keyword query, they return spatial web objects, e.g., stores and restaurants, near the query location. The results consist of single objects that each satisfy the query in isolation.

In contrast, we aim to find groups of objects such that the objects in a group collectively satisfy a query. Several recently proposed hybrid indexes that tightly integrate spatial indexing (e.g., the R-tree) and text indexing (e.g., inverted lists). In these indexes, each entry *e* in a tree node stores a *keyword summary field* that concisely summarizes the keywords in the subtree rooted at *e*. This enables irrelevant entries to be pruned during query processing. The *IR2-tree* and the *bR*-tree* [14] augment the R-tree with signatures and bitmaps, respectively. We use the IR-tree [8], covered in Section 3.1, as our index structure due to two features of the IR-tree. First, the fanout of the tree is independent of the number of words of objects in the dataset. Second, during query processing, only (a few) posting lists relevant to the query keywords need to be fetched.

However, we note that our proposed algorithms are not tied to the IR-tree, but can be used also with the other tightly combined index. Most existing works on spatial keyword queries retrieve single objects that are close to the query point and are relevant to the query keywords. In contrast, we retrieve groups of objects that are close to the query point and collectively meet the keywords requirement. To the best of our knowledge, the only work that retrieves groups of spatial keyword objects relates to the *m*CK query [14, 15] that takes a set of *m* keyword as argument.

It returns *m* objects of minimum diameter that match the *m* keywords. It is assumed that each object in the result corresponds to a unique query keyword. In contrast, our query takes both a spatial location and a set of keywords as arguments, and its semantics are quite different from those of the *m*CK query.

2. Literature Survey

Literature survey is the most important step in software development process. Before developing the tool it is necessary to determine the time factor, economy n company strength. Once these things are satisfied, ten next steps are to determine which operating system and language can be used for developing the tool. Once the programmers start building the tool the programmers need lot of external support. This support can be obtained from senior programmers, from book or from websites. Before building the system the above consideration are taken into account for developing the proposed system.

Various spatial queries using R-tree [11] and R*-tree [5] have been extensively studied. Besides the popular nearest neighbor query [23] and range query [18], closest-pair queries for spatial data using R-trees have also been investigated [13], [9], [25]. Nonincremental recursive and

iterative branchand- bound algorithms for *k*-closest pairs queries have been discussed by Corral et al. [9]. An incremental algorithm based on priority queues for the distance join query has been discussed by Hjaltason and Samet [13].

Moreover, their queries specify a set of spatial locations, while our queries specify keywords with no specific spatial location. Various studies have also been done on finding association rules and co-location patterns in spatial databases [16], [24], [29], the aim being to find objects that frequently occur near to each other. Objects are judged to be near to each other if they are within a specified threshold distance of each other. Our study here is a useful alternative which foregoes the distance threshold, but instead allows users to verify their hypothesis through spatial discovery.

The objects returned are required to intersect with the query MBR and contain all the user-specified keywords. A hybrid index of R*-tree and inverted index, called the KR*-tree, is used for query processing. Felipe et al. [10] proposed another similar query combining *k*-NN query and keyword search, and uses a hybrid index of R-tree and signature file, called the *IR*2. Our *m*CK query differs from these two queries. First, our query specifies keywords with no specific location. Second, all the userspecified keywords do not necessarily appear in one result tuple. They can appear in multiple tuples as long as the tuples are closest in space.

3. Objective and Scope

Main contributions are summarized as follows:

1) This is the first work that addresses the keyword based search. We also identify three crucial issues that an effective spatial data based and desired keyword based search engine should meet.
2) We have defined and used a method for combining spatial data index and IR Trees.
3) We propose three important guidelines in identifying the user desired keyword *search for* node type, and design a formula to compute the confidence level of a certain node type to be a desired *search for* node based on the guidelines.
4) We design formulae to compute the confidence of each candidate node type as the desired *search via* node to model natural human intuitions, in which we take into account the pattern of keywords co-occurrence in query.
5) We implement the proposed techniques in a keyword search technique prototype called IR2Trees Extensive experiments, show its effectiveness, efficiency and scalability.

4. Methodology

4.1 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. 6.1 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 ði _ 1Þth 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 gap-keeping can be applied effectively. The tool needed is a space filling curve (SFC) such as Hilbert- or Z-curve.

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

Figure 5: Converted values of the points in Fig. 1a based on 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.

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; 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.

4.2 Building R-Tree

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).

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.

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

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 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.

5. Conclusion

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 much faster. Compared to IR2-tree, its superiority is overwhelming since its query time is typically lower by a factor of orders of magnitude. 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 milli-seconds.

References

[1] 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.
[2] X. Cao, G. Cong, and C.S. Jensen, “Retrieving Top-k Prestige- Based Relevant Spatial Web Objects,” Proc. VLDB Endowment, vol. 3, no. 1, pp. 373-384, 2010.
[3] X. Cao, G. Cong, C.S. Jensen, and B.C. Ooi, “Collective Spatial Keyword Querying,” Proc. ACM SIGMOD Int’l Conf. Management of Data, pp. 373-384, 2011.
[4] I.D. Felipe, V. Hristidis, and N. Rishe, “Keyword Search on Spatial Databases,” Proc. Int’l Conf. Data Eng. (ICDE), pp. 656-665, 2008.
[5] R. Hariharan, B. Hore, C. Li, and S. Mehrotra, “Processing Spatial- Keyword (SK) Queries in Geographic Information Retrieval (GIR) Systems,” Proc.
[6] D. Zhang, Y.M. Chee, A. Mondal, A.K.H. Tung, and M. Kitsuregawa, “Keyword Search in Spatial Databases: Towards Searching by Document,” Proc. Int’l Conf. Data Eng. (ICDE), pp. 688-699, 2009.