ISBN 978-981-11-0008-6
Proceedings of 2016 6th International Workshop on Computer Science and Engineering (WCSE 2016)
Tokyo, Japan, 17-19 June, 2016, pp. 518-524
doi: 10.18178/wcse.2016.06.087

Multi-IRS: Multiple Trees Indexing for Generic Location-Aware Rank Query

Utharn Buranasaksee [1 +] and Kriengkrai Porkaew[2]

[1] Rajamangala University of Technology Suvarnabhumi, Phranakhon Si Ayutthaya, Thailand
[2] King Mongkut's University of Technology Thonburi, Bangkok, Thailand

Abstract. The mobile usage nowaday makes the data on the Internet becomes more location-aware. Searching two-dimensional space with the text requires a powerful index structure that can combine two data types in the same index. Though there have been many indexes proposed to solve location-aware rank query problem by combining such information within the same data structure, in the big data era, many new datatypes are introduced and required to search with the geolocation information. Integrating multiple datatypes to spatial-textual objects requires a new index structure that can efficiently perform searching those generic datatypes. Though there were some existing studies that proposed the framework such as inverted R-tree with synopses (IRS), the framework is not able to achieve optimized performance due to the index creation process remains same as the traditional method. This paper presents the multiple trees indexing that can improve the optimization of the index structure based on the given query at the runtime. In the experimental, our proposed method can significantly outperform the state-of-the-art method on the real dataset.

Keywords: multiple, index, generic, location-aware, query.

1. Introduction

With the increased usage of mobile phone, more information on the Internet becomes more geo-tagged. As a result, more and more location-aware datasets have been created. Therefore, the need of location-aware information searching gains more attention. However, the spatial information is two-dimensional and in the Euclidean distance while the textual information is a set of keywords. Indexing two different datatypes together would not be done in straightforward approach as the performance is poor when separated indexes are maintained [1]. Recently, the development of the searching techniques that index both spatial information and textual information have been increasingly proposed. Many of the techniques make use of text relevancy by creating additional textual information along with the node in the spatial data structure such as Inverted R-Tree (IR-Tree). One of the main applications of spatio-textual is to search the objects based on the given spatial proximity and text relevancy which is called as Location-aware Top-K Query (LKQ) problem [2].

Fig. 1 Sample Book Review from Amazon website.

With the big data era, new datatypes that are associated with the location-aware objects has been introduced. For example, Fig. 1 shows a sample of a book review obtained from Amazon website. For the

+ Corresponding author. Tel.: +66 (87) 8766345
E-mail address: utharn.b@rmutsb.ac.th

book review, Amazon gives its reviewer location, book categories, helpfulness, overall, and review time. The book contains many datatypes including spatial location, text, number, and date. The user may want to very helpful reviews ("helpfulness > 4") that posted within the year 2015 ("review time = 2015"), and the reviewer stays in the same state as the user ("location nears California"). Therefore, searching on such datasets requires a more powerful index data structure that can serve the demand. While the existing approaches were proposed to solve the LKQ problem, those indexes cannot be performed efficiently on the data that contain the other attributes as the validity of other attributes needs to be checked afterward in the different step.

Therefore, a generic location-aware rank query (GLRQ) problem was proposed [3] to search the objects based on the given spatial proximity, text relevancy, and the other attributes. A GLRQ query is composed of a query location, a set of keywords, a query predicate, and a ranking function. A query predicate is a constraint on the other attributes which allow the user to search the objects on specific conditions. A constraint can be a single value or a range of interesting value such as (price = $25), or (review time >= 1 January 2015). A ranking function is a user-provided function that returns the most suitable objects measured by a combination of spatial location, text relevancy, and the other attributes. However, if the object does not satisfy with the query predicate, the ranking function will return the score as zero.

To solve GLRQ problem, Liu et al. [3] proposed the use of a synopses tree that can be applied to the existing indexes refer as Inverted-file R-Tree with a Synopses Tree (IRS-Tree). This technique improves the system by pruning unmatching objects during the index traversal. However, those IRS-Trees are still not optimized in some situation such that the matching attributes are spread over space. As a result, less number of objects are pruned. There are two main reasons for these shortcomings. First, the query predicate is given at the runtime. Therefore, there is no way to determine the query beforehand. Second, the tree construction process creates only one type to base on the priority of the attributes such as spatial proximity, then text relevancy, then the other attributes. As a result, the tree is constructed to serve various types of the query, not a specific type of query.

In this work, we point out the optimization that can improve the existing indexes in GLRQ problem. The optimization can be used to enhance the performance based on the query given at the runtime. Second, we proposed a multiple index structure called Multi-IRS that generates multiple indexes which will be used to serve a different type of the query. Since there are more than one indexes to choose when to perform searching, we propose an execution system which will determine the query at the runtime and choose the most optimized index for that query. Finally, through the extensive experimental in GLRQ, we compare the efficiency of the proposed method and the state-of-the-art method regarding query performance. We show that our Multi-IRS method significantly outperforms the state-of-the-art method.

We organize the content in this paper as follows. In Section 1, we point out the optimization problem in the existing GLRQ indexes. Then the existing spatial keyword search indexes are reviewed in Section 2. We formulate our problem in details in Section 3. Then, in Section 4, our proposed index is illustrated. Finally, the experimental study is discussed and the conclusion is drawn in Section 5 and 6 respectively.

2. Preliminary Studies

We classify the existing spatial keyword search indexes into two categories.

2.1. Textual-first approach

This approach creates an index structure that is based on textual information first. Then a tree structure that will point the objects of which geo-location is spread over space. As the result, the indexes constructed using this approach such as S2I [4] and TAS [5] would contain the textual information, e.g. branching by a character in English, in the main tree with which a sub-tree is containing the objects that share the same keyword is associated.

To solve GLRQ query, the indexes using textual-first approach can be extended using B+-Tree lookup table technique [6] which will store information of the other attributes along with each sub-tree. The B+-Tree is sorted by the object ID to improve the retrieval process. However, Kwon and Whang [6] defines the

problem statement slightly different from our GLRQ. Therefore, in this work, we focus on index optimization in spatial-first approach.

2.2. Spatial-first approach

This approach creates an index structure that is based on spatial first. Then the inverted file that stores textual information is created and associated with each node during the spatial-based index construction. The spatial index part usually deploys using object-partitioning such as R-tree [7], or R*-tree [8], or using space-partitioning trees such as Quad-tree [9] or grid-based [10]. The inverted files that are combined with an R-Tree and Quad-Tree are called IR-Tree [2, 4, 11, 12, 13, 14, 15], and IL-Quad Tree [16] respectively. The inverted files typically contain textual information such as term frequency of the objects underneath that node. The aggregate functions such as maximum are used to determine the maximum score which will be used in the ranking function during the search process. All the IR-Tree and IL-Quad Tree variants try to reduce the disk access by using information in the inverted files to prune out unmatching branches.

To extend the existing LKQ indexes to search efficiently generic location-aware which contains the other attributes, Liu et al. [3] proposed using a synopses tree. A synopses tree works the same way as the inverted file. Therefore, an IR-Tree that is extended to solve GLRQ problem refer as an IR-Tree with a synopses tree (IRS). Liu et al. use the histogram technique and store a synopsis entry for each attribute as a bit array. A bit array is slotted into buckets. Each bucket is used to indicate the satisfiability of the value of all the node underneath that node. The boundary of all the buckets is defined in the global histogram to centralize the value maintaining. When searching GLRQ problem, the specified value range of each attribute is converted into a query predicate. A query predicate is stored like a synopsis entry which will be used to match the synopsis entry of the node until checking at the leaf node, the value in the attribute is compared with the value in the query.

Since the existing IRS technique does not change the way of the tree construction, the IRS will not be optimized for the query that contains a query predicate. The optimization issue which will be discussed in details is caused by when the synopsis entry in high-level nodes almost contains 1's and results in less number of branches is pruned. Furthermore, the problem becomes more complicated when the location-aware objects contain many other attributes.

3. Problem Statement

According to GLRQ problem defined in [3], a generic location-aware rank query contains the constraints of spatial, text, and other attributes. The query returns top-k results according to user-provided ranking function. The constraints can be specified as a maximum distance, a set of keywords, and a selection of value in each attribute.

In GLRQ problem, we want to split an overflowed node into two nodes by grouping its child into two groups such that the average number of I/O cost is minimized. Since the ranking function in GLRQ problem will return the score as zero if the object does not satisfy any query predicates, the satisfiability of the query predicate overwrites the score in the ranking function. On the other hand, if the object satisfies the query predicate, we say that the query relationally overlaps with the object on the given attribute.

Let R be a node in the data structure that will be split into two nodes R_1 and R_2. The probability $P_r(R_i)$ that the node R_i will be accessed by an arbitrary GLRQ query Q with a query predicate P can be determined as follows.

$$P_r(R_i) = \frac{n(Q.d \cap R_i.d)}{n(Q.d)} \cdot \frac{n(Q.P \cap R_i.syn)}{n(Q.P)} \quad (1)$$

where $n(Q.d \cap Ri.d)$ and $n(Q.P \cap Ri.syn)$ are the number of queries that spatially and rationally overlap with a node R_i accordingly, $n(Q.d)$ and $n(Q.P)$ are the number of spatial-based and predicate-based queries respectively.

$$I/O\ cost = c \cdot P_r(R_1) + c \cdot P_r(R_2) \quad (2)$$

$$= \frac{c}{n(Q.d) \cdot n(Q.syn)} (n(Q.d \cap R_1.d) \cdot n(Q.P \cap R_1.syn) + n(Q.d \cap R_1.d) \cdot n(Q.P \cap R_2.syn)) \quad (3)$$

Since *c*, *n(Q.d)*, and *n(Q.P)* are independent of the choice of split policy, we will consider only the other part of the equation. Since we do not have distribution information, we assume that all the queries are distributed evenly in terms of index space and attribute value, and are approximately equal in size and attribute selection.

At this point, the number of queries that spatially overlap with a certain node is proportional to the Minkowski sum [17] of the query, whereas the number of queries that relationally overlap with a certain node is proportional to the product of the synopses between the node and the query.

This analysis shows that I/O cost depends on the node size, the query size, and the product of the synopses between the node and the query. In this work, we will focus on the analysis of synopses since the analysis of R-Tree has been done in [18]. For any hierarchical structure, the synopsis entry in the high-level nodes is the result of superimposition to that in the low-level nodes. Therefore, the number of 1's in the high-level nodes will be the result of the grouping nodes that mathematically close among themselves on the given attribute.

The number contributes to the product of the I/O cost. This observation tells us that heuristic approach that IRS should use, at the high level of the tree, is that the product of synopses between the node and the query nominates the Minkowski sum of the node as it applies to all the objects in the data structure. On the other hand, at the low-level node, the number of the objects underneath that node is smaller. Therefore, the Minkowski sum of the node and the query nominates the product of synopses between the node and the query.

As a conclusion, we suggest that the product of the synopses tree should be used in determining how to split a node when the node is overflowed or which node to expand when a new entry is inserted.

4. Proposed Approach

4.1. Multiple index structure

As we discussed in the previous section, generating a single IRS Tree that will be optimized to specific predicate query would result in worse performance when the query predicate does not present or presents in a different attribute. For example, an IRS Tree is generated based on attribute A_1 and spatial proximity. When the GLRQ query Q is present with a query predicate on attribute A_2, the performance is worse than using a normal IRS Tree that uses spatial proximity as a rule to group the objects.

In this work, we propose a multiple index structure called Multi-IRS which is an index structure that contains multiple IRS that is generated based on the other attributes and spatial proximity.

Let all the objects O in a location-aware dataset contains the spatial location, keywords, and t other attributes. Let an R-Tree that holds all the objects O has f entries in each node. The number of the inner node in an R-Tree *N(R)* can be determined as follows.

$$N(R) = \sum_{i=1}^{h} \left(\frac{N(O)}{f^i}\right) \cdot t \tag{4}$$

where *N(O)* is the number of objects in the dataset, and h is the height of the tree. Since, on disk-based R-Tree, f can be much larger than t, the number of the node in Multi-IRS tree *N(M)* can be approximately calculated as follows.

$$N(M) \approx \frac{N(O)}{f} \cdot t \tag{5}$$

This analysis shows the scalability of our proposed Multi-IRS as its size will only grow linearly. Though the index storage can be multiplied by t, the IRS subtree in our proposed index are independent among themselves. Hence, the index construction process can be simultaneously generated.

4.2. Execution system

After generating multiple IRS on the given datasets, we propose the execution system which will be used to determine which IRS will be used with the given GLRQ query at the runtime.

According to Equation 3, the analysis shows that if the synopses tree of the node in IRS relationally overlaps more with the query, the average I/O cost grows as there are many objects to be processed and pruned afterward. Therefore, if the given query predicate will satisfy the least number of the objects, the

average I/O cost will be decreased. The execution system can determine such values by taking the advantage of the existing data structure called global histogram which is generated during a synopses tree generation.

The execution system processes the query predicate based on the number of given query predicates. When a single query predicate is present, the execution system first determines if the query predicate matches a small proportion of the objects in the dataset. If so, the execution system picks the IRS that was generated based on the attribute of the query predicate. On the other hand, if multiple query predicates are present, our execution system will choose the best query predicate among the predicates first. Finally, the IRS that was generates based on the attribute of that predicate is chosen.

To select the best query predicate, the system uses the statistics in the global histogram and calculate how many objects that the query predicate will need to process. Though this is an approximate number and can be varied from another condition such as spatial proximity and text relevancy, the process in this step is quick and the overhead of the process does not grow when the number of the objects increases. Then the system compares and chooses the predicate that satisfies the least proportion of the objects.

5. Experimental Study

We conduct the experiment between our proposed method and the state-of-the-art method.

5.1. Experimental setting

As the index structure built for GLRQ problem will outperform those naïve approaches such as LKQ index, we compare our proposed index structure with the index proposed for GLRQ problem. To the best of our knowledge, we found that LINQ framework is the only state-of-the-art method that was proposed to solve GLRQ problem. Therefore, we compare our Multi-IRS with LINQ framework. Our experimental uses k-nearest neighbor (k-NN) type of searching.

To compare the query performance, under our prototype, we implement the code in the in-memory data structure. Therefore, we measure the simulated I/O cost such as disk access as the algorithm will be disk-based in a real environment. We conduct all the experiments in Intel Core i3 3.3Ghz Windows 10 with 8GB of main memory.

We use Amazon review dataset on books [19] which contains many numeric attributes and randomly bind with the Tiger/Line dataset [20]. The dataset contains 1M objects with five other attributes which are helpfulness, overall, review time, price, and sale rank order. We do not want the keyword distribution to affect the performance of spatial location and other attributes. Therefore, the keywords are not associated with the objects. The R-Tree construction process is done by OMT packing [21] which allows all the objects to utilize the maximize capacity in both indexes. The page size we use to simulate the data for each node is set to 4096 bytes. After calculating, we set the fanout of the R-Tree can be set up to 100.

To make the queries what user would likely use, we generate the query set by the following steps. First, we randomly pick an object in the dataset and uses its location as a center point. Then we randomly generate the type and a simple selection of each query predicate. Thus, each query predicate will cover from a small to a large set of the objects. Then the maximum distance is randomly generated from the center point.

Table 1 shows the parameters for the evaluation. The default parameter values are represented in bold. We generate 100 queries for each setting combination.

Table 1: Our Experimental Settings

Fanout	10 20 30 40 **50** 60 70 80 90 100
Top-K	**10** 20 30 40 50 60 70 80 90 100
No of Predicates	1 2 **3** 4 5

5.2. Experimental result

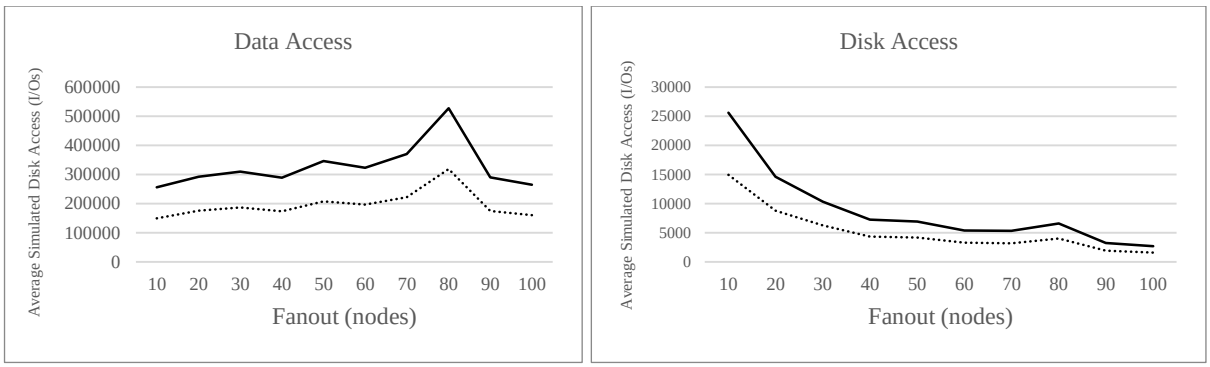
Varying the fanout. We study the query performance of the methods when the number of fanouts in each node increase from 10 to 100. We can see that the disk access of the IRS tree decreases when the fanout increase as shown in Fig. 2b, whereas the number of data access increases in Fig. 2a. This is apparent

because the number of nodes in the tree reduces when the fanout increases. However, when the fanout increases from 50, the performance of both indexes start to drop and continue dropping steadily as the fanout increases to 100. This would be because there will be too many entries to access once the node is processed.

Varying the predicate. We study the query performance when the number of query predicates increases from 1 to 5. It can be seen from Fig. 3 that the performance advantage of Multi-IRS over LINQ framework steadily increases as the number of query predicates increases. The results show that Multi-IRS method uses 2.2 times less disk I/O than IRS method when the number of predicates is 5. The is because Multi-IRS chooses the best query predicate that prunes the most objects among the query predicates. Then the Multi-IRS method chooses an IRS-Tree that is generated based on the attribute of the predicate. In addition, as the number of the query predicates increases the execution system has more choice to select the best query predicate among the given query predicates.

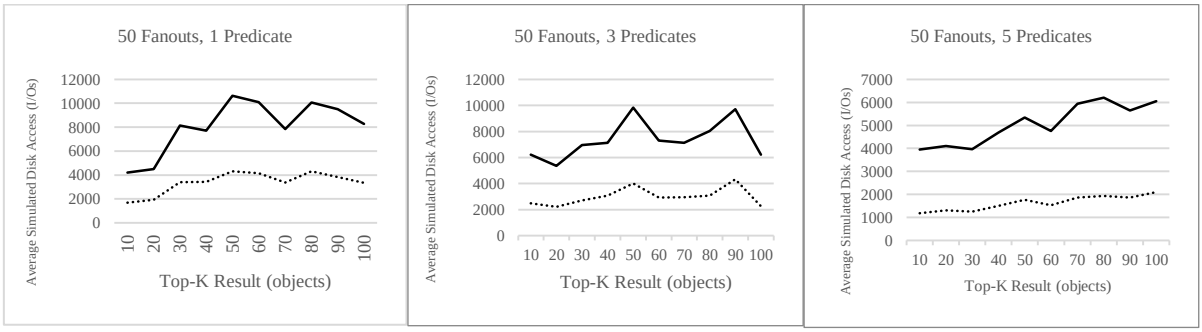
6. Conclusion

In this paper, we address the optimization problem in the generic location-aware rank query (GLRQ) and propose a novel index structure called Multi-IRS. The Multi-IRS generates multiple index structure which will perform best based on the query type. Then the execution system determines the query at the runtime and chooses the index structure that will perform best for the given query. Experimental results show that the proposed index structure outperforms the state-of-the-art methods.



(a) (b)

Fig. 2: Average I/O cost varying by fanout (Multi-IRS is dotted line; IRS is solid line).



(a) (b) (c)

Fig. 3: Average I/O Cost Varying by the number of query predicates (Multi-IRS is dotted line; IRS is solid line).

7. References

[1] Zhou, Y., Xie, X., Wang, C., Gong, Y., & Ma, W. Y. (2005, October). Hybrid index structures for location-based web search. In Proceedings of the 14th ACM international conference on Information and knowledge management (pp. 155-162). ACM.

[2] Cong, G., Jensen, C. S., & Wu, D. (2009). Efficient retrieval of the top-k most relevant spatial web objects. Proceedings of the VLDB Endowment, 2(1), 337-348.

[3] Liu, X., Chen, L., & Wan, C. (2015). LINQ: A framework for location-aware indexing and query processing. Knowledge and Data Engineering, IEEE Transactions on, 27(5), 1288-1300.

[4] Zhang, D., Tan, K. L., & Tung, A. K. (2013, March). Scalable top-k spatial keyword search. In Proceedings of the 16th International Conference on Extending Database Technology (pp. 359-370). ACM.

[5] Basu Roy, S., & Chakrabarti, K. (2011, June). Location-aware type ahead search on spatial databases: semantics and efficiency. In Proceedings of the 2011 ACM SIGMOD International Conference on Management of data (pp. 361-372). ACM.

[6] Kwon, H. Y., & Whang, K. Y. (2015). Scalable and efficient processing of top-k multiple-type integrated queries. World Wide Web, 1-25.

[7] Guttman, A. (1984). R-trees: a dynamic index structure for spatial searching (Vol. 14, No. 2, pp. 47-57). ACM.

[8] Beckmann, N., Kriegel, H. P., Schneider, R., & Seeger, B. (1990). The R*-tree: an efficient and robust access method for points and rectangles (Vol. 19, No. 2, pp. 322-331). ACM.

[9] Finkel, R. A., & Bentley, J. L. (1974). Quad trees a data structure for retrieval on composite keys. Acta informatica, 4(1), 1-9.

[10] Khodaei, A., Shahabi, C., & Li, C. (2010, August). Hybrid indexing and seamless ranking of spatial and textual features of web documents. In Database and Expert Systems Applications (pp. 450-466). Springer Berlin Heidelberg.

[11] De Felipe, I., Hristidis, V., & Rishe, N. (2008, April). Keyword search on spatial databases. In Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on (pp. 656-665). IEEE.

[12] Rocha-Junior, J. B., Gkorgkas, O., Jonassen, S., & Nørvåg, K. (2011). Efficient processing of top-k spatial keyword queries. In Advances in Spatial and Temporal Databases (pp. 205-222). Springer Berlin Heidelberg.

[13] Li, Z., Lee, K. C., Zheng, B., Lee, W. C., Lee, D., & Wang, X. (2011). Ir-tree: An efficient index for geographic document search. Knowledge and Data Engineering, IEEE Transactions on, 23(4), 585-599.

[14] Wu, D., Cong, G., & Jensen, C. S. (2012). A framework for efficient spatial web object retrieval. The VLDB Journal—The International Journal on Very Large Data Bases, 21(6), 797-822.

[15] Wu, D., Yiu, M. L., Cong, G., & Jensen, C. S. (2012). Joint top-k spatial keyword query processing. Knowledge and Data Engineering, IEEE Transactions on, 24(10), 1889-1903.

[16] Zhang, C., Zhang, Y., Zhang, W., & Lin, X. (2013, April). Inverted linear quadtree: Efficient top k spatial keyword search. In Data Engineering (ICDE), 2013 IEEE 29th International Conference on (pp. 901-912). IEEE.

[17] Lee, I. K., Kim, M. S., & Elber, G. (1998). Polynomial/rational approximation of Minkowski sum boundary curves. Graphical Models and Image Processing, 60(2), 136-165.

[18] Porkaew, K. (2000). Database support for similarity retrieval and querying mobile objects.

[19] McAuley, J., Targett, C., Shi, Q., & van den Hengel, A. (2015, August). Image-based recommendations on styles and substitutes. In Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval (pp. 43-52). ACM.

[20] R-Tree Portal. (December 2014). “Tiger/Line Dataset” [Online] Available: http://chorochronos.datastories.org

[21] Lee, T., & Lee, S. (2003, June). OMT: Overlap Minimizing Top-down Bulk Loading Algorithm for R-tree. In CAiSE Short Paper Proceedings (Vol. 74).