Location Aware Indexing

Yagnesh Kamble
Computer Engineering department
Sardar Patel Institute of Technology
Mumbai, India

Shubham Godshalwar
Computer Engineering department
Sardar Patel Institute of Technology
Mumbai, India

Ashna Bajaj
Computer Engineering department
Sardar Patel Institute of Technology
Mumbai, India

Reeta Koshy
Professor, Computer Engineering department
Sardar Patel Institute of Technology
Mumbai, India

ABSTRACT
This project closely models a framework to process Generic Location-Aware Rank Queries. A restaurant-finder application has been created to demonstrate how a Generic Location-Aware Ranked Query (GLRQ) can be processed by deploying three data structures in sync with each other – the synopses tree, the R-tree and inverted files. The synopses tree, created using histograms, handles the numeric attributes. The R-tree filters results based on their location, while the inverted files filter according to specified keywords (eg: lunch, breakfast, italian, karaoke), if any. Existing methods of processing such queries perform the pruning of the search space in two stages – first according to location and keyword, and then according to specified predicates (or vice versa), which is usually not efficient. The method used here trumps the aforementioned because the pruning is carried out simultaneously. This is reasonably faster, especially when working with large datasets, which has been experimentally demonstrated.

General Terms
Query processing, spatial indexing

Keywords
Location-aware, synopses tree, IR-tree

1. INTRODUCTION
There has been a massive increase in the usage of locations all over the Internet, especially on social networking platforms like Facebook, Twitter, Instagram, etc, which allow users to geo-tag their posts, as has been discussed in [1]. As a result, it is essential to have an efficient searching method that enables the user to search across a variety of filters such as location ('Search places near me'), numerical attributes ('Rating > 4', 'No. of comments > 100'), and keywords ('Search posts tagged with Street Food').

This framework was created because existing methods do not optimally process GLRQs, which are queries that contain numerical predicates, keywords, as well as location specifications. Since we have developed a restaurant-finder application, related examples have been used to further emphasize the inadequacy of the existing methods.

In the naive LKQ (Location-Keyword Query) approach[2], the query is assumed to be location-aware and contain only keywords. Thus, constraints on numerical attributes are also converted to keywords, as is illustrated in the following example.

Consider a query that searches for all nearby restaurants having a rating greater than 3.5 (out of 5). Over here, the predicate 'rating>3.5' is converted to rating=3.5, 3.6, 3.7, 3.8, 3.9 and so on, where each value of rating is treated as a keyword. This greatly complicates the query.

The queries can also be processed using the LKQ-first or Predicate-first method. In this method, the data is first filtered first according to numerical attributes, then according to the location and keyword, or vice versa as the case may be. This is especially inefficient if the latter filtering process yields results which are a very small subset of the former. In Predicate-first, the numerical attributes are first looked at. Suppose they return 250 results. Now, after location-keyword filtering on those 250, only 3 of them satisfy the constraints. Thus, the remaining 247 results were unnecessarily fetched.

Our framework, however, performs simultaneous pruning of predicates as well as locations and keywords. This is done by using three primary data structures: the synopses tree, the R-tree and inverted files in conjunction with each other.

2. LITERATURE SURVEY
2.1 LINQ - A Framework For Location-Aware Indexing And Query Processing
The crux of the system design is derived from this paper[1]. It gives a method of evaluating generic location-aware rank queries (GLRQs). If these queries are evaluated in the usual method of location-keyword queries (LKQs), they prove to be very inefficient. The method proposed in this paper makes use of a data structure called the synopses tree, along with the R-tree and inverted files data structures. The synopses tree greatly reduces the cost of pruning, while ensuring that accuracy is preserved. These data structures are used to perform score-based pruning and predicate-based pruning simultaneously in a manner which is much faster than performing them in separate stages. This method has been tested on a number of real and synthetic data sets to prove the efficiency over the normal method adopted by an RDBMS.

2.2 Efficient Retrieval Of The Top-K Most Relevant Spatial Web Objects
This paper[2] introduces us to the concept of IR-trees, and how they can be used to query objects that have two parameters – a location, and a set of keywords associated with the object. This framework integrates the inverted file for text retrieval and the R-tree for spatial proximity querying to obtain an inverted file R-tree.

Each node of the IR-tree records a summary of the location information and textual content of all the objects in the sub-tree rooted at that node. The query-processing algorithms that uses the location index information to estimate the spatial distance of a query to the objects in the node's sub-tree, and it uses the text index to estimate the text relevance for the objects.

The paper presents an algorithm for building the IR-tree, given the location (in the form of a Minimum Bounding Rectangle) for each object, along with the associated textual document. It also gives algorithms for how the query is to be processed using the IR-tree created.

2.3 Synopses for Massive Data: Samples, Histograms, Wavelets and Sketches

Since the synopses tree is an important data structure used in the system design, this paper[3] has been useful in introducing the concept of synopses. It explains how synopses can be built using any of the four available methods – samples, histograms, wavelets and sketches. However, our concentration is focused on the histograms part.

Multi-dimensional histograms are uses to summarise datasets. Since it is very expensive to construct and maintain multi-dimensional histograms, the nD data distribution can be factored into multiple 2D distributions. The effectiveness of this method has been verified. By this way, we can provide accurate approximations of the joint data distributions with low cost.

2.4 Spatial Keyword Query Processing: An Experimental Evaluation

This paper[4] provides methods to combine spatial and keyword-based pruning on queries containing such constraints. It illustrates the use of inverted files to handle text, and provides two methods to handle locations – R-trees and grids. It introduces the three main types of queries containing spatio-textual attributes as are required by the user. This also forms a basis of our undertaken project, and merely has to be integrated with the synopses tree to handle numeric attributes.

3. SYSTEM DESIGN

3.1 Architecture

The backbone of the framework consists of three data structures, which work in synchronization with each other to provide an efficient searching mechanism. These data structures are:

The R-tree – filters results based on spatial proximity

Inverted files – filters results based on keyword matches

The synopses tree – filters results based on numerical constraints

Their usage has been further illustrated below.

3.1.1 Synopses tree

This is a newly created structure that contains synopses of the numerical attributes in the data set. A tree-like data structure is used, in which the root node stores the synopses of the entire dataset. Its children store synopses of subsequently smaller datasets, such that the union of the datasets represented by the children forms the dataset of the parent. In other words, every child node should be a subset of the parent node, and the parent should be completely represented by its children.

The synopsis of a dataset is constructed by using 2-dimensional histograms [3]. Over here, each attribute is a single dimension. The attributes are divided into pairs of 2, using which 2D histograms are constructed.

Each histogram is divided into four regions, or buckets. These buckets represent ranges of values. For example:

Consider a histogram of rating (range 0-5) vs. health (range 0-100). The buckets can be created as follows:

B1 contains all elements where rating<=3.5 and health<=30

B2 contains all elements where rating<=2.5 and health>=30

B3 contains all elements where rating>2.5 and health<=60

B4 contains all elements where rating>2.5 and health>60

Each bucket contains a list of items satisfying the constraints. The first element of the bucket is always 1 or 0; 0 indicating that the bucket is empty, while 1 indicating that it isn't.

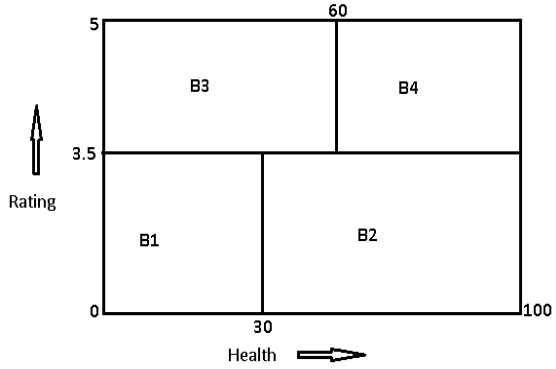

Figure 1: An example of a 2D histogram divided into 4 buckets

3.1.2 Inverted files

These data structures are used to swiftly return values that match the keywords specified. This is done by creating a dictionary of the keywords, in the form of key-value pairs. The key here is the keyword, and the values are all those items in the dataset that match the keyword. This method is used for efficient retrieval especially when the dataset is large.

3.1.3 R-Trees

R-trees, closely modeled on B-trees, provide a way to store the spatial attributes of objects[2]. This is done by using Minimum Bounding Rectangles (MBRs). In an R-tree, each node is an MBR. An MBR is a region drawn on the map that contains the exact locations of the objects in it. However, MBRs can be nested, or can overlap.

Each node MBR bounds its children. A node can have many objects in it. The leaves of the R-tree point to the actual objects.

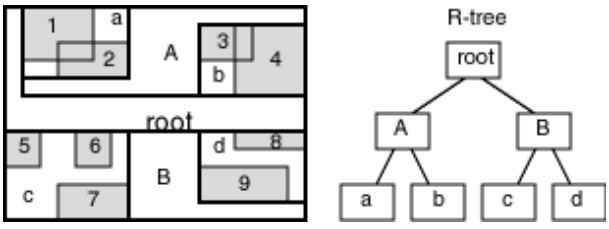

Figure 2: Structure of the R-tree

As shown below, these three data structures are used in sync to retrieve the results of a query fired by the user. Experimental results prove that the time taken to search using this method is much faster than the traditional way of searching the database record by record.

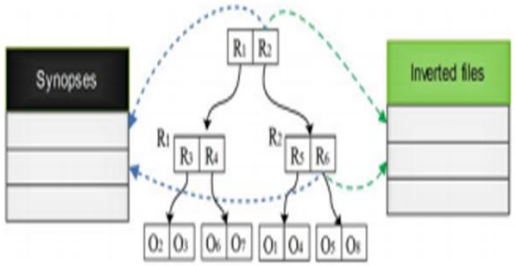

Figure 3: The three data structures linked with each other [1]

4. THE SEARCHING ALGORITHM

1. The synopses tree and inverted files are created from the database contents and stored
2. The query is fired by the user from the UI
3. The keywords are extracted, and the inverted files are checked to return the matching results, say R1
4. Simultaneously, the numerical attributes are passed to the synopses tree, and the results satisfying the constraints are returned, say R2
5. An intersection of R1 and R2 produces the final list of results, say R3
6. They are filtered according to the location, and displayed to the user on the screen
7. However, if it happens that the number of items in R3 does not reach the minimum number of results that should ideally be returned to the user, then more items are added to R3 by increasing the radius of the location, or including values from R1 and R2 in decreasing order of relevance

5. IMPLEMENTATION DETAILS

All experiments have been carried out on a machine having an Intel core i5 processor having 4 GB RAM and 1.7 GHz quad core processor. The database used is a MySQL synthetic dataset having around 1000 records, and the data structures have been created using Python scripts. The user interface has been developed using HTML, CSS and PHP.

6. RESULTS

The following results give the execution time by using both methods - the data structures and naive LKQ methods - to retrieve results from a MySQL database of 1000 records having attributes of rating (0-5), health (0-100), and a list of keywords.

A minimum value of rating(r) and health (h) and two keywords were specified. The objective was to return tuples satisfying the given predicates (min. rating and health) and containing the mentioned keywords.

The time taken (in seconds) by both the methods are as follows:

Table 1: Comparative study of time taken by both methods to process the same query

Method using data structures	Naïve LKQ method
0.0813	0.1436
0.0634	0.0922
0.0689	0.0896
0.0744	0.0905

7. CONCLUSION

This paper thus demonstrates that the use of data structures in searching a database when provided with multiple search parameters (keywords, numerical attributes, and location) is significantly faster than the conventional method of retrieval from a database.

However, the given scope is but a small part of the actual potential of this method. The usage of 2D histograms can be extended to multi-dimensional histograms to handle more numerical attributes. Experiments can also be conducted on multiple real datasets to further emphasize the efficiency of our method.

8. REFERENCES

[1] Xiping Liu, Lei Chen, Changxuan Wan, LINQ: A Framework for Location-aware Indexing and Query Processing in IEEE Transactions on Knowledge and Data Engineering, Vol. 27, No. 5, pp. 1288-1300 .

[2] G.Cong, C.S. Jensen, and D.Wu. Efficient retrieval of the top k-most relevant spatial web objects, Proc. VLDB Endowment, Vol. 2, pp. 337-348, 2009.

[3] G. Cormode, M. Garofalakis, P.J. Haas, and C. Jermaine. Synopses for massive datasets: Samples, Histograms, Wavelets, Sketches. Found. Trends Databases, vol. 4 nos. 1-3, pp. 1-294, 2012.

[4] Lisi Chen, Gao Kong, Christian S. Jensen, Dingming Wu. Spatial Keyword Query Processing: An Experimental Evaluation, International Conference on Very Large Data Bases, August 2013.