Keyword Search in Spatial Databases: Towards Searching by Document

Dongxiang Zhang, Yeow Meng Chee, Anirban Mondal, Anthony K. H. Tung, Masaru Kitsuregawa

Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on. IEEE, 2009.

Group 9

Atish Majumdar	1410110081
Prasanna Natarajan	1410110298
Vedant Chakravarthy	1410110489

I. Introduction

In a database of spatial objects, every tuple is associated with certain information represented by keywords. This paper introduces a new type of spatial keyword query, known as the m-closest keywords (mCK) query. The aim of the mCK query is to find tuples which are located in the closest spatial proximity that align with the user-entered keywords. In order to efficiently process mCK queries, a new index is introduced, known as the bR*-tree, which is an extension of the R * tree. Following this, a priori based search approaches are used to decrease the total search space.

There is extensive research being conducted on various forms of spatial queries, above and beyond the general one- nearest neighbour queries, range queries, and spatial joins. Keyword based queries have received significant attention in the research community. This paper presents a novel query, the m-closest keywords (mCK) query. An mCK query returns the closest tuples in any space which match certain keywords, given a set of m keywords as input. The main focus of the paper is to apply this mCK query in searching by document.

A given tuple can be associated with as many as m keywords, hence the total number of responses that can be possibly returned from a mCK query is at most m. A problem which arises from this is that the value of m might become very large while searching by document, in which case the naive mCK query processing would become very computationally as well as temporally intensive, as the number of possibilities to examine all possible set of m tuples of objects matching the keywords would be needed. This process is extremely expensive and hence is not a good approach. This paper proposes to improve upon this by using a new index, known as the bR*- tree, for processing queries. The bR*- tree extends the R*-tree to efficiently summarize keywords and their spatial information. Following this, a-priori based search strategies are used to significantly reduce the search space. For application of said search strategies, some constraints are required to prune the dataset. The paper defines two such monotone constraints, the distance mutex and the keyword mutex. Finally, using all the above, the algorithm is tested and is found to be effective in reducing mCK query response time, but also provides scalability with regards to the number of keywords in the query.

II. Motivation

Bob is in search of a new locality to shift to in his city. He wants a certain set of facilities to be available to him in close proximity, namely, a pet play area, a hardware store, a shooting range and a grocery store. He would like to move to a neighbourhood that has these in the closest proximity since he would like to visit them frequently and easily.

Currently, Bob would require to find each hardware store in the locality and map out distances to the closest pet area, hardware store and grocery store. He can then find the best locality for him to live in based on these criteria.

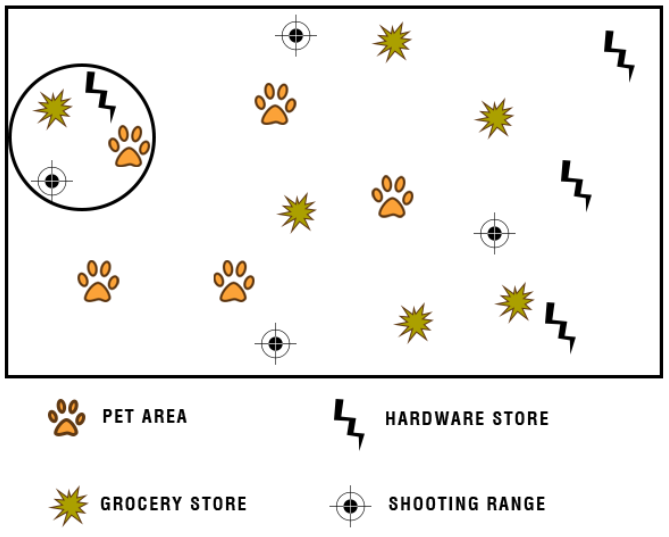

Fig 1. Bob's Housing Problem

This system, however, can help problems like these that require spatial data in a manner that relates multiple queries to a required region in an efficient way. Here, Bob simply uses Pet Area, Hardware Store, Grocery Store and Shooting Range as keywords to the mCK query, which will return to him encircled spatial area, which encompasses all the given requirements in the closest proximity.

III. Problem Statement

mCK Query Problem: Given a spatial database with d-dimensional tuples represented in the form $(l_1, l_2, ..., l_{d-1}, w)$ and a set of m query keywords $Q = \{w_{q1}, w_{q2}, ..., w_{qm}\}$, the mCK Query Problem is to find m tuples $T = \{T_1, T_2, ..., T_m\}$, $T_i.w \in Q$ and $T_i.w \neq T_j.w$ if $i \neq j$, and diam(T) is minimum.

Diameter: LetS be a tuple set endowed with a distance metric $dist(\cdot,\cdot)$. The diameter of a subset $T \subseteq S$ is defined by

$$diam(\mathcal{T}) = \max_{T,T' \in \mathcal{T}} dist(T, T')$$

In other words, the problem that this paper is trying to solve is that of querying spatial data with keyword associations, returning tuples which are the in the closest proximity to each other.

IV. BR* TREE: R* TREE WITH BITMAPS AND KEYWORD MBRS

BR*-tree is an extension of the R*-tree.

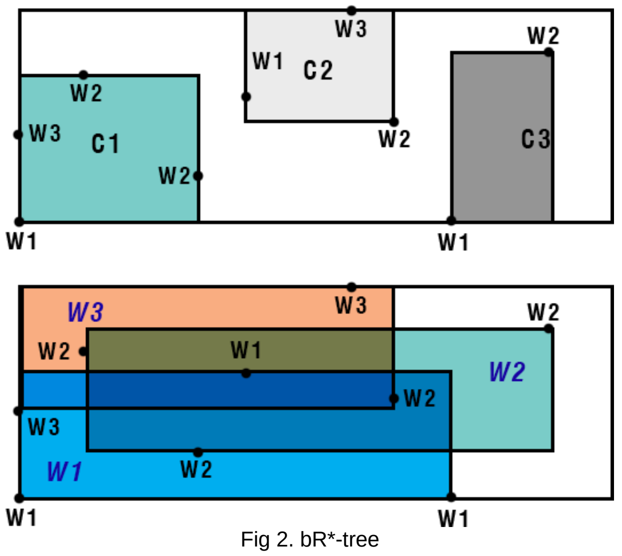

Fig 2. bR*-tree

	Bitmap
Parent Node	111
C_1	111
C_2	111
C_3	110

Table 1: Bitmap for bR*-tree

Apart from containing a node's MBR (as is the case in the R-tree model), summary of the keywords is also contained in the node in the form of a bitmap in the node. Eg: B = 01001 reveals that the database in question has 5 keywords, current node can only be associate with 2 things, given query Q = 00110, if B&Q=0, given node can be removed from search space.

The keyword MBR is also stored for better pruning. The keyword MBR of the w_i is the MBR of all the objects in the nodes that are associated with w_i. Helps find the approximate area in a node that each keyword is distributed.

After being augmented with the bitmap and keyword MBR, non-leaf nodes of the bR*-tree contain entries of the form

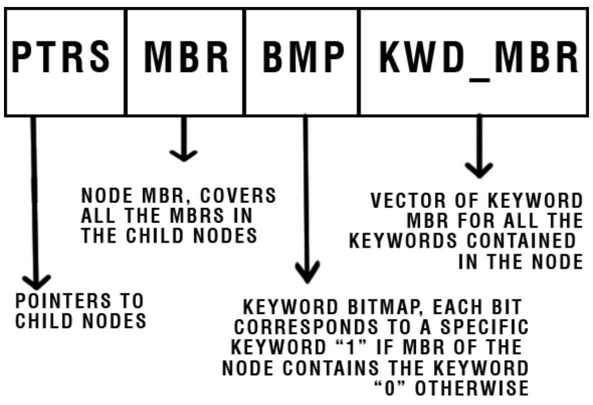
PTRS
MBR
BMP
KWD_MBR
NODE MBR, COVERS ALL THE MBRS IN THE CHILD NODES
VECTOR OF KEYWORD MBR FOR ALL THE KEYWORDS CONTAINED IN THE NODE
POINTERS TO CHILD NODES
KEYWORD BITMAP, EACH BIT CORRESPONDS TO A SPECIFIC KEYWORD "1" IF MBR OF THE NODE CONTAINS THE KEYWORD "0" OTHERWISE

Fig 3. Node Structure in bR* tree

V. SEARCH ALGORITHMS

Assuming that all the data objects are represented as a hierarchical bR*-tree, the mCK query finds m keywords in the leaf entries matching the query key.

The search algorithm begins from root node. The candidate search space is any space where the target keywords may be located. Therefore, it is divided into two parts: the space within one child node, that is, where all the m query keywords are contained in this node and the space across multiple child nodes, that is, where all m query keywords are contained in multiple child nodes together and they are close to each other.

Put one example: our own (similar to Fig.3)

In order to find the m closest keywords located at the leaf entries the algorithm traverses the bR*-tree. This traversal is a recursive process, therefore, the same process that happened in the root node will be followed in all the selected children nodes (the search space). That is any node set from the 1st iteration is replaced with its subsets after pruning, thus spawning a larger number of new node sets. The number of nodes in the new node set is non-decreasing and their nodes are one level lower in the tree. When the algorithm encounters leaf node, it retrieves all the combinations of m tuples from these entries and calculates the closest m keywords that match the user query keywords to see if a closer result can be found. One additional constraint to reduce the search space is that the number of child nodes will never be greater than m, as the target m tuples can only reside in at most m child nodes.

The two algorithms follow the following steps :

1. find a relatively small diameter for branch-and-bound pruning before the actual search starts.
2. the search starts from root node and choose a child node with the smallest MBR that contains all the query keywords and traverse down that node.
3. Step 2 is repeated until leaf nodes are reached or until the algorithm is unable to find any child node with all the query keywords.
4. Then, an exhaustive search within the current node is done.
5. The diameter of the result is used as the initial diameter for searching.

A. Searching in One Node

The algorithm proposed uses the a priori algorithm, which is used for reducing search space for any combinatorial problem, to reduce the search space. The a priori algorithm has certain advantages and make use of these algorithms the paper defines two monotonic constraints namely, distance mutex and keyword mutex. The distance mutex and keyword mutex are constraints used to prune the search space.

B. Searching in Multiple Nodes

The a priori algorithm also works quite well for this case too. The two constraints can be used to prune within multiple nodes:

1) Pruning via Distance Matrix

Distance mutex is a monotonic contraint. Thus, a node set is defined as a distance mutex if the distance between two nodes A and B in the node set N is greater than $\delta*$. Thus, when a node set is a distance mutex, it can be pruned to acheive a search space smaller than the default search space of the total dataset.

2) Pruning via Keyword Mutex

Keyword mutex is a monotonic constraint. A node set is defined as a keyword mutex when for some n different keywords such that only each node uniquely contributes a keyword, then there will always exist two different keywords whose distance is greater than $\delta*$. Thus, when a node set is a keyword mutex, it can be pruned to acheive a search space smaller than the default search space of the total dataset.

VI. Possible Improvements

As the paper had assumed only one keyword to be associated with a particular tuple, a possible improvement in that regard would be to allow multiple keyword associations with a single tuple. This would lead to the requirement of processing a large number of keywords for queries.

A possible improvement in this regard would be to index frequently paired keywords, which are usually searched for in conjunction. This could possibly be more efficient, as more and more association rules would be formed post a certain threshold, and a complete lookup would not be required in such cases.

As a bitmap is maintained for each keyword, in case of a large number of keywords, it would require a very high number of comparisons. A possible alternative to this is to cluster similar or frequently paired keywords, and return results based on if the cluster a rating beyond a certain threshold, say 0.6-0.99 on a scale of 0 to 1. This would require additional computation upon the keywords, but decrease the required number of comparisons made when a query has to be processed.

Currently, a result is only returned if it matches all the keywords entered by the user, which might be unlikely when there are a high number of keywords. The algorithm should account for maximum utility, where it checks for maximum number of keywords matched instead of checking if all the keywords are matched.

We could use compression algorithms to reduce space required to store all spatial data, as well as keywords on the disk.

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules.In Proc. 20th Int. Conf. Very Large Data Bases, VLDB, pages 487–499,1994.

[2] S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: A system for keyword-based search over relational databases. In Proceedings of ICDE, 2002.

[3] W. Aref, D. Barbara, S. Johnson, and S. Mehrotra. Efficient processing of proximity queries for large databases. Proc. ICDE, pages 147–154, 1995.

[4] Zhang, Dongxiang, Mondal, et al. "Keyword search in spatial databases: Towards searching by document." Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on. IEEE, 2009.