ISSN No. 0976-5697

Volume 3, No. 3, May-June 2012

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

Ranking Spatial Data using Quality Preferences

Murali Krishna M*
Department of CSE
Kaushik College of Engineering
Visakhapatnam, India
maadugula@gmail.com

Ramesh Naidu G
Department of CSE
Kaushik College of Engineering
Visakhapatnam, India
rameshgonnet@gmail.com

Abstract: Day to day growth of objectives required selection on basis of features. A spatial preference query ranks objects based on the qualities of features in their spatial neighbourhood. For example, consider a real estate agency office that holds a database with available flats for lease. A customer may want to rank the flats with respect to the appropriateness of their location, defined after aggregating the qualities of other features (e.g., restaurants, cafes, hospital, market, etc.) within a distance range from them. In this paper, we formally define spatial preference queries and propose appropriate indexing techniques and search algorithms for them. Our methods are experimentally valuated for a wide range of problem settings.

Keywords: spatial preference, branch and bound algorithm, and query processing.

I. INTRODUCTION

A query to a web search engine usually consists of a list of keywords, to which the search engine responds with the best or "top" k pages for the query. This top-k query model is prevalent over multimedia collections in general, but also over "structured" data for applications where users do not expect exact answers to their queries, but instead a rank of the objects that best match the queries. A top-k query in this context is then simply an assignment of target values to the attributes of a relation. To answer a top-k query, a database system identifies the objects that best match the user specification, using a given scoring function .Example 1. Consider a relation with information about restaurants in the New York City area. Each tuple (or object) in this relation has a number of attributes, including Address, Rating, and Price, which indicate, respectively, the restaurant's location, the overall food rating for the restaurant represented by a grade between 1 and 30, and the average price for a diner. A user who lives at 2590 Broadway and is interested in spending around $25 for a top quality restaurant might then ask a top-10 query fAddress="2590 Broadway", Price=$25, Rating=30g. The result to this query is a list of the 10 restaurants that match the user's specification the closest, for some definition of proximity.

Processing top-k queries efficiently is challenging for a number of reasons. One critical such reason is that, in many web applications, the relation attributes might not be available other than through external web-accessible form interfaces. For instance, in our example above, the Rating attribute might be available through the Zagat-Review website,1 which, given an individual restaurant name, returns its food rating as a number between 1 and 30 (random access). This site might also return a list of all restaurants ordered by their food rating (sorted access). Similarly, the Price attribute might be available through the New York Time's NYT-Review website. 2 Finally, the scoring associated returns the distance (in miles) between the restaurant and the user addresses.

To process a top-k query over web-accessible databases, we then have to interact with sources that export different interfaces and access capabilities. In our restaurant example, a possible query processing strategy is to start with the Zagat-Review source, which supports sorted access, to identify a set of candidate restaurants to explore further. This source returns a rank of restaurants in decreasing order of food rating. To compute the final score for each restaurant and identify the top-10 matches for our query, we then obtain the proximity between each restaurant and the user-specified address by querying MapQuest, and check the average dinner price for each restaurant individually at the NYT-Review source.

Hence, we interact with three autonomous sources and repeatedly query them for a potentially large set of candidate restaurants. Our query scenario is related to a (centralized) multimedia query scenario where attributes are reached through several independent multimedia "subsystems," each producing scores that are combined to compute a top-k query answer. While multimedia systems might support sorted and random attribute access, there are important differences between processing top-k queries over multimedia systems and over web sources. First, web sources might only support random access (e.g., MapQuest returns the distance between two given addresses). Second, attribute access for centralized multimedia systems might be faster than for web sources, because accessing web sources requires going over the Internet. Finally, and importantly, unlike in multimedia systems where attribute access requires "local" processing, applications accessing web sources can take full advantage of the intrinsic parallel nature of the web and issue probes to several web sources simultaneously, possibly issuing several concurrent probes to each individual source as well.

In this article, we present algorithms to efficiently process top-k queries over web sources that support just random access and, optionally, sorted access as well. We first introduce an efficient sequential top-k query processing

algorithm that interleaves sorted and random accesses during query processing and schedules random accesses at a fine-granularity per-object level. Then, we use our sequential technique as the basis to define a parallel query processing algorithm that exploits the inherently parallel nature of web sources to minimize query response time. As we will see, making the algorithms parallel results in drastic reductions in query processing time.

II. BACKGROUND AND RELATED WORK

Object ranking is a popular retrieval task in various applications. In relational databases, we often want to rank tuples using an aggregate score function on their attribute values [1]. For example, consider a database of a real estate agency, containing information about flats available for rent. potential customer may want to view the top-10 flats with the largest sizes and lowest prices. The score of each flat in this case is expressed by the sum of two individual scores: size and price, after they have been scaled to the same range (e.g., between 0 and 1, where 1 indicates the highest preference; highest possible size and lowest possible price). Another popular object ranking application is document ranking based on the relevance of the keywords (terms) they contain to a user query (also expressed by a set of terms).This problem has been the primary research in information retrieval (IR) for over two decades [2]. The ranking function in this problem is again an aggregation of the relevance of the query terms with the document, often enriched with some global ranking scores of the documents according to their popularity [3].

In spatial databases, ranking is often associated to nearest neighbor (NN) retrieval. Given a query location, we are often interested in retrieving the set of nearest objects to it that qualify a condition (e.g., restaurants). Assuming that the set of interesting objects is indexed by a hierarchical spatial access method (e.g., the R-tree [4]), we can use distance bounds while traversing the index to derive the answer in a branch-and-bound fashion [5]. Tao et al. [6] noted that top-k queries can be modeled as (weighted) nearest neighbor queries, in the multi-dimensional space defined by the involved attribute domains, where the query point is formed by taking the maximum value of each dimension.

Motivated by this observation, they adapted the algorithm of [7] for this problem. Nevertheless, it is not always possible to use multidimensional indexes for top-k retrieval. First, such indexes usually break-down in high dimensional spaces [8, 3]. Second, top-k queries may involve an arbitrary set of attributes (e.g., size and price) from a set of possible ones (e.g., size, price, distance to the beach, number of bedrooms, floor, etc.) and indexes may not be available for all possible attribute combinations (i.e., they are too expensive to create and maintain). Third, information for different rankings to be combined (i.e., for different attributes) could appear in different databases (in a distributed database scenario) and unified indexes may not exist for them. A stream of research [8, 1, 5, 7] for top-k queries has focused on the efficient merging of object rankings that may arrive from different (distributed) sources. The motivation of these methods is to minimize the number of accesses to the input rankings until the objects with the top-k aggregate scores have been identified. To achieve this, upper and lower bounds for the objects seen so far are maintained while traversing the sorted lists.

In the next paragraphs, we first review the R-tree, which is the most popular spatial access method and the NN search algorithm of [9] and survey recent research of feature based spatial queries.

A. Spatial Query Evaluation on R-trees:

The most popular spatial access method is the R-tree [9], which indexes minimum bounding rectangles (MBRs) of objects. Figure 2 shows a collection R = {p1, . . . , p8} of spatial objects (e.g., points) and an R-tree structure that indexes them. R-trees can efficiently process main spatial query types, including spatial range queries, nearest neighbor queries, and spatial joins. Given a spatial region W, a spatial range query retrieves from R the objects that intersect W. For instance, consider a range query that asks for all objects within distance 3 from q, corresponding to the shaded area in Figure 2. Starting from the root of the tree, the query is processed by recursively following entries, having MBRs that intersect the query region. For instance, e1 does not intersect the query region, thus the sub-tree pointed by e1 cannot contain any query result. In contrast, e2 is followed by the search algorithm and the points in the corresponding node are examined recursively to find the query result p7.

A nearest neighbor (NN) query takes as input a query object q and returns the closest object in D to q. For instance, the nearest neighbor of q in Figure 2 is p7. Its generalization is the k-NN query, which returns the k closest objects to q, given a positive integer k. NN (and k-NN) queries can be efficiently processed using the bestfirst (BF) algorithm of [4], provided that D is indexed by an R-tree. A min-heap H which organizes R-tree entries based on the (minimum) distance of their MBRs to q is initialized with the root entries. In order to find the NN of q in Figure 2, BF first inserts to H entries e1, e2, e3, and their distances to q. Then the nearest entry e2 is retrieved from H and objects p1; p7; p8 are inserted to H. The next nearest entry in H is p7, which is the nearest neighbor of q. In terms of I/O, the BF algorithm is shown to be no worse than any NN algorithm on the same R-tree [4]. The aggregate R-tree (a R-tree) [10] is a variant of the R-tree, where each non-leaf entry augments an aggregate measure for some attribute value (measure) of all points in its subtree. As an example, the tree shown in Figure 2 can be upgraded to a MAX a R-tree over the point set, if entries e1; e2; e3 contain the maximum measure values of sets fp2; p3g; fp1; p8; p7g; fp4; p5; p6g, respectively. Assume that the measure values of p4; p5; p6 are 0:2; 0:1; 0:4, respectively. In this case, the aggregate measure augmented in e3 would be maxf0:2; 0:1; 0:4g = 0:4. In this paper, we employ MAX a R-trees for indexing the feature datasets (e.g., restaurants), in order to accelerate the processing of top-k spatial preference queries.

Given a feature dataset F and a multi-dimensional region R, the range top-k query selects the tuples (from F) within the region R and returns only those with the k highest qualities. Hong et al. [11] indexed the dataset by a MAX aR-tree and developed an efficient tree traversal algorithm to answer the

query. Instead of finding the best k qualities from F in a specified region, our (rangescore) query considers multiple spatial regions based on the points from the object dataset D, and attempts to find out the best k regions (based on scores derived from multiple feature datasets F_c).

III. PROBLEM FORMULATION

Given an *object* dataset *O* and a set of c *feature* datasets $\{F_i \mid i \in [1, c]\}$, the top-k spatial preference query returns the k data objects $\{p_1,p_2,p_3 \ldots p_k\}$ from *O* with the highest score. The score of a data object p € *O* is defined by the scores of feature objects t € F_i in its spatial neighbourhood. Each feature object t is associated with a *non-spatial score* w(t) that indicates the goodness (quality) of t and its domain of values is the range [0, 1].

The range (rng) score of p , given a radius r :

$$\tau_i^{rng}(p) = \max\{w(t) \mid t \in Fi: d(p,t) \leq r\}$$

The nearest neighbour (nn) score of p:

$$\tau_i^{nn}(p) = \max\{w(t) \mid t \in Fi, \forall v \in Fi: d(p,t) \leq d(p.v)\}$$

IV. MAPPING TO DISTANCE SCORE SPACE

Top-k spatial preference queries return a ranked set of spatial data objects. The main difference to traditional top-k queries is that the score of each data object *p* € *O* is obtained by the feature objects in its spatial neighbourhood. Thus, determining the partial score of a data object p based on the feature set F_i requires that the *pairs* of objects (p, t) with *t* € F_i need to be examined. Consequently, the *search space* that needs to be explored to determine the partial score is the Cartesian product between O and Fi. As the total number of pairs with respect to all feature datasets is significantly larger than the cardinality of dataset O processing top-k spatial preference queries is particularly challenging.

In this section, we formally define the search space of the top-*k* spatial preference queries by defining a mapping of the data objects *O* and any feature dataset F_i to a *distance-score space*. Then, we prove that only a subset of the pairs (*p, t*), where *p* € *O* and *t* € F_i, are sufficient to answer all top-*k* spatial preference queries. This drastically reduces the search space for any given query, thereby saving computational costs significantly. In addition, we prove that this subset of pairs is the minimal subset of pairs necessary.

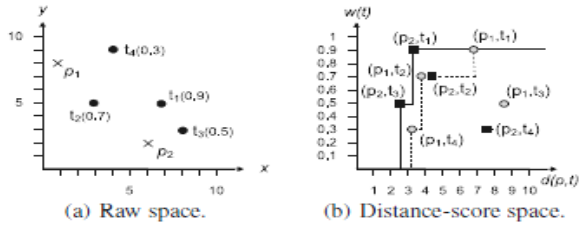


Figure.1. Mapping to the distance score space.

In a pre processing step, the subset of pairs is computed and stored using a multi-dimensional index. As a result, we avoid computing pairs of the Cartesian product on-the-fly during query processing, leading to an efficient algorithm for processing top-*k* spatial preference queries. ing

V. QUERY PROCESSING

In this section, we present the *Skyline Feature Algorithm (SFA)* for processing top-k spatial preference queries. First, we present an algorithm that exploits the distance-score space and returns the data objects in descending order of their partial scores. Then, we present the algorithmic details of SFA, which produces the result of the top-k spatial preference query by coordinating access to the partial scores of data objects. For ease of presentation, in the following, we refer to a pair (*p, t*), where p € O and t € F_i, as a *data point* indexed by R_i^o.Access to Partial Scores. During query processing, the data points in R_i^o are retrieved sorted in descending order of their partial scores. Furthermore, only node entries of the R-tree R_i^o that satisfy the spatial constraint are processed. First, we present in details our algorithm for retrieving data points sorted based on the range score (Flowchart). Then, we describe the necessary modifications for supporting the influence and nearest neighbour scores.

NextObject takes as input the radius r that defines the range constraint and a heap H that contains node entries and data points in descending order of partial scoreτ_i^{θ} ( ). Initially, the heap H contains the root of R_i^o. Each time, the entry *e* at the top of the heap H, i.e., with maximum partial score, is retrieved (lines 3, 10). As long as e is not a data point (line 4), NextObject inserts in the heap H (line 7) the children entries of e whose distance is smaller or equal to the radius r (line 6).

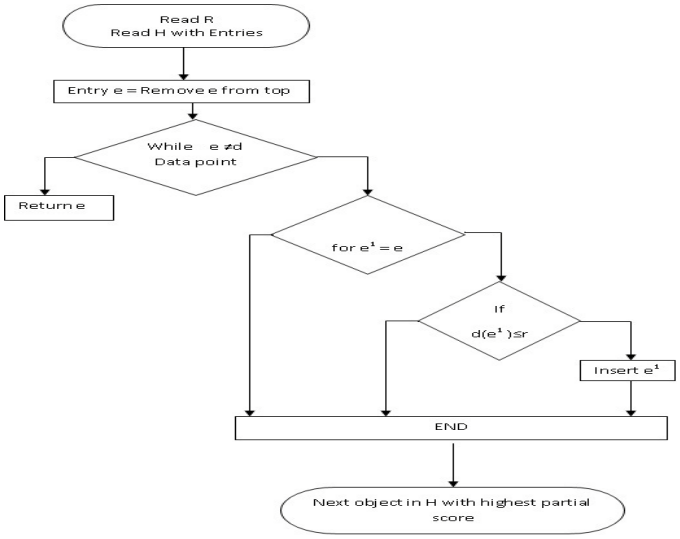


Figure.2. Flowchart for query processing

When the next entry is a data point, it is returned as the data point with the highest partial score τ_i^{θ}( ) in S_i^o.

VI. EXPERIMENTAL EVALUATION

This section, we compare the efficiency of the proposed algorithms using real and synthetic datasets. Each dataset is indexed by a R-tree with 4K bytes page size. We used an LRU memory buffer whose default size is set to 0.5% of the sum of tree sizes (for the object and feature trees used). Our algorithms were implemented in C++ and experiments were run on a Pentium IV 2.3GHz PC with 512 MB of RAM. In all

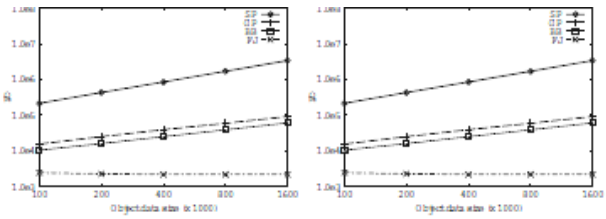
experiments, we measure only the I/O cost (i.e., number of age faults) of the algorithms as their

A. Experimental Setting:

We used both real and synthetic data for the experiments. For each synthetic dataset, the coordinates of points are random values uniformly and independently generated for different dimensions. By default, an object dataset contains 200K points and a feature dataset contains 100K points. The real datasets are described in Table 1. All of them are geographical datasets of China, available at the *Digital Chart of the World*.

Table 1: Rane of Parameter Values

Parameter	Values
Buffer size	0.1,0.2,0.5,1,2,5,10
Object data size	100,200,400,800,1600
Feautre data size	50,100,200,400,800
Quality skewness	1,2,4,8,16,32,64
Number of results,k	1,2,4,8,16,32,64
Number of features,m	1,2,3,4,5
Query range	10,20,50,100,200



(a) Range score (b) NN score

Figure 3: Effect of buffer size, range scores.

Figure 3 compares the cost of the algorithms with respect to the object data size |D|. Since the cost of FJ is dominated by the cost of joining feature datasets, it is insensitive to |D|. On the other hand, the cost of the other methods (SP, GP, BB) increases with |D|, as score computations need to be performed for more objects in D.

VII. CONCLUSION

In this paper, we studied a top-k spatial preference query, which provides a novel type of ranking for spatial objects based on qualities of features in their neighbourhood. We presented several algorithms for processing top-*k* spatial preference queries. First, we introduced a baseline algorithm SP that computes the scores of every object by performing spatial queries on feature datasets. SP is optimized by an incremental computation technique that reduces the number of component score computations for the objects. Next, we presented the GP, a variant of SP that reduces I/O cost by computing scores of objects in the same leaf node concurrently. Based on GP, we developed algorithm BB, which prunes non-leaf entries in the object tree that cannot lead to better results. For this, we developed techniques for deriving upper bound scores for non-leaf entries in the object tree by accessing feature trees. Finally, we propose algorithm FJ, which performs a multi-way join on feature trees to obtain combinations of feature points that commonly affect a spatial region and then search for the objects (in the object tree) affected by these combinations.

Our experimental results show that BB outperforms SP and GP, since SP and GP examine every object in the object tree, whereas BB applies pruning techniques to reduce the number of objects to be examined (and thus their score computations). FJ is more efficient than BB for two or less feature sets, because FJ effectively discovers combinations of features that may lead to results with high scores. BB and FJ mainly access object data and feature data, respectively. Thus, BB is the best method when the object dataset is small whereas FJ is the best algorithm when there are few and small feature datasets. Apart from the problem variants discussed in Section 5, in the future, we plan to design a cost model for BB and FJ so that the query optimizer is able to decide the best query algorithm (either BB or FJ) for a particular problem input. Another interesting research direction is to investigate efficient processing of top-*k* spatial preference queries on non indexed data

VIII. REFERENCES

[1] M. L. Yiu, X. Dai, N. Mamoulis, and M. Vaitis, “Top-k Spatial Preference Queries,” in ICDE, 2007.

[2] N. Bruno, L. Gravano, and A. Marian, “Evaluating Top-k Queries over Web-accessible Databases,” in ICDE, 2002.

[3] A. Guttman, “R-Trees: A Dynamic Index Structure for Spatial Searching,” in SIGMOD, 1984.

[4] G. R. Hjaltason and H. Samet, “Distance Browsing in Spatial Databases,” TODS, vol. 24(2), pp. 265–318, 1999.

[5] R. Weber, H.-J. Schek, and S. Blott, “A quantitative anaysis and performance study for similarity-search methods in highdimensional spaces.” in VLDB, 1998.

[6] K. S. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, “When is “nearest neighbor” meaningful?” in ICDT, 1999.

[7] R. Fagin, A. Lotem, and M. Naor, “Optimal Aggregation Algorithms for Middleware,” in PODS, 2001.

[8] I. F. Ilyas, W. G. Aref, and A. Elmagarmid, “Supporting Top-k Join Queries in Relational Databases,” in VLDB, 2003.

[9] N. Mamoulis, M. L. Yiu, K. H. Cheng, and D. W. Cheung,“Efficient Top-k Aggregation of Ranked Inputs,” ACM TODS, vol. 32, no. 3, p. 19, 2007.

[10] D. Papadias, P. Kalnis, J. Zhang, and Y. Tao, “Efficient OLAP Operations in Spatial Data Warehouses,” in SSTD, 2001.

[11] S. Hong, B. Moon, and S. Lee, “Efficient Execution of Range Topk Queries in Aggregate R-Trees,” IEICE Transactions, vol. 88-D,no. 11, pp. 2544–2554, 2005..