IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 17, Issue 3, Ver. V (May – Jun. 2015), PP 07-10
www.iosrjournals.org

Efficient Refining Of Why-Not Questions on Top-K Queries

P. Haripriya[1], J. Jegan Amarnath[2]
[1]*P.G student, Sri Sairam Engineering College, Chennai.*
[2]*Assistant Professor, Sri Sairam Engineering College, Chennai,*

***Abstract**: After decades of effort working on database performance, the quality and the usability of database systems have received more attention in recent years. In particular, answering the why-not questions after a search is made has become more important. In this project, the problem of answering why-not questions on top-k queries and refining the user query is solved. Generally many users love to pose those kinds of queries when they are making multi-criteria decisions. However, they would also want to know why their expected answers do not show up in the query results. The different algorithms are developed to answer such why-not questions efficiently. Top-K dominating questions are those which have more than one or two results. When this case occurs, the result is ordered according to highest ranking among the records. A search is made and result is displayed, if the expected tuple does not appear then user raises a why-not query. This query is refined using algorithm and then the result is calculated. A penalty function is added such that the result can be returned efficiently and without any fault.*

***Keyword:** why-not questions, Top-K and Dominating queries, penalty.*

I. Introduction

Database technology has made great strides in the past decades. Today, we are able to process ever larger numbers of ever more complex queries on ever more humongous data sets efficiently. Internet search engines have popularized keyword based search. Users provide keywords to the user interface and a ranked list of documents is displayed to the user. A why-not question is being posed when a user wants to know why her expected tuples do not show up in the query result. A certain effort has worked on answering why-not questions on traditional relational or the SQL queries. But none of those can answer why-not questions on preference queries like top-k queries yet. Answering the why-not questions gives the purpose of using the data mining algorithms. The main goal is to find a refined top-k query that include non-empty set of missing objects and the user's initial query. A non-empty set includes all the contents and the set is found to be not empty. When the user provides a query with the count or search for a top-k query then the process analyses and produces the result without the non-empty set.

For example, a user of DBLife may be surprised to find out that the system believes that a person was not on the program committee of conference of another. In fact he may have actually been on the program committee, but this fact does not appear in the extracted data, perhaps due to bugs in extractors, or in accuracies in sources, or incomplete coverage of sources. Therefore, it is important to help developers debug the system and to help users understand why they got the result they did. On the other hand, if the result really shouldn't be in the result, it is a must to explain to the user why this is the case so that they can gain confidence in the non-answer. [1][2].Top-k dominating queries, or just dominating queries, is a form of top-k query that users may pose why-not questions on. While a top-k dominating query frees users from specifying the set of weightings by ranking the objects based on the number of (other) objects that they could dominate.

Both the why-not and Dominating Top-k queries are explained with two algorithms to provide the reason for the missing records. The main goals can be explained as the problem formulation, the problem analysis, and the algorithms of answering why-not questions on top-k queries and dominating queries. Also given thing is that there are an infinite number of points (weightings) in the weighting space we should put limited amount of the records into S in order to obtain a good approximation of the answer. Searching for the particular keyword through traditional information retrieval techniques for enabling keyword search in document collections use data structures such as inverted lists that efficiently identify documents containing a query keyword is another method. A straight forward mapping of this idea to databases is a symbol table that stores information at row level granularity that is we keep the list of rows that contains the keyword. Alternative symbol table designs are possible where we can leverage the physical design of the database. For example, if a column has an index then we only need column level granularity. For this purpose we only store the list of columns for each keyword where they occur.

II. Related Work

The concept of why-not is first discussed in [6]. This work answers a user's why-not question on Select-Project-Join(SPJ) queries by telling her which query operator(s) eliminated her desired answers.

After that, this line of work is gradually expanded. In [7] and [8], the missing answers of SPJ [7] and SPJUA (SPJ + Union + Aggregation) queries are explained by a data-refinement approach, i.e., it tells the user how the data should be modified if user wants the missing answer back to the result.

Answering why-not for a Top-k query was explained by Zhian He and Eric Lo [1]. An algorithm is discussed for answering the queries posed by the user on a Top-K query. Also a defined dimensional disk space has been provided for the records or the tuples on the Top-k rankings and positions. The main goal is about the basic top-k query where users need to specify the set of weightings and the query where users do not need to specify the set of weightings because the ranking function ranks an object higher if it can dominate more objects. The target is focussed mainly to give an explanation to a user who is wondering why her expected answers are missing in the query result. Since the problems are non-identical, a different explanation models for top-k queries and top-k dominating queries is given.

Islam, M.S.,Rui Zhou, Chengfei Liu has proposed a method for answering the why-not question on Reverse Skyline queries. This query recovers all data points whose dynamic skylines contain the query point. The benefit and the semantics of answering why-not questions in reverse skyline queries are defined. A technique to modify the why-not point and the query point to include the why-not point in the reverse skyline of the query point is given. This point can be placed anywhere within a region safely without losing any of the existing reverse skyline points. Considering the safe region of the query point answering a why-not question is done. The procedure also efficiently combines both query point and data point modification techniques to produce meaningful answers.

Vermeulen, Vanderhulst, Luyten, Coninx, Karin started a method of answering the why-not question through the pervasive crystal. The condition becomes distressed when they are unable to understand and control a pervasive computing environment. Also the other works have shown that allowing users to pose why and why not questions about context-aware applications resulted in better understanding and stronger feelings of trust. Though why-not questions have been used before to aid in debugging and to clarify graphical user interfaces, it is not clear how they can be integrated into pervasive computing systems. In existing framework with support for why and why-not question is extended for the search of missing keywords. So a new method called Pervasive Crystal which is a system for asking and answering why and why-not questions in pervasive computing environments was derived.

Islam M.S has proposed a related process where a database is efficiently used in this process without wasting the sample space. There is a growing interest in allowing users to ask questions on received results in the hope of improving the usability of database systems. Islam M.S. has proposed this approach which aims at answering the so called why and why-not questions on received results with respect to different query settings in databases. The goals of this research can be explained as studying the problem of answering the why and the why-not questions in relative databases, explain the efficient strategies for answering these questions in terms of different settings and finally developing a framework that can take advantage of the existing data indexing and query evaluation techniques for the purpose of answering such questions in the databases. The progressed research work contributes completely towards improving the usability of traditional database systems. The similarity between the current and the related work is that an algorithm for refining those why-not questions is given and it is efficient in time. Analyzing and answering a dominating Top-k query is not an easy task and it is also solved by giving different weightings to the set of the records.

III. Problem And Analysis

A table is called a trusted table if it is assumed to be correct and complete, so we do not have to consider updates or insertions to it when computing the provenance of non-answers. An attribute is called a trusted attribute if its values in existing tuples are correct and therefore updates to them can be ignored. But that new values can appear in trusted attributes when new tuples are inserted. The user must either choose to trust tables or individual attributes that appear in a database else the corresponding objects. In a database, each object p with d attribute values can be represented as a point p = |p[1] p[2] ...p[d]| in a d-dimensional **data space** Rd. Now we assume that all attribute values are numeric and a smaller value means a better score for simplicity. A top-k query is composed of a scoring function which gives a result set size r, and a weighting vector w = w [1] w[2]. The scoring function as any monotonic function is accepted and the **weighting space** subject to the constraints $w[i] = 1$ and $0 \leq w[i] \leq 1$ is assumed. The query result would then be a set of k objects whose scores are the smallest.

The penalty function is developed in case of interruption of the process. Nevertheless, the solution works for all kinds of monotonic penalty functions. A technique to skip many of those progressive top-k operations so as to improve the algorithm's efficiency is also presented. A much more aggressive and effective

stopping condition that makes most of those operations stop is also presented. Two techniques together can significantly reduce the overall running time of the algorithm. Since general QP solver requires the solution space be convex, first divide Wri into Cnj −1. Each convex coordinate corresponds to a quadratic programming problem. After solving all these quadratic programming problems, the best wri would be identified. For all rankings to be considered there are n+1 j=1 Cnj −1 = 2n (n is the number of incomparable objects with m) quadratic programming problems in the worst case.

An approach for finding out the multiple missing objects is proposed. The main goal is considered through varying the data size, query dimension, count or the number of missing objects, performance. A sampling-based algorithm that finds the best approximate answer is proposed. A progressive top-k query q based on the weighting vector w in the user's original query q is posed using any progressive top-k query evaluation algorithm, and stop when m comes forth to the result set with a ranking ro. If m does not appear in the query result, then report to the user that m does not exist in the database and the process terminates. If m exists in the database, then randomly sample a list of weighting vectors S = [w1, w2 . . . ws] from the weighting space.

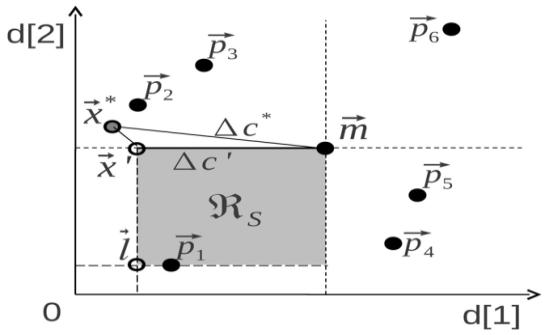

Fig: 1. Restricted Sample space

IV. Answering Dominating Top-K Query

The basic idea for refining why-not top-k dominating queries is similar to the idea of answering top-k why-not questions. First in the case where there is only one missing object m execute a top-k dominating query q_o using a progressive top-k dominating query evaluation algorithm and stop when m comes forth to the result set with a ranking ro. If m does not appear in the query result, inform the user that m does not exist in the database and the process terminates. tie at rank k-th, only one of them is returned). Initially, a user poses a top-k dominating query qo(ko).After she gets the result, she may pose a why-not question on qo with a set of missing objects M = {m1, . . . , mj}. By using only the query-refinement approach here, we can only modify the value of k in order to make M appear in the result. That may result in a refined query whose k's value is increased significantly if there are some missing objects that are actually dominated by many points. As such, we also use the data-refinement approach [7], [8] here. That is, we may either adjust the value of k, the values of m1, . . . , mj, or both.

Now, formally, the problem is: Given a why-not question {M, qo(ko)}, where M is a non-empty set of missing objects, qo(ko) is the user's initial top-k dominating query, the goal is to find a new value k_ and a value replacement M_ for M, such that all the objects in M_ appear in the result of refined dominating query q(k) with the smallest penalty based on the weightings.

V. Algorithm

[PHASE-1] The algorithm first executes a progressive top-k dominating query evaluation algorithm to locate the list of objects, together with their scores, and the list is given as L with rank 1, 2, 3,... until the missing object m shows up in the result in rank rth. Now denote that operation as (L, ro) = DOMINATING(UNTIL-SEE- _m). After that, it samples data values _x1, _x2, . . . , _xs from the restricted sample space Rs and adds them into S.

[PHASE-2] Next, for **some** data value sample _xi of S, modify m's values to be xi and then determine the ranking ri of m after the value modification. Note that the ranking ri basically can be determined by executing a progressive top-k dominating algorithm once again on the database. The Technique is given as (a) below to illustrate a much efficient way to determine the ranking ri, without actually invoking the progressive top-k dominating algorithm. Therefore the technique in why-not top-k processing can be applied here to skip ranking calculations for some data value samples.

[PHASE-3] After PHASE-2, we should have s + 1 "refined queries and modified values" pairs: <q>o(ro), m = m,<q>1(r1), m = <x1>, . . . , <q>s+1(rs+1), m = <xs+1>. The pair with the least penalty is returned to the user as the answer.

Technique —Efficient ranking computation for a sample point

This method describes how to efficiently compute the ranking ri of m if setting m's value to _xi. First, we compute the new score of m which is the number of objects dominated by m, when its values equal to sample _xi. This technique can be easily done by any skyline-related algorithm or by posing a simple range query on an R-tree. Next update the scores of all objects in L (stored in PHASE-1) as the value of m is changed to _xi. do not update the scores of objects not in L because they were either dominated by _m or incomparable with m. So, their scores would not get changed. For the objects in L that do not dominate _m, their scores are unchanged because if they did not dominate _m before, they also cannot dominate m now (because _m gets a better value _xi). Only for those objects in L that dominate _m, we check whether every such object dominates _xi (which is _m's new value), if yes, its score is unchanged; otherwise its score is reduced by one. With all the updated scores in place, we can easily determine the new ranking ri of _m. We represent this operation as:

ri = COMPUTE-RANK(_m, _xi).

A case study is done for a NBA database selecting top-k players for center, guard and other positions. The search is made according to their rankings and the why-not questions posed by the users are answered respectively based on their weightings. The technique proposed in the algorithm is tested and experimented with a sample database containing the records. The effectiveness of techniques are also very promising. Without using any optimization technique, the algorithm requires about 1500 seconds and 400 seconds on uniform dataset and anti-correlated dataset, respectively. But when optimization techniques are enabled, the algorithm runs about two orders of magnitude faster — it requires only about 10 seconds and 2 seconds on uniform dataset and anti-correlated dataset, respectively.

VI. Conclusion

The refining of why-not questions on Top-k queries is studied. There are different techniques for answering a why-not questions but this algorithm helps users to get a efficient answer to their questions. While a search is made the why-not query must be refined such that the errors will be avoided at an initial state. The basic top-k query where users need to specify the set of weightings, and the top-k dominating query where users do not need to specify the set of weightings because the ranking function ranks an object higher if it can dominate more objects. The target is to give an explanation to a user who is wondering why her expected answers are missing in the query result. Since the problems are different, so a different explanation models for top-k queries and top-k dominating queries is used . For the former, the user gets a refined query with approximately minimal changes to the k value and their weightings. For the latter, user gets a refined query with approximately minimal changes to the k value and the missing objects' data values. In the future work the case of non-numeric attributes will be studied.

References

[1]. Sanjay Agrawal, Surajit Chaudhuri, Gautam Das, "DBXplorer: A System for Keyword-Based Search over Relational Databases" 2010.

[2]. Zhian He, Eric Lo, "Answering Why-Not Questions on Top-K Queries", IEEE transactions on knowledge and data engineering, vol. 26, no. 6, june 2014.

[3]. Islam , M.S ; Rui Zhou ; Chengfei Liu, "On answering why-not questions in reverse skyline queries", IEEE transactions on Mining techniques, April 2013.

[4]. Islam, M.S., "On answering why and why-not questions in Databases", Data Engineering Workshop (ICDEW),IEEE international conference,2013.

[5]. Vermeulen,J; Vanderhulst, G. ; Luyten,K. ; Conix, karin ," PervasiveCrystal: Asking and Answering Why and Why Not Questions about Pervasive Computing Applications", IEEE Conference publications, 2010.

[6]. Jiajun Gu ; Kitagawa, H. ," Extending Keyword Search to Metadata on Relational Databases" , Information-Explosion and Next Generation search (IENGS),. 2008.

[7]. Melanie Herschel, MauricioA.Hern´andez, "Explaining Missing Answers to SPJUA Queries", IEEE conference publications, 2008

[8]. E. Tiakas, A. N. Papadopoulos, and Y. Manolopoulos, "Progressive processing of subspace dominating queries," VLDB J., vol. 20, no. 6, pp. 921–948, 2011.

[9]. A. Vlachou, C. Doulkeridis, Y. Kotidis, and K. Nørvåg, "Reverse top-k queries," in Proc. ICDE, Long Beach, CA, USA, 2010, pp. 365–376.

[10]. M. L. Yiu and N. Mamoulis, "Efficent processing of top-k dominating queries on multi-dimensional data," in Proc. VLDB, Vienna, Austria, 2007, pp. 541–552.

[11]. S. Borzsonyi,D. Kossmann, and K. Stocker, "The skyline operator,"ACM Trans. Database Syst., vol. 25, no. 2, pp. 129–178, 2000.

[12]. A. Motro, "Query generalization: A method for interpreting null answers," in Proc. Expert Database Workshop, 1984, pp. 597–616.