A Novel Approach to Query Modification Based on User's Why-not Question

Jianfeng Zhang[1], Weihong Han[1], Yan Jia[1], Peng Zou[2], Hua Fan[1]

[1]School of Computer Science, National University of Defense Technology, Changsha, Hunan, P.R.China, 410073

[2]Academy of Equipment Command and Technology, Beijing, P. R. China

Abstract

Top-k query is an efficient way to show the most important objects to user from massive amounts of data. After huge effort working on database performance, recently, an explain capability has attract more attention in recent years. In top-k query, since people may not specify his/her accurate preference, he may feel frustrated with the top-k results and propose a question such that "why my expecting tuple is not appeared in top-k results as long as tuple p has been appeared in top-k results". Based on this motivation, in this paper, we propose a new method to approach this problem. Given the inputs as the original top-k query, the expecting tuple and the comparable tuple, our algorithm returns a new query to the user which makes the smallest change in the original top-k results. Finally, an extensive performance study using both synthetic and real data set is reported to verify its effectiveness and efficiency.

Keywords: *Top-k Query; Refined Query; Result Explaining; Why Not.*

1. Introduction

Recently, the support of rank aware query processing has attracted much more attentions in database research area. Top-k queries return only a limited of k objects that best match the user's preference, thus avoiding the huge and overwhelming result sets.

Although database system researchers have made tremendous advanced on functionality and performance related issues in the past decades, research on improving database usability has not attracted as much as it deserves. Recent years, the feature of explaining missing objects in database queries, or the so-called "why-not" questions has received growing attentions [1]. For example, users often feel frustrated when they find their expected tuple m is not in the query results but their unexpected tuple p is in the query results without any explanation. So they may propose such a question: "since tuple p is in top-k results, why not tuple m?" If the database system can give a good explanation for it, it would be very useful for uses to understand and modify the query.

Example 1. Take Fig.1 for example. There are five network security alerts in this Table, and each alert is composed of three attributes: Asset, Threat and Occurrence. All these attributes are normalized to a range from 0 to 1. In order to retrieve the Top-2 risky alert, user need to use the scoring function to rank all the alerts. Here we use the simplest but very common scoring function ($f_{\vec{w}}(p) = \vec{w} \cdot \vec{p} = \sum_{i=1}^{d} w[i] \times p.A_i$) to aggregate score of each alerts. Suppose user initiates the query with weight setting [0.333, 0.333, 0.334], then the Top-2 alerts are {5, 4}. Unfortunately, user may be confused by this Top-2 results. In his instinctive thinking, he may consider alert-1 is more risky than alert-4. Now alert 4 is in top-2 results, why not alert-1.

Alert ID	Asset	Threat	Occurence
1	0.7	0.3	0.1
2	0.2	0.4	0.7
3	0.5	0.2	0.2
4	0.4	0.8	0.5
5	0.8	0.6	0.8

QUERY:

```
SELECT * FROME Alert
ORDER BY W[1]*Asset
        +W[2]*Threat
        +W[3]*Occurence
DESC LIMIT K
```

Fig.1 A Set of Some Alerts

In this paper, we present methods to answer user's questions by modifying the original top-k query. Generally speaking, a Top-k query depends on two parameters: the number of objects to be shown in result (i.e. the k value) and user's preference on each attribute (i.e. the weight vector $\vec{w}$). In order to answer user's questions, we need to modify the original query to get a new query. To solve this problem, we first define an evaluation model to evaluate the difference between the original query and the new query. The difference is estimated by the changes of weight vectors and the top-k results. Next, we find a new query with the least change of weight vectors as the temporary optimal new query. Based on this new query, we use a sample method to get some optimal weight vectors and evaluate the quality of query under each

weight vector. Finally, we return the new query with the least changes of original query as the answer.

Our contributions can be summarized as follows:

1. We formulate our problem in explaining user's why not questions on the top-k query. We compactly represent the problem and formally define it.
2. We propose a greedy approach to find the approximate solution to answer user's why not questions, and present several efficient techniques to improve the efficiency of our algorithm.
3. We present an implementation and evaluation of the proposed algorithm in synthetic and real data sets. Extensive experimental results on synthetic and real data sets demonstrate the effectiveness and efficiency of our algorithm

The remainder of this paper is organized as follows. In Section 2, we review the related works. In Section 3, we present a formal problem definition. In Section 4 we propose a greedy algorithm to search the approximate solution to answer user's why not questions. An extensive empirical evaluation using both synthetic data sets and real data sets is reported in Section 5. Finally, our work of this paper is summarized in Section 6.

2. Related Works

There has been some different ways to answer why-not questions. To the best of our knowledge, the major existing approaches to explain why-not questions on top-k query can be summarized as follows.

The first approach explains why-not question by modifying some objects in the database which will include both the original results and the specified missing objects. For example, in [2], the author answers user's why-not question on Select-Project-Join (SPJ) queries by telling him/her which query operators eliminated his/her desired objects.

The second approaches explain missing objects by identifying the manipulation operations in the query plan that is responsible for excluding the missing objects. In [3], the missing answers of Select-Project-Join-Union-Aggregation (SPJUA) queries are explained by telling the user how to modify the data. In [4], the author proposes novel algorithms to generate good quality refined queries that not only similar to the original query but also produce precise query results with minimal irrelevant objects, and these algorithms can answer the SPJ queries both the basically and complex why-not questions with aggregation that involve comparison constraints.

The third typical approaches proposed in the literature to handle the many answers problem are to utilize scoring functions and return only the Top-k ranked results. A represented work that is also related to us is reserve Top-k query[1], which is defined by a given product p and returns the weighting vector for which p in the top-k set. Two versions of reverse top-k queries, namely monochromatic and bichromatic, are presented in this paper. Based on the geometrical properties of the result set, an algorithm for evaluating monochromatic reverse top-k queries is proposed. Thereafter, the author presents an efficient threshold based algorithm for computing bichromatic reverse top-k queries. Another related work is answering why-not questions on top-k query [5]. The authors present methods to answer why-not questions on Top-k queries through modifying both the k value and the set of weighting together. By returning the user a refined query with approximate minimal changes to the k value and their weightings, the user could get not only his/her desired query, but also learn what was/were wrong with her initial query. However, it only considers the situation that makes the missing objects be appeared in top-k results, and ignore the effects on the original top-k results.

3. Problem Statement

In this section, we present some basics regarding to top-k query, and then we formally define the problem how to modify a query based on quality function. Table I summarizes some notions frequently used in this paper.

Table I: The summary of frequently used notions

Notion	***Meaning***
D	A multidimensional data set
d	The dimensionality of D
p, q, m	Objects in data set
$\prec$	A preference relationship
$\|S\|$	The number of objects in S
$\vec{w}$	The weight vector for the scoring function
$w[i]$	The i-th coordinate value of $\vec{w}$

3.1 Preliminaries

We have a data space D of n objects. Each object is described by d attributes. We use $p.A_i$ to refer to the value of an attribute A_i for an object p. For ease of discussion, we assume that all of these numerical attributes are normalized to range from 0 to 1. Furthermore, without loss of generality, we assume that greater score values are more preferable.

Top-k queries are defined based on the scoring function f that aggregates the individual scores into an overall scoring value, which enables the ranking of the data points in data space D. In the simplest but very common case, a linear aggregation function is adopted, which is specified as a weighted sum of scores. Each dimension d_i has an associated query-dependent weight $w[i]$ indicating d_i 's relative importance for the query. The result of a top-k query is a ranked list of the k objects with the best scoring values, where $score(\vec{p},\vec{w}) = \vec{p}\cdot\vec{w}$.

In the Euclidean space a linear top-k query can be represented by a vector $\vec{w}$. As discussed in [6] the magnitude of the query vector does not influence the query result, as long as the direction remains the same, i.e. representing the relative importance between different dimensions. Therefore, we make the assumption that $\sum_{i=1}^{d} w[i] = 1$.

3.2 Problem Definition

In this section, we formally define the problem how to modify a query based on quality function.

Definition 1 (Top-k query): Given a positive integer k and a weight vector $\vec{w}$, the result of $Q(k,\vec{w})$ is a ranked list of objects such that $Q(k,\vec{w}) \in D, |Q(k,\vec{w})| = k$ and $\forall p_1, p_2 : p_1 \in Q(k,\vec{w}), p_2 \in D - Q(k,\vec{w})$ it holds that $score(\vec{p}_1,\vec{w}) > score(\vec{p}_2,\vec{w})$.

Definition 2 (The k-th object): Given a positive integer k and a weight vector $\vec{w}$, the result of $Tuple(k,\vec{w})$ is an object p such that $p \in Q(k,\vec{w}) - Q(k-1,\vec{w})$.

At the beginning, user gives a top-k query $Q_0(k_0,\vec{w}_0)$. Based on the returned results, she/he may propose a question such that: Since object p is in the top-k results, why not miss object m in the top-k results? To solve this problem, we try to define *a new query answer* $Q'(k',\vec{w}')$ which satisfies the conditions as follows.

- object m is in the top-k results of $Q'(k',\vec{w}')$
- $score(\vec{m},\vec{w}') > score(\vec{p},\vec{w}')$

Generally, there exist too many new queries which satisfy the above conditions. So in order to evaluate the quality of the new query, we define an evaluation model as follows.

$$Eval(Q(k',\vec{w}')) = \lambda_w \Delta W + \lambda_c \Delta C \quad (1)$$

where λ_w, λ_c are the user's tolerance to the changes of w and top-k results on her/his original query, and $\lambda_w + \lambda_c = 1$. $\Delta w = \| \vec{w}' - \vec{w}_0 \|_2$ is used to calculate the changes of w, and ΔC is used to measure the changes of the initial top-k results, which is calculated by the following equation.

$$\Delta C = | Q(k_0,\vec{w}_0) \cup Q(k',\vec{w}') | - | Q(k_0,\vec{w}_0) \cap Q(k',\vec{w}') |$$

Since ΔC is much greater than ΔW, we normalize them in Section 4.3. Through this definition, we can see that the new query with the smaller evaluated value is better than these queries with larger values, because it makes a litter change on the original query.

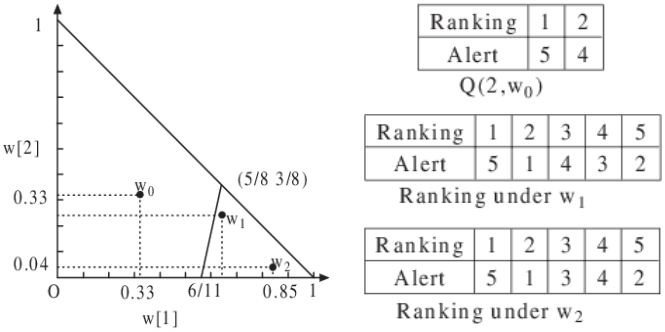


Fig.2 A 3-Dimension example

Example 2. Let us give an example. We use Fig. 2 to explain how to answer user's question that is proposed in example 1. First the results of initial query $Q(2,\vec{w}_0)$ are alert-5 and alert-4. In order to satisfy user's preference $score(\vec{m},\vec{w}') > score(\vec{p},\vec{w}')$, we get some weight vectors as candidate, such as $\vec{w}_1$ and $\vec{w}_2$. Take $\vec{w}_1$ for example, both $Q(2,\vec{w}_1)$ and $Q(3,\vec{w}_1)$ satisfy the conditions of a new query answer. However, we think that $Q(3,\vec{w}_1)$ is a better new query answer than $Q(2,\vec{w}_1)$, because it only adds a new alert-1 into the original top-k results ($\Delta C = 1$). In the same way, $Q(2,\vec{w}_2)$ is an better new query answer than any other queries $Q(k',\vec{w}_2)$ when $k' > 2$. Finally, using evaluation model defined in Eq. (1), we find that $Q(3,\vec{w}_1)$ is better than $Q(2,\vec{w}_2)$ because they have the same ΔC, and $\Delta W_1 < \Delta W_2$.

3.3 Problem Analysis

As discussed above, we try to define a new query which makes m be ranked before p (formally described as $score(\vec{m},\vec{w}') > score(\vec{p},\vec{w}')$). In the data space, we say if an object p domains an object m, and then $score(\vec{p},\vec{w}') > score(\vec{m},\vec{w}')$ is always hold, so this situation is out of our consideration. Otherwise if an object p is incomparable with m, then a hyper plane

$H:(\vec{m}-\vec{p})\cdot\vec{w}=0$ partitions weight space into two part $W_>$ and $W_<$. The $\vec{w}$ located in $W_>$ makes $score(\vec{m},\vec{w})>score(\vec{p},\vec{w})$. The $\vec{w}$ located in $W_<$ means that $score(\vec{m},\vec{w})<score(\vec{p},\vec{w})$. So if we want to get a new query that satisfies the condition $score(\vec{m},\vec{p})>score(\vec{p},\vec{w})$, we only need to consider the weight space $W_>$. Based on each $\vec{w}$ in $W_>$, we can get a new query to satisfy the conditions. Furthermore, based on the above quality function, we can evaluate all the new queries, and return the best new query to user as the refined query. However, there are infinite $\vec{w}$ in $W_>$, it's very hard to enumerate all the possible $\vec{w}$. Therefore, we propose a new optimal algorithm to solve this problem.

4. Detailed Techniques

In this section, we present our method to solve the problem how to refine a query based on the quality function. First we give a basic solution to explain our main idea. Then base on this basic solution, we give some techniques to improve the performance of our algorithm.

4.1 Basic Solution

Let us start the discussion from the basic solution. Basically, we solve this problem by four core phases. First, we get the weight space that satisfies $(\vec{m}-\vec{p})\cdot\vec{w}>0$, and sample s weight vectors from this space, which is donated by $S=\{\vec{w}_1,\vec{w}_2,\cdots,\vec{w}_s\}$. Second, based on S, we use the progressive top-k algorithm [7-9] to get the top-k results and the terminal conditions are discussed in Section 4.2. Third, for each $\vec{w}_i\in S$, we terminate the progressive top-k algorithm after executing k_i steps. Then we can evaluate the quality of each query $Q(k_i,\vec{w}_i)$ by Eq. (1). Finally, we choose the query with the least value as the new query which is returned to user as the answer.
In the following sections we try to improve the efficiency of the basic solution.

4.2 The Terminal Conditions of Progressive Top-k Results

In this section, we try to explain how to terminate the progressive top-k algorithm. As discussed above, we want to get the new query with the least value which is calculated by Eq. (1). So for each $\vec{w}_i\in S$, we need to minimize the ΔC. However through the definition of $\Delta C=|Q(k_0,\vec{w}_0)\cup Q(k',\vec{w}_i)|-|Q(k_0,\vec{w}_0)\cap Q(k',\vec{w}_i)|$, we can see that ΔC depends on $Q_0(k_0,\vec{w}_0)$ and $Q(k',\vec{w}_i)$. Since k_0, $\vec{w}_0$ and $\vec{w}_i$ are certain, the terminal conditions of the progressive top-k algorithm under $\vec{w}_i$ plays a crucial role.

For the sake of brevity, we denote ΔC_k instead of ΔC under the new query $Q(k,\vec{w}_i)$. First, we will give the algorithm (as shown in Algorithm 1) on how to calculate the terminal condition. Then we will give a detailed analysis of this algorithm.

Algorithm 1: GetOptimalK

Input: Dataset D, Weight vector $\vec{w}$, User's initialized query $Q_0(k_0,\vec{w}_0)$
Output: new query $Q(k,\vec{w})$

1. $Q(k_m,\vec{w})\leftarrow$ Running progressive top-k algorithm until to see m;
2. $a=|q\,|\,q\in Q(k_m,\vec{w})\cap Q_0(k_0,\vec{w}_0)|$;
3. $UpperBound\leftarrow k_m+2(k_0-a)$;
4. $\Delta C=\Delta C_{min}=k_0+k_m-2a$;
5. $k_{min}=k_m$;
6. $k=k_m+1$;
7. while $k\le UpperBound$ do
   - 7.1 if $Tuple(k,\vec{w})\in Q_0(k_0,\vec{w}_0)$ then
     - $\Delta C=\Delta C-1$;
   - 7.2 else
     - $\Delta C=\Delta C+1$;
   - 7.3 if $\Delta C<\Delta C_{min}$ then
     - $\Delta C_{min}=\Delta C; k_{min}=k$;
   - 7.4 $k=k+1$;

return $Q(k_{min},\vec{w})$;

To prove the correctness of the above algorithm, in the following, we need to prove that the optimal answer that minimizes Eq. (1) in terms of ΔC has an upper bound (as shown in line 3 Algorithm 1).

Theorem 1: The progressive top-k algorithm could find the optimal answer in $k_m+2(k_0-a)$ steps, where k_m is the rank of m under the weight vector $\vec{w}$ (line 1), a is the number of objects which is both in $Q_0(k_0,\vec{w}_0)$ and $Q(k_m,\vec{w})$ (line 2).

Proof: we proof this theorem by two cases: (i) $Q(k_m,\vec{w})$ may not be the optimal answer. (ii) k' in the optimal answer $Q(k',\vec{w})$ has an upper bound $k_m+2(k_0-a)$. In case (i), we can construct an example to show that $Q(k_m,\vec{w})$ may not the optimal answer. For example,

if $Tuple(k_m+1,\vec{w}) \in Q_0(k_0,\vec{w}_0)$, then ΔC decreases after we increase k_m. So this case shows that the missing object m has appeared in top-k results, but we cannot stop progressive top-k algorithm because of $Eval(Q(k_{m+1},\vec{w})) < Eval(Q(k_m,\vec{w}))$. In case (ii), we try to prove $\forall k'' > k_m + 2(k_0 - a)$, $\Delta C_{k''} > \Delta C_{k_m}$. The proven is as follows.

$$\begin{aligned}\Delta C_{k''} &= |\, Q(k_0,\vec{w}_0) \cup Q(k'',\vec{w}) \,| - |\, Q(k_0,\vec{w}_0) \cap Q(k'',\vec{w}) \,| \\ &= (k_0 + k'' - a'') - a'' = k_0 + k'' - 2a'' \\ &\quad \because\ k'' > k_m + 2(k_0 - a) \\ &> k_0 + k_m + 2(k_0 - a) - 2a'' \\ &\quad \because a'' \le k_0 \\ &\ge k_0 + k_m - 2a = k_0 + (k_m - a) - a \\ &= |\, Q(k_0,\vec{w}_0) \cup Q(k_m,\vec{w}) \,| - |\, Q(k_0,\vec{w}_0) \cap Q(k_m,\vec{w}) \,| \\ &= \Delta C_{k_m}\end{aligned}$$

where $a'' = |\, q \mid q \in Q(k'',\vec{w}) \cap Q_0(k_0,\vec{w}_0) \,|$. Through this case, we can see that $Eval(Q(k_m,\vec{w})) < Eval(Q(k'',\vec{w}))$. So we can see that the progress top-k algorithm could be terminated after $k_m + 2(k_0 - a)$ steps. Combining these two cases together, we can guarantee that our algorithm can get the optimal answer correctly.

4.3 Pruning the weight space

In this section, we will discuss where to get the candidate weight vectors. As discussed above, based on the top-k results returned by the original query, user may propose a question such as p is in top k results, why not object m. In the basic idea, to answer this question, we sample some weight vectors from the weight space that constrained by some inequalities $\Gamma = \{(\vec{m} - \vec{p}) \cdot \vec{w} > 0, w[i] \in [0,1], \sum w[i] = 1\}$. Actually, we only need sample a subspace of Γ. First we will give main steps about how to get the subspace (As shown in Algorithm 2). Then we will try our best to explain why we only need this subspace to be sampled.

Algorithm 2: GetSubRegion

Input: User's initialized query $Q_0(k_0,\vec{w}_0)$, Parameters λ_w, λ_c, Missing object m, Comparable object p

Output: Sampling Region S_{region}

1. $\Gamma = \{(\vec{m} - \vec{p}) \cdot \vec{w} > 0, w[i] \in [0,1], \sum w[i] = 1\}$
2. $\vec{w}_1 \leftarrow$ Project $\vec{w}_0$ to Γ.
3. $Q(k_1,\vec{w}_1) \leftarrow$ GetOptimalK($\vec{w}_1$)
4. $\Delta W_1 = \| \vec{w}_1 - \vec{w}_0 \|_2$
5. $\Delta C_1 = |\, Q(k_0,\vec{w}_0) \cup Q(k_1,\vec{w}_1) \,| - |\, Q(k_0,\vec{w}_0) \cap Q(k_1,\vec{w}_1) \,|$
6. $\Delta W_{max} = \Delta W_1 + \lambda_c \Delta C_1 / \lambda_w$
7. $S_{region} = \{\| \vec{w} - \vec{w}_0 \|_2 < \Delta W_{max}\} \cap \Gamma$

return S_{region};

Now we will give a brief walkthrough of Algorithm 2. First we project $\vec{w}_0$ to Γ, and get the projection point $\vec{w}_1$ [10-13]. Second, base on the $\vec{w}_1$, we use Algorithm 1 to get the optimal query $Q(k_1,\vec{w}_1)$. Then we evaluate the quality of $Q(k_1,\vec{w}_1)$, and get the maximal value of ΔW. Finally, we can prune the weight space, and only reserve the weight space which is described as S_{region} (Theorem 2).

Theorem 2. For any $\vec{w}_i \in W - S_{region}$, it holds that $Eval(Q(k_1,\vec{w}_1)) < Eval(Q(k_i,\vec{w}_i))$, which means that the quality of any query $Q(k_i,\vec{w}_i)$ whose weight vector $\vec{w}_i$ locates out of region S_{region} is worse than the quality of $Q(k_1,\vec{w}_1)$.

Proof:

$$\begin{aligned}Eval(Q(k_i,\vec{w}_i)) &= \lambda_w \Delta W_i + \lambda_c \Delta C_i \\ &> \lambda_w \Delta W_{max} + \lambda_c \Delta C_i \\ &= \lambda_w \Delta W_1 + \lambda_c \Delta C_1 + \lambda_c \Delta C_i \\ &= Eval(Q(k_1,\vec{w}_1)) + \lambda_c \Delta C_i\end{aligned}$$

So when $\vec{w}_i \in W - S_{region}$, $\Delta W_i = \| \vec{w}_i - \vec{w}_0 \|_2 > \Delta W_{max}$, $Eval(Q(k_1,\vec{w}_1)) < Eval(Q(k_i,\vec{w}_i))$ is always hold.

Based on Theorem 2, we can improve our algorithm from three parts. First we try to improve our evaluation model. Second, we try to improve the efficiency of sampling. Finally, we do our best to stop the progressive top-k algorithm as earlier as possible.

1) Normalizing the evaluation model

As we mentioned in Section 3.2, ΔC is much greater than ΔW, so we normalize each of them by their maximal values. We normalize Δw using ΔW_{max}, because for any $\vec{w}_i \in S_{region}$, $\Delta W = \| \vec{w}_i - \vec{w}_0 \|_2 \le \Delta W_{max}$. In a similar way, we normalize ΔC by ΔC_1 in Algorithm 2. Because for any $\vec{w}_i \in S_{region}$, $\Delta W_i > \Delta W_1$ always holds, if $\Delta C_i > \Delta C_1$, the quality of new query under $\vec{w}_i$ must be worse than the query under $\vec{w}_1$. In this case, we discard this $\vec{w}_i$. Now we have a normalized evaluation model as follows.

$$Eval(Q(k',\vec{w}')) = \lambda_w \frac{\Delta W}{\Delta W_{max}} + \lambda_c \frac{\Delta C}{\Delta C_1} \qquad (2)$$

2) How to get the weight vector candidate S?

Instead of sampling the whole weight space, we sample the candidate weight vectors from the S_{region}. Now another problem is how many weight vectors we should sample. In a straightforward thinking, the bigger the $|S|$, the more approximate the answer. However, the bigger size of S means we need to execute more progressive top-k query which increases the running time. To solve this problem, we adopt the method which is discussed in Section III.C in [5]. We say a new query is the best T% query if its quality is better than $(1-T)$% new queries in the whole answer space, and we hope the probability of getting at least one such new query is larger than a certain threshold $\Pr$:

$$1-(1-T\%)^{|S|} \geq \Pr \quad (3)$$

This Equation is general. The sample size is independent of the data size and the dimension but controlled by two parameters T% and $\Pr$.

3) Improve the terminal condition in Section 4.2

After getting the candidates $S=\{\vec{w}_1,\vec{w}_2\cdots,\vec{w}_n\}$ for weight vectors, as shown in Section 4.2, we can use Algorithm 1 to compute the optimal k for each weight vector $\vec{w}\in S$. We can see that for specific $\vec{w}_i$, we use the progressive top-k algorithm to get the top-k objects one by one, and the optimal k under $\vec{w}_i$ is calculated until meeting the upper bound. In order to decrease the execution time, in this section, we try to terminate the progressive top-k algorithm before meeting the upper bound of k. The main idea is that we terminate the progressive top-k algorithm and discard this $\vec{w}_i$ if the quality under this $\vec{w}_i$ will not be better than the quality we calculate before (Suppose that the optimal answer we have gotten is $Q_{opt}(k_{opt},\vec{w}_{opt})$).

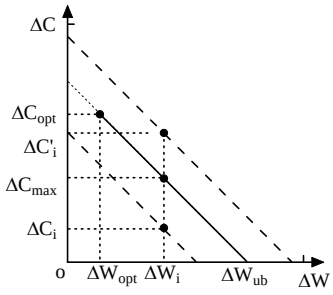


Fig.3 Terminal Conditions

According to Fig. 3, now we will give details about how to improve the performance of progressive top-k algorithm. First, we have an optimal query $Q_{opt}(k_{opt},\vec{w}_{opt})$ (At the beginning, this optimal query is set to be the new query under $\vec{w}_1$). Based on this optimal query, we can evaluate the quality of this query through the Eq. (2) .

$$E_{min}=Eval(Q(k_{opt},\vec{w}_{opt}))=\lambda_w\frac{\Delta W_{opt}}{\Delta W_{max}}+\lambda_c\frac{\Delta C_{opt}}{\Delta C_1}$$

Given a w_i, we consider these two situations.

The first situation is $\Delta W_i>\Delta W_{ub}=E_{min}\Delta W_{max}/\lambda_w$. In this situation, we can see that the quality of new query under $\vec{w}_i$ will never be better than the query $Q_{opt}(k_{opt},\vec{w}_{opt})$. This property could be proven using the same methods in Theorem 2.

The second situation is $\Delta W_i\leq\Delta W_{ub}$. First, we get the upper bound of ΔC_{max}. When using progressive top-k algorithm to get the top-k results, we can get a lower bound of ΔC (represented by ΔC_{lb}) at each step. In case (1), if the missing object m does not appear in top-k results when $\Delta C_{lb}>\Delta C_{max}$, we can discard this weight vector since it could not get a better new query than $Q_{opt}(k_{opt},\vec{w}_{opt})$. In case (2), when the missing object appears in the top-k results, $\Delta C_{lb}<\Delta C_{max}$ is satisfied. As mentioned in Theorem 1, we continue to execute progressive top-k algorithm to find the optimal k which minimizes ΔC. During this procedure, when the terminal condition $\Delta C_{lb}>\Delta C_{max}$ or $k>UpperBound$ (As discussed in Algorithm 1) is satisfied, we can terminate the top-k algorithm under this weight vector.

4.4 Algorithm

The pseudo code of our complete idea is presented in Algorithm 3. It is self explain and mainly discussed above, so we do not give it a walkthrough here.

Algorithm 3: GetOptimalQuery

Input: Dataset D, User's initialized query $Q_0(k_0,\vec{w}_0)$, Parameters λ_w,λ_c, Missing object m, Comparable object p

Output: new query $Q_{opt}(k_{opt},\vec{w}_{opt})$

1. $S_{region}\leftarrow$ *GetSubRegion*
   //Sample S_{region} and sort them by ΔW
2. $S\leftarrow\{\vec{w}_2,\cdots,\vec{w}_s\}$
   // $\vec{w}_1$ is the projection point of $\vec{w}_0$
3. $Q_{opt}(k_{opt},\vec{w}_{opt})\leftarrow$ *GetOptimalK*($\vec{w}_1$)
4. For each $\vec{w}_i\in S$
   4.1 $\Delta W_i=\|\vec{w}_i-\vec{w}_0\|_2$
   4.2 $E_{min}=Eval(Q_{opt}(k_{opt},\vec{w}_{opt}))$

4.3 If $\Delta W_i > E_{min}\Delta W_{max} / \lambda_w$ then
Continue
4.4 $\Delta C_{max} = (E_{min} - \lambda_w \frac{\Delta W_i}{\Delta W_{max}})\Delta C_1 / \lambda_c$
//running progressive top-k algorithm
4.5 $k = 1$; $\Delta C_{lb} = 0$
4.6 While $Tuple(k, \vec{w}_i) \neq m$ do
if $Tuple(k, \vec{w}_i) \notin Top(k_0, \vec{w}_0)$ then
$\Delta C_{lb} = \Delta C_{lb} + 1$
If $\Delta C_{lb} > \Delta C_{max}$ then
continue
$k = k + 1$;
4.7 $a = | q \mid q \in Q(k, \vec{w}_i) \cap Q_0(k_0, \vec{w}_0) |$
4.8 $UpperBound \leftarrow k + 2(k_0 - a)$;
4.9 $\Delta C = \Delta C_{min} = k_0 + k - 2a$;
4.10 $k_{min} = k$; $k = k + 1$;
4.11while $k \leq UpperBound$ and $\Delta C_{lb} \leq \Delta C_{max}$ do
if $Tuple(k, \vec{w}_i) \in Q_0(k_0, \vec{w}_0)$ then
$\Delta C = \Delta C - 1$;
else
$\Delta C = \Delta C + 1$; $\Delta C_{lb} = \Delta C_{lb} + 1$
if $\Delta C < \Delta C_{min}$ then
$\Delta C_{min} = \Delta C; k_{min} = k$;
$k = k + 1$;
//evaluate the quality of new query
4.12 If $Eval(Q(k_{\min}, \vec{w}_i)) < E_{min}$
$E_{min} = Eval(Q(k_{min}, \vec{w}_i))$
$Q_{opt}(k_{opt}, \vec{w}_{opt}) \leftarrow Q(k_{min}, \vec{w}_i)$
return $Q_{opt}(k_{opt}, \vec{w}_{opt})$;

Now we give a brief complexity analysis of our algorithm. We use the algorithm in [10] to compute the distance from a point to a simplex, the complexity of this algorithm is $O(n^4)$, where n is the number of vertices of a simplex. In our case, since the simplex is defined by $\Gamma = \{(\vec{m} - \vec{p}) \cdot \vec{w} > 0, w[i] \in [0,1], \sum w[i] = 1\}$, which makes n be small. Then in the loop to compute the new query under each $\vec{w}_i$, the time cost mainly comes from executing the progressive top-k. As discussed in [8], the cost of the progressive top-k algorithm is $k + | skyline(D) |$. The complexity of our algorithm is $| S | (k + | skyline(D) |)$.

5. Experimental Study

We conduct a thorough performance evaluation on the efficiency and effectiveness of our techniques. Since this work is the first work in query modification (as discussed in related works), our performance evaluation is conducted against our techniques only. Specifically, we focus on evaluating our GetOptimalQuery algorithm in Section 4.4. As there is no existing work, we compare our algorithm under different quality function in Eq. (2) (TMW stands for "tolerate modifying $\vec{w}$" which sets $\lambda_c = 0.9$ and $\lambda_w = 0.1$. TMR stands for "tolerate modifying top-k results" which sets $\lambda_c = 0.1$ and $\lambda_w = 0.9$. NM stands for "Never mind" which sets $\lambda_c = 0.5$ and $\lambda_w = 0.5$).

5.1 Experiment setup

All the experiments are implemented by Java and compiled by JDK 1.7, and we run all the experiments on Ubuntu Linux Operating system with Intel Core-2 Duo Processor and 2GB memory. We use both real and synthetic data sets in our evaluation process.

Real dataset is extracted from NBA players' game-by-game statistics (http://www.nba.com), containing 16916 game statistics of all NBA players from 1973 to 2009 [14]. Each record represents a NBA player by regular season, player name, points per game (PTS), rebounds per game(REB), assists per game(AST), steals per game(STL), blocks per game(BLK), field goal percentage (FGP), field throw percentage (FTP), and three points percentage (TPP). Based on this real dataset, we give an interesting case to show the meaning of our method.

Table II: Parameters setting

Parameter	***Ranges***
Data size	1K, **10K**, 50K, 100K,
Dimension	2, **3**, 4, 5
Original k_0	5, **10**, 15, 20

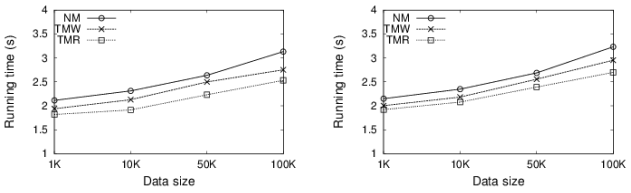
Synthetic datasets are generated using classical method in [15] with respect to the following parameters. Table II summarizes parameter ranges, and the default values are in bold font. Note that the sample size is determined by the Eq. (3), here we set the default T to be 0.2 and Pr to be 0.8 (resulting the sample size of 800 weight vectors). In the experiments below, these parameters use default values unless otherwise specified. Based on these datasets, we test the efficiency of our algorithm.

5.2 Experiment Results

The first experiment is done on the NBA data set. In this experiment, we use three attributes PTS, FGP and FTP to find the top-5 players in NBA history. Therefore, we issue an original top-5 query with the preference $\vec{w}_0$ = (0.333, 0.333, 0.334). The result of this query is {Michael Jordan (1986), Michael Jordan (1987), Michael Jordan (1989), Michael Jordan (1988), George Gervin (1979)}. Then we may be confused with these top-5 results and propose such a question: "since George Gervin is in top-5 results, why not Kobe Bryant". In order to answer this question we use our proposed algorithm with the parameters $Q_0(5, \vec{w}_0)$, missing player Kobe Bryant and the comparable player George Gervin. Using TMW mode, in 3422ms, we get a new query with k=6 and $\vec{w} = (0.379,\ 0.272,\ 0.349)$. The new query indicates us to put more weights on PTS ability if we wish to see that Kobe Bryant is ranked before George Gervin. The corresponding result of $Q(6, \vec{w})$ is {Michael Jordan (1986), Kobe Bryant (2005), Michael Jordan (1987), Bob Mcadoo (1974), Michael Jordan (1989) George_Gervin (1979)}. From this case, we can see that we answer the user's why-not questions through little change of the original top-k query results, only add two new players and remove a player.

Statement. In the following experiments, we test the performance of our algorithm on the synthetic data sets. Given an original query $Q_0(k_0, \vec{w}_0)$, we set the object which is ranked at last of Q_0 to be the comparable object p. Then the missing object m is random selected from $D - Q_0$. As discussed above, there are two case relationships between m and p. If p dominates m (m will never dominates p), we drop this m, and do not run our method under this situation. Otherwise we get the running time of our algorithm and computer the average time cost.

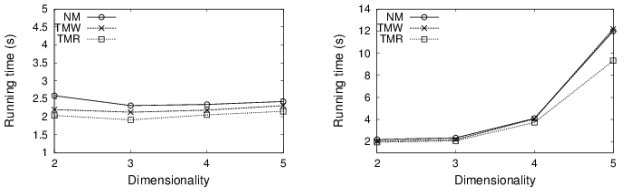
The second experiment tests the running time of our algorithm under different data size. As depicted in Fig. 4, this experiment is conducted against the 4 data size and the 3 tolerant models. It shows that our algorithm scales linearly with the data size. The time cost under TMR (tolerate modifying top-k results is) model is the best because ΔW_{ub} in Fig. 3 is closer to ΔW_{opt} than the other models. Therefore, many weight vectors are discarded at the beginning of Algorithm 4 because they cannot get a better new query answer than the existing optimal answer.



(a) Uniform Data (b) Anti-correlated Data

Fig.4 Varying Data Size

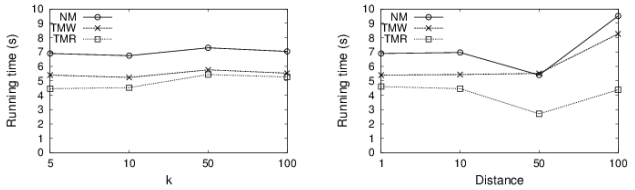
The third experiment, conducted against Uniform and Anti-correlated datasets with dimension range from 2 to 5, evaluates effects of dimension. The results are reported in Fig.5. It demonstrates that the time cost scales linearly with the dimension. We also notice that the running times do not increase under the Uniform Data, but at a faster rate on Anti-correlated Data. Generally speaking, executing a progressive top-k algorithm in high dimension on anti-correlated data set needs more time because of the more attribute and more top-k candidate.



(a) Uniform Data (b) Anti-correlated Data

Fig.5 Varying Dimensionality

The fourth experiment is to test running time under different original k_0 and missing object m. We find that there is almost no relationship between the running time and k_0 or m. So we only show results on anti-correlated data set. In Fig. 6(a), we vary k_0 (This is the same means as we vary the object p), and we select the missing object which is ranked at $k_0 + 10$ in original query (If m is dominated by p, we use $k_0 + 11$ instead). In Fig. 6(b), distance=10 means there at least exists 10 objects between p and m in the original query.

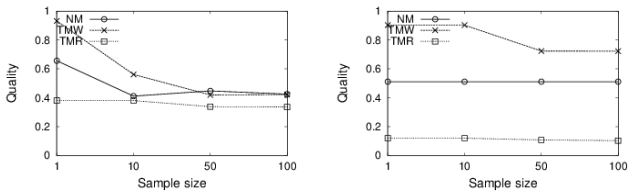


(a) Varying k (b) Varying missing object

Fig.6 Varying Original Query

The final experiment is to evaluate the effects of sample size. We try to analyze the effects which sample size makes on the quality of the new query. Since the sample size is controlled by T and Pr, we do not show the

results of varying T or Pr separately. Fig. 7 shows the quality of optimal new query under different sample sizes. We can see that the quality of the optimal new query under larger sample size is better than the one under smaller sample size. Note that in Fig. 7(b), the quality of NM model is keep the same under different sample size because the quality of new query under the sample weight is worse than the quality of new query under $\vec{w}_1$.



(a) Uniform Data (b) Anti-correlated Data

Fig.7 Varying Sample Size

6. Conclusions

In this paper, we have introduced a new model for solving the problem how to modify the query to answer user's question. At the beginning, we explain the motivation of our problem. Then we have presented this problem and outlined the challenges. Based on the formally definition, we give our solutions for practically realizing such an algorithm. Our experimental evaluation of an implementation of this algorithm in real and synthetic data sets demonstrates the efficiency of our approach.

Acknowledgments

This work is supported by the National 863 Program of China (2010AA012505, 2011AA010702, 2012AA01A401 and 2012AA01A402), National Natural Science Foundation of China (60933005), National Science Technology Support Program of China (2012BAH38B04) and National 242 Information Security Program of China (2011A010). The authors would like to thank the reviewers for their detailed comments which significantly improved this paper.

References

[1] A. Vlachou, C. Doulkeridis, Y. Kotidis et al., "Reverse top-k queries," ICDE, 2010, pp. 365-376.
[2] A. Chapman, and H. V. Jagadish, "Why not?," SIGMOD Conference, 2009, pp. 523-534.
[3] D. Nanongkai, A. D. Sarma, A. Lall et al., "Regret-Minimizing Representative Databases," PVLDB, vol. 3, no. 1, pp. 1114-1124, 2010.
[4] Q. T. Tran, C.-Y. Chan, and C.-Y. Chan, "How to ConQueR why-not questions.," SIGMOD Conference, 2010, pp. 15-26.
[5] Z. He, and E. Lo, "Answering Why-not Questions on Top-k Queries.," ICDE, IEEE Computer Society, 2012, pp. 750-761.
[6] P. Tsaparas, T. Palpanas, Y. Kotidis et al., "Ranked Join Indices," ICDE, 2003, pp. 277-288.
[7] Y.-C. Chang, L. D. Bergman, V. Castelli et al., "The Onion Technique: Indexing for Linear Optimization Queries," Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA, ACM, 2000, pp. 391-402.
[8] L. C. Lei Zou, "Dominant Graph: An Efficient Index Structure to Answer Top-K Queries," ICDE, 2008, pp. 536-545.
[9] R. Fagin, A. Lotem, and M. Naor, "Optimal aggregation algorithms for middleware," in Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, Santa Barbara, California, United States, 2001, pp. 102-113.
[10] V. M. Oleg Golubitsky, and S. M. Watt, "An Algorithm to Compute the Distance from a Point to a Simplex," East Coast Computer Algebra Day, 2012.
[11] R. A. Waltz, J. L. Morales, J. Nocedal et al., An interior algorithm for nonlinear optimization that combines line search and trust region steps, Mathematical Programming 107, 2006.
[12] P. Vincent, and Y. Bengio, "K-Local Hyperplane and Convex Distance Nearest Neighbor Algorithms," Advances in Neural Information Processing Systems, The MIT Press, 2001, pp. 985-992.
[13] C. Michelot, "A finite algorithm for finding the projection of a point onto the canonical simplex of R^n," Journal of Optimization Theory and Applications, vol. 50, pp. 195-200, 1986.
[14] "Databasebasketball_2009_v1.zip," http://www.databasebasketball.com/stats_download.htm.
[15] S. Börzsönyi, D. Kossmann, and K. Stocker, "The Skyline Operator," ICDE, 2001, pp. 421-430.

Jianfeng Zhang is a Ph.D student in School of Computer Science, National University of Defense Technology. He received the B.S. and M.S. degrees in Computer Science from National University of Defense Technology in 2006 and 2009 respectively. His current research is in network security, database system and data mining.

Weihong Han received the PhD degree from National University of Defense Technology. Currently she is an associate professor, and her research interests include network & information security, database and data mining.

Yan Jia received the PhD degree from National University of Defense Technology in 2001. She is a doctoral tutor at National University of Defense Technology. Her research interests include massive database system, information security and data mining.

Peng Zou is a professor and Ph.D. supervisor at National University of Defense Technology. His research interests include network & information security, distributed computing.

Hua Fan is currently a Ph.D student at National University of Defense Technology. He received the B.S. and M.S. degrees in Computer Science from National University of Defense Technology in 2006 and 2009, respectively. He His research interests include stream data management and Sensor network.