View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Institutional Knowledge at Singapore Management University

Singapore Management University
Institutional Knowledge at Singapore Management University

Research Collection School Of Information Systems

School of Information Systems

12-2016

Answering why-not and why questions on reverse top-k queries

Qing LIU
Zhejiang University

Yunjun GAO
Zhejiang University

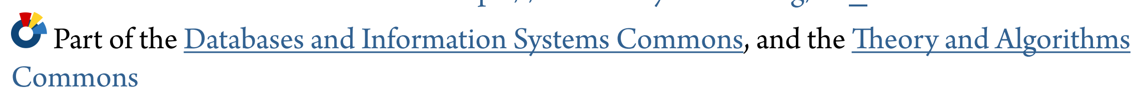
Gang CHEN
Zhejiang University

Baihua ZHENG
Singapore Management University, bhzheng@smu.edu.sg

Linlin ZHOU
Zhejiang University

DOI: https://doi.org/10.1007/s00778-016-0443-4

Follow this and additional works at: https://ink.library.smu.edu.sg/sis_research



Part of the Databases and Information Systems Commons, and the Theory and Algorithms Commons

Citation

LIU, Qing; GAO, Yunjun; CHEN, Gang; ZHENG, Baihua; and ZHOU, Linlin. Answering why-not and why questions on reverse top-k queries. (2016). *VLDB Journal*. 25, (6), 867-892. Research Collection School Of Information Systems.
Available at: https://ink.library.smu.edu.sg/sis_research/3303

This Journal Article is brought to you for free and open access by the School of Information Systems at Institutional Knowledge at Singapore Management University. It has been accepted for inclusion in Research Collection School Of Information Systems by an authorized administrator of Institutional Knowledge at Singapore Management University. For more information, please email libIR@smu.edu.sg.

The VLDB Journal manuscript No.
(will be inserted by the editor)

Answering Why-not and Why Questions on Reverse Top-k Queries

Qing Liu · Yunjun Gao · Gang Chen · Baihua Zheng · Linlin Zhou

Received: date / Accepted: date

Abstract *Why-not* and *why* questions can be posed by database users to seek clarifications on unexpected query results. Specifically, why-not questions aim to explain why certain expected tuples are absent from the query results while why questions try to clarify why certain unexpected tuples are present in the query results. This paper systematically explores the *why-not and why questions on reverse top-k queries*, owing to its importance in multi-criteria decision making. We first formalize why-not questions on reverse top-k queries, which try to include the missing objects in the reverse top-k query results, and then, we propose a *unified framework* called *WQRTQ* to answer why-not questions on reverse top-k queries. Our framework offers three solutions to cater for different application scenarios. Furthermore, we study why questions on reverse top-k queries, which aim to exclude the undesirable objects from the reverse top-k query results, and extend the framework *WQRTQ* to efficiently answer why questions on reverse top-k queries, which demonstrates the flexibility of our proposed algorithms. Extensive experimental evaluation with both real and synthetic data sets verifies the effectiveness and efficiency of the presented algorithms under various experimental settings.

Keywords Reverse top-k query · Why-not question · Why question · Result explanation · Algorithm

Qing Liu · Yunjun Gao (Contact author) · Linlin Zhou · Gang Chen
College of Computer Science
Zhejiang University, Hangzhou, China
E-mail: {liuq, gaoyj, zlinlin, cg}@zju.edu.cn

Yunjun Gao · Gang Chen
The Key Lab of Big Data Intelligent Computing of Zhejiang Province
Zhejiang University, Hangzhou, China
E-mail: {gaoyj, cg}@zju.edu.cn

Baihua Zheng
School of Information Systems
Singapore Management University, Singapore
E-mail: bhzheng@smu.edu.sg

1 Introduction

The capability and usability of database are two research directions in database community. Specifically, the capability of database mainly focuses on the performance and functionality of database systems, which have been significantly improved in the past decades. However, the usability of database is far from meeting user needs due to many characteristics stemming from the users' expectations for interacting with databases [34]. As pointed out in [42], the explain capability, which provides users the explanations for unexpected query results, is one of the important and essential features that is missing from today's database systems. In reality, users always expect the *precise* and *complete* results from the database query. Unfortunately, the database query sometimes returns results that are different from users' expectation, e.g., some expected tuples are missing or some unexpected tuples are present. If a user encounters such cases, intuitively, she wants to pose a *why-not* question to figure out *why* her expected tuples *are not* returned or a *why* question to find out *why* her unexpected tuples *are* returned. If the database system can offer such clarifications, it helps the users understand initial query better and know how to change the query until the satisfactory results are found, hence improving the usability of database.

Currently, there are three categories of methods to answer why-not questions. The first category of methods finds the manipulations which are responsible for excluding users' desired tuples. The typical examples include answering users' why-not questions on Select-Project-Join (SPJ) queries [13] and Select-Project-Join-Union-Aggregation (SPJUA) queries [6]. The second category of approaches provides a set of data modifications (e.g., insertion, update, etc.) so that the missing tuples can present in the query result. This category also mostly focuses on SPJ queries [31,50] and SPJUA queries [27,28]. The third category revises the initial

query to generate a refined query whose result contains the user specified missing tuples. Why-not questions on Select-Project-Join-Aggregation (SPJA) queries [42], top-k queries [24,25], reverse skyline queries [33], to name but a few, all belong to this category. Moreover, the existing provenance techniques, such as *non-annotation method* [9,19], and *annotation approach* [4,18], can be employed to address why questions. Nonetheless, both why-not and why questions are *query-dependent*, and none of existing work can answer why-not and why questions on reverse top-k queries, which is an important and essential building block for multi-criteria decision making. Therefore, in this paper, we study the problem of *answering why-not and why questions on reverse top-k queries*.

Before presenting the reverse top-k query, we first introduce the top-k query. Given a dataset P, a positive integer k, and a preference function f, a top-k query retrieves the k points in P with the *best scores* based on f. The points returned by the top-k query match users' preferences best, and help users to avoid receiving an overwhelming result set. Based on the top-k query, Vlachou et al. [43] propose the reverse top-k query from the manufacturers' perspective, which has a wide range of applications such as market analysis [36,43,45,46] and location-based services [44]. Given a dataset P, a positive integer k, a preference function set *W* (in terms of weighting vectors), and a query point q, a reverse top-k query returns the preference functions in *W* whose top-k query results contain q. Figure 1 illustrates an example of reverse top-k queries. Figure 1(a) records the price and heat production for each computer brand (e.g., Apple, DELL, etc.), and Figure 1(b) lists the customer preferences in terms of weighting vectors by assigning a weight to every attribute. Without loss of generality, we adopt a linear preference function, i.e., $f(\overrightarrow{w}, p)$ = w[*heat*] $\times$ *p.heat* + w[*price*] $\times$ *p.price*, to compute the score of a point p w.r.t. a weighting vector $\overrightarrow{w}$. Figure 1(c) depicts the score of every computer for different customers, and we assume that smaller values are more preferable. Based on Figure 1(c), if Apple issues a reverse top-3 (k = 3) query at a query point/computer q, Anna and Tony are retrieved as they rank the query computer q as one of their top-3 options. In other words, reverse top-k queries can help Apple to identify the potential customers who are more likely to be interested in its product(s), and thus to assess the impact of product(s) in the market.

Unfortunately, reverse top-k queries only return query results to users *without any explanation*. If the query result does not contain some expected tuples, it may disappoint users. Consider the aforementioned example again. Suppose Kevin and Julia are Apple's existing customers, however, they are not in the result of the reverse top-3 query of q. Apple may feel frustrated, and ask "*Why Kevin and Julia do not take Apple as one of their choices? What actions should be taken to win them back?*" If the database system can offer such clarifications, it will help Apple to retain existing customers as well as to attract more new customers, and hence to increase/maintain its market share. In view of this, for the first time, we explore why-not questions on reverse top-k queries, which could be an important and useful tool for market analysis. Given an original reverse top-k query q and a why-not weighting vector set W_m that is missing from the query result, why-not questions on reverse top-k queries suggest how to refine the original query via changing q to q' and/or changing W_m and k to W'_m and k' such that i) W'_m (that might be equivalent to W_m) *does* present in the query result of top-k' query q'; and ii) the penalty caused by changing (q, W_m, k) to (q', W'_m, k') is minimum. Note, the penalty is evaluated by the penalty models proposed in Section 3 to quantify the changes.

In addition to why-not questions on reverse top-k queries, we also explore why questions on reverse top-k queries in this work, which also has a large application base. Back to the above example in Figure 1. Assume that the query computer q is designed for professional developers. After issuing a reverse top-3 query, Apple finds that Tony, a high school student, is also interested in the computer q. It may puzzle Apple *"Why does Tony also like this computer? Are the configurations of* q *appealing to not only professional developers but also students? What actions should be taken such that only the professional developers will choose this computer?"* If the database system can offer answers to these questions, it can help Apple to design products that capture the real preferences and requirements of their target customers better. Towards this, in this paper, we study why questions on reverse top-k queries. Specifically, given an original reverse top-k query and a why weighting vector set W_p that is unexpected but present in the query result, why questions on reverse top-k queries suggest how to refine the original query with minimum penalty such that W_p is excluded from the refined query result. Note that the penalty models used to quantify the modification of the refined reverse top-k query for why questions are proposed in Section 4.

In this paper, we present a *unified framework* called WQRTQ, which provides three solutions to answer why-not questions on reverse top-k queries to cater for different application scenarios. The first solution is to modify a query point q using the *quadratic programming* (e.g., Apple changes the configurations of the computer as a solution to win back certain customers). The second solution adopts a *sampling based method*, which modifies a weighting vector set W_m and a parameter k (e.g., Apple can employ proper marketing strategies to influence the customers' preferences so that the new computer launched by Apple will appear in their wish-list). The third solution is to modify q, W_m, and k simultaneously, which integrates the *quadratic program-*

computer	p_1 Dell	p_2 HP	p_3 Sony	p_4 IBM	p_5 Acer	p_6 ASUS	p_7 NEC	q Apple
price	2	6	1	9	7	5	3	4
heat	1	3	9	3	5	8	7	4

(a) The information of computers

costomer	occupation	w[*price*]	w[*heat*]
Kevin ($\vec{w}_1$)	developer	0.1	0.9
Anna ($\vec{w}_2$)	developer	0.3	0.7
Tony ($\vec{w}_3$)	student	0.5	0.5
Julia ($\vec{w}_4$)	student	0.9	0.1

(b) The customer preferences

computer *id*	p_1	p_2	p_3	p_4	p_5	p_6	p_7	q
score for Kevin	1.1	3.3	8.2	3.6	5.2	7.7	6.6	4
score for Anna	1.3	3.9	6.6	4.8	5.6	7.1	5.8	4
score for Tony	1.5	4.5	5	6	6	6.5	5	4
score for Julia	1.9	5.7	1.8	8.4	6.8	5.3	3.4	4

(c) The scores of computers

Fig. 1 Example of reverse top-k queries

ming, *sampling method*, and *reuse technique*. This is a combination of previous two solutions, which means both the customers' preferences need to be changed and the settings of the computer will be modified to attract those customers back. In addition, we extend WQRTQ to efficiently answer why questions on reverse top-k queries, which demonstrates the flexibility of our proposed algorithms. Extensive experiments using both real and synthetic datasets show that our proposed algorithms can produce clarifications and suggest changes efficiently.

In brief, the key contributions of this paper are summarized as follows:

- We solve why-not questions on reverse top-k queries. To our knowledge, there is no prior work on this problem.
- We present a unified framework WQRTQ, including three different approaches, to answer why-not questions on reverse top-k queries.
- We study a complimentary problem to why-not questions on reverse top-k queries, namely, why questions on reverse top-k queries, and extend our WQRTQ framework with new algorithms to tackle it.
- We conduct extensive experimental evaluation using both real and synthetic datasets to demonstrate the effectiveness and efficiency of the proposed algorithms under a variety of experimental settings.

A preliminary version of this work has been published in [21]. As an extension, we make following fresh contributions in the paper, which include (i) why questions on reverse top-k queries (Section 4); (ii) enhanced experimental evaluation that incorporates the new class of problem (Section 5); and (iii) more comprehensive related work (in Section 6). In addition, we have further improved the presentation and organization of the paper.

The rest of this paper is organized as follows. Section 2 presents problem formulation. Section 3 elaborates our framework and solutions to answer why-not questions on reverse top-k queries. Section 4 describes our algorithms to answer why questions on reverse top-k queries. Section 5 reports experimental results and our findings. Finally, Section 6 reviews related work, and Section 7 concludes the paper with some directions for future work.

Table 1 Symbols and description

Notation	Description
$f(\overrightarrow{w}, p)$	The score of a point p w.r.t. a weighting vector $\overrightarrow{w}$
W_m/W_p	The why-not/why weighting vector set
$TOPk(\overrightarrow{w})$	The set of top-k points w.r.t. a weighting vector $\overrightarrow{w}$
$H(\overrightarrow{w}, p)$	The hyperplane that is perpendicular to $\overrightarrow{w}$ and contains the point p
$SR(q)$	The safe region of q
$IR(q)$	The invalid region of q
$EIR(q)$	The enhanced invalid region of q
$HS(\overrightarrow{w}, p)$ / $\overline{HS}(\overrightarrow{w}, p)$	The half space/complementary half space formed by $\overrightarrow{w}$ and p
I	A point set that contains all the points incomparable with q
D	A point set that contains all the points dominating q

2 Problem Formulation

In this section, we first introduce the concept of reverse top-k queries, and then provide the formal definition of why-not and why questions on reverse top-k queries, respectively. Table 1 summarizes the notations used throughout this paper.

2.1 Reverse Top-k Queries

Given a d-dimensional dataset P, a point $p \in P$ is represented in the form of $p = \{p[1], \cdots, p[d]\}$, where $p[i]$ refers to the i-th dimensional value of P. The top-k query ranks the points based on the user specified scoring function f that aggregates the individual score of a point into an overall scoring value. In this paper, we utilize a *linear scoring function* (or *weighted sum function*) that is commonly used in the literature [24,25,43,46]. Specifically, within a data space, each dimension i is assigned a weight $w[i]$ indicating the relative importance of the i-th dimension for the query. The weights for all dimensions can be denoted as a weighting vector $\overrightarrow{w} = \{w[1], \cdots, w[d]\}$, in which $w[i] \geq 0$ $(1 \leq i \leq d)$ and $\sum_{i=1}^{d} w[i] = 1$. Then, the aggregated score of any data point p $(\in P)$ with respect to $\overrightarrow{w}$ is $f(\overrightarrow{w}, p) = \sum_{i=1}^{d}(w[i] \times p[i])$. Without loss of generality, we assume that *smaller* scoring values are *preferable* in this paper. Below, we formally define the top-k query.

Definition 2.1 (**Top-k query**). Given a d-dimensional data set P, a positive integer k, and a weighting vector $\overrightarrow{w}$, a top-k query (*TOPk*) returns a set of points, denoted as $TOPk(\overrightarrow{w})$,

such that (i) $TOPk(\overrightarrow{w}) \subseteq P$; (ii) $|TOPk(\overrightarrow{w})| = k$; and (iii) $\forall p_1 \in TOPk(\overrightarrow{w}), \forall p_2 \in P - TOPk(\overrightarrow{w})$, it holds that $f(\overrightarrow{w}, p_1) \leq f(\overrightarrow{w}, p_2)$.

Take the dataset P depicted in Figure 1 as an example. In Figure 1(c), it is observed that the three smallest scores for $\overrightarrow{w}_1$ are $f(\overrightarrow{w}_1, p_1) = 1.1$, $f(\overrightarrow{w}_1, p_2) = 3.3$, and $f(\overrightarrow{w}_1, p_4) = 3.6$. Hence, we have $TOP3(\overrightarrow{w_1}) = \{p_1, p_2, p_4\}$. It is worth mentioning that, if the points share the same score at ranking k-th, only one of them is randomly returned. Based on the definition of the top-k query, we formulate reverse top-k queries by following [43].

Definition 2.2 (Reverse Top-k Query). Given a d-dimensional dataset P, a d-dimensional weighting vector set *W*, a query point q, and a positive integer k, a reverse top-k (***RTOPk***) query retrieves a set of weighting vectors, denoted as $RTOPk(q)$, such that (i) $RTOPk(q) \subseteq W$, and (ii) for every $\overrightarrow{w_i} \in RTOPk(q)$, it holds that $q \in TOPk(\overrightarrow{w_i})$.

A RTOPk query finds the weighting vectors in *W* whose top-k query results contain q. Back to Figure 1 again. As $TOP3(\overrightarrow{w_2}) = \{p_1, p_2, q\}$, $\overrightarrow{w_2}$ belongs to $RTOP3(q)$. After exploring all the potential weighting vectors, we have $RTOP3(q) = \{\overrightarrow{w_2}, \overrightarrow{w_3}\}$. Based on the reverse top-k query, we formally define why-not and why questions on reverse top-k queries in Section 2.2 and Section 2.3, respectively.

2.2 Why-not Questions on Reverse Top-k Queries

In this subsection, we formalize the definition of why-not questions on reverse top-k queries.

Definition 2.3 (Why-not Questions on RTOPk Queries). Given a RTOPk query issued from a query point q on a dataset P based on a weighting vector set *W*, and a why-not weighting vector set $W_m \subseteq W - RTOPk(q)$, the goal of answering why-not questions on RTOPk queries is to find (q', W'_m, k') such that (i) $\forall \overrightarrow{w} \in RTOPk(q)$, $\overrightarrow{w} \in RTOPk'(q')$; (ii) $\forall \overrightarrow{w'_i} \in W'_m$, $\overrightarrow{w'_i} \in RTOPk'(q')$; and (iii) the penalty of changing (q, W_m, k) to (q', W'_m, k'), as defined in Equation (1), is minimum, and then to return the result of RTOP$k'(q')$.

$$Penalty(q', {W_m}', k') = \gamma Penalty(q') + (1-\gamma) Penalty(W'_m, k') \quad (1)$$

In above definition, condition (i) is to guarantee all the weighting vectors that are returned by original RTOPk query at q shall be still returned even after the modification; condition (ii) is to make sure the set of expected weighting vectors that are missing in previous RTOPk query at q will be returned by the modified RTOPk' query (in the form of W'_m which is very close to W_m if not equivalent to W_m); and condition (iii) is to guarantee that the recommended modification is optimal as quantified by $Penalty$. Our above definition only guarantees RTOP$k(q) \subseteq$ RTOP$k'(q')$, and $W'_m \subset$ RTOP$k'(q')$, while RTOP$k'(q')$ - RTOP$k(q)$ - W'_m might not be empty. This means the modification may also return some weighting vectors $\overrightarrow{w}$ that do not belong to either original result set or W'_m. However, why-not questions on reverse top-k queries mainly focus on how to include expected tuples W_m that are missing in the result set back to the result set, and hence we do not consider RTOP$k'(q')$ - RTOP$k(q)$ - W'_m in above definition.

In general, why-not question on $RTOP(q)$ will be issued when an expected weighting vector set W_m is not returned by $RTOP(q)$, and it provides an explanation on the absence of W_m via a refinement (q', W'_m, k'). To be more specific, it tries to include W_m back to the result set via modifying the query point q which stands for the product in our example and/or (W_m, k) which stands for the user preferences in our example, with minimum penalty. In this paper, we have proposed three different solutions to perform the modification. To be more specific, our first solution only changes q' (i.e., $W'_m = W_m$ and $k' = k$ and hence $Penalty(W'_m, k') = 0$) which is catered for the cases where the missing tuples can be re-included by changing the query point. Our second solution only changes W_m and k (i.e., $q' = q$ and hence $Penalty(q') = 0$), which is catered for the cases where the query point has been finalized and hence cannot be changed but parameters W_m and k are flexible. Our third solution changes all three parameters, catered for the cases where the modifications suggested by previous two solutions have their penalties above the limit set by manufacturers or customers. These three solutions will be detailed in Section 3

According to Definition 2.3, for why-not questions on RTOPk queries, the target is to find (q', W'_m, k') such that $\forall \overrightarrow{w'_i} \in W'_m$, $\overrightarrow{w'_i} \in RTOPk'(q')$. Based on Definition 2.2, $\overrightarrow{w'_i} \in RTOPk(q) \longrightarrow q \in TOPk(\overrightarrow{w'_i})$ and $\overrightarrow{w'_i} \notin RTOPk(q) \longrightarrow q \notin TOPk(\overrightarrow{w'_i})$. Hence, why-not questions on RTOPk queries can be re-phrased as: given a RTOPk query based on (q, W_m, k) having $\forall \overrightarrow{w_i} \in W_m$, $q \notin TOPk(\overrightarrow{w_i})$, how to refine the RTOP$k$ query (i.e., to find the tuple (q', W'_m, k')) with minimum penalty such that $\forall \overrightarrow{w'_i} \in W'_m, q' \in TOPk'(\overrightarrow{w'_i})$, as shown in Figure 2(a).

In addition, it is worth mentioning that the why-not questions on top-k queries and reverse top-k queries are two different problems. (i) If the why-not weighting vector set W_m consists of only one weighting vector, our second solution, i.e., modifying W_m and k, is identical with the approach of why-not questions on top-k queries. However, we propose another two new solutions to answer the why-not questions on reverse top-k queries, i.e., modifying the query point q, and modifying q, W_m and k. (ii) If W_m consists of more than one weighting vector, the approach of why-not reverse top-k queries cannot be applied to address our problem since

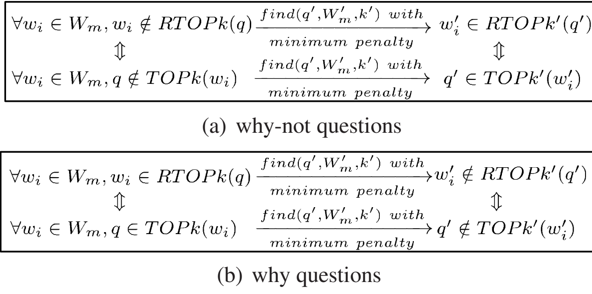


Fig. 2 Transformation of why-not and why questions

the penalty of modified reverse top-k queries is not minimum.

The transformed problem might look similar as the problem of why-not questions on top-k queries [24,25]. Note that given a why-not point set $P_m \subseteq P$ and a weighting vector $\overrightarrow{w}$ having $\forall p_i \in P_m$, $p_i \notin TOPk(\overrightarrow{w})$, why-not questions on top-k queries find (w', k') with minimum penalty such that $\forall p_i \in P_m$, $p_i \in TOPk'(\overrightarrow{w'})$. However, we want to highlight that these two problems are inherently different. First, these two problems have totally different inputs. The inputs of our problem contain a why-not weighting vector set that captures the preferences of customers and a query point q representing a product of the manufacturer, while why-not questions on top-k queries take as inputs a why-not point set that denotes the attributes of products and a weighting vector representing a customer preference. Second, they serve different purposes. Our problem tries to make the product q as one of the top-k choices for the set of a given customer preferences, but why-not questions on top-k queries try to make all the specified products appear in the top-k result of a given weighting vector.

2.3 Why Questions on Reverse Top-k Queries

In real life, users are interested in not only the missing tuples that are absent from the query results, but also the undesirable tuples that are returned as part of the result but are not expected to be present. In the following, we formally define the why questions on reverse top-k queries.

Definition 2.4 (**Why Questions on RTOPk Queries**). Given a RTOPk query issued from a query point q on a dataset P and a weighting vector set W, and a why weighting vector set $W_p \subseteq RTOPk(q)$, why questions on RTOPk queries is to find (q', W'_p, k') such that (i) $\forall \overrightarrow{w'_i} \in W'_p, \overrightarrow{w'_i} \notin RTOPk'(q')$; and (iii) the penalty of (q', W'_p, k'), defined in Equation (2), is minimum, and to return the result of RTOP$k'(q')$.

$$Penalty(q', {W_p}', k') = \gamma Penalty(q') + (1-\gamma) Penalty(W'_p, k') \quad (2)$$

Definition 2.4 looks similar as Definition 2.3, as why-not questions on RTOPk queries are symmetric to why questions on RTOPk queries. However, we want to highlight that why-not questions on RTOPk always suggest a modification such that original results of RTOP$k(q)$ are still present in the new result of RTOP$k'(q')$, as guaranteed by condition (i) in Definition 2.3; while why questions on RTOPk queries cannot guarantee all the original results (excluding W'_m) still remain after we perform the modification. For why questions on RTOPk queries, if the solution exists by keeping all the expected original query result, we will return the corresponding solution with minimum penalty, otherwise we actually ignore the condition of keeping all previous expected results. In our future, we want to study how to remain all the original reverse top-k query results that are expected even after modification.

The goal of answering why questions is to find (q', W'_p, k') with minimum penalties such that the specified why weighting vector(s) will be excluded from the refined query results. Similarly, based on Definition 2.2, why questions on RTOPk queries can be re-phrased as: given a RTOPk query based on (q, W_p, k) having $\forall \overrightarrow{w_i} \in W_p$, $q \in TOPk(\overrightarrow{w_i})$, how to refine the original query (i.e., to find the tuple (q', W'_p, k')) with minimum penalty such that $\forall \overrightarrow{w'_i} \in W'_p$, $q' \notin TOPk'(\overrightarrow{w'_i})$, as shown in Figure 2(b).

It is worth mentioning that the difference between why-not question and why question is two-fold. First, why question takes the objects in the original query result as inputs while why-not question takes the non-answers as inputs. Second, why-not and why question serves opposite purpose, i.e., the goal of why question is to exclude the undesirable objects from the query result while why-not question tries to include the desirable objects in the query result.

3 Answering Why-not Questions

In this section, we propose a unified framework to answer why-not questions on reverse top-k queries, and then detail the framework, which contains three solutions based on the modification of different parameters. Note that, in all our proposed algorithms, we assume the dataset is indexed by an R-tree [2].

3.1 Framework Overview

First, we present a unified framework called WQRTQ (i.e., **W**hy-not **Q**uestions on **R**everse **T**op-k **Q**ueries) to answer why-not questions on reverse top-k queries. As illustrated in Figure 3, WQRTQ takes as inputs an original reverse top-k query and the corresponding why-not weighting vector set W_m, and returns to the users the refined reverse top-k query with minimum penalty. Specifically, it consists of the following three solutions.

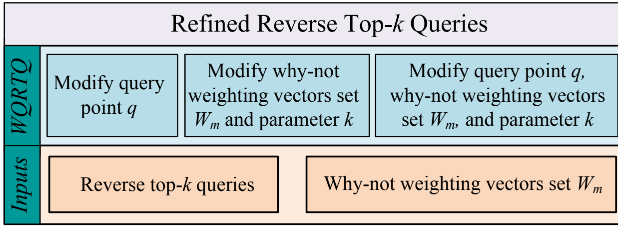


Fig. 3 Framework of WQRTQ

(1) **Modifying q**. The first solution is to change query point q only, from q to q', which is to be detailed in Section 3.2. To this end, we introduce the concept of safe region (see Definition 3.2). As long as the query point q' falls into the safe region, the why-not weighting vector set W_m will appear in the reverse top-k query result of q'. After getting the safe region, we use the quadratic programming to get q' with the minimum change as compared to q.

(2) **Modifying W_m and k**. The second solution, to be presented in Section 3.3, is to modify a why-not weighting vector set W_m and a parameter k into W'_m and k' respectively, such that the modified W'_m belongs to the result of the reverse top-k' query of q. Towards this, we present a sampling-based method to obtain W'_m and k' with the minimum penalty. In particular, we sample a certain number of weighting vectors that may contribute to the final result, and then locate the optimal W'_m and k' according to the sample weighting vectors.

(3) **Modifying q, W_m, and k**. Our third solution is to modify a query point q, a why-not weighting vector set W_m, and a parameter k simultaneously, as to be detailed in Section 3.4. After refining, the modified weighting vector set W'_m is contained in the reverse top-k' query result of q'. This solution utilizes the techniques of quadratic programming, sampling method, and reuse.

It is worth mentioning that all three solutions always return a non-empty result, i.e., a refinement can always be identified. Specifically, the first solution can always find a non-empty safe region within which a refinement can be located; the second solution employs a sampling method to refine the original query and it can locate the answer once the sample weighting vectors are obtained; and above two statements guarantee that the third solution, as a combination of the first solution and the second solution, will always return a non-empty result.

3.2 Modifying q

Intuitively, if Apple finds some existing customers cs that are not interested in its new computer, it can adjust some computer parameters before putting it into production so that the modified computer can re-appear in the lists of the top-k options of those customers cs. In view of this, we propose the first solution to refine the original reverse top-k query, namely, modifying a query point q, as formally defined below.

Definition 3.1 (**Modifying q**). Given a d-dimensional data set P, a positive integer k, a query point q, and a why-not weighting vector set W_m with $\forall \overrightarrow{w_i} \in W_m, q \notin TOPk(\overrightarrow{w_i})$, the modification of a query point q is to find q' such that (i) $\forall \overrightarrow{w_i} \in W_m$, $q' \in TOPk(\overrightarrow{w_i})$; (ii) $\forall \overrightarrow{w_j} \in RTOPk(q)$, $\overrightarrow{w_j} \in RTOPk(q')$; and (iii) the penalty of q', defined in Equation (3), is minimum.

$$Penalty(q') = \frac{|q - q'|}{|q|} = \frac{\sqrt{\sum_{i=1}^{d} (q[i] - q'[i])^2}}{|q|} \tag{3}$$

Assuming that the attributes of an object are independent of each other for simplicity, we use Equation (3) to quantify the modification of the product, which is also employed by Padmanabhan et al. [39] to measure quality distortation for the upgraded product. Note Equation (3) is equivalent to Equation (1) for our first solution as $Penalty(W'_m, k') \equiv 0$ when $W'_m = W_m \wedge k' = k$. For example, in Figure 1, Kevin and Julia are not in the reverse top-3 result of q. If Apple modifies computer's parameter $q(4, 4)$ to $q'(3, 2.5)$ or $q''(2.5, 3.5)$, the new computer q' or q'' becomes one of the top-3 options for both Kevin and Julia. According to Definition 3.1, q'' is more preferable as $Penalty(q') = 0.318 > Penalty(q'') = 0.279$. In some applications, the attributes of an object may have several constraints. Under such circumstance, we can add the corresponding constraints to the Equation (3). Our proposed approach is still applicable by adding those constraints and can support other monotonic functions.

Intuitively, the search space of the query point is the whole data space. However, ensured by the following lemma, we only consider decreasing $q[i]$'s value.

Lemma 3.1 *Given a query point q, let q' is the modified query point with the minimum penalty having $\forall \overrightarrow{w_i} \in W_m$, $q' \in TOPk(\overrightarrow{w_i})$, then $\forall i \in [1, d], q'[i] \leq q[i]$.*

Proof Assume that $\exists j \in [1, d], q'[j] > q[j]$. Then, we can find another point $q'' = \{q''[i] \mid i = j, q''[i] = q[i]; i \neq j, q''[i] = q'[i]\}$. Since (i) $\forall \overrightarrow{w_i} \in W_m$, $q' \in TOPk(\overrightarrow{w_i})$ and (ii) the scoring function is monotonic, it also holds that $\forall \overrightarrow{w_i} \in W_m, q'' \in TOPk(\overrightarrow{w_i})$. In addition, $Penalty(q'') < Penalty(q')$. Therefore, q' is not the qualified modified query point with the minimum penalty, which contradicts the condition of the lemma. Thus, our assumption is invalid and the proof completes. □

As an example, assume that $q(4, 4)$ in Figure 1 is modified to $q'(5, 1)$. We can always find another query point (e.g., $q''(4, 1)$ in this case) that has smaller scoring value and meanwhile generate smaller penalty. In other words, the

search space for q' can be shrunk to [0, q]. Lemma 3.1 also ensures that modifying query point doesn't lose any original reverse top-k query result. Specifically, let q' be the modified query point, and $\overrightarrow{w} \in RTOPk(q)$ be any original result. As $\forall i \in [1, d], q'[i] \leq q[i]$, $f(\overrightarrow{w}, q') \leq f(\overrightarrow{w}, q)$, and hence $q \in TOPk(\overrightarrow{w}) \longrightarrow q' \in TOPk(\overrightarrow{w})$.

Furthermore, to get a qualified q', we find that it is possible to locate a region within [0, q], namely, q's *safe region* as defined in Definition 3.2, that definitely bounds the modified query point q'.

Definition 3.2 (**Safe Region**). Given a d-dimensional data set P, a positive integer k, a query point q, and a why-not weighting vector set W_m, a region in the data space is said to be safe for q (i.e., q's safe region), denoted as $SR(q)$, such that $\forall q' \in SR(q)$ and $\forall \overrightarrow{w_i} \in W_m$, $q' \in TOPk(\overrightarrow{w_i})$.

In other words, if q is modified to q' by moving the query point q anywhere within $SR(q)$, all the why-not weighting vectors will appear in a given reverse top-k query result. Obviously, if we can identify such $SR(q)$, our first solution only needs to return the point in $SR(q)$ that is closest to q. In the sequel, we explain how to derive $SR(q)$. In a d-dimensional space, given a weighting vector $\overrightarrow{w}$ and a point p, we can get a hyperplane, denoted as $H(\overrightarrow{w}, p)$, which is perpendicular to $\overrightarrow{w}$ and contains the point p. Then, we have the lemma below.

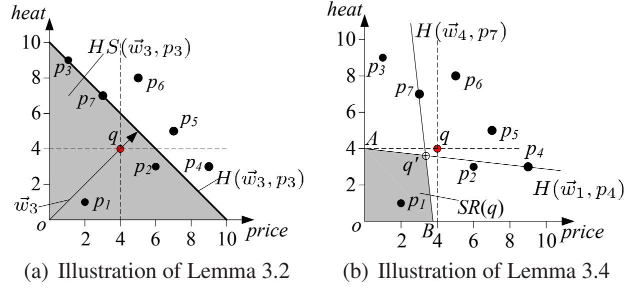
Lemma 3.2 *Given a hyperplane $H(\overrightarrow{w}, p)$ formed by $\overrightarrow{w}$ and p, (i) if a point p' lies on $H(\overrightarrow{w}, p)$, $f(\overrightarrow{w}, p') = f(\overrightarrow{w}, p)$; (ii) if a point p'' lies below $H(\overrightarrow{w}, p)$, $f(\overrightarrow{w}, p'') < f(\overrightarrow{w}, p)$; and (iii) if a point p''' lies above $H(\overrightarrow{w}, p)$, $f(\overrightarrow{w}, p''') > f(\overrightarrow{w}, p)$.*

Proof The proof is straightforward and hence is omitted. □

According to Lemma 3.2, all the points lying on/below/above the hyperplane $H(\overrightarrow{w}, p)$ have the same/smaller/larger scoring values, as compared with p w.r.t. $\overrightarrow{w}$. Figure 4(a) explains Lemma 3.2 in a 2D space, where the hyperplane $H(\overrightarrow{w_3}, p_3)$ is formed by $\overrightarrow{w_3}$ and p_3 in Figure 1. Given points p_1 below $H(\overrightarrow{w_3}, p_3)$, p_5 above $H(\overrightarrow{w_3}, p_3)$, and p_7 on $H(\overrightarrow{w_3}, p_3)$, we have $f(\overrightarrow{w_3}, p_1) < f(\overrightarrow{w_3}, p_3)$, $f(\overrightarrow{w_3}, p_5) > f(\overrightarrow{w_3}, p_3)$, and $f(\overrightarrow{w_3}, p_7) = f(\overrightarrow{w_3}, p_3)$. These findings are also consistent with their scores listed in Figure 1(c). Based on Lemma 3.2, the concept of half space is stated below.

Definition 3.3 (**Half Space**). Given a hyperplane $H(\overrightarrow{w}, p)$, the half space formed by $\overrightarrow{w}$ and p, denoted as $HS(\overrightarrow{w}, p)$, satisfies that $\forall p' \in HS(\overrightarrow{w}, p)$, $f(\overrightarrow{w}, p') \leq f(\overrightarrow{w}, p)$.

In other words, $HS(\overrightarrow{w}, p)$ includes all the points lying on and below the hyperplane $H(\overrightarrow{w}, p)$. Figure 4(a) illustrates the half space $HS(\overrightarrow{w_3}, p_3)$ formed by $\overrightarrow{w_3}$ and p_3, i.e., the shaded area in Figure 4(a). Based on Lemma 3.2 and Definition 3.3, we present the following lemmas to explain the construction of q's safe region.



(a) Illustration of Lemma 3.2 (b) Illustration of Lemma 3.4

Fig. 4 Example of Lemma 3.2 and Lemma 3.4

Lemma 3.3 *Given a weighting vector $\overrightarrow{w}$, and a point p which is the top k-th point of $\overrightarrow{w}$, if $q' \in HS(\overrightarrow{w}, p)$, $q' \in TOPk(\overrightarrow{w})$.*

Proof The proof is straightforward and hence omitted. □

Lemma 3.4 *Given a why-not weighting vector set $W_m = \{\overrightarrow{w_1}, \overrightarrow{w_2}, \cdots, \overrightarrow{w_n}\}$, and a set $\Lambda = \{p_1, p_2, \cdots, p_n\}$ of points ($\forall p_i \in \Lambda$ is the top k-th point w.r.t. its corresponding why-not weighting vector $\overrightarrow{w_i} \in W_m$), the safe region of a query point q refers to the common area covered by all the half spaces formed by $\overrightarrow{w_i}$ and p_i, i.e., $SR(q) = \cap_{1 \leq i \leq n} HS(\overrightarrow{w_i}, p_i)$.*

Proof The proof is straightforward according to Lemma 3.3 and Definition 3.3. □

Figure 4(b) depicts an example of Lemma 3.4, which utilizes the dataset shown in Figure 1. Assume that $\overrightarrow{w_1}$ and $\overrightarrow{w_4}$ are two why-not weighting vectors, and the corresponding the 3-rd points with lowest scores for $\overrightarrow{w_1}$ and $\overrightarrow{w_4}$ are p_4 and p_7, respectively. Therefore, the safe region of q w.r.t. $\{\overrightarrow{w_1}, \overrightarrow{w_4}\}$ is the overlapping of $HS(\overrightarrow{w_1}, p_4)$ and $HS(\overrightarrow{w_4}, p_7)$, i.e., the shaded area (i.e., quadrilateral $AoBq'$) highlighted in Figure 4(b). Note that the safe region formed by the $(k-1)$-th points (denoted as $SR'(q)$) doesn't contain the optimal q'. This is because the hyperplane formed by the $(k-1)$-th points is always below the hyperplane formed by the k-th points.

After getting the safe region of q, we need to find the optimal query point q' with the minimum cost w.r.t. q. Take Figure 4(b) as an example again. Point q' is the desirable refined query point. However, a safe region is a convex polygon bounded by hyper-planes. The above safe region computation does not scale well with the dimensionality because computing the intersection of half spaces becomes increasingly complex and prohibitively expensive in high dimensions [3]. Actually, finding the optimal query point q' with the minimum cost w.r.t. q is an optimization problem. Moreover, the penalty function of q' defined in Equation (3) can be seen as a quadratic function. In light of this, we employ the quadratic programming to find the optimal q' without computing the exact safe region. Specifically, the quadratic

programming can be represented in the following form:

$$\begin{aligned} &min\ f(x) = \tfrac{1}{2}x^T Hx + x^T c \\ &s.t. \begin{cases} Ax \le b \\ lb \le x \le ub \end{cases} \end{aligned} \quad (4)$$

Equation (4) derives the optimal x that minimizes $f(x)$ under the constraints $Ax \le b$ and $lb \le x \le ub$, in which $f(x)$ is an objective function; H and A are matrixes; x, c, b, lb, and ub are vectors; and superscript T denotes transposition. Our problem is actually an optimization problem, with the goal to obtain q' having the smallest penalty. Hence, we utilize the quadratic programming to obtain the optimal q'. Since the denominator in Equation (1) is a positive constant, for simplicity, in this paper, we assume that the objective function for our problem is $f(q') = \sum_{i=1}^{d}(q[i] - q'[i])^2$ $= \frac{1}{2}(q')^T Hq' + (q')^T c$, where H = diag(2, 2, $\cdots$, 2) is a $d \times d$ diagonal matrix with all eigenvalues being 2, and c = $(-2q[1], -2q[2], \cdots, -2q[d])$ is a d-dimensional vector.

In addition, given a why-not weighting vector set W_m = $\{\overrightarrow{w_1}, \overrightarrow{w_2}, \cdots, \overrightarrow{w_n}\}$ and a point set $\Lambda = \{p_1, p_2, \cdots, p_n\}$ ($p_i \in \Lambda$ is the top k-th point of $\overrightarrow{w_i} \in W_m$), the optimal (modified) q' falling within the safe region $SR(q)$ must satisfy that, $\forall \overrightarrow{w_i} \in W_m$ and $\forall p_i \in \Lambda$, $f(\overrightarrow{w_i}, q') \le f(\overrightarrow{w_i}, p_i)$ according to Definition 3.2, which can be represented by $Aq' \le b$ in Equation (4), where A defined below is a $n \times d$ matrix and $b = (f(\overrightarrow{w_1}, p_1), f(\overrightarrow{w_2}, p_2), \cdots, f(\overrightarrow{w_n}, p_n))$. As mentioned earlier, the varying range of q is $[0, q]$. Consequently, $0 \le q' \le q$ corresponds to $lb \le x \le ub$.

$$A = \begin{vmatrix} w_1[1] & w_1[2] & \ldots & w_1[d] \\ w_2[1] & w_2[2] & \ldots & w_2[d] \\ \vdots & \vdots & \ddots & \vdots \\ w_n[1] & w_n[2] & \ldots & w_n[d] \end{vmatrix}$$

Based on the above analysis, we propose the algorithm called MQP-I to **m**odify the **q**uery **p**oint q, whose pseudocode is presented in Algorithm 1. First, we adopt the branch-and-bound method to find the top k-th point for every why-not weighting vector (lines 1-12). Then, we use the interior-point quadratic programming algorithm **QuadProg** [38] to get the optimal refined query point q' (lines 13-14). In particular, **QuadProg** iteratively finds an approximate Newton direction associated with the Karush-Kuhn-Tucker system of equations which characterizes a solution of the logarithmic barrier function problem. Totally, **QuadProg** finds an optimal solution in $O(d \times L)$ iterations, where d is the dimensionality, and L denotes the size of a quadratic programming problem [38]. Specifically, $L = \lceil log(d^3 + 1)\rceil + \lceil log(\theta + 1)\rceil + \lceil log(\omega + 1)\rceil + \lceil log(d + n)\rceil$ with $\omega = max(f(\overrightarrow{w_1}, p_1)$, $f(\overrightarrow{w_2}, p_2)$, $\cdots$, $f(\overrightarrow{w_n}, p_n))$ and $\theta = max(q[1], q[2], \cdots, q[d])$. Moreover, each iteration involves $O(d^2)$ arithmetic operations. Hence, **QuadProg** solves problem in no more than $O(d^3 \times L)$ arithmetic operations. Assume $|RT|$ is the cardinality of R-tree, we present the time complexity of MQP-I in Theorem 3.1 below.

Algorithm 1 Modifying query point q (MQP-I)

Input: an R-tree RT on a set P of data points, a query point q, a parameter k, a why-not weighting vector set W_m
Output: q'
/*HP is a min-heap; Λ is a set storing the top k-th point for each why-not weighting vector; H and A are matrixes; c, b, lb, and ub are vectors. */

```
1: for each weighting vector w_i ∈ W_m do
2:     initialize the min-heap HP with all root entries of RT;
3:     count ← 0;
4:     while HP is not empty do
5:         de-heap the top entry e of HP;
6:         if e is a data point then
7:             count ← count + 1;
8:             if count = k then
9:                 add e to Λ; break
10:        else  //e is an intermediate (i.e., a non-leaf) node
11:            for each child entry e_i ∈ e do
12:                insert e_i into HP;
13: set H, A, c, b, lb, and ub by using W_m, Λ, and q;
14: q' ← QuadProg(H, A, c, b, lb, ub);
        // interior-point quadratic programming algorithm in [38]
15: return q';
```

Theorem 3.1 *The time complexity of MQP-I algorithm is* $O(\,|RT| \times |W_m| + d^3 \times L)$.

Proof MQP-I algorithm consists of two phases. The first phase is to find the top k-th point for each why-not weighting vector. In the worst case, it needs to traverse the whole R-tree $|W_m|$ times, whose time complexity is $O(|RT| \times |W_m|)$. The second phase is the quadratic programming, whose time complexity is $O(d^3 \times L)$. Therefore, the total time complexity of MQP-I is $O(|RT| \times |W_m| + d^3 \times L)$. □

3.3 Modifying W_m and k

Imagine that, if the computer q in Figure 1 has been put into production, changing attribute values might not be feasible. Fortunately, as pointed out by Carpenter and Nakamoto [10], consumer preferences could be actually influenced by proper marketing strategies, such as advertising, which is proved by the example of Wal-Mart [1]. Hence, alternatively, Apple can adopt proper marketing strategies to influence their customers to change their preferences, such that the new computer q re-appears in customers' wish list again. Moreover, some of the existing works [24] and [33] also answer the why-not questions via modifying the preferences. To this end, we develop the second solution to refine the original reverse top-k query by modifying the customers' preferences. Since customers' preferences are application-dependent and the reverse top-k query studied in this paper involves two types of customers' preferences, i.e., W_m and k, our second solution is to modify a why-not weighting

vector set W_m and a parameter k. In reality, the change of W_m can be achieved by proper marketing strategies as mentioned above and the modification of k can be achieved by controlling the information exposed to the users.

Firstly, we introduce the penalty model to quantify the total changes of W_m and k. We use ΔW_m and Δk to measure the cost of the modification of W_m and k respectively, as defined in Equation (5).

$$\begin{cases} \Delta k = max(0, k' - k) \\ \Delta W_m = \sum_{i=1}^{|W_m|} \sqrt{\sum_{j=0}^{d} (w_i[j] - w_i'[j])^2} \end{cases} \tag{5}$$

It is worth noting that, there is a possibility that the modified k' value may be smaller than the original k value. In this case, we set Δk to 0. For example, assume that $(W_m, k = 6)$ is modified to $(W_m', k' = 3)$. Since q belongs to the top-3 query result of every refined why-not weighting vector, it must also be in the corresponding top-6 query result. Consequently, it is unnecessary to change the value of original k. In other words, $k' > k$. Note that the condition $k' > k$ can not be used to avoid the exploration of the "invalid search space". It is because the weighting vectors whose rank of q is lower than k also can contribute to the final results. In addition, ΔW_m refers to the sum of every why-not weighting vector penalty. In a word, we utilize the sum of ΔW_m and Δk to capture the total change of customer preferences. Given the fact that the customers' tolerances to the changes of W_m and k are different, we utilize a non-negative parameter α (≤ 1) to capture customers' relative tolerance to the changes of k. Then, a normalized penalty model is defined in Equation (6). Note that, the larger the value of α is, the bigger the role that Δk plays in determining the penalty. Again Equation (6) is equivalent to Equation (1) for this solution as $Penalty(q') \equiv 0$ when $q' = q$.

$$Penalty(W_m{}', k') = \alpha \frac{\Delta k}{\Delta k_{max}} + (1 - \alpha) \frac{\Delta W_m}{(\Delta W_m)_{max}} \tag{6}$$

Here, Δk_{max} refers to the maximum value of Δk which is set to $(k'_{max} - k)$ with k'_{max} calculated by Lemma 3.5 below.

Lemma 3.5 *Given a set $R = \{r_1, r_2, \cdots, r_n\}$, where $r_i \in R$ is the actual ranking of a query point q under the corresponding why-not weighting vector $\overrightarrow{w_i} \in W_m$, then $k'_{max} = max(r_1, r_2, \cdots, r_n)$.*

Proof Assume that we have a refined W_m' and k' with $\Delta W_m' = 0$, the corresponding $k' = max(r_1, r_2, \cdots, r_n)$. Any other possible refined W_m'' and k'' with $\Delta W_m'' > 0$ must have its $k'' < max(r_1, r_2, \cdots, r_n)$ or it cannot be the optimal result. Consequently, $k'_{max} = max(r_1, r_2, \cdots, r_n)$, and the proof completes. □

As shown in Figure 1, the actual rankings of q under why-not weighting vectors $\overrightarrow{w_1}$ and $\overrightarrow{w_4}$ are 4 and 4 respectively, and thus, $k'_{max} = 4$.

Similarly, $(\Delta W_m)_{max}$ refers to the maximum value of (ΔW_m), and it has been proven in [24] that $\Delta \overrightarrow{w_i} \leq \sqrt{1 + \sum_{j=1}^{d} (w_i[j])^2}$. As $\Delta W_m = \sum_{i=1}^{|W_m|} (\Delta \overrightarrow{w_i}) \leq \sum_{i=1}^{|W_m|} \sqrt{1 + \sum_{j=1}^{d} (w_i[j])^2}$, we have $(\Delta W_m)_{max} = \sum_{i=1}^{|W_m|} \sqrt{1 + \sum_{j=1}^{d} (w_i[j])^2}$. Based on the above analysis, we re-form the normalized penalty model below.

$$Penalty(W_m{}', k') = \frac{\alpha \cdot max(0, k' - k)}{max(r_1, r_2, \cdots, r_n) - k} + \frac{(1 - \alpha) \cdot \sum_{i=1}^{|W_m|} \sqrt{\sum_{j=1}^{d} (w_i[j] - w_i'[j])^2}}{\sum_{i=1}^{|W_m|} \sqrt{1 + \sum_{j=1}^{d} (w_i[j])^2}} \tag{7}$$

Given the fact that customer preferences are application-dependent, Equation (7) provides a reasonable estimation of the differences between customer preferences in terms of the reverse top-k query. Based on Equation (7), we formally define the problem of modifying W_m and k as follows.

Definition 3.4 (**Modifying W_m and k**). Given a d-dimensional dataset P, a positive integer k, a query point q, and a why-not weighting vector set $W_m = \{\overrightarrow{w_1}, \overrightarrow{w_2}, \cdots, \overrightarrow{w_n}\}$ ($\forall \overrightarrow{w_i} \in W_m, q \notin TOPk(\overrightarrow{w_i})$), the modification of W_m and k is to find $W_m' = \{\overrightarrow{w_1'}, \overrightarrow{w_2'}, \cdots, \overrightarrow{w_n'}\}$ and k', such that (i) $\forall \overrightarrow{w_i'} \in W_m'$, $q \in TOPk'(\overrightarrow{w_i'})$; (ii) $\forall \overrightarrow{w_j} \in RTOPk(q)$, $\overrightarrow{w_j} \in RTOPk'(q)$, and (iii) the $Penalty(W_m', k')$ is minimized.

Take Figure 1 as an example again and assume that $\alpha = 0.5$ for simplicity. If we modify Kevin's and Julia's weighting vectors to $\overrightarrow{w_1'} = (0.18, 0.82)$ and $\overrightarrow{w_4'} = (0.75, 0.25)$ respectively, Kevin and Julia will appear in the reverse top-3 query result of q with *Penalty* = 0.121. Alternatively, we can modify k to k' = 4 and remain the weighting vectors unchanged, Kevin and Julia will also appear in the reverse top-4 query result of q with *Penalty* = 0.5. Based on Definition 3.4, the first modification is better. It is worth mentioning that since (i) $k' > k$ and (ii) the query point q is not changed, the modification of W_m and k doesn't influence the original query result and hence the condition $\forall \overrightarrow{w_j} \in RTOPk(q)$, $\overrightarrow{w_j} \in RTOPk'(q)$ is guaranteed.

Since the function $Penalty(W_m', k')$ is not differentiable when $k' = k$, it is impossible to use a gradient descent based method to compute (W_m', k') with minimal cost. Another straightforward way is to find the optimal (W_m', k') by evaluating all the candidates. Although the total number of candidate (W_m', k') is infinite in an infinite weighting vector space, it is certain that only tuples (W_m', k') satisfying Lemma 3.6 are the candidate tuples for the final result.

Lemma 3.6 *Given a why-not weighting vector set $W_m = \{\overrightarrow{w_1}, \overrightarrow{w_2}, \cdots, \overrightarrow{w_n}\}$, a refined $W_m' = \{\overrightarrow{w_1'}, \overrightarrow{w_2'}, \cdots, \overrightarrow{w_n'}\}$ and*

k', and a set $R' = \{r'_1, r'_2, \cdots, r'_n\}$ ($r'_i \in R'$ is the actual ranking of q under $\overrightarrow{w'_i} \in W'_m$), if a tuple (W'_m, k') is a candidate tuple, it holds that (i) $k' = max(r'_1, r'_2, \cdots, r'_n)$; and (ii) $\forall r'_i \in R'$ $(1 \leq i \leq n)$, there does not exist another weighting vector $\overrightarrow{w''_n}$ under which the ranking of q is r'_i and $|\overrightarrow{w''_i} - \overrightarrow{w_i}| < |\overrightarrow{w'_i} - \overrightarrow{w_i}|$.

Proof First, assume that the statement (i) is not valid, i.e., an answer tuple (W'_m, k') has $k' > max(r'_1, r'_2, \cdots, r'_n)$ or $k' < max(r'_1, r'_2, \cdots, r'_n)$. If $k' > max(r'_1, r'_2, \cdots, r'_n) = k''$, $Penalty(W'_m, k') > Penalty(W'_m, k'')$, and hence, it cannot be the optimal answer. If $k' < max(r'_1, r'_2, \cdots, r'_n)$, then $\exists \overrightarrow{w'_i} \in W'_m$, $q \notin TOPk'(\overrightarrow{w'_i})$, which contradicts with the statement (i) of Definition 3.4. Thus, our assumption is invalid and the statement (i) is true. Second, assume that statement (ii) is invalid, i.e., for an answer tuple (W'_m, k'), there is a $\overrightarrow{w''_i}$ with $|\overrightarrow{w''_i} - \overrightarrow{w_i}| < |\overrightarrow{w'_i} - \overrightarrow{w_i}|$ and meanwhile the actual ranking of q under $\overrightarrow{w''_i}$ being r'_i. If $|\overrightarrow{w''_i} - \overrightarrow{w_i}| < |\overrightarrow{w'_i} - \overrightarrow{w_i}|$, then $(\Delta W'_m)$ is not minimal. Therefore, $Penalty(W'_m, k')$ is not minimum, and (W'_m, k') cannot be the final result, which contradicts with the condition of Lemma 3.6. Hence, our assumption is invalid, and the statement (ii) must be true. The proof completes. □

According to Lemma 3.6, the qualified candidates W'_m and k' interact with each other, which can facilitate their search process. If we fix one parameter, the other one can be computed accordingly. Since the weighting vector space for W'_m is infinite, it is impossible to fix W'_m. Consequently, we try to fix k'. Given a specified dataset and a query point, the range of k' can be determined by the number of the points *incomparable* with q and the number of the points *dominating* q. Specifically, if a point p_1 dominates another point p_2, it holds that, for every $i \in [1, d]$, $p_1[i] \leq p_2[i]$ and there exists at least one $j \in [1, d]$, $p_1[j] < p_2[j]$. If p_1 neither dominates p_2 nor is dominated by p_2, we say that p_1 is *incomparable* with p_2. For instance, in Figure **??**, the query point q is dominated by p_1, and it is incomparable with p_3. Given a d-dimensional dataset P and a query point q, we can find all the points that dominate q and all the points that are incomparable with q, preserved in sets D and I respectively. Thus, a possible ranking of q could be $R_q = \{(|D|+1), (|D|+2), \cdots, (|D|+|I|+1)\}$, which is also the range of k'.

If we fix the query point q's ranking r_i with $r_i \in R_q$, for every why-not weighting vector $\overrightarrow{w_i}$, we can find its corresponding $\overrightarrow{w'_i}$ with the minimal $|\overrightarrow{w'_i} - \overrightarrow{w''_i}|$ by using the quadratic programming. After finding all these weighting vectors for each $r_i \in R_q$, we can get the optimal W'_m and k'. However, for a single why-not weighting vector, if all rankings of q have to be considered, there are in total $2^{|I|}$ quadratic programming problems in the worst case, as proved in [24]. Totally, for the entire why-not weighting vector set W_m, it needs to solve $|W_m| \times 2^{|I|}$ quadratic programming problems, which is very costly. Nonetheless, if we can find $\overrightarrow{w''_i}$ that approximates the minimum $|\overrightarrow{w'_i} - \overrightarrow{w_i}|$, it would save the search significantly even though it is not the exact answer. Hence, in the second solution, we trade the quality of the answer with the running time, and propose a sampling based algorithm, which finds an approximate optimal answer.

The basic idea of the sampling-based algorithm is as follows. We first sample a certain number of weighting vectors from the sample space, and then, we use these sample weighting vectors to find (W'_m, k') with minimum penalty. In particular, there are three issues we have to address: (i) how to get high quality sample weighting vectors; (ii) how to decide a proper sample size S_W; and (iii) how to use the sample weighting vectors to obtain (W'_m, k') with minimum penalty. Next, we discuss the three issues in detail.

First, how can we get the high quality sample weighting vectors as the quality of sample weighting vectors impacts that of the final answer? It is worth noting that, the full d-dimensional weighting vector space is the hyperplane $\sum_{i=1}^{d} w[i] = 1$ in which $w[i] \geq 0$ $(1 \leq i \leq d)$. However, if we take the whole weighting vector space as a sample space, the penalty of the modified why-not weighting vectors may be big. Hence, we have to narrow down the sample space. According to the statement (ii) of Lemma 3.6, for a fixed k', the modified weighting vector $\overrightarrow{w'_i} \in W'_m$ has the minimum $|\overrightarrow{w'_i} - \overrightarrow{w_i}|$ w.r.t. $\forall \overrightarrow{w_i} \in W_m$. Thus, we should sample the weighting vector that can approximate to the minimum $|\overrightarrow{w'_i} - \overrightarrow{w_i}|$. As proved in [24], for a fixed k', the weighting vector $\overrightarrow{w'_i}$, which has the minimum $|\overrightarrow{w'_i} - \overrightarrow{w_i}|$ w.r.t. $\overrightarrow{w_i}$, exists in one of the hyperplanes formed by I and q. Specifically, for a point $p \in I$, the hyperplane formed by p and q is: $(\overrightarrow{p} - \overrightarrow{q}) \cdot \overrightarrow{w} = 0$. Therefore, all the hyperplanes intersecting with $\sum_{i=1}^{d} w[i] = 1$ constitute the sample space.

Second, how shall we decide an appropriate sample size S_W? It is well known that, the bigger the sample size, the higher the quality of the result. Nonetheless, it is impossible to sample an infinite number of weighting vectors since a larger sample size increases the cost. In this paper, we employ a general equation $1 - (1 - T\%)^S \geq Pr$ to help users decide the sample size as with [24]. Specifically, if we hope the probability of at least one refined query to be the best-$T\%$ refined query is no smaller than a certain threshold Pr, then the sample size should be $S \geq \log(1 - Pr)/\log(1 - T\%)$. In this paper, we take the sample size S_W as a user specified parameter, which can better meet users' requirements. Alternatively, it is also a good and useful solution to consider a time-based heuristic that takes an input time threshold to compute a good solution within the threshold. We would study this in our future works.

Third, how to use the sample weighting vectors to get (W'_m, k') with the minimal penalty? There are two possible solutions. The first solution is, for every why-not weighting vector $\overrightarrow{w_i} \in W_m$, to find a sample weighting vector $\overrightarrow{w_s} \in W_s$ with minimum $|\overrightarrow{w_i} - \overrightarrow{w_s}|$, and then replace $\overrightarrow{w_i} \in W_i$ with $\overrightarrow{w_s} \in W_s$. After replacing all why-not weighting vectors, we can obtain a refined W'_m. The corresponding k' can be computed according to statement (i) of Lemma 3.6. The second method is to select randomly $|W_m|$ sample weighting vectors to replace W_m, and we then can get a candidate refined tuple (W'_m, k'). The optimal (W'_m, k') can be found from the entire candidates. For the first solution, we can ensure that the refined W'_m is optimal, while the total penalty of W'_m and k' may not be the minimum. For the second solution, if all candidate tuples are considered, there are in total $|S|^{|W_m|}$ instances, whose computation cost could be very expensive. Thus, we present an efficient approach that only examines up to $|S|$ instances, supported by Lemma 3.7.

Lemma 3.7 *Given a candidate tuple (W'_m, k'), and a weighting vector $\overrightarrow{w}$ (the ranking of q under $\overrightarrow{w}$ is bigger than k'), if $\exists \overrightarrow{w'_i} \in W'_m$ such that $|\overrightarrow{w_i} - \overrightarrow{w}| < |\overrightarrow{w'_i} - \overrightarrow{w_i}|$ ($\overrightarrow{w_i}$ is the original why-not weighting vector w.r.t. $\overrightarrow{w'_i}$), there exist another candidate tuple (W''_m, k''), where W''_m contains $\overrightarrow{w}$.*

Proof If $\exists \overrightarrow{w'_i} \in W'_m$ such that $|\overrightarrow{w_i} - \overrightarrow{w}| < |\overrightarrow{w'_i} - \overrightarrow{w_i}|$, we can obtain a new W''_m from W'_m by replacing all these $\overrightarrow{w'_i}$ with $\overrightarrow{w_i}$, and its corresponding k''. Although $k'' > k'$, $\Delta W''_m < \Delta W'_m$. Thus, (W''_m, k'') is a candidate tuple for the final result including $\overrightarrow{w}$. □

According to Lemma 3.7, we can get the optimal refined W_m and k by examining the sample weighting vectors one by one. To be more specific, for every sample weighting vector, we compute its corresponding ranking of q. We also sort the whole sample weighting vectors in ascending order of the ranking of q. Next, we initialize a candidate tuple (W'_m, k') to the first sample weighting vector and its corresponding ranking of q. For each remaining sample weighting vector $\overrightarrow{s}$, we examine whether it can contribute to the final result. Based on Lemma 3.7, if $\exists \overrightarrow{w'_i} \in W'_m$, $|\overrightarrow{w_i} - \overrightarrow{s}| < |\overrightarrow{w'_i} - \overrightarrow{w_i}|$, we replace all such $\overrightarrow{w_i}$ with $\overrightarrow{s}$ and get a new (W''_m, k''). Thereafter, we obtain some candidate tuples (W'_m, k'), and the one with the minimal penalty is the final answer.

Based on the above discussion, we propose our sampling based algorithm called MWK-I to **m**odify $\boldsymbol{W_m}$ and $\boldsymbol{k}$, with its pseudo-code shown in Algorithm 2. Initially, MWK-I invokes a function **FindIncom** that follows the branch-and-bound traversal to form the set I of points incomparable with q and the set D of points dominating q (line 2). It traverses the nodes of R-tree based on breadth-first order. If a node is dominated by q, it is discarded; otherwise, it is expanded.

Algorithm 2 Modifying W_m and k (MWK-I)

Input: an R-tree RT on a set P of data points, a query point q, a parameter k, a why-not weighting vector set W_m, a sample size $|S|$
Output: W'_m and k'
/*HP is a min-heap; D is the set of points dominating q; I is the set of points incomparable with q; k'_{max} is the maximal value of k'; S is the set of sample weighting vectors; CW is a candidate W'_m; P_{min} is the penalty of current optimal candidates W'_m and k'. */

```
1: k'_max ← ∞, HP ← ∅
2: FindIncom(RT, q, HP, D, I)
3: sample |S| weighting vectors from the hyperplanes formed by I
   and q, maintained by S
4: for each weighting vector s_i ∈ S do
5:     compute the ranking rs_i of q based on D and I
6: sort vectors in S based on ascending order of rs_i values
7: for each weighting vector w_i ∈ W_m do
8:     compute the ranking r_i of q based on D and I
9: k'_max ← max_{∀w_i∈W_m}(r_i)
10: CW ← the first sample weighting vector in S
11: W'_m ← W_m, k' ← k'_max, P_min ← Penalty(W'_m, k')
12: for each remaining s_i ∈ S and its corresponding rs_i do
13:     if k'_max < rs_i then break
14:     for each (cw_i, w_i) ∈ CW × W_m do //updates CW using s_i
15:         if |s_i − w_i| < |cw_i − c_i| then
16:             cw_i ← s_i
17:         if CW is updated then
18:             if (pe ← Penalty(CW, rs_i)) < P_min then
19:                 W'_m ← CW, k' ← max(k, rs_i), P_min ← pe
20: return W'_m and k'
```

The set D preserves all the points dominating q, and the set I stores all the points that are incomparable to q. Then, the algorithm samples $|S|$ weighting vectors from the hyperplanes formed by I and q, maintained by S (line 3). For every sample weighting vector $\overrightarrow{s_i}$, it computes the ranking rs_i of q, and then sorts vectors $\overrightarrow{s_i}$ in S based on ascending order of rs_i (lines 4-6). Thereafter, the maximum value of k' is obtained (lines 7-9) for pruning later. Next, MWK-I examines, for each sample weighting vector $\overrightarrow{s_i}$, whether $\overrightarrow{s_i}$ can contribute to the final result based on Lemma 3.7, and then gets the tuple (W'_m, k') with the minimum penalty (lines 12-19). Theorem 3.2 presents the time complexity of MWK-I.

Theorem 3.2 *The time complexity of MWK-I algorithm is $O(|RT| + |S| \times |W_m|)$, with $|S|$ the cardinality of a sample weighting vector set and $|W_m|$ the cardinality of a why-not weighting vector set.*

Proof The time complexity of MWK-I is mainly determined by the computation of D and I as well as using the sample weighting vectors to get the optimal result. In the worst case, **FindIncom** has to traverse the whole R-tree RT to form sets D and I, with time complexity $O(|RT|)$. In addition, the time complexity of using the sample weighting vectors to get the optimal results is determined by the cardinality of the why-not weighting vector set and the sample size, i.e.,

$O(|S| \times |W_m|)$. Thus, the total time complexity of MWK-I is $O(|RT| + |S| \times |W_m|)$, and the proof completes. □

3.4 Modifying q, W_m, and k

The two solutions proposed above can return the refined query with the minimum penalty, but there might be some cases where the returned penalty is still beyond the manufacturers' or customers' limits of acceptability. Therefore, manufacturers (e.g., Apple) might want to reach a compromise between what customers want and they can offer. In other words, both manufacturers and customers should change their preferences to narrow down the gap, which can be addressed through bargaining, e.g., manufacturers and customers collaborate in finding an optimal solution [23]. Hence, in this subsection, we propose the third solution to refine the reverse top-k query by modifying both manufacturers' product (i.e., q) and customers' preferences (i.e., W_m and k).

First, we present the penalty model to quantify the modifications of q, W_m, and k. As defined in Equation (1), penalty $Penalty(q', {W_m}', k')$ considers both $Penalty(q')$ defined in Equation (3) and $Penalty({W_m}', k')$ defined in Equation (7). Weighting parameter γ is introduced to capture a user's relative tolerance to the change of q, as compared with that of (W_m, k). Both $Penalty(q')$ and $Penalty({W_m}', k')$ have the values in the range of (0,1], and thus, there is no need to normalize them and $Penalty(q', {W_m}', k')$ is also in the range of (0,1].

Note that, similar penalty functions have been used in industry, e.g., *joint outcome* that is the sum score of the manufacturers and the customers for the final agreement is used to measure the bargaining solution [23]. This further justifies that our penalty function is practical. For example, in Figure 1, if we modify $q, \overrightarrow{w_1}$, and $\overrightarrow{w_4}$ to $q'(3.8, 3.8)$, $\overrightarrow{w_1'}(0.135, 0.865)$, and $\overrightarrow{w_4'}(0.8, 0.2)$ respectively, $\overrightarrow{w_1'}$ and $\overrightarrow{w_4'}$ become the reverse top-3 query result of q' with $penalty = 0.06$ ($\gamma = 0.5$). Based on Equation (1), we formulate the problem of modifying q, W_m, and k as follows.

Definition 3.5 (**Modifying q, W_m, and k**). Given a d-dimensional dataset P, a positive integer k, a query point q, and a why-not weighting vector set $W_m = \{\overrightarrow{w_1}, \overrightarrow{w_2}, \cdots, \overrightarrow{w_n}\}$ with $\forall \overrightarrow{w_i} \in W_m$, $q \notin TOPk(\overrightarrow{w_i})$, the modification of q, W_m, and k is to find q', $W_m' = \{\overrightarrow{w_1'}, \overrightarrow{w_2'}, \cdots, \overrightarrow{w_n'}\}$, and k', such that (i) $\forall \overrightarrow{w_i'} \in W_m'$, $q' \in TOPk'(\overrightarrow{w_i'})$; (ii) $\forall \overrightarrow{w_j} \in RTOPk(q), \overrightarrow{w_j} \in RTOPk'(q')$, and (iii) the $Penalty(q', W_m', k')$ is minimized.

For the third solution, we need to get a new tuple (q', W_m', k') whose penalty is minimized. There are two potential approaches. The first one is to locate (W_m', k') first and then determine the corresponding q'. The second method is to find the candidate q' and then the corresponding (W_m', k'). From MWK-I algorithm presented in Section 3.3, we know that the optimal (W_m', k') can be obtained only when the query point q is fixed. This is because the set I used for the sampling is dependent on q. Thus, we adopt the second method in our third solution. Since there are infinite candidate query points, it is impossible to evaluate all the potential candidates (q', W_m', k'). Hence, we again employ the sampling technique to modify q, W_m, and k. The basic idea is as follows. We first sample a set of candidate query points. For every sample query point q', we use MWK-I algorithm to find the optimal (W_m', k'). Finally, the tuple (q', W_m', k') with the smallest penalty is returned. In the sequel, we explain (i) how to sample query points, and (ii) how to invoke MWK-I repeatedly.

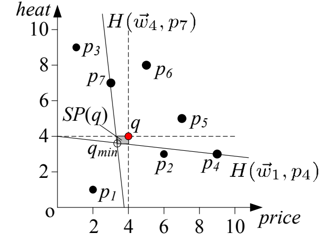


Fig. 5 Example of the sample space of q

For the first issue, we need to find out the sample space of q and its sample size S_q. Recall that, according to Definition 3.2, if the query point falls into the safe region of q, the why-not weighting vectors must appear in the reverse top-k query result. Thus, if we sample a query point (e.g., q') from the safe region, there is no need to modify (W_m and k), and the penalty of (q', W_m, k) will not be smaller than that of (q_{min}, W_m, k), in which q_{min} is the result returned by the first solution (i.e., modifying q presented in Section 3.2). Therefore, (q', W_m, k) cannot be the final result, and we should sample the query point from the space that is out of the safe region. Furthermore, if we sample a query point (e.g., q'') out of the safe region, the corresponding refined tuple (q'', W_m'', k'') must satisfy the condition $\Delta(W_m'', k'') > 0$. The tuple (q'', W_m'', k'') is the optimal result only when $|q'' - q| < |q_{min} - q|$; otherwise, $Penalty(q'', W_m'', k'') > Penalty(q_{min}, W_m, k)$, and hence, it cannot be the final result. Therefore, we know that only the query points falling within the range $[q_{min}, q]$ could be qualified sample query points. Thus, the sample space of q, denoted as $SP(q)$, is $\{q' | q_{min} < q' < q\}$. Take Figure 5 as an example, q_{min} is returned by the first solution (i.e., modifying q presented in Section 3.2), and the shaded area formed by q_{min} and q is the sample space of q. In addition, we also suppose the sample size S_q of query points is specified by users.

The second issue is the iterative call of MWK-I algorithm. Recall that, the first step of MWK-I algorithm is to find the points that are incomparable with the query point. Our third solution needs to invoke MWK-I for each sample query point to find the candidate (q', W_m', k'), which requires traversing the R-tree $|Q|$ times with high cost. Thus,

Algorithm 3 Modifying q, W_m, and k (MQWK-I)

Input: an R-tree RT on a set P of data points, a query point q, a parameter k, a why-not weighting vector set W_m, sample sizes $|S|$ and $|Q|$ for the sample weighting vector and the sample query point
Output: q', W'_m, q'
/*Q is a set of sample query points; MinPenalty is the penalty of the current optimal candidates q', W'_m, and k'. */
1: $Q \leftarrow \emptyset$, $MinPenalty \leftarrow \infty$
2: $q_{min} \leftarrow$ **MQP-I**(RT, q, k, W_m)
3: $Q \leftarrow |Q|$ query points sampled from the space determined by q_{min} and q
4: **for** each each query point $q_i \in Q$ **do**
5: $(W''_m, k'') \leftarrow$ **MWK-I**$(RT, q_i, k, W_m, |S|)$
6: **if** $Penalty(q_i, W''_m, k) < MinPenalty$ **then**
7: $q' \leftarrow q_i$, $W'_m \leftarrow W''_m$, $k' \leftarrow k''$
8: $MinPenalty \leftarrow Penalty(q', W'_m, k')$
9: **return** q', W'_m, k'

we employ the reuse technique to avoid repeated traversal of the R-tree. To this end, we use a heap to store the visited nodes for reusing unless they are expanded. Correspondingly, the **FindIncom** function needs to be revised as well. In particular, when **FindIncom** encounters a data point or an intermediate node dominated by q, it has to be preserved for the reuse later.

Based on the above discussion, we propose the algorithm called MQWK-I to **m**odify $\boldsymbol{q}$, $\boldsymbol{W}_m$, and $\boldsymbol{k}$. The pseudocode of the algorithm is listed in Algorithm 3. First of all, MQWK-I invokes MQP-I algorithm to get the minimal q_{min} (line 2). Next, it samples $|Q|$ query points from the sample space determined by q_{min} and q, preserved by the set Q (line 3). Then, for every sample query point q', MQWK-I derives the corresponding optimal (W'_m, k') using MWK-I algorithm (line 5). Finally, the tuple (q', W'_m, k') with the minimum penalty is returned (line 9). Note that MQWK-I doesn't lose the existing reverse top-k query result. It is because $q' < q$ and MWK-I algorithm also doesn't lose the existing reverse top-k query result. The time complexity of MQWK-I algorithm is presented in Theorem 3.3.

Theorem 3.3 *The time complexity of MQWK-I algorithm is* $O(|RT| \times |W_m| + d^3 \times L + |Q| \times (|RT| + |S| \times |W_m|))$.

Proof The time complexity of MQWK-I consists of the computation of q_{min} and the iterative call of MWK-I. The time complexity of q_{min} computation is equal to that of MQP-I, i.e., $O(|RT| \times |W_m| + d^3 \times L)$. The iterative call MWK-I takes $O(|Q| \times (|RT| + |S| \times |W_m|))$. Therefore, the total time complexity of MQWK-I is $O(|RT| \times |W_m| + d^3 \times L + |Q| \times (|RT| + |S| \times |W_m|))$. The proof completes. □

4 Answering Why Questions

In this section, we extend the framework WQRTQ to answer why questions on reverse top-k queries. Similarly, we first give an overview of the framework and then detail the algorithms.

4.1 Framework Overview

We extend WQRTQ shown in Figure 3 to answer why questions on reverse top-k queries. Specifically, it takes as inputs an original reverse top-k query and the corresponding why weighting vector set W_p, and returns the refined reverse top-k query with minimum penalty as the result, by using one of the three solutions proposed. Specifically, the presented three solutions include:

(1) **Modifying $\boldsymbol{q}$**. The first solution is to change a query point q into q' such that the why weighting vectors are excluded from the reverse top-k result of q'. For this solution, we introduce the concept of the *invalid region* of q, within which if the query point falls, the why weighting vectors do not appear in the reverse top-k query result. Because of the complexity of the construction of invalid regions, we also employ quadratic programming to find optimal refined q. However, the inputs of the quadratic programming are different from the first solution of why-not questions.

(2) **Modifying $\boldsymbol{W_p}$ and $\boldsymbol{k}$**. The second solution is to modify a why weighting vector set W_p and a parameter k into W'_p and k' respectively. After the modification, W'_p is excluded from the result of the reverse top-k' query result of q. Our second solution is also a sampling-based method, which is to be detailed in Section 4.3. Note that the sampling technique used to support why question is very different from that used to support why-not questions

(3) **Modifying $\boldsymbol{q}$, $\boldsymbol{W_p}$, and $\boldsymbol{k}$**. Our third solution is to modify a query point q, a why weighting vector set W_p, and a parameter k, such that the modified weighting vector set W'_p is excluded from the reverse top-k' query result of q'. The tuple (q', W'_p, k') with the smallest penalty is returned by using the techniques of quadratic programming, sampling method, and reuse. Note that, modifying $\boldsymbol{q}$, $\boldsymbol{W_p}$, and $\boldsymbol{k}$ to support why questions is different from that to support why-not questions, because they have different sample spaces of q, and they invoke modifying $\boldsymbol{W_p}$, and $\boldsymbol{k}$ that are different too.

Note that, all the above three approaches may retain the existing result set, although it is not guaranteed. We would like to study how to retain the original results before the modification for why questions on RTOPk queries in our future work.

4.2 Modifying q

If the computer designed for professional developers appears in the wish list of many high school/college students, there must be some mismatches between the target market

and the real market, meaning that the design of the computer might not be proper. As a solution, Apple might want to change certain specifications/settings of the computer so that it could meet the requirements from its target users better. After modifying, although the modified computer may be less popular for the non-target users, it will be more popular among the target users. Hence, it can help the manufacturer to design more appropriate products that meet the requirements from their target users better and hence they are able to attract more target users. Accordingly, our first solution is to modify the query point q so that unexpected data points do not appear in the result set.

Definition 4.1 (**Modifying q**). Given a d-dimensional data set P, a positive integer k, a query point q, and a why weighting vector set W_p with $\forall \overrightarrow{w_i} \in W_p$, $q \in TOPk(\overrightarrow{w_i})$, the modification of a query point q is to find q' such that (i)$\forall \overrightarrow{w_i} \in W_p$, $q' \notin TOPk(\overrightarrow{w_i})$; and (ii) the penalty of q', as defined in Equation (3), is minimized.

Note that we still use Equation (3) to quantify the distortion of the product after modification. The smaller the penalty is, the better the solution is as manufacturers prefer a smaller modification, as justified in [39].

If we exclude a why weighting vector $\overrightarrow{w_i} \in W_p$ from the reverse top-k query result, the rank of q w.r.t $\overrightarrow{w_i}$ must be bigger than k. Recall that the scoring function is monotonic, and a smaller scoring value is ranked higher. If for $\forall i \in [1, d]$, the condition $q'[i] \leq q[i]$ is satisfied, the score of q' must be smaller than that of q, indicating that $\overrightarrow{w_i}$ cannot be excluded from the reverse top-k query result. For instance, assume that q(4, 4) in Figure 1 is modified to q'(3, 3), and then the why weighting vector $\overrightarrow{w_3} \in W_p$ will still be in the reverse top-3 query result of q'. This observation implies that, for the modified q', there must be at least one dimension i such that $q'[i]$ is bigger than $q[i]$. Therefore, the search space for q' can be shrunk to $\Omega - \{q' | 0 \leq q' \leq q\}$, where Ω represents the whole data space. In order to find a new p' satisfying the conditions listed in Definition 4.1, we introduce a new concept, termed as *invalid region*.

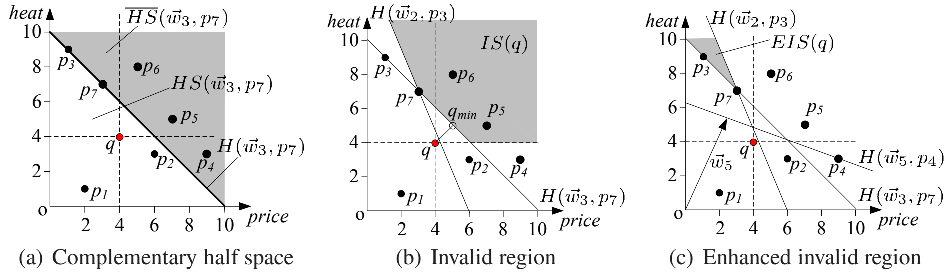
Definition 4.2 (**Invalid Region**). Given a d-dimensional dataset P, a positive integer k, a query point q, and a why weighting vector set W_p, a region in the data space is defined as the invalid region of q, denoted as $IR(q)$, such that $\forall q' \in IR(q)$ and $\forall \overrightarrow{w_i} \in W_p$, $q' \notin TOPk(\overrightarrow{w_i})$.

If q is modified to q' by moving the query point q anywhere within $IR(q)$ w.r.t. a why weighting vector set W_p, all the why weighting vectors in W_p will be *excluded* from a specified reverse top-k query result. If we can obtain such $IR(q)$, the answer of our first solution for the why questions will be the point in $IR(q)$ that is closest to q. In order to facilitate the formation of invalid region, we introduce the concept of *complementary half space*. Specifically, a complementary half space w.r.t. $\overrightarrow{w}$ and p, denoted as $\overline{HS}(\overrightarrow{w}, p)$, is formed by all the points lying above the hyperplane $H(\overrightarrow{w}, p)$ such that $\forall p' \in \overline{HS}(\overrightarrow{w}, p)$, $f(\overrightarrow{w}, p') > f(\overrightarrow{w}, p)$. For example, in Figure 6(a), the shaded area is the complementary half space $\overline{HS}(\overrightarrow{w_3}, p_7)$ formed by $\overrightarrow{w_3}$ and p_7. If we move the query point q within the complementary half space formed by a why weighting vector and its corresponding the k-th point, the why weighting vector will be absent from the reverse top-k query result of q. With the help of complementary half space, the invalid region of q can be formed easily. Given a why weighting vector set $W_p = \{\overrightarrow{w_1}, \overrightarrow{w_2}, \cdots, \overrightarrow{w_m}\}$, and a set $\Lambda = \{p_1, p_2, \cdots, p_m\}$ of points ($\forall p_i \in \Lambda$ is the k-th point w.r.t. its corresponding why weighting vector $\overrightarrow{w_i} \in W_p$), the invalid region of a query point q, i.e., $IR(q)$, is the common area covered by all the complementary half spaces formed by $\overrightarrow{w_i}$ and p_i, formally, $IR(q) = \cap_{1 \leq i \leq m} \overline{HS}(\overrightarrow{w_i}, p_i)$. Figure 6(b) illustrates an example of the construction of q's invalid region, in which we employ the data set depicted in Figure 1 as P and suppose $W_p = \{\overrightarrow{w_2}, \overrightarrow{w_3}\}$. The corresponding 3-rd points with smallest scores for $\overrightarrow{w_2}$ and $\overrightarrow{w_3}$ are p_3 and p_7, respectively. The shaded area in Figure 6(b) represents the final invalid region of q.

After getting the invalid region of q by computing the intersection of all the complementary half spaces, we can find the optimal query point q' with the minimum cost w.r.t. q. Take Figure 6(b) as an example again. Point q_{min} is the desirable refined query point. However, we find that some query points in the invalid region of q may invalidate certain existing reverse top-k query results, which is undesirable in real applications. Back to the Figure 6(b). Assume that W_p contains only one why weighting vector $\overrightarrow{w_3}$, and $\overrightarrow{w_2}$ is an existing reverse top-k query result. Then, $IR(q) = \overline{HS}(\overrightarrow{w_3}, p_7)$ and q_{min} remains as the desirable refined query point. However, q_{min} is above the hyperplane $H(\overrightarrow{w_2}, p_3)$ (p_3 is the 3-rd point for $\overrightarrow{w_2}$), meaning $f(\overrightarrow{w_2}, p_3) \leq f(\overrightarrow{w_2}, q_{min})$. Therefore, if the query point q is changed to q_{min}, the weighting vector $\overrightarrow{w_2}$ will also be excluded from the result, and the reverse top-k query result of q_{min} will be empty. In order to tackle this issue, we propose the enhanced invalid region of q, as defined in Definition 4.3, to exclude only unexpected results W_p but retain all other existing results Γ. We also derive Lemma 4.1 to facilitate the formation of the enhanced invalid region of q.

Definition 4.3 (**Enhanced Invalid Region**). Given a d-dimensional dataset P, a positive integer k, a query point q, a why weighting vector set W_p, and a sub-result set Γ of a reverse top-k query that is still desirable (i.e., $W_p \cap \Gamma = \varnothing$ and $\forall \overrightarrow{w_i} \in \Gamma \cup W_p$, $q \in TOPk(\overrightarrow{w_i})$), the enhanced invalid region of q, denoted as $EIR(q)$, refers to a region in the data space such that $\forall q' \in EIR(q)$, (i) $\forall \overrightarrow{w_i} \in W_p$, $q' \notin TOPk(\overrightarrow{w_i})$; and (ii) $\forall \overrightarrow{w_j} \in \Gamma$, $q' \in TOPk(\overrightarrow{w_j})$.

Lemma 4.1 *Given a why weighting vector set $W_p = \{\overrightarrow{w_1}, \overrightarrow{w_2}, \cdots, \overrightarrow{w_m}\}$, and its corresponding data point set $\Lambda_w = \{p_{w1}, p_{w2}, \cdots, p_{wm}\}$ that preserves the top k-th point w.r.t.*



(a) Complementary half space (b) Invalid region (c) Enhanced invalid region

Fig. 6 Example of modifying q for why questions

$\overrightarrow{w_i} \in W_p$, let area A_p refer to the common area covered by all the complementary half spaces formed by $\overrightarrow{w_i} \in W_p$ and $p_{wi} \in \Lambda_w$, i.e., $A_p = \cap_{1 \le i \le m} \overline{HS}(\overrightarrow{w_i}, p_{wi})$. Given a sub-result set $\Gamma = \{\overrightarrow{v_1}, \overrightarrow{v_2}, \cdots, \overrightarrow{v_l}\}$ of a reverse top-k query that is still desirable (i.e., $W_p \cap \Gamma = \varnothing$ and $\forall \overrightarrow{w_i} \in \Gamma \cup W_p$, $q \in TOPk(\overrightarrow{w_i})$) and its corresponding data point set $\Lambda_\Gamma = \{p_{v1}, p_{v2}, \cdots, p_{vl}\}$ that preserves the top k-th point w.r.t. $\overrightarrow{w_j} \in \Gamma$, let area A_Γ refer to the common area covered by all the half spaces formed by $\overrightarrow{v_j} \in \Gamma$ and $p_{vj} \in \Lambda_\Gamma$, i.e., $A_\Gamma = \cap_{1 \le j \le l} HS(\overrightarrow{v_j}, p_{vj})$. Then, we have $EIR(q) = A_p \cap A_\Gamma$.

Proof The proof is straightforward and hence omitted. □

For instance, in Figure 6(c), assume that we have $W_p = \{\overrightarrow{w_3}\}$ and $\Gamma = \{\overrightarrow{w_2}\}$. Then, the shaded area in Figure 6(c) is the enhanced invalid region of q, and the point p_7 is the optimal refined query point since it is the closest point in $EIR(q)$ to q. Obviously, $EIR(q)$ provides an easy solution to locate the optimal refined query point. Nonetheless, we also notice that $EIR(q)$ could be empty. Take Figure 6(c) as an example again. If we have $W_p = \{\overrightarrow{w_3}\}$ and $\Gamma = \{\overrightarrow{w_2}, \overrightarrow{w_5}\}$, $EIR(q) = \overline{HS}(\overrightarrow{w_3}, p_7) \cap HS(\overrightarrow{w_2}, p_3) \cap HS(\overrightarrow{w_5}, p_4) = \varnothing$. Consequently, in the case where $EIR(q) = \varnothing$, we still employ the invalid region of q to find the optimal refined query point. In addition, it is worth mentioning that, for the monochromatic reverse top-k query, its result is a set of intervals, which contains infinite weighting vectors. Hence, it is impossible to form the enhanced invalid region of q. In this case, we also utilize the invalid region of q to find the optimal modified query point.

Based on the above discussion, we understand that modifying q can be achieved as soon as we form $IR(q)$ or $EIR(q)$. Unfortunately, $IR(q)$ and $EIR(q)$ are convex polygons bounded by hyperplanes with high computation cost and poor scalability [3]. Therefore, we employ the quadratic programming algorithm to find the optimal modified query point. The initialization of the parameters for the quadratic programming algorithm is similar as that for why-not questions presented in Section 3.2, and thus, we skip details to avoid redundancy and to save space.

In summary, we propose an algorithm called MQP-II to modify the query point q for why questions on reverse top-k queries, with its pseudo-code listed in Algorithm 4.

Algorithm 4 Modifying query point q (MQP-II)

```
Input: an R-tree RT on a set P of data points, a query point q,
       a parameter k, a why weighting vector set W_p, an existing
       reverse top-k results set Γ
Output: q'
/*Λ_w and Λ_Γ are two sets storing the top k-th point for why weigh-
  ting vectors and existing reverse top-k results. */
1:  for each weighting vector w_i ∈ W_p do
2:      find the top k-th point and add it to Λ_w
3:  for each weighting vector v_j ∈ Γ do
4:      find the top k-th point and add it to Λ_Γ
5:  set H, A, c, b, lb, and ub based on W_p, Λ_w, Γ, Λ_Γ, and q
6:  q' ← QuadProg(H, A, c, b, lb, ub)   // finding q' in EIR(q)
7:  if q' = ∅ then   // EIR(q) = ∅
8:      set H, A, c, b, lb, and ub based on W_p, Λ_w, and q
9:      q' ← QuadProg(H, A, c, b, lb, ub)   // finding q' in IR(q)
10: return q'
```

First, we adopt the branch-and-bound method to find the k-th point for every why weighting vector and the existing reverse top-k results (lines 1-4). Specifically, the branch-and-bound method is the same as that (i.e., lines 2-12) in Algorithm 1. Then, we use the interior-point quadratic programming algorithm **QuadProg** [38] to get the optimal refined query point q' (lines 5-9). Otherwise (i.e., $EIR(q) \neq \varnothing$), we get the optimal refined query point in $EIR(q)$ (lines 5-6). If $EIR(q) = \varnothing$, we get the optimal refined query point in $IR(q)$ (lines 7-9).

Theorem 4.1 *The time complexity of MQP-II algorithm is $O(|RT| \times (|W_p| + |\Gamma|) + 2 \times d^3 \times L)$, in which L is the same as the one in Theorem 3.1.*

Proof MQP-II algorithm consists of two phases. The first phase is to find the k-th point for every why weighting vector and the existing reverse top-k results. In the worst case, it needs to traverse the whole R-tree $(|W_p| + |\Gamma|)$ times, whose time complexity is $O(|RT| \times (|W_p| + |\Gamma|))$. The second phase is the quadratic programming. In the worst case, it needs to invoke the quadratic programming algorithm twice. Since the time complexity of quadratic programming algorithm is $O(d^3 \times L)$ [38], the time complexity of the second phases is $O(2 \times d^3 \times L)$. Therefore, the total time complexity of MQP-II is $O(|RT| \times (|W_p| + |\Gamma|) + 2 \times d^3 \times L)$. The proof completes. □

4.3 Modifying W_p and k

As mentioned in Section 3.2, manufacturers can modify q (e.g., Apple changes the design of a computer) in order to exclude certain unexpected customers from the result set. However, we would like to highlight that this is not the only option. Back to the previous example. The unexpected users of the computer that is newly designed for professional developers are some high school/college students. As we know, the customer's preferences of products are usually formed by their knowledge of the products and their prior experiences. Nonetheless, most, if not all, customers initially have very limited knowledge about a new product, and they gradually become knowledgeable after an exposure or experience with the product. In other words, those high school students show interests in the new computer because of their limited knowledge about the product and the preference they assume might not reflect their real preference. Consequently, Apple could educate customers in terms of how to set proper preference or influence them to change their preferences through proper marketing strategies (e.g., advertising or marketing campaign). To this end, we develop the second solution to answer why questions on reverse top-k queries by modifying customer's preferences, i.e., the why weighting vector set W_p and the parameter k.

In Section 3.3, we propose Equation (6) to estimate the changes between the original customer preferences and the modified customer preferences w.r.t. the reverse top-k query. For the second solution of why questions on reverse top-k queries, we still employ Equation (6) to quantify the difference between the modified customer preferences and the original customer preferences, and we still assume that the smaller the difference is, the better the modification is.

In order to utilize Equation (6) in our solution, we need to derive the values for both Δk_{max} and $(\Delta W_p)_{max}$. Here, Δk_{max} represents the maximum change of k's value. Unlike why-not questions on reverse top-k queries, for why questions on reverse top-k queries, we only need to decrease the value of k. In the following, we use an example to explain why only decrease, but not increase, in k value is possible. Assume that $(W_p, k = 6)$ is modified to $(W'_p, k' = 10)$. If q does not belong to the top-10 query result of every refined why weighting vector $\overrightarrow{w'_i} \in W'_p$, it definitely will not be in the corresponding top-6 query result. Consequently, there is no need to increase the original k value. As soon as we can get the possible minimum value of the modified k', i.e., k'_{min}, Δk_{max} will be set to $\Delta k_{max} = k - k'_{min}$. Below, we present Lemma 4.2 to facilitate the computation of k'_{min}.

Lemma 4.2 *Given a query point q, a why weighting vector set $W_p = \{\overrightarrow{w_1}, \overrightarrow{w_2}, \cdots, \overrightarrow{w_m}\}$, a set $R_p = \{r_1, r_2, \cdots, r_m\}$, where $r_i \in R_p$ is the actual ranking of q under the corresponding why weighting vector $\overrightarrow{w_i} \in W_p$, then $k'_{min} = min_{\forall r_i \in R_p}(r_i) - 1$.*

Proof Assume that we have a refined W'_p and k' with $\Delta W'_p = 0$ and $k' = min_{\forall r_i \in R_p}(r_i) - 1$. Any other possible refined W''_p and k'' with $\Delta W''_p > 0$ must have its $k'' > min_{\forall r_i \in R_p}(r_i) - 1$, otherwise W''_p and k'' cannot be the optimal result as $Penalty(W'_p, k') < Penalty(W''_p, k'')$. On the other hand, $min_{\forall r_i \in R_p}(r_i) - 1$ has its minimum value 0. Given the fact that k'_{min} shall be a non-negative integer, $min_{\forall r_i \in R_p}(r_i) - 1$ already reaches the minimum value of k'_{min}, and it cannot be further reduced. Therefore, $k'_{min} = min_{\forall r_i \in R_p}(r_i) - 1$, and the proof completes. □

For example, in Figure 6(c), suppose we have $W_p = \{\overrightarrow{w_2}, \overrightarrow{w_3}\}$. As depicted in Figure 1(c), we understand the actual ranking of the query point q under $\overrightarrow{w_2}$ and $\overrightarrow{w_3}$ are 3 and 2 respectively, i.e., $R_p = \{3, 2\}$. Accordingly, $k'_{min} = min(2, 3) - 1 = 1$. Unfortunately, the range of $[k'_{min}, k)$ is still very loose and some $k' \in [k'_{min}, k)$ may exclude certain existing reverse top-k query results that we would like to retain in the result set. Back to Figure 6(c). Assume that there is another weighting vector $\overrightarrow{w_5}$, which is one of the existing reverse top-3 query result of q. From Figure 6(c), we know that the actual ranking of the query point q under $\overrightarrow{w_5}$ is 3. If we set k' to 1, $\overrightarrow{w_5}$ will be excluded from the reverse top-3 query result of q. In order to make sure all the desirable results are still retained in the result set, we propose Lemma 4.3 to guide the approximation of k'_{min} value.

Lemma 4.3 *Given a reverse top-k query issued at a query point q, a why weighting vector set $W_p = \{\overrightarrow{w_1}, \overrightarrow{w_2}, \cdots, \overrightarrow{w_m}\}$, a sub-set $\Gamma = \{\overrightarrow{v_1}, \overrightarrow{v_2}, \cdots, \overrightarrow{v_l}\}$ of the result that we would like to retain in the result set even after we modify W_p and k with $W_p \cap \Gamma = \varnothing$, a set $R_\Gamma = \{r_{v1}, r_{v2}, \cdots, r_{vl}\}$, where $r_{vj} \in R_\Gamma$ is the actual ranking of q under the corresponding existing reverse top-k result $\overrightarrow{v_j} \in \Gamma$, then $k'_{min} = max_{\forall r_{vj} \in R_\Gamma}(r_{vj})$.*

Proof If the existing reverse top-k result $\overrightarrow{v_j} \in \Gamma$ is retained in the modified query result, the modified value of k' should not be smaller than the actual ranking of a query point q under $\overrightarrow{v_j}$. Consequently, if the whole existing reverse top-k result set is retained in the modified query result, the modified value of k' should not be smaller than the maximal value of the actual ranking of the query point q under existing reverse top-k result set. Hence, $k'_{min} = max_{\forall r_{vj} \in R_\Gamma}(r_{vj})$. The proof completes. □

As an example, in Figure 6(c), suppose $\Gamma = \{\overrightarrow{w_5}\}$, $k'_{min} = 3$. It is worth mentioning that Γ may be empty if the reverse top-k query result only contains the why weighting vectors. In addition, for why questions on monochromatic reverse top-k queries, we set $k'_{min} = min_{\forall r_i \in R_p}(r_i) - 1$ since the monochromatic reverse top-k query result is infinite, and thus, it is impossible to compute the maximal value of actual ranking of a query point q under existing reverse top-k query result. In this case, we assume that Γ is also an

empty set. Recall that we have already stated in Section 3.3 that $(\Delta W_p)_{max} = \sum_{i=1}^{|W_p|} \sqrt{1 + \sum_{j=1}^{d} (w_i[j])^2}$. Based on the above discussion, we present the penalty model to quantify the modification of W_p and k in Equation (8).

$$Penalty({W_p}', k') = \alpha \cdot \frac{max(0, k - k')}{k - k'_{min}} + (1 - \alpha) \cdot \frac{\sum_{i=1}^{|W_p|} \sqrt{\sum_{j=1}^{d} (w_i[j] - w'_i[j])^2}}{\sum_{i=1}^{|W_p|} \sqrt{1 + \sum_{j=1}^{d} (w_i[j])^2}} \quad (8)$$

where

$$k'_{min} = \begin{cases} min_{\forall r_i \in R_p}(r_i) - 1 & \Gamma = \varnothing \\ max_{\forall r_{vj} \in R_\Gamma}(r_{vj}) & \Gamma \neq \varnothing \end{cases} \quad (9)$$

Accordingly, based on the cost function, we formulate the problem of modifying W_p and k for why questions on reverse top-k queries in Definition 4.4.

Definition 4.4 (**Modifying $\boldsymbol{W_p}$ and $\boldsymbol{k}$**). Given a d-dimensional dataset P, a positive integer k, a query point q, and a why weighting vector set $W_p = \{\overrightarrow{w_1}, \overrightarrow{w_2}, \cdots, \overrightarrow{w_m}\}$ ($\forall \overrightarrow{w_i} \in W_p, q \in TOPk(\overrightarrow{w_i})$), the modification of W_p and k for why questions on reverse top-k queries is to find $W'_p = \{\overrightarrow{w'_1}, \overrightarrow{w'_2}, \cdots, \overrightarrow{w'_m}\}$ and k', such that (i) $\forall \overrightarrow{w'_i} \in W'_p$, $q \notin TOPk'(\overrightarrow{w'_i})$; and (ii) the $Penalty(W'_p, k')$ is minimized.

As mentioned earlier, the possible ranking of q can be determined according to the set D of points dominating q and the set I of points incomparable with q. Then, for every possible ranking of q, we can employ the quadratic programming algorithm to find the $\overrightarrow{w'_i}$ such that $|\overrightarrow{w'_i} - \overrightarrow{w_i}|$ is minimum, for $\overrightarrow{w_i} \in W_p$. If the $\overrightarrow{w'_i}$s under all possible rankings of q are found, we can get the optimal W'_p and k'. However, this brute-force approach suffers from high time complexity, i.e., it needs to solve $|W_p| \times 2^{|I|}$ quadratic programming problem. To address this efficiency issue, we again employ the sampling based algorithm to modify W_p and k.

The basic idea is to first sample a certain number of weighting vectors from the sample space and then use these sample weighting vectors to find (W'_p, k') with minimum penalty. In particular, the sample space is also the same as that of MWK-I algorithm. This is because for $\overrightarrow{w'_i} \in W'_p$, $|\overrightarrow{w'_i} - \overrightarrow{w_i}|$ must be minimal. Otherwise, $\Delta W'_p$ is not minimal. As proved in [24], the qualified weighting vectors exist in the hyperplanes formed by I and q. That is to say, for any point $p \in I$, the hyperplane formed by p and q w.r.t. a why weighting vector $\overrightarrow{w_i} \in W_p$ is $(\overrightarrow{p} - \overrightarrow{q}) \cdot \overrightarrow{w}_i = 0$; and all the hyperplanes intersecting with $\sum_{i=1}^{d} w[i] = 1$ constitute the sample space. In addition, we also assume that the sample size S_W is specified by the user.

After getting the sample weighting vectors, we try to find the (W'_p, k') with minimal penalty according to Lemma 3.7. To be more specific, assume that we have a candidate tuple (W'_p, k'), for a sample weighting vector $\overrightarrow{s}$ with the ranking of q under $\overrightarrow{s}$ being smaller than k'. If $\exists \overrightarrow{w'_i} \in W'_p$, $|\overrightarrow{w_i} - \overrightarrow{s}| < |\overrightarrow{w_i} - \overrightarrow{w'_i}|$, we replace all such $\overrightarrow{w_i}$ with $\overrightarrow{s}$, and get a new candidate (W''_p, k''), where k'' equals to the ranking of q under $\overrightarrow{s}$. Although $\Delta k'' > \Delta k'$, we have $\Delta W''_p < \Delta W'_p$. Thus, (W''_p, k'') is also a candidate tuple for the final result. Thereafter, we obtain all candidate tuples (W'_p, k'), and the one with the minimal penalty is the final answer.

Based on the above discussion, we present an algorithm called MWK-II to modify W_p and k for why questions on reverse top-k queries. Since MWK-II shares a similar logic as MWK-I, i.e., Algorithm 2, we skip the pseudo-code of MWK-II but only explain the differences. First, MWK-I sorts the sample weighting vectors in *ascending* order of rs_i values (line 6 of Algorithm 2) while MWK-II needs to sort the sample weighting vectors in *descending* order of rs_i. Second, MWK-I needs to compute the value of k'_{max} (lines 7-9 of Algorithm 2) while MWK-II needs to compute the value of k'_{min} based on Equation (9). Third, MWK-I terminates the examination of sample weighting vectors if $k'_{max} < rs_i$ (line 13 of Algorithm 2) while the termination condition for MWK-II is $k'_{min} > rs_i$. Similarly, the time complexity of MWK-II algorithm is $O(|RT| + |S| \times |W_p|)$, where $|S|$ is the cardinality of a sample weighting vector set.

4.4 Modifying q, W_p, and k

The two solutions proposed above assume that only the products' configurations or the customers' preferences are improper, and they propose ideas to change their setting(s) accordingly. However, in some circumstances, both the products' configurations and the customers' preferences could be improved. In other words, both the product's configuration and the customers' preferences shall be changed so that the manufacturers can design the appropriate product for particular application better, and the customers find the products that really suit their needs. Towards this, we present the third solution to refine the reverse top-k query for why questions by modifying both the product (i.e., q) and customers' preferences (i.e., W_p and k).

The penalty model, as presented in Equation (2), is to measure the cost caused by modifying q, W_p and k to q', W'_p, and k' respectively, which is similar as the cost model defined in Equation (1) for why-not questions. To be more specific, again we use $Penalty(q')$ defined in Equation (3) and $Penalty(W'_p, k')$ defined in Equation (8) to measure the cost of modifying q and (W_p, k) respectively, and weighting parameter γ ($\in [0, 1]$) represents the relative tolerance to the change of q. Based on Equation (2), we formalize the

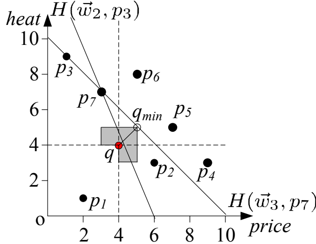


Fig. 7 Example of the sample space of q for why questions

problem of modifying (q, W_p) and k for why questions on reverse top-k queries in Definition 4.5.

Definition 4.5 (**Modifying q, W_p and k**). Given a d-dimensional dataset P, a positive integer k, a query point q, and a why weighting vector set $W_p = \{\overrightarrow{w_1}, \overrightarrow{w_2}, \cdots, \overrightarrow{w_m}\}$ with $\forall \overrightarrow{w_i} \in W_p$, $q \in TOPk(\overrightarrow{w_i})$, the modification of q, W_p, and k is to find q', $W'_p = \{\overrightarrow{w'_1}, \overrightarrow{w'_2}, \cdots, \overrightarrow{w'_m}\}$, and k', such that (i) $\forall \overrightarrow{w'_i} \in W'_p$, $q' \in TOPk'(\overrightarrow{w'_i})$; and (ii) the $Penalty(q', W'_p, k')$ is minimized.

Because of the similarity between the third solution for why-not questions on reverse top-k queries and that for why questions on reverse top-k queries, we extend MQWK-I to perform the modification for why questions on reverse top-k queries. The basic idea consists of three steps: (i) sampling a set of candidate query points; (ii) for every sample query point q', using MWK-II algorithm to find the optimal (W'_p, k'); and (iii) returning the tuple (q', W'_p, k') with the smallest penalty according to Equation (**??**). It is worth mentioning that the sample space of q for why questions on reverse top-k queries is different from that of why-not questions on reverse top-k queries. If the query point locates inside $IR(q)$ or $EIR(q)$, the why weighting vectors will definitely be excluded from the reverse top-k query result. Thus, if we sample a query point (e.g., q') from the invalid region $IR(q)$ or enhanced invalid region $EIR(q)$, $Penalty(W'_p, k') = 0$. Hence, $Penalty(q', W'_p, k') \geq Penalty(q_{min}, W_p, k)$ (q_{min} is optimal refined query point). Therefore, (q', W'_p, k') cannot be the final result, and we should sample the query point out of the invalid region and the enhanced invalid region. Furthermore, given a refined tuple (q'', W''_p, k'') where q'' is sampled out of both $IR(q)$ and $EIR(q)$, then $Penalty(W''_p, k'') > 0$. The tuple (q'', W''_p, k'') becomes the optimal result only if $Penalty(q'') < Penalty(q_{min})$. Otherwise, $Penalty(q'', W''_p, k'') \geq Penalty(q_{min}, W_p, k)$, and thus, it cannot be the final result. As mentioned in Section 4.3, the qualified search space of q is $\Omega - \{q' | 0 \leq q' \leq q\}$. Therefore, the sample space of q for why questions on reverse top-k queries is: $SP(q) = Rec(q, q_{min}) - \{q' | 0 \leq q' \leq q\}$, in which $Rec(q, q_{min})$ is an area centered at q and has the coordinate-wise distance to q_{min} as its extent. Figure 7 depicts an example of q's sample space for why questions, where q_{min} is the optimal refined query point in Figure 6(b). The shaded area is the sample space of q.

Based on the above discussion, we present an algorithm called MQWK-II to modify q, W_p, and k. MQWK-II proceeds as follows. First, it invokes MQP-II algorithm to get the minimal q_{min}. Second, it samples $|Q|$ query points from the sample space. For each sample query point q', it employs MWK-II algorithm to compute the corresponding optimal (W'_p, k') integrating the reuse technique. Finally, the tuple (q', W'_p, k') with the minimal penalty will be returned as the result. Since the logic of MQWK-II is similar as that of MQWK-I, we skip the pseudo-code of MQWK-II for save spacing. Based on the time complexity analysis of MQWK-I presented in Section 3.4, we conclude that the time complexity of MQWK-II is $O(|RT| \times (|W_p| + |\Gamma|) + 2 \times d^3 \times L + |Q| \times (|RT| + |S| \times |W_p|))$.

5 Experimental Evaluation

In this section, we evaluate the effectiveness and efficiency of our proposed algorithms via extensive experiments, using both real and synthetic datasets.

5.1 Experimental Settings

In our experiments, we use two real datasets, i.e., *NBA* and *Household*. *NBA* (http://www.nba.com) contains 17K 13 -dimensional points, where each point corresponds to the statistics of an NBA player in 13 categories such as the number of points scored, rebounds, assists, etc. *Household* (http://www.ipums.org) is a 127K 6-dimensional dataset. Each tuple in the dataset represents the percentage of an American family's annual income spent on six types of expenditures (e.g., gas, electricity). We also create three synthetic datasets, i.e., *Independent*, *Correlated* and *Anti-correlated*. In *Independent* dataset, all attribute values are generated independently using a uniform distribution; *Correlated* dataset denotes an environment in which points good in one dimension are also good in one or all of the other dimension(s); and *Anti-correlated* dataset denotes an environment in which points good in one dimension are bad in one or all of the other dimension(s). Since the experimental results on anti-correlated and correlated are similar, we only report the experimental results on anti-correlated data due to the space limits. The why-not/why weighting vectors are selected from the weighting vector sets that are generated by following the independent distribution as with [24,25].

We study the performance of the presented algorithms under various parameters, including dimensionality d, dataset cardinality $|P|$, k, actual ranking of q under W_m/W_p, the cardinality of a why-not (why) weighting vector set $|W_m|$ ($|W_p|$), the sample size S_W and S_q, parameters α and γ. The ranges of the parameters and their default values denoted by bold are summarized in Table 2, which follows existing

Table 2 Parameter ranges and default values

Parameter	Setting
Dimensionality d	2, **3**, 4, 5, 10
Dataset cardinality $\|P\|$	10K, 50K, **100K**, 500K, 1000K
k	**10**, 20, 30, 40, 50
Actual ranking of q under W_m	11, **101**, 1001, 10001
Actual ranking of q under W_p	**9**, 19, 29, 39, 49
$\|W_m\|$ or $\|W_p\|$	**1**, 2, 3, 4, 5
S_W or S_q	100, 200, 400, **800**, 1600
α or γ	0.1, **0.5**, 0.9

work [24,25,43,46]. In every experiment, only one parameter is changed, whereas others are fixed to their default values. We adopt total running time (in seconds) and penalty as the main performance metrics. All experiments presented in this paper are implemented in C++, and conducted on a Windows PC with 2.8 GHz CPU and 4 GB main memory. Each dataset is indexed by an R-tree, where the page size is set to 4096 bytes. Note that our algorithms are not memory-based. Therefore, in our experiments, the R-tree is stored on the disk and only a part of it is loaded to the buffer.

5.2 Evaluation of Why-not Questions Algorithms

In this section, we evaluate the algorithms for answering why-not questions on reverse top-k queries, namely, MQP-I, MWK-I, and MQWK-I. First, we investigate the impact of dimensionality d on the algorithms. We utilize synthetic datasets, and report the efficiency of different algorithms in Figure 8. Note that, each numbers with three decimal points listed in every diagram refers to the penalty of the corresponding algorithm at a specific setting. In general, the performance of three algorithms degrades with the growth of dimensionality. This is because all three algorithms need to traverse the R-tree that has a poor efficiency in a high dimensional space, resulting in the performance degradation of three algorithms. Moreover, for MQP-I and MQWK-I, the quadratic programming takes more time in finding the optimal q' as d grows, which also leads to the degradation of MQP-I and MQWK-I. It is also observed that, all the algorithms return the answers with small penalty. However, the penalty neither increases nor decreases with the growth of cardinality. The reason is that, the penalty is only affected by the sample size, while other parameters have no influence on it.

Second, we vary the dataset cardinality $|P|$ from 10^4 to 10^6, and report its impact on the algorithms, as reported in Figure 9. As expected, the total running time of three algorithms ascends as $|P|$ grows. Nevertheless, the penalties of MQP-I, MWK-I, and MQWK-I are small. This is because the larger the dataset cardinality is, the bigger the R-tree is. Thus, three algorithms need to traverse more R-tree nodes with the growth of $|P|$, incurring longer total running time. It is observed that the runtime of independent dataset is very different from that of anti-correlated dataset as shown in Figure 9, which is caused by the datasets' different distributions.

Third, we explore the influence of k on three algorithms, and report the results in Figure 10. It is observed that, all the algorithms degrade their performances as k increases. The reason is that, k'_{max} ascends as k grows, and thus, MWK-I takes more time in getting the optimal tuple (W'_m, k') using sample weighting vectors, which results in the degradation of MWK-I. For MQP-I algorithm, if the value of k becomes larger, the cost of finding the k-th point also increases, and hence, the performance degrades. Since MQWK-I integrates MQP-I and MWK-I, it degrades as well. Again, the penalties of three algorithms are still small.

Fourth, we inspect the impact of actual ranking of q under the why-not weighting vector set W_m by fixing d at 3, $|P|$ at 100K, sample size at 800, $|W_m|$ = 1, and k = 10. Figure 11 depicts the results on both real and synthetic datasets. Clearly, the total running time of three algorithms increases while the penalties remain small. For MWK-I algorithm, when the actual ranking of q under W_m ascends, k'_{max} also grows, incurring longer total running time. For MQP-I algorithm, if the ranking of q is low, L (defined in Theorem 3.1) becomes larger, and thus, the quadratic programming takes more time to find q'. Based on the above two reasons, MQWK-I also degrades its performance.

Next, we explore the influence of the cardinality $|W_m|$ of a why-not weighting vector set on the algorithms, and Figure 12 plots the results. We observe that MQP-I, MWK-I, and MQWK-I can find the optimal solution with small penalty. Again, the total running time of all algorithms increases gradually when $|W_m|$ ascends. The degradation of MWK-I is mainly caused by the second phase of the algorithm, i.e., using the sample weighting vectors to find the approximate optimal answer. The performance descent of MQP-I is due to the computation of the k-th point for more why-not weighting vectors. Similarly, MQWK-I degrades as well. Moreover, it is observed that, in Figure 12(b), the performance of MQP-I degrades rapidly and the performance gap between MQP-I and MWK-I becomes smaller. It is because the increase of why-not weighting vectors under *NBA* makes the size of the quadratic programming problem (i.e., L defined before Theorem 3.1 in section 3.2) grow rapidly, resulting in the rapid degradation of MQP-I.

Then, we evaluate the effect of sample size S_W and S_q on the algorithms respectively. To this end, we vary sample size from 100 to 1600, and fix other parameters to their default values. Figure 13 and Figure 14 report the results on S_W and S_q respectively. In Figure 13, the total running time of algorithms MQWK-I and MWK-I grows when S_W ascends, although the growth is relatively moderate for MWK-I. This is because the algorithms take more time to examine the samples. Moreover, it is obvious that the penalty of algorithms MQWK-I and MWK-I drops as sample size

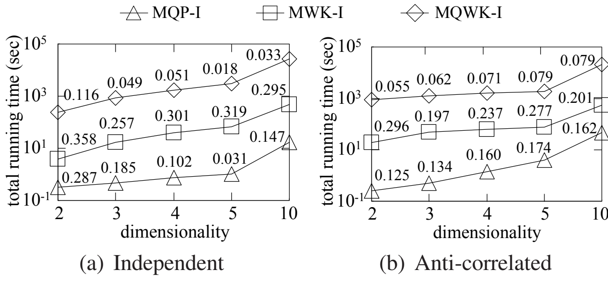


Fig. 8 Why-not questions cost vs. dimensionality

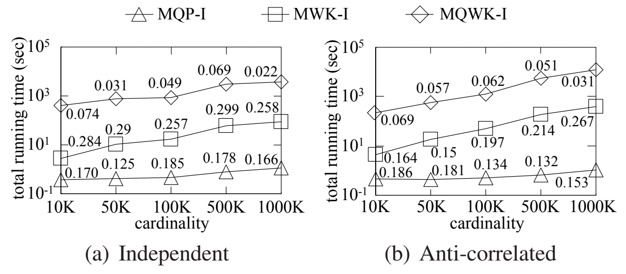


Fig. 9 Why-not questions cost vs. dataset cardinality

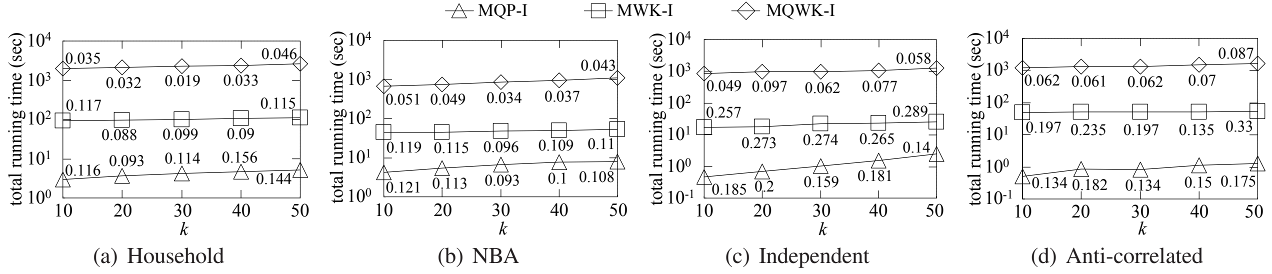


Fig. 10 Why-not questions cost vs. k

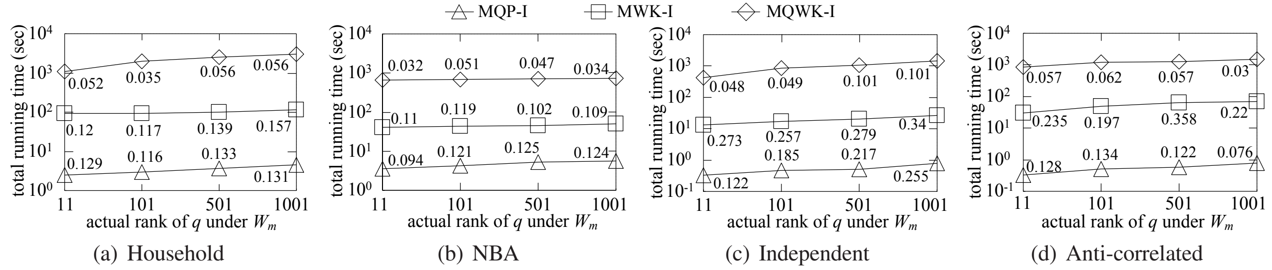
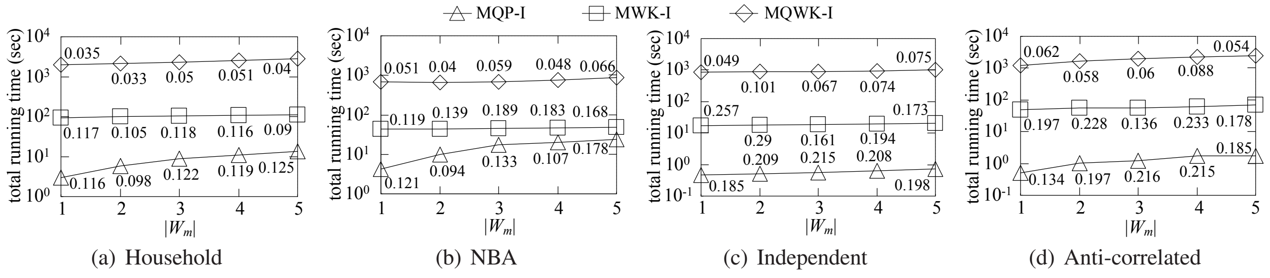


Fig. 11 Why-not questions cost vs. actual ranking under W_m

grows. The reason behind is that the bigger the sample size, the higher the quality result. Note that, the penalty sometimes decreases very fast with increasing sample size, and sometimes it does not change. There are two potential reasons. First, it is caused by the randomness since the sample weighting vectors are randomly sampled from the sample space. Second, different dataset distributions may also lead to this phenomenon. In Figure 14, as expected, the total running time of algorithm MQWK-I changes and the penalty of the algorithm MQWK-I decreases with the growing of S_q. However, the total running time and the penalty of algorithm MWK-I almost remain the same in Figure 14. This is because algorithm MWK-I doesn't need to sample the query point. Moreover, in both Figure 13 and Figure 14, the total running time and the penalty of MQP-I algorithm do not change with the growth of S_W/S_q, since MQP-I does not use the sampling technique. Compared Figure 13 with Figure 14, it is found that S_q has a bigger impact on MWK-I than S_W. Recall that MQWK-I needs to iteratively invoke MWK-I. If S_q increases, MQWK-I calls more MWK-I. If S_W increases, the time for each calling of MQWK-I increases. From Figure 13, we find the time of MWK-I ascends gently as S_W increases. Therefore, the growth of S_q leads to more rapid degradation of MQWK-I .

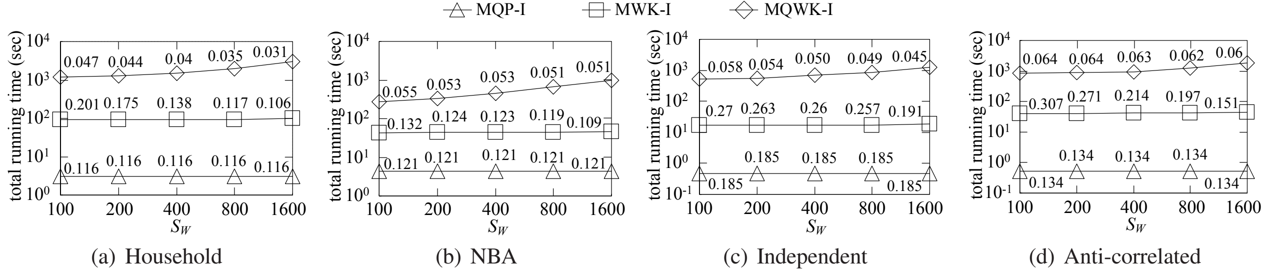
Finally, we evaluate the effect of parameters α and γ on the algorithms, whose results are shown in Figure 15 and Figure 16 respectively. It is observed that both α and γ have little influence on the total running time. In addition, α influences the penalty of algorithms MQWK-I and MWK-I; and γ affects the penalty of algorithm MQWK-I. The reason behind is that α and γ only determine the penalty model of the algorithms but not the running time of the algorithms, which also can be confirmed by Theorem 3.1, Theorem 3.2, and Theorem 3.3 .

In summary, from all the experimental results, we can conclude that our proposed algorithms, viz., MQP-I, MWK-I, and MQWK-I, are efficient, and scale well under a variety of parameters. Among three algorithms, MQP-I that only modifies the query point is most efficient with relatively high penalty while MQWK-I that modifies both the query point and customer preferences incurs the smallest penalty with relatively long running time. Moreover, It is found that the algorithms' performance decreases with the growth of dimentionality and cardinality. Therefore, it is necessary to



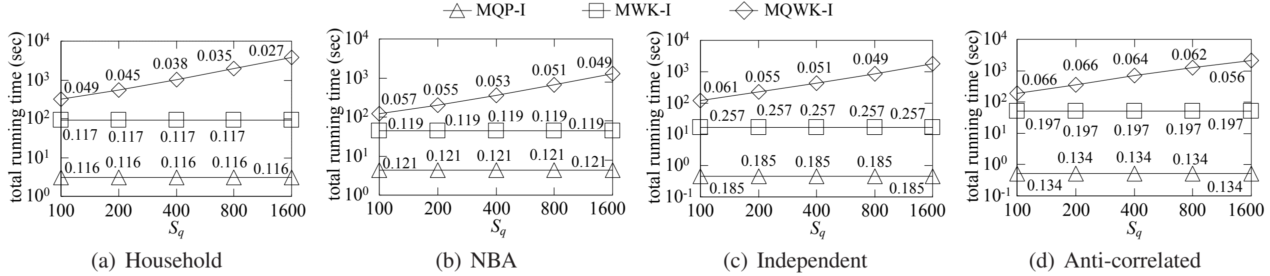
(a) Household (b) NBA (c) Independent (d) Anti-correlated

Fig. 12 Why-not questions cost vs. $|W_m|$



(a) Household (b) NBA (c) Independent (d) Anti-correlated

Fig. 13 Why-not questions cost vs. S_W



(a) Household (b) NBA (c) Independent (d) Anti-correlated

Fig. 14 Why-not questions cost vs. S_q

propose more efficient algorithms to answer why-not questions on reverse top-k queries.

5.3 Evaluation of Why Questions Algorithms

This subsection shows the performance of algorithms MQP-II, MWK-II, and MQWK-II, which are designed to answer why questions on reverse top-k queries. First, we explore the influence of dimensionality d on three algorithms by using synthetic datasets. Specifically, k = 10, $|P|$ = 100K, $|W_p|$ = 1, sample size is set to 800, actual ranking of q under W_p is 9, and d is in the range of [2, 5]. Figure 17 shows the efficiency of three algorithms. Obviously, similar with the why-not questions algorithms, the total running time of three algorithms ascends as $|d|$ grows. The reasons behind is two-fold. First, the efficiency of R-tree is poor in a high dimensional space. Second, the performance of quadratic programming algorithm degrades with the growth of d.

Second, we evaluate the effect of dataset cardinality $|P|$ on the algorithms, and report the results in Figure 18. As $|P|$ grows, the cost of three algorithms increases, which is consistent with our expectation, and confirms that $|P|$ has a direct impact on the performance. This is because with the growth of $|P|$, MQP-II, MWK-II, and MQWK-II all need to traverse more data points, and hence, the performance of algorithms degrades.

Third, we investigate the influence of k on the algorithms. Figure 19 depicts the results on both real and synthetic datasets. As expected, the total running time of three algorithms ascends as $|k|$ grows. For MWK-II algorithm, it gets the optimal tuple (W_p', k') using sample weighting vectors under which the query point q's rank is between $[k'_{min}, k]$. If k grows, the range $[k'_{min}, k]$ enlarges, and thus, more sample weighting vectors are examined, resulting in the degradation of MWK-II. For MQP-II algorithm, if the value of k becomes larger, the cost of finding the k-th point also ascends, and hence, the performance degrades. Moreover, in Figure 19(b), the curve for MQP-II is the highest. If k increases, the why-not weighting vectors also change, making the size of the quadratic programming problem increase rapidly. Therefore, the MQP-II degrades rapidly.

Fourth, we study the impact of actual ranking of q under the why weighting vector set W_p by fixing d = 3, $|P|$ = 100K, sample size = 800, $|W_p|$ = 1, and k = 10. As shown in Figure 20, the total running time of three algorithms increases. For MWK-II algorithm, when the actual ranking of

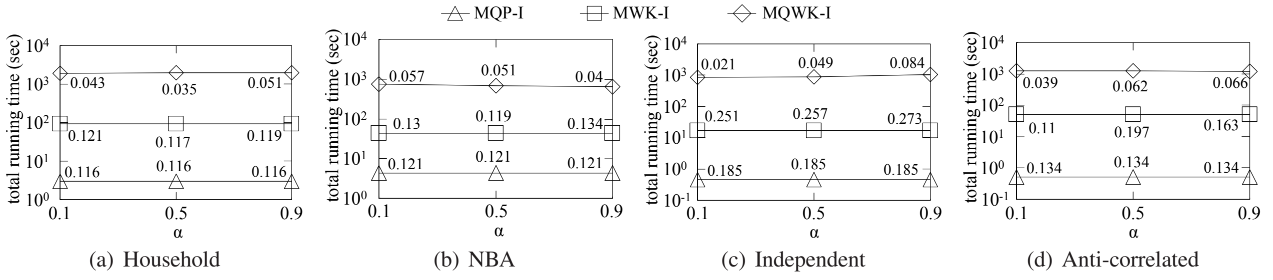


Fig. 15 Why-not questions cost vs. α

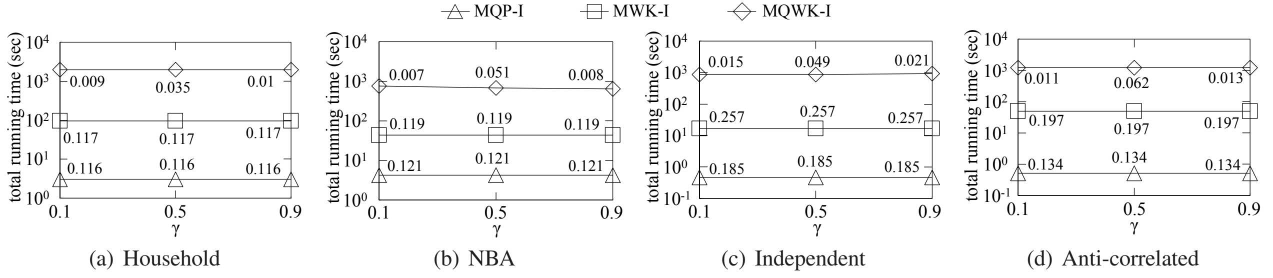


Fig. 16 Why-not questions cost vs. γ

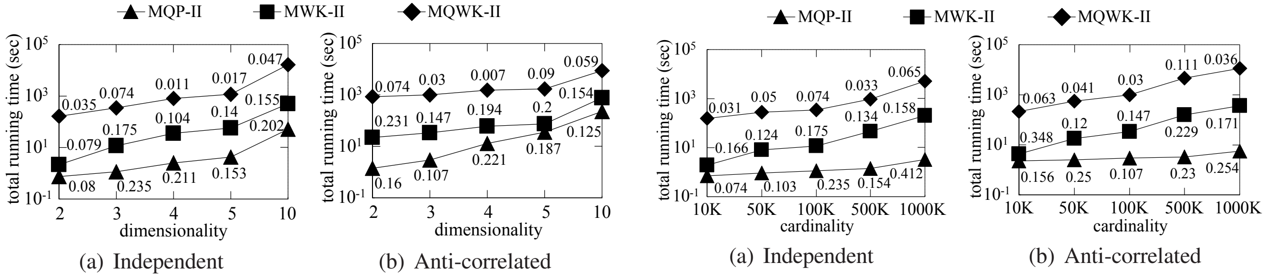


Fig. 17 Why questions cost vs. dimensionality

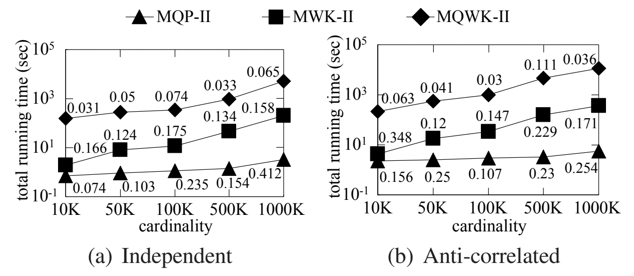


Fig. 18 Why questions cost vs. dataset cardinality

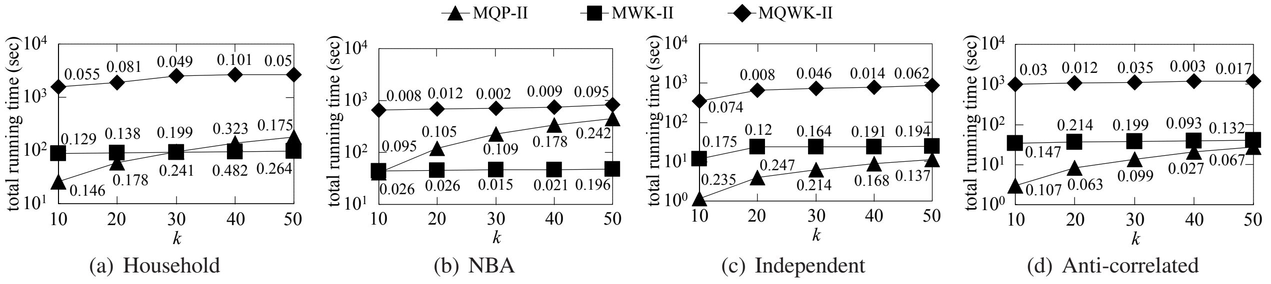
q under W_p grows, k'_{min} also ascends, incurring longer total running time. For MQP-II algorithm, if the ranking of q is low, the quadratic programming takes more time. Hence, MQWK-II also degrades.

Next, we vary the cardinality of a why weighting vector set $|W_p|$ from 1 to 5, and verify its effect on the algorithms using both real and synthetic datasets. As shown in Figure 21, the total running time of all algorithms increases gradually when $|W_p|$ ascends. The second phase of MWK-II algorithm leads to its degradation, where it examines the sample weighting vectors to find the optimal (W'_p, k'). In addition, MQP-II needs to compute the k-th point for more why weighting vectors, and thus results in the worse performance. Similarly, MQWK-II degrades as well.

Then, we explore the impact of sample size S_W and S_q on three algorithms with results reported in Figure 22 and Figure 23 respectively. We observe that (i) with the growth of S_W, the total running time of algorithms MQWK-II and MWK-II grows while the penalty of these two algorithms decrease; (ii) with the growth of S_q, the total running time of algorithm MQWK-II grows while its penalty decreases. The reason is obvious since (i) the algorithms take more time to examine the sample weighting vectors; and (ii) the bigger the sample size, the higher the quality result. It is observed that, only in this set of experiments, the penalty changes in a relatively stable trend. This is because, the penalty is only affected by the sample size, but not the other parameters.

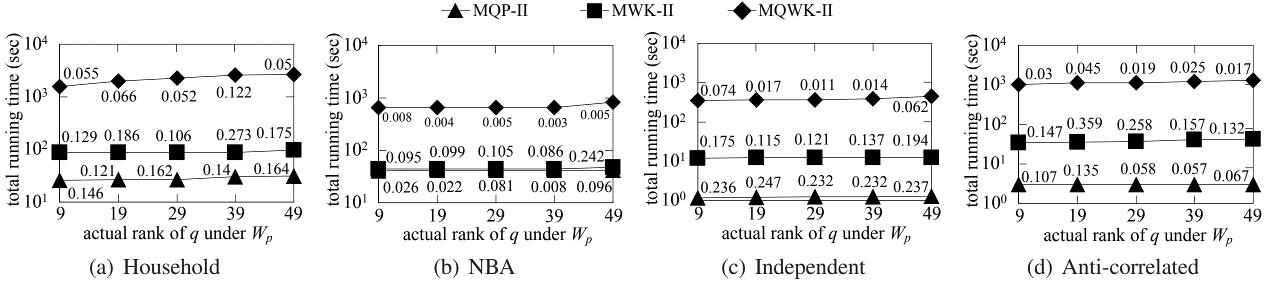
Finally, we evaluate the effect of parameters α and γ on the algorithms, with results reported in Figure 24 and Figure 25 respectively. As expected, α and γ rarely affect the total running time of the algorithms. On the other hand, (i) since the penalty model of MWK-II (i.e., Equation (8)) includes α, α affects the penalty of algorithm MWK-II; and (ii) since the penalty model of MQWK-II (i.e., Equation (2)) contains α and γ, both α and γ affect penalty of algorithm MQWK-II.

All the the above experimental results demonstrate that MQP-II, MWK-II, and MQWK-II scale well under a variety of parameters, which can further verify the flexibility of the framework WQRTQ. Similar as the observations made in the experiments on why-not question algorithms, MQP-II is most efficient in terms of running time, and MQWK-II has the smallest penalty with relatively long running time.



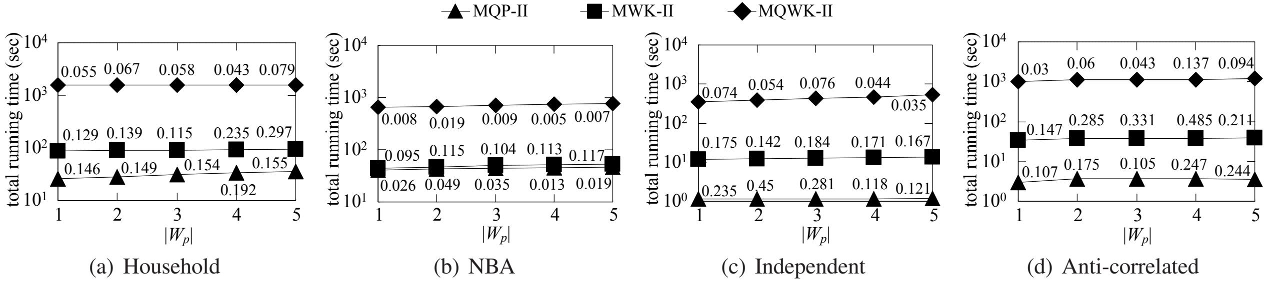
(a) Household (b) NBA (c) Independent (d) Anti-correlated

Fig. 19 Why questions cost vs. k



(a) Household (b) NBA (c) Independent (d) Anti-correlated

Fig. 20 Why questions cost vs. actual ranking under W_p



(a) Household (b) NBA (c) Independent (d) Anti-correlated

Fig. 21 Why questions cost vs. $|W_p|$

6 Related work

In this Section, we review previous work on top-k queries, reverse top-k queries, data provenance, and why-not questions.

Top-k queries. Top-k query has received much attention in the database community because of its usefulness. Existing algorithms include convex hull based algorithm *Onion* [12], view based algorithms *LPTA* [20] and *PREFER* [29,30], layered index based algorithm *AppIR* [48], branch-and-bound algorithm *BRS* [41], dominant graph based top-k query algorithm [51], and top-k query algorithms using cache [47], among which *BRS* is I/O optimal.

Reverse top-k queries. Vlachou et al. [43] firstly introduce the reverse top-k query and consider its two variants, namely, monochromatic and bichromatic versions. To efficiently answer the monochromatic reverse top-k query, Vlachou et al. [43] and Chester et al. [17] present several algorithms in a 2-dimensional (2D) space. The bichromatic top-k query algorithms include *RTA*, *GRTA*, and *BBR* [43,46]. In addition, Yu et al. [49] develop a dynamic index to support reverse top-k queries, and Ge et al. [22] employ all top-k queries to boost the reverse top-k query. More recently, reverse top-k queries are widely studied in market analysis [36,45], location-based services [44], and uncertain circumstances [35]. It is worth noting that, all the current reverse top-k queries only return the results without any explanation, and thus, the existing techniques designed for reverse top-k queries cannot answer corresponding why-not questions efficiently.

Data provenance. Data provenance explores the derivation of a piece of data that is in a query result [40]. It can help users understand why data tuples exist within a result set. Current approaches for computing data provenance include non-annotation method [9,19] and annotation approach [4, 18]. Nonetheless, it cannot be applied to clarify the missing tuples in the query result set.

Why-not questions. Huang et al. [31] firstly explores the provenance of the non-answers (i.e., the why-not question, whose name was proposed in [13]). Since then, lots of efforts have been put into answering why-not questions. The existing approaches can be classified into three categories: (i) *manipulation identification* (e.g., the why-not questions on SPJ queries [13] and SPJUA queries [6]), (ii) *database modification* (e.g., the why-not questions on SPJ queries [31, 50] and SPJUA queries [27,28]), and (iii) *query refinement*

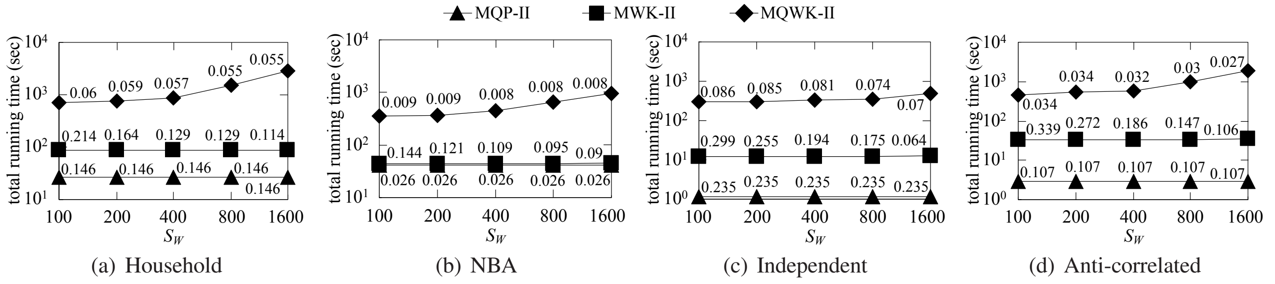


Fig. 22 Why questions cost vs. S_W

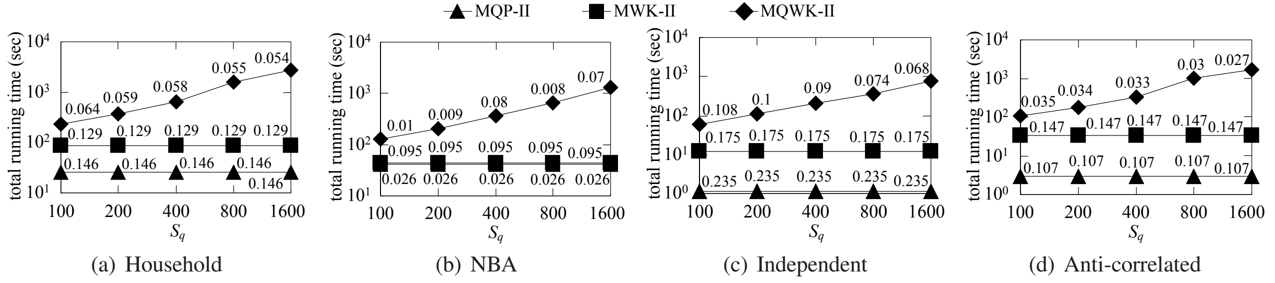


Fig. 23 Why questions cost vs. S_q

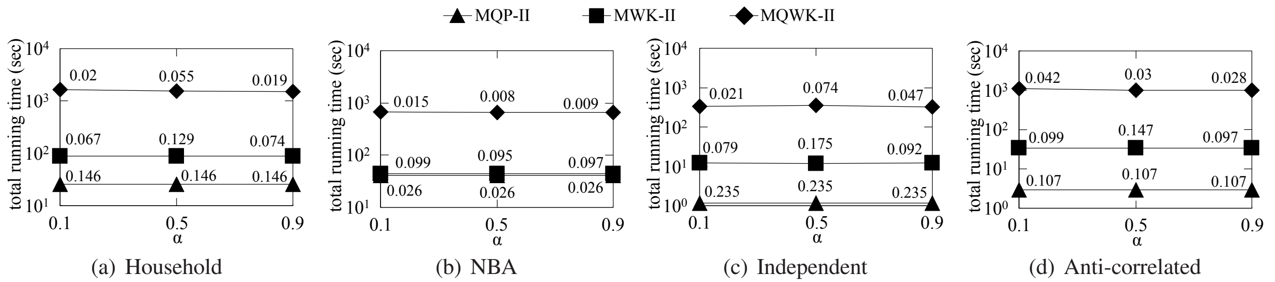


Fig. 24 Why questions cost vs. α

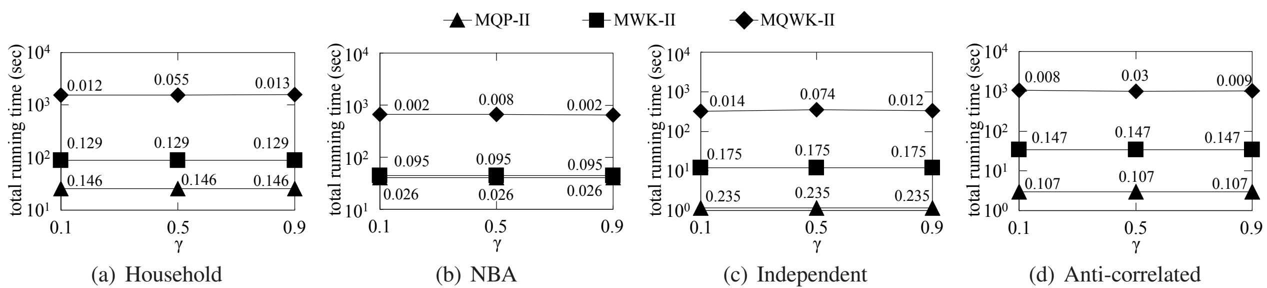
(e.g., the why-not questions on SPJA queries [42], top-k queries [24,25], top-k dominating queries [25], reverse skyline queries [33], image search [5], spatial keyword top-k queries [15,16], similar graph match [32], and metric probabilistic range queries [14]). In addition, Herschel [26] tries to identify hybrid why-not explanations for SQL queries, which combines manipulation identification and query refinement. Ten Cate et al. [11] present a new framework for why-not explanations by leveraging concepts from an ontology to provide high-level and meaningful reasons. Bidoit et al. [7,8] provide a new formalization of why-not explanation as polynomials. Meliou et al. [37] aim to find the causality and responsibility for the non-answers of the query. Here, causality is the cause of non-answers to the query, and responsibility captures the notion of degree of causality.

It is noteworthy that our work follows the query refinement model to answer why-not questions on reverse top-k queries, i.e., we modify the parameter(s) and/or a query point and/or why-not point(s) of an original query to include the missing tuples in a refined query result. However, since why-not questions are *query-dependent*, different queries require different query refinement, which explains the reason that existing query refinement techniques cannot be applied directly in our problem, and justifies our main contribution, that is to design proper query refinement approaches to support why-not questions on reverse top-k queries.

7 Conclusions

In this paper, for the first time, we study the problem of why-not and why questions on reverse top-k queries. We propose a unified framework called WQRTQ to answer why-not questions on reverse top-k queries. Specifically, WQRTQ consists of three solutions, i.e., (i) modifying a query point q, (ii) modifying a why-not weighting vector set W_m and a parameter k, and (iii) modifying q, W_m, and k. Furthermore, we utilize the quadratic programming, sampling method, and reuse technique to boost the performance of our algorithms. In addition, we extend WQRTQ to answer why questions on reverse top-k queries, which demonstrates the flexibility of our proposed algorithms. Extensive experiments with both real and synthetic data sets verify the effectiveness and efficiency of our presented algorithms.

A promising direction for future work is to study the why-not-and-why questions, i.e., not only to include the why-



(a) Household (b) NBA (c) Independent (d) Anti-correlated

Fig. 25 Why questions cost vs. γ

not weighting vectors into the result but also to exclude the why weighting vectors from the result. A naive method is to first call the algorithms of why-not questions to include the why-not weighting vectors, and then call the algorithms of why questions to exclude the why weighting vectors. However, the efficiency of the naive method may not be desirable. Hence, we would like to develop more efficient algorithms to answer why-not-and-why questions in our future work.

Acknowledgements This work was supported in part by the National Key Basic Research and Development Program (i.e., 973 Program) No. 2015CB352502 and 2015CB352503, NSFC Grants No. 61522208, 61379033, and 61472348, and the Fundamental Research Funds for the Central Universities. We also would like to express our gratitude to anonymous reviewers for providing valuable and helpful comments to improve this paper.

References

1. Arnold, S.J., Handelman, J., Tigert, D.J.: The impact of a market spoiler on consumer preference structures (or, what happens when wal-mart comes to town). J. Retailing and Consumer Services **5**(1), 1–13 (1998)
2. Beckmann, N., Kriegel, H., Schneider, R., Seeger, B.: The r*-tree: An efficient and robust access method for points and rectangles. In: SIGMOD, pp. 322–331 (1990)
3. Berg, M., Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational geometry: Algorithms and applications. Springer, New York, USA (1997)
4. Bhagwat, D., Chiticariu, L., Tan, W.C., Vijayvargiya, G.: An annotation management system for relational databases. VLDB J. **14**(4), 373–396 (2005)
5. Bhowmick, S.S., Sun, A., Truong, B.Q.: Why not, wine?: Towards answering why-not questions in social image search. In: MM, pp. 917–926 (2013)
6. Bidoit, N., Herschel, M., Tzompanaki, K.: Query-based why-not provenance with nedexplain. In: EDBT, pp. 145–156 (2014)
7. Bidoit, N., Herschel, M., Tzompanaki, K.: Efficient computation of polynomial explanations of why-not questions. In: CIKM, pp. 713–722 (2015)
8. Bidoit, N., Herschel, M., Tzompanaki, K.: Efq: Why-not answer polynomials in action. In: VLDB, pp. 1980–1991 (2015)
9. Buneman, P., Khanna, S., Tan, W.C.: Why and where: A characterization of data provenance. In: ICDT, pp. 316–330 (2001)
10. Carpenter, G.S., Nakamoto, K.: Consumer preference formation and pioneering advantage. J. Marketing Research **26**(3), 285–298 (1989)
11. ten Cate, B., Civili, C., Sherkhonov, E., Tan, W.C.: High-level why-not explanations using ontologies. In: PODS, pp. 31–43 (2015)
12. Chang, Y.C., Bergman, L., Castelli, V., Li, C.S., Lo, M.L., Smith, J.R.: The onion technique: Indexing for linear optimization queries. In: SIGMOD, pp. 391–402 (2000)
13. Chapman, A., Jagadish, H.V.: Why not? In: SIGMOD, pp. 523–534 (2009)
14. Chen, L., Gao, Y., Wang, K., Jensen, C.S., Chen, G.: Answering why-not questions on metric probabilistic range queries. In: ICDE, p. to appear (2016)
15. Chen, L., Lin, X., Hu, H., Jensen, C.S., Xu, J.: Answering why-not questions on spatial keyword top-*k* queries. In: ICDE, pp. 297–290 (2015)
16. Chen, L., Xu, J., Lin, X., Jensen, C.S., Hu, H.: Answering why-not spatial keyword top-*k* queries via keyword adaption. In: ICDE, p. to appear (2016)
17. Chester, S., Thomo, A., Venkatesh, S., Whitesides, S.: Indexing reverse top-*k* queries in two dimensions. In: DASFAA, pp. 201–208 (2013)
18. Chiticariu, L., Tan, W.C., Vijayvargiya, G.: Dbnotes: A post-it system for relational databases based on provenance. In: SIGMOD, pp. 942–944 (2005)
19. Cui, Y., Widom, J.: Lineage tracing for general data warehouse transformations. VLDB J. **12**(1), 41–58 (2003)
20. Das, G., Gunopulos, D., Koudas, N., Tsirogiannis, D.: Answering top-*k* queries using views. In: VLDB, pp. 451–462 (2006)
21. Gao, Y., Liu, Q., Chen, G., Zheng, B., Zhou, L.: Answering why-not questions on reverse top-*k* queries. In: VLDB, pp. 738–749 (2015)
22. Ge, S., U, L.H., Mamoulis, N., Cheung, D.W.: Efficient all top-*k* computation: A unified solution for all top-*k*, reverse top-*k* and top-*m* influential queries. IEEE Trans. Knowl. Data Eng. **25**(5), 1015–1027 (2013)
23. Goh, K.Y., Teo, H.H., Wu, H., Wei, K.K.: Computer-supported negotiations: An experimental study of bargaining in electronic commerce. In: ICIS, pp. 104–116 (2000)
24. He, Z., Lo, E.: Answering why-not questions on top-*k* queries. In: ICDE, pp. 750–761 (2012)
25. He, Z., Lo, E.: Answering why-not questions on top-*k* queries. IEEE Trans. Knowl. Data Eng. **26**(6), 1300–1315 (2014)
26. Herschel, M.: Wondering why data are missing from query results?: Ask conseil why-not. In: CIKM, pp. 2213–2218 (2013)
27. Herschel, M., Hernandez, M.: Explaining missing answers to spjua queries. In: VLDB, pp. 185–196 (2010)
28. Herschel, M., Hernandez, M.A., Tan, W.C.: Artemis: A system for analyzing missing answers. In: VLDB, pp. 1550–1553 (2009)
29. Hristidis, V., Koudas, N., Papakonstantinou, Y.: Prefer: A system for the efficient execution of multi-parametric ranked queries. In: SIGMOD, pp. 259–270 (2001)
30. Hristidis, V., Papakonstantinou, Y.: Algorithms and applications for answering ranked queries using ranked views. VLDB J. **13**(1), 49–70 (2013)
31. Huang, J., Chen, T., Doan, A.H., Naughton, J.F.: On the provenance of non-answers to queries over extracted data. In: VLDB, pp. 736–747 (2008)

32. Islam, M., Liu, C., Li, J.: Efficient answering of why-not questions in similar graph matching. IEEE Trans. Knowl. Data Eng. **27**(10), 2672–2686 (2015)
33. Islam, M.S., Zhou, R., Liu, C.: On answering why-not questions in reverse skyline queries. In: ICDE, pp. 973–984 (2013)
34. Jagadish, H.V., Chapman, A., Elkiss, A., Jayapandian, M., Li, Y., Nandi, A., Yu, C.: Making database systems usable. In: SIGMOD, pp. 13–24 (2007)
35. Jin, C., Zhang, R., Kang, Q., Zhang, Z., Zhou, A.: Probabilistic reverse top-k queries. In: DASFAA, pp. 406–419 (2014)
36. Koh, J.L., Lin, C.Y., Chen, A.L.P.: Finding k most favorite products based on reverse top-t queries. VLDB J. **23**(4), 541–564 (2014)
37. Meliou, A., Gatterbauer, W., Moore, K.F., Suciu, D.: Why so? or why no? functional causality for explaining query answers. In: MUD, pp. 3–17 (2010)
38. Monteiro, R.D.C., Adler, I.: Interior path following primal-dual algorithms, part ii: Convex quadratic programming. Math. Program. **44**(1-3), 43–66 (1989)
39. Padmanabhan, V., Rajiv, S., Srinivasan, K.: New products, upgrades, and new releases: A rationale for sequential product introduction. J. Marketing Research **34**(4), 456–472 (1997)
40. Tan, W.C.: Provenance in databases: Past, current, and future. IEEE Data Eng. Bull. **30**(4), 3–12 (2007)
41. Tao, Y., Hristidis, V., Papadias, D., Papakonstantinou, Y.: Branch-and-bound processing of ranked queries. Inf. Syst. **32**(3), 424–445 (2007)
42. Tran, Q.T., Chan, C.Y.: How to conquer why-not questions. In: SIGMOD, pp. 15–26 (2010)
43. Vlachou, A., Doulkeridis, C., Kotidis, Y., Norvag, K.: Monochromatic and bichromatic reverse top-k queries. IEEE Trans. Knowl. Data Eng. **23**(8), 1215–1229 (2011)
44. Vlachou, A., Doulkeridis, C., Norvag, K.: Monitoring reverse top-k queries over mobile devices. In: MobiDE, pp. 17–24 (2011)
45. Vlachou, A., Doulkeridis, C., Norvag, K., Kotidis, Y.: Identifying the most influential data objects with reverse top-k queries. In: VLDB, pp. 364–372 (2010)
46. Vlachou, A., Doulkeridis, C., Norvag, K., Kotidis, Y.: Branch-and-bound algorithm for reverse top-k queries. In: SIGMOD, pp. 481–492 (2013)
47. Xie, M., Lakshmanan, L.V.S., Wood, P.T.: Efficient top-k query answering using cached views. In: EDBT, pp. 489–500 (2013)
48. Xin, D., Chen, C., Han, J.: Towards robust indexing for ranked queries. In: VLDB, pp. 235–246 (2006)
49. Yu, A., Agarwal, P.K., Yang, J.: Processing a large number of continuous preference top-k queries. In: SIGMOD, pp. 397–408 (2012)
50. Zong, C., Yang, X., Wang, B., Zhang, J.: Minimizing explanations for missing answers to queries on databases. In: DASFAA, pp. 254–268 (2013)
51. Zou, L., Chen, L.: Dominant graph: An efficient indexing structure to answer top-k queries. In: ICDE, pp. 536–545 (2008)