The Power of Two: Simplified User Interaction for the Indistinguishability Query

Lam Do
Denison University
Granville, OH, USA
do_k2@denison.edu

Oghap Kim
Denison University
Granville, OH, USA
kim_o1@denison.edu

Chloe Chai
Denison University
Granville, OH, USA
chai_c1@denison.edu

Ashwin Lall
Denison University
Granville, OH, USA
lalla@denison.edu

Abstract

We propose a simplified user interaction framework for the indistinguishability query. The indistinguishability query determines all of the user's near optimal tuples without explicit knowledge of their utility function. This approach uses a simple interactive framework where the user picks their favorite tuple during rounds of questions. We propose an alternative interactive framework that is more truthful and focuses on comparing two attributes at a time. This allows us to use more powerful 2D techniques for higher-dimensional data while also simplifying the decision-making on the part of the user.

We provide a strongly truthful algorithm that displays the user with only real tuples from the database. In addition, we give an algorithm that displays synthetic tuples that are more realistic than previous work. We introduce a new definition — tolerably truthful — that guarantees that all tuples have attribute values within the ranges found within the database. Our tolerably truthful algorithm has a provable approximation guarantee for the indistinguishable output set. We also verify the efficacy of our algorithms with experiments on both real and synthetic data sets and through a user study.

ACM Reference Format:
Lam Do, Oghap Kim, Chloe Chai, and Ashwin Lall. 2025. The Power of Two: Simplified User Interaction for the Indistinguishability Query. In *The International Conference on Scalable Scientific Data Management 2025 (SSDBM 2025), June 23–25, 2025, Columbus, OH, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3733723.3733725

1 Introduction

Selecting the best option among many can be challenging, especially when there are numerous attributes. In order to solve this problem, existing multi-criteria decision-making techniques allow users to look at a manageable number of options interesting to them from the database without having to sift through all possible alternatives, allowing the user to save time and energy. Suppose that an energy consultant, Erin, wants to choose an energy source with high energy efficiency (EE) and reliability rating (RR). The challenge is that she does not know her relative valuation of these attributes, and it may also be time-consuming to look at all the options in a database to find her favorite. Traditional methods assume that Erin has a utility function and utilize various techniques to assist her in identifying her optimal choice. These traditional approaches include *top-k*, *skyline*, and *regret minimization*.

Top-k [11], which relies on having Erin's utility function, computes and returns k tuples with the highest utility in the database. However, requiring an explicit utility function from Erin may be unreasonable because it is difficult for her to know her exact relative valuation for each interesting attribute.

Another method of querying which does not rely on having Erin's utility function is the *skyline operator* [3], which displays the tuples in the database that are not dominated by any other tuple. A tuple dominates another when its values are at least as good in all attributes and strictly better than the other in at least one attribute. The collection of non-dominated tuples is called the "skyline". Although this approach displays tuples contained in the skyline which could be considered the "best" of the database, the output is not personalized to Erin's preferences and therefore may include many tuples that are not interesting to her. Moreover, it is not possible to control the output size of the skyline operator; thus, the final result from the query may be too time-consuming for Erin to look through when the skyline set is large.

The *regret minimization technique* [20] outputs k tuples that aim to reduce the maximum "regret ratio". The regret ratio indicates how disappointed the user is with the final result of tuples shown to them compared to looking at the entire database [21]. Even though regret minimization includes some near-optimal tuple for the user, it does not consider all of the near-optimal tuples and may cause Erin to miss out on tuples that appear interesting to her. Therefore, Erin may find a near favorite tuple in the final output set but the rest of the tuples may be unexciting to her. This can lead Erin to feel like she is missing out on other tuples that would have been a good choice for her.

In this paper, we adopt the Indistinguishability Query [14] to address the Fear of Missing Out (FOMO), where users may worry that the database query might exclude good options, causing them to miss out on interesting tuples. The Indistinguishability Query guarantees that all optimal and near-optimal tuples are returned to

SSDBM 2025, June 23–25, 2025, Columbus, OH, USA

ACM ISBN 979-8-4007-1462-7/25/06
https://doi.org/10.1145/3733723.3733725

the user without the explicit knowledge of their utility function. It involves an initial interactive phase that learns the user's implicit utility function by repeatedly asking them to choose their favorite out of a number of tuples displayed. The interactive framework is based on a widely-recognized assumption [14, 20, 32] that it is easier for the user to choose their favorite from a small selection of choices than to know their exact utility function. Then, the query outputs a set of tuples that are ϵ-indistinguishable from the optimal. For a pair of tuples to be considered ϵ-indistinguishable, the utility of those tuples needs to be within $(1 + \epsilon)$ multiplicative factor of each other. Since the goal of the Indistinguishability Query is to find the optimal tuple along with all the near-optimal tuples, Erin is shown the other near-optimal options even if they are slightly worse than optimal, for they may have other appealing attributes to Erin such as land use or availability. For instance, Erin may find a source with slightly lower EE and RR more enticing if it has low land use.

Motivated by this novel approach, we propose an alternative interactive framework that focuses on comparing two attributes at a time as opposed to all at once. In [32], a halfspace technique was used to narrow down all of the coefficients of the user's utility function after every question. Another previous work [14] attempted to narrow down the feasible region of the utility function that captures all coefficients in each round. This paper aims to work with only two dimensions at a time. This allows us to use geometric 2D techniques while still allowing us to process data in high dimensions. Moreover, there is a benefit to the user to make head-to-head comparisons between two attributes, rather than trying to evaluate many different attributes at once. In addition, we follow the convention of [24] in showing the user only two options at a time, further simplifying their decision-making.

We argue that presenting Erin with two energy sources at a time with just two attributes makes her decisions considerably easier. For example, if Erin is interested in four attributes: Energy Efficiency (EE), Resource Availability (RA), Price Stability (PS), and Reliability Rating (RR), previous interactive methods [14, 20, 32] would display sources along with all of their specs like this:

- Source 1: 22 EE, 34 RA, 200 PS, 5 RR
- Source 2: 20 EE, 28 RA, 220 PS, 3 RR
- Source 3: 25 EE, 36 RA, 180 PS, 4 RR

In contrast, our algorithm focuses on showing only two sources at a time with two attributes each. For example:

- Source 1: 34 RA, 5 RR
- Source 2: 28 RA, 3 RR

This head-to-head comparison method leverages the simplicity of binary choices, which are cognitively easier and faster for users to make [8].

In the performance evaluation section, we demonstrate that our algorithm delivers excellent performance in finding the user's indistinguishability set. Altogether, our algorithm requires less effort from the user, making the interaction easier and yielding results comparable to previous methods. We verified this with a user study.

Our work also aims to improve user satisfaction by being more truthful than previous works. For an algorithm to be considered strongly truthful, this means that every tuple that the user sees throughout the interactive process must exist in the database. In weakly truthful algorithms, artificial tuples may be used to give provable performance guarantees, however they may never show the user a tuple that is better than the user's optimal in the final output as this would lead the user to feel that there has been a bait-and-switch.

A significant deficiency of prior weakly truthful algorithms [14] is that they may use unrealistic attribute values (e.g. 200 EE and 0 RR) for the sake of extracting the most information from each question. Seeing impractical tuples may be off-putting to the user, discouraging them from continuing. Motivated by this deficiency, we propose a new definition of truthfulness: *tolerably truthful*. A tolerably truthful algorithm only displays realistic tuples that are created with attribute values in the same range as real tuples from the database. This approach makes artificial tuples more realistic.

Contributions: The main contributions of this paper are:

- We introduce a simplified user interaction framework for multi-criteria decision-making.
- We define a new concept of truthfulness called *tolerably truthful* (TT).
- We give a strongly truthful and a tolerably truthful algorithm for executing the indistinguishability query.
- We illustrate performance evaluations for all the above algorithms on real and artificial data sets. We also include results from a small user study.

Organization: We address the related works in the next section. In Section 3, we provide definitions, including the formal problem definition. We present the strongly truthful Breakpoint algorithm that uses real tuples in Section 4 and the tolerably truthful algorithm for performing the query using realistic artificial tuples in Section 5. We show the efficacy of our algorithms through experiments on real and synthetic datasets in Section 6. Conclusions and future work can be found in Section 7.

2 Related Work

Many efforts have been made to improve methods of multi-criteria based decision making.

Top-k and Skyline. There has been numerous studies done on the top-k query [9, 15, 16, 25, 27]. However, this query becomes difficult to execute whenever the user does not know their specific utility function. Therefore, many methods have been made to account for this downside, including the skyline query [3–5, 10, 15, 17, 18, 22, 28, 29, 31]. By using dominating tuples, the skyline query assures that it returns the user's optimal tuple. Yet, the final output includes numerous tuples from the skyline that may appear uninteresting to the user and misses out on the user's near-optimal tuples. Additionally, there is no way of controlling the output set, making it harder for the user to go through the options when there is a large number of undominated tuples. In contrast, the Indistinguishability Query includes both optimal and near-optimal tuples.

Regret minimization. Regret Minimization [1, 2, 7, 13, 21, 23, 33] aims to display a small number of tuples that still guarantee a small maximum regret ratio. The maximum regret ratio is the percentage of regret that a user faces on seeing just the small set rather than looking through all the tuples in the worst case. Unlike the skyline operator, it allows control over the size of the final output shown

to the user. However, it does not include all of the near-optimal tuples.

User Interaction. Interactive techniques [20, 26, 30, 32] utilize both artificial and real tuples from the database and outputs some near-optimal tuples to the user, but not all of them. Compared with existing methods, our solution aims to output the same indistinguishable set, however our user interaction limits the number of tuples shown to the user in each round to two, motivated by the idea it is easier for a user to compare two tuples than for them to compare many tuples at one time [12, 24].

Indistinguishability. Rather than discovering one of the user's near-optimal tuples, or even their optimal tuple, the Indistinguishability Query [14] uses interaction to find *all* of the user's near-optimal tuples so that they do not have to fear missing out on a good option. The size of the output can be controlled by a parameter ϵ that indicates how far from the optimal the output tuples may lie. The user may prefer one of these tuples to their optimal when taking into account unconsidered criteria such as land use or availability. While the query allows for strongly truthful algorithms, a result [14] demonstrates that showing only real tuples can result in an arbitrary number of false positives. To get around this, the paper shows the user artificial tuples; however, these tuples may not be truthful and could include tuples with unrealistic values (e.g., 0 for RA or 1000 for EE), which may lead to the user feeling dissatisfied if they are shown tuples that are not actually available to them, or that are not plausibly real options.

To resolve this problem, this paper focuses on different ways and approaches to make the user interaction phase more truthful. We propose both strongly truthful and *tolerably truthful* algorithms that displays realistic tuples to the user.

3 Preliminaries

3.1 Definitions

In this paper, we refer to the set of tuples (sometimes called points) in the input database as D, where the size of D is n ($|D| = n$), and each tuple in D has d attributes or dimensions. The set D is a subset of $\mathbb{R}_+^d$. The number of attributes of interest selected by the user, d, may be less than the total number of attributes in the database. We assume that higher values are more desirable for each attribute. If there are attributes where lesser values are considered better, like carbon footprint or price, we can adjust those attributes by subtracting their values from the maximum. We do not consider categorical attributes in this work as there is no standard way to assign values to categories.

Example: Erin is looking to choose an energy source and selects three attributes energy efficiency (EE), resource availability (RA), and reliability rating (RR) ($d = 3$) out of a large number of attributes in a large database.

We can express the user's unknown utility function as $f : \mathbb{R}_+^d \to \mathbb{R}_+$, which when applied calculates the user's utility of a given point. The utility function is commonly represented by a linear function [6, 7, 21, 32] represented as $f(p) = u \cdot p$, where $u \in \mathbb{R}_+^d$.

Example: Erin has an unknown utility such as

$$f(EE, RA, RR) = 2EE + 4RA + 20RR,$$

with utility vector $u = (2, 4, 20)$.

source	EE	RA	RR	2EE + 4RA + 20RR
s_1	22	34	4	260
s_2	20	28	5	252
s_3	29	34	3	254
s_4*	34	40	2	268
s_5	20	25	3	200

Table 1: Example indistinguishability query. The highlighted tuples are 0.05-indistinguishable from the optimal (c_4) for a user with utility function 2EE + 4RA + 20RR.

One of the contributions of this work is to model what happens when showing the user only pairs of attributes at a time. In such circumstances, we assume that the user's utility for the attributes shown is computed with the same coefficients but with only the attributes shown.

Example: If Erin has utility function $f(EE, RA, RR) = 2EE + 4RA + 20RR$, but is only shown attributes EE and RR for a source, she will compute the (partial) utility of this option as $f'(EE, RR) = 2EE + 20RR$.

The idea of ϵ-indistinguishability was defined in [14].

Definition 3.1. (ϵ-indistinguishability) [14] Given a utility function f and some $\epsilon > 0$, we say that two tuples p_1 and p_2 are ϵ-indistinguishable if

$$f(p_1) \leq (1 + \epsilon) f(p_2)$$

and

$$f(p_2) \leq (1 + \epsilon) f(p_1).$$

This means that for some $\epsilon > 0$, the user cannot discern between two tuples that have a utility within a factor $(1 + \epsilon)$ of each other.

Example: Two sources with attributes 22 EE, 34 RA, 4 RR and 20 EE, 28 RA, 5 RR have utilities 260 and 252 with the above utility function for Erin. These tuples are 0.05-indistinguishable to Erin as $260 \leq 252(1 + 0.05)$.

Our goal is to compute and return the set of points that are ϵ-indistinguishable from the optimal point for a user based on their utility function. This is called the indistinguishability query [14].

Definition 3.2. (Indistinguishability Query) [14] For any database D and $\epsilon > 0$ and unknown utility function f, for which the unknown optimal tuple in D is $p^* = \arg\max_{p \in D} f(p)$, we define

$$\mathcal{I}_{f,\epsilon} = \{p \in D : p \text{ and } p^* \text{ are } \epsilon\text{-indistinguishable}\}.$$

We consider $\mathcal{I}_{f,\epsilon}$ to be the set of tuples from D that includes the user's optimal point and all points that are ϵ-indistinguishable from it. The set $\mathcal{I}_{f,\epsilon}$ may be abbreviated to $\mathcal{I}$ for clarity when f and ϵ are clear from the context.

Example: Consider the example database in Table 1. If Erin's utility function is 2EE + 4RA + 20RR, then the indistinguishability query with $\epsilon = 0.05$ outputs Erin's optimal source (s_4 with utility 268) and all sources with utility at least $268/1.05 \approx 255.2$, or the set $\{s_1, s_4\}$.

Definition 3.3. (α-approximation) [14] A set S is an α-approximation for $\mathcal{I}$ if $I \subseteq S$ and for each $p' \in S - \mathcal{I}$ we have that

$$p^* \cdot u - (1 + \epsilon) p' \cdot u \leq \alpha,$$

Notation	Meaning
D	set of all tuples, $\|D\| = n$
d	number of attributes
u	user's utility vector
p^*	optimal point for u
ϵ	user's desired indistinguishability threshold
$\mathcal{I}_{u,\epsilon}$ or $\mathcal{I}$	ϵ-indistinguishable optimal points
q	number of rounds of questions asked
α	approximation factor for a non-exact solution

Table 2: Notation used in this paper

where $f(p) = u \cdot p$ is the user's linear utility function and $p^* = \arg\max_{p \in D} p \cdot u$.

While an α-approximation allows a few tuples that are not ϵ-indistinguishable from the optimal to be output, the $\mathcal{I} \subseteq S$ condition means that it never omits any elements of $\mathcal{I}$ (i.e., there are no false negatives) so that the user can rest assured that they are not missing any tuples of interest. This form of approximation guarantees that any additional tuples that are included are very close to being ϵ-indistinguishable. As α approaches zero, the approximation approaches the set $\mathcal{I}$.

Example: Suppose Erin's favorite source has utility 268 and the query is run with $\epsilon = 0.05$. A 2.5-approximation would guarantee that the utility value v of any output tuple would be such that $268 - (1 + .05)v \leq 2.5$ or $v \geq (268 - 2.5)/1.05 \approx 252.9$, rather than $268/1.05 \approx 255.2$ desired by the 0.05-indistinguishability query. For the example in Table 1, this would mean that this approximation would also include the source s_3 in the output set.

The notation defined in this section is given in Table 2.

3.2 User Interaction

To find the set of indistinguishably optimal points, we will need to ask the user questions to narrow down her utility function. Similar to [14, 20, 32], we model this interaction in the form of rounds in which the user is shown a small subset of tuples for q rounds (sometimes called questions) and is asked for their favorite tuple in each round. After the last round, the final set of tuples is output to the user. Note that in each round the user is only shown the attributes that they indicated were of interest to them. While it is unreasonable to expect the user to produce their utility function, it is reasonable to ask them to compare a pair of tuples and pick their favorite.

Example: Erin is interested in three attributes of an energy source: EE, RA, and RR. In an interactive round, the system shows Erin two sources : Source 1 (24 EE, 34 RA, 4 RR) and Source 2 (20 EE, 28 RA, 5 RR). Erin will select the source that has a higher utility for her. She will continue this process for a fixed number of rounds q or until she terminates the process. At the conclusion, she is shown a set of tuples of interest to her based on what was learned during the interactive rounds.

One of our contributions in this paper is to simplify this interactive process to show the user only two attributes at a time. The advantage for the user is that they can look at fewer values and make a head-to-head comparison of their tradeoffs between pairs of attributes. This method also improves the performance of our algorithms as we can more directly compare pairs of components of the user's utility function. We are also able to use 2D geometric techniques that are infeasible in higher dimensions. In addition, we restrict the number of tuples shown in each round to two.

Example: In one round, Erin may be asked to compare Source 1 (22 EE, 4 RR) and Source 2 (20 EE, 5 RR). In another round Erin may be asked to compare Source 1 (34 RA, 4 RR) and Source 2 (28 EE, 5 RR).

3.3 Tolerably Truthful

While the interactive rounds would ideally show the user real tuples in the database, this may not be amenable to computing the indistinguishability query very accurately. The tuples in the database may not be suited for estimating the user's utility function meaning that we would have no provable bounds on the accuracy of the output. Moreover, it has been shown [14] that there exist databases for which any algorithm that outputs all of $\mathcal{I}$ may also output an arbitrary number of false positives, many of which may be very far from the user's optimal. For these reasons, we also consider showing the user artificially constructed tuples.

In [20], the authors propose the idea of *truthful* artificial tuples, defined as tuples that are not superior to the user's favorite in the eventual output set. The reason for this definition is that it would be undesirable to show the user unrealistically good options to make them excited about continuing to search only to have inferior options at the end of the search process. The user may feel as though a bait-and-switch scheme has been perpetrated. By guaranteeing that the artificial tuples are no better than the user's ultimate optimal, we avoid any such issues.

A weakness of the definition of truthful is that it allows for highly unrealistic values in the artificial tuples. For example, it allows an algorithm to show impossible values such as 0 reliability rating or 250 energy efficiency when constructing artificial tuples. These might be unsettling to the user and cause them to stop using the interactive process. To get around this issue, we propose the following definition.

Definition 3.4. (Tolerably Truthful) An algorithm is tolerably truthful if it is truthful and it only displays tuples p with the following property for all dimensions $1 \leq i \leq d$:

$$min_i \leq p[i] \leq max_i,$$

where min_i and max_i are the minimum and maximum values for dimension i in the database, respectively.

First, a tolerably truthful algorithm is truthful. According to [20], an algorithm is considered truthful if the tuples shown during interaction are no better for the user than the user's eventual favorite tuple from the final output set. Second, a tolerably truthful algorithm displays tuples with values bounded by the minimum and maximum values of real points in the database.

Example: If the min/max values in that database of attributes EE and RR are 15/50 and 2/5, respectively, then a tolerably truthful algorithm may show Erin a source with EE of 25 and RR of 4, even if no such source exists in the database as long as this hypothetical source is not better than Erin's favorite source in the final output set.

3.4 Breakpoints

As mentioned earlier, we will show users only two attributes at a time to learn their utility function. Recall that the utility of a tuple is defined by a utility vector u so that the utility of a point p is given by $u \cdot p$. When showing only a pair of attributes, we assume that the user's utility is given by the same utility vector coefficients. For example, when shown only attributes i and j, the user's utility on these two attributes will be computed as $u[i] \cdot p[i] + u[j] \cdot p[j]$.

This gives us a way to bound the ratio of the components of the user's utility function. For each dimension i, we will aim to find each ratio $\frac{u[i]}{u[a]}$ with respect to a chosen anchor dimension a. If the user prefers point p to point q when only dimensions i and a are shown to them, we have:

$$u[a] \cdot p[a] + u[i] \cdot p[i] \geq u[a] \cdot q[a] + u[i] \cdot q[i]$$

or

$$\frac{u[i]}{u[a]} \geq \frac{q[a] - p[a]}{p[i] - q[i]},$$

if $p[i] - q[i] > 0$ and otherwise

$$\frac{u[i]}{u[a]} \leq \frac{q[a] - p[a]}{p[i] - q[i]}.$$

Example: Suppose Erin chooses Source 1 (4 RR, 24 EE) over Source 2 (5 RR, 20 EE). Then we know that the coefficients of her utility function for attributes EE and RR (with anchor RR), given by $u[EE]$ and $u[RR]$, are such that

$$\frac{u[EE]}{u[RR]} \geq \frac{5-4}{24-20} = \frac{1}{4}.$$

We call the value $\frac{q[a]-p[a]}{p[i]-q[i]}$ a breakpoint and note that it can be related to the slopes of the points p and q when restricted to dimensions a and i:

$$\frac{q[a] - p[a]}{p[i] - q[i]} = \frac{-1}{slope_{a,i}(p, q)},$$

where the slope of two points p and q when restricted to dimensions a, i is:

$$slope_{a,i}(p, q) = \frac{q[i] - p[i]}{q[a] - p[a]}.$$

Example: In the previous example with Source 1 (4 RR, 24 EE) over Source 2 (5 RR, 20 EE), the slope of these two points is $(24 - 20)/(4 - 5) = -4$. We can verify from the previous example that the breakpoint is $-1/slope = 1/4$.

By using this breakpoint method to narrow down these ratios, we can reconstruct the user's utility function normalized to the anchor attribute's coefficient $u[a]$:

$$\frac{u[1]}{u[a]}p[1] + \frac{u[2]}{u[a]}p[2] + \ldots + \frac{u[a]}{u[a]}p[a] + \cdots + \frac{u[d]}{u[a]}p[d] = \frac{u \cdot p}{u[a]}.$$

Since normalizing the utility function by a constant doesn't change the relative evaluation of tuples (e.g., the optimal tuple remains the same), this method allows us to estimate the user's utility function.
Example: Suppose Erin has a utility function $f(EE, RA, RR) = 2EE + 4RA + 20RR$ or utility vector $u = (2, 4, 20)$ and we choose RR as the anchor dimension, then we estimate the re-normalized utility vector $(2/20, 4/20, 20/20)$.

3.5 Problem Definition

We are now ready to formally define the problem solved in this paper:
Problem definition: Given a database of n tuples $D \subseteq \mathbb{R}_+^d$ and indistinguishability parameter $\epsilon > 0$, approximately compute a user's ϵ-indistinguishable set $\mathcal{I}$ by interactively asking the user to pick their favorite tuple out of two tuples while showing only two attributes at a time.

4 Strongly Truthful Algorithm

In this section, we present a strongly truthful heuristic algorithm designed to estimate a user's utility through interaction and output an approximate indistinguishability set. The algorithm works by choosing an anchor dimension a and then estimating each of the values $u[i]/u[a]$ $(1 \leq i \leq d)$ as accurately as possible. These estimates are used to then prune out tuples that are not in $\mathcal{I}$ given what we know about the estimated utility function.

Our algorithm uses as a subroutine an algorithm [19] (that we call CountSlopes) that takes as input a set of n 2D points and a range $[l, h]$ and outputs the number of pairs of points that have slope within this range. The algorithm cleverly reduces the problem of counting slopes to that of counting inversions in a list, which can then be solved in $O(n \log n)$ time by using an algorithm based on Merge Sort.

Our algorithm is given in Algorithm 1. It begins by selecting an anchor dimension (Lines 1-2). The anchor dimension is chosen as the one with the highest minimum number of breakpoints (i.e. number of slopes between $[-\infty, 0]$) with all other dimensions. This ensures that the chosen dimension has the most breakpoints available for further interaction, which helps refine utility estimates.

After determining the anchor dimension a, the algorithm proceeds to estimate utility ratios $\frac{u[i]}{u[a]}$ for all dimensions $1 \leq i \leq d, i \neq a$. This is done by bounding the possible ranges for these ratios, with $L_i = 0$ as the lower bounds and $H_i = \infty$ as the upper bounds initially. The algorithm maintains the invariant $L_i \leq \frac{u[i]}{u[a]} \leq H_i$. For the anchor dimension, the ratio $\frac{u[a]}{u[a]}$ is always 1, thus $L_a = H_a = 1$.

The main loop (lines 5-18) updates the L_i and H_i values based on rounds of user interaction. In each round, the algorithm prioritizes the dimension with the widest range of utility ratios $[L_i, H_i]$ (i.e., largest $H_i - L_i$). To refine the bounds, the algorithm counts the number of slopes within the range $\left[\frac{-1}{L_i}, \frac{-1}{H_i}\right]$ with the CountSlope algorithm and attempts to find the median slope that divides the number of breakpoints in half. Instead of calculating all slopes between every point in the database (which would be computationally expensive), the algorithm uses a heuristic approach of sampling points T times and selecting the pair that best approximates the desired median. After the user chooses the point they prefer out of the two displayed, the algorithm updates the bounds for L_i or H_i appropriately. The process then repeats for the next dimension with the widest range.
Example: Suppose the database has just four tuples with (EE, RR) values $s_1(20, 5)$, $s_2(25, 3)$, $s_3(30, 4)$, and $s_4(40, 2)$. The possible breakpoints for u_{EE}/u_{RR} are (s_1 and s_2) $(25 - 20)/(5 - 3) = 2.5$, (s_1 and s_3) $(30 - 20)/(5 - 4) = 10$, (s_1 and s_4) $(40 - 20)/(5 - 2) \approx 6.67$, ($s_2$ and s_4) $(40 - 25)/(3 - 2) = 15$, and (s_3 and s_4) $(40 - 30)/(4 -$

2) = 5. Note that s_2 and s_3 give a negative breakpoint (since s_3 dominates s_2) and so we exclude this case. The algorithm will start by showing the tuples that give the median breakpoint (s_1 and s_4 with breakpoint 6.67) so that it can narrow down the number of remaining breakpoints by half with each question.

Finally, the algorithm removes points that are $(1+\epsilon)$-dominated by any other points for all utility functions that have $u_i \in [L_i, H_i]$. We follow the heuristic procedure proposed in [14]. Instead of comparing the utility of all points p to that of all other points q in D to see if they are $(1+\epsilon)$-dominated (which would again be computationally expensive), we are going to take the lower bound L_i of all dimensions, apply that on all points in the database and keep track of highest value utility, $V = \max_{p \in D} p \cdot (L_1, ..., L_d)$. Then, we scale all points p up by $(1+\epsilon)$ and calculate their utility with $(H_1, \ldots, H_d)$. If they are lower than V, it means that they cannot possibly be in the indistinguishability set, and we prune them out. Put another way, we removed all points in the set $\{p \in D : (1+\epsilon) \cdot p \cdot (H_1, ..., H_d) < V\}$.

Example: Suppose after several rounds of questions we have narrowed down the user's utility function $u \in [2.25, 2.5] \times [0.75, 0.8] \times [1, 1]$ (where u is normalized to the third dimension as anchor). We use the lower bound utility function $u_L = (2.25, 0.75, 1)$ to determine the largest utility value among all tuples in the database, V (i.e., V is a lower bound on the utility of the optimal tuple). We then prune all tuples whose utility with utility function $u_H = (2.5, 0.8, 1)$ is less than $V/(1+\epsilon)$ since such tuples cannot possibly be in the final set $\mathcal{I}$.

4.1 Running Time

The running time of the first part of the algorithm (Lines 1-2) is $O(d^2 n \log n)$ since CountSlopes runs in $O(n \log n)$ time.

The `while` loop on Line 5 runs q times. We repeat Lines 8-13 T times. Our goal is to reduce T to a small number without negatively impacting the performance of the algorithm. In Section 6, we show that it is reasonable to pick $T = 100$. In Lines 7 and Line 11, we use the algorithm CountSlopes that runs in $O(n \log n)$ time. This makes the main `while` loop take $O(qTn \log n)$ time.

The final pruning out step in Lines 19-21 is adapted from [14], and has a running time of $O(nd)$.

Overall, the running time of Breakpoint is $O(d^2 n \log n + qTn \log n)$.

4.2 Proof of Correctness

We show that the invariant $L_i \le \frac{u[i]}{u[a]} \le H_i$ is maintained using the following two lemmas for updating L_i and H_i in Lines 15-18 of Algorithm 1.

Lemma 4.1. *If the user prefers point p_1, then $\frac{u[i]}{u[a]} \ge \frac{-1}{\sigma}$.*

Proof. If the user prefers point p_1, we know that

$$\begin{aligned} u[i] \cdot p_1[i] + u[a] \cdot p_1[a] &\ge u[i] \cdot p_2[i] + u[a] \cdot p_2[a], \quad \text{or} \\ \frac{u[i]}{u[a]} \cdot (p_1[i] - p_2[i]) &\ge p_2[a] - p_1[a], \quad \text{or} \\ \frac{u[i]}{u[a]} &\ge \frac{p_2[a] - p_1[a]}{p_1[i] - p_2[i]} \ (\text{as } p_1[i] > p_2[i]) \\ \frac{u[i]}{u[a]} &\ge \frac{-1}{\sigma}. \end{aligned}$$

Algorithm 1 Breakpoint(D, s, ϵ, T, a)

Input: A database of tuples, $D \subset [0,1]^d$; anchor dimension a, indistinguishability parameter ϵ, number of samples T
Output: A subset of D

1: For each $1 \le i \le d$ and $1 \le j \le d$, $i \ne j$, count the number of slopes between $-\infty$ and 0 for dimension i and j, denoted as $count(i, j)$.
2: Set anchor dimension $a = \arg\max_{1 \le i \le d} \min_{j \ne i} count(i, j)$.
3: For each dimension, $1 \le i \le d$, $i \ne a$, let $L_i = 0$ and $H_i = \infty$.
4: Let $L_a = 1$, $H_a = 1$.
5: **while** user is still giving feedback **do**
6: Find the dimension i with the largest $H_i - L_i$.
7: Let μ_i be the number of slopes in range $[\frac{-1}{L_i}, \frac{-1}{H_i}]$.
8: **repeat**
9: Randomly select 2 tuples p_1, p_2 from D such that $p_1[i] > p_2[i]$ and compute their slope σ.
10: **if** $\frac{-1}{L_i} \le \sigma \le \frac{-1}{H_i}$ **then**
11: Define mid as the number of slopes in range $[\frac{-1}{L_i}, \sigma]$.
12: Store p_1 and p_2 and their slope σ if the value $|\frac{\mu_i}{2} - mid|$ is minimized.
13: **until** T attempts have been made.
14: Display p_1, p_2 to the user.
15: **if** the user chooses p_1 **then**
16: Update $L_i = \frac{-1}{\sigma}$.
17: **else**
18: Update $H_i = \frac{-1}{\sigma}$.
19: Initialize subset $C = D$.
20: **for all** $p \in D$ **do**
21: Remove all points from C that are $(1+\epsilon)$-dominated by p for all utility functions in $[L_i, H_i]$.
22: **return** C.

Thus, the new lower bound of the ratio $\frac{u[i]}{u[a]}$ is $\frac{-1}{\sigma}$. If the user prefers point p_1, we update $L_i = \frac{-1}{\sigma}$.

□

Lemma 4.2. *If the user prefers point p_2, then $\frac{u[i]}{u[a]} \le \frac{-1}{\sigma}$.*

Proof. If the user prefers point p_2, we know that

$$\begin{aligned} u[i] \cdot p_1[i] + u[a] \cdot p_1[a] &\le u[i] \cdot p_2[i] + u[a] \cdot p_2[a], \quad \text{or} \\ \frac{u[i]}{u[a]} \cdot (p_1[i] - p_2[i]) &\le p_2[a] - p_1[a], \quad \text{or} \\ \frac{u[i]}{u[a]} &\le \frac{p_2[a] - p_1[a]}{p_1[i] - p_2[i]} \ (\text{as } p_1[i] > p_2[i]) \\ \frac{u[i]}{u[a]} &\le \frac{-1}{\sigma}. \end{aligned}$$

Thus, the upper bound of the ratio $\frac{u[i]}{u[a]}$ is $\frac{-1}{\sigma}$. If the user prefers point p_2, we update $H_i = \frac{-1}{\sigma}$.

□

Theorem 4.3. *The invariant $L_i \le \frac{u[i]}{u[a]} \le H_i$ is maintained throughout interaction.*

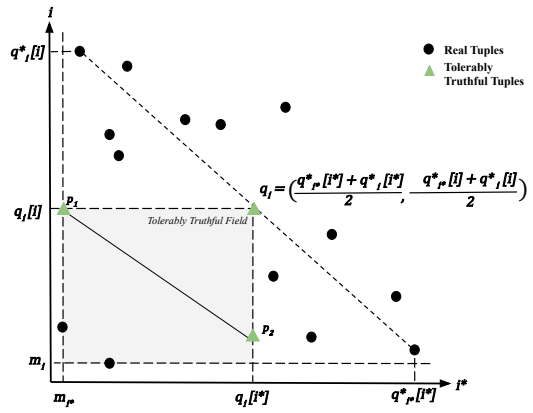


Figure 1: TT - Breakpoint Algorithm Depiction

Proof. L_i and H_i is initialized to 0 and ∞ before the loop starts, with the exception of $L_a = H_a = 1$. We know that $L_i \leq \frac{u[i]}{u[a]} \leq H_i$, thus, the invariant holds during initialization.

Throughout interaction, the user prefers either p_1 or p_2 in each round. Since we proved that L_i and H_i are updated correctly based on the user's implicit utility function in Lemma 4.1 and Lemma 4.2, we conclude that the invariant $L_i \leq \frac{u_i}{u_a} \leq H_i$ is maintained throughout interaction. □

While the strongly truthful Breakpoint algorithm does a good job of narrowing down each $\frac{u[i]}{u[a]}$ value, there is no guarantee for how well it can approximate the set $\mathcal{I}$. This is because we can make no assumptions about the dataset and, in the worst case, it may perform poorly. Still, we are able to show that the Breakpoint algorithm does reasonably well on some real data sets in Section 6. In order to get an approximation guarantee on $\mathcal{I}$, we need to introduce artificial tuples, which we do in the next section.

5 Tolerably Truthful Algorithm

In this section, we present a tolerably truthful algorithm that approximates the indistinguishability set with an asymptotic guarantee. The algorithm narrows down the range of the unknown user's utility function by asking rounds of questions, pruning out tuples that are significantly far away from being ϵ-indistinguishable from the user's optimal.

In order to remain tolerably truthful, we have to choose the tuples shown to the user very carefully. For any pair of dimensions i and i^*, we construct what we call a *tolerably truthful field*, a rectangle within which we are guaranteed that all tuples help us maintain the property of remaining tolerably truthful. This is illustrated in Figure 1. The bottom left of the rectangle is defined to be the minimum value of the two attributes (m_i and m_i^* in the figure) so that all tuples in the field have values greater than the minimum attributes. The upper right of the rectangle is defined to be the midpoint of the tuples with maximum values in the respective attributes (q_i in the figure). This guarantees that all tuples in the rectangle are less than the maximum attributes. We will also prove that the utility of this tuple q_i is no greater than the utility of the user's optimal tuple in the output set , thus preserving the property of being truthful.

Algorithm 2 TT-Breakpoint(D, ϵ)

Input: A database of tuples, $D \subset [0,1]^d$, indistinguishability parameter ϵ
Output: A subset of D

1: For each $1 \leq i \leq d$, let $m_i = min_{e \in D} e_i$ and keep track of maximum point $q_i^* = \arg\max_{e \in D} e_i$.
2: Let $i^* = 1$, $i = 2$.
3: **while** $i \leq d$ **do** ▷ Find largest utility coefficient.
4: Let point q_i be $\left(\frac{q_{i^*}^*[i^*]+q_i^*[i^*]}{2}, \frac{q_{i^*}^*[i]+q_i^*[i]}{2}\right)$.
5: Display $p_1 = (m_{i^*}, q_i[i])$ and $p_2 = (m_{i^*}+a, q_i[i]-a)$ where $a = \min(q_i[i^*] - m_{i^*}, q_i[i] - m_i)$
6: Update $i^* = i$ if user picks p_1
7: Let $i = i + 1$.
8: For each dimension, $1 \leq i \leq d, i \neq i^*$, initialize L_i to 0 and H_i to 1. Define $L_{i^*} = H_{i^*} = 1$.
9: For each $1 \leq i \leq d$, let $q_i = \left(\frac{q_{i^*}^*[i^*]+q_i^*[i^*]}{2}, \frac{q_{i^*}^*[i]+q_i^*[i]}{2}\right)$.
10: Let $i = 1$.
11: **while** remain questions to be asked **do**
12: **if** $i = i^*$ **then**
13: Update $i = (i \bmod d) + 1$.
14: Define $b = \frac{(L_i+H_i)}{2}$ as the desired ratio breakpoint.
15: Define $\hat{b} = -\frac{1}{b}$ as the desired slope breakpoint.
16: Let $width = q_i[i^*] - m_{i^*}$ and $height = q_i[i] - m_i$.
17: **if** $q_i[i] + (width \cdot \hat{b}) < m_i$ **then** ▷ Check if the point intersects with bottom of TT field
18: Create point $p_2 = (m_{i^*} - \frac{height}{\hat{b}}, q_i[i] - height)$.
19: **else**
20: Create point $p_2 = (m_{i^*} + width, q_i[i] + width \cdot \hat{b})$.
21: Display $p_1 = (m_{i^*}, q_i[i])$ and p_2 to user.
22: **if** user chooses p_1 **then**
23: Update $L_i = b$.
24: **else**
25: Update $H_i = b$.
26: Update $i = (i \bmod d) + 1$.
27: Initialize subset $C = D$.
28: **for all** $e \in D$ **do**
29: Remove all points from C that are (1+ ϵ)-dominated by e for all utility functions in $[L_1, H_1]$, $[L_2, H_2]$, ..., $[L_d, H_d]$
30: **return** C.

Our tolerably truthful algorithm, TT-Breakpoint, is given in Algorithm 2. At the beginning of the algorithm, it loops through the database dimensions to find the minimum value m_i and maximum point q_i^* for each dimension i in Line 1. Then, the algorithm computes the anchor dimension i^*, representing the dimension with the largest coefficient in the utility vector as follows in Lines 2 - 7. As described earlier, the artificial point q_i is chosen to be the midpoint between the maximum points of each dimension. In Line 5, the algorithm builds two points using q: $p_1 = (m_{i^*}, q_i[i])$ and $p_2 = (m_{i^*} + a, q_i[i] - a)$ where $a = min(q_i[i^*] - m_{i^*}, q_i[i] - m_i)$. By using point q_i and the minimum value of each dimension, the

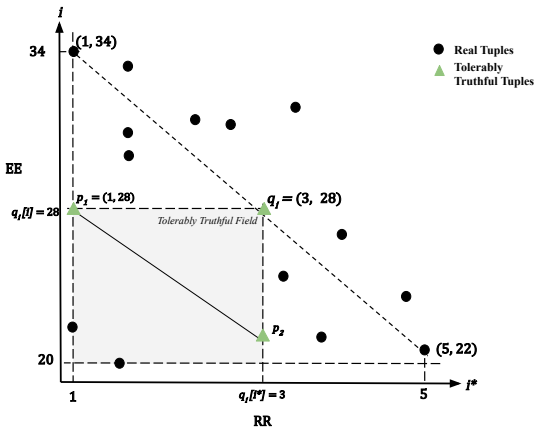


Figure 2: TT-Breakpoint Algorithm Example

algorithm generates two points dominated by this point q_i that generate slope of -1 as shown in Figure 1. Since the breakpoint can be computed to be $\frac{-1}{slope}$ (see Section 3.4), whenever the slope is -1, the $\frac{u_i}{u_{i^*}}$ breakpoint is 1. This method demonstrates that selecting one of the two points indicates the user's preference for the corresponding dimension. Therefore, when the user chooses a tuple, the algorithm can update whichever attribute the user prefers to find the larger coefficient. This can be illustrated directly where if the user picks $p_2 = (m_{i^*} + a, q_i[i] - a)$, then

$$\begin{aligned} u[i^*](m_{i^*} + a) + u[i](q_i[i] - a) &\geq u[i^*]m_{i^*} + u[i]q_i[i] \\ u[i^*]((m_{i^*} + a) - m_{i^*}) &\geq u[i]\,(q_i[i] - (q_i[i] - a)) \\ u[i^*]a &\geq u[i]a \\ u[i^*] &\geq u[i]. \end{aligned}$$

We can similarly show that $u[i^*] \leq u[i]$ if the user picks p_1. Using this technique, the algorithm loops through all the dimensions to find the dimension with the greatest $u[i]$. This process allows us to bound $\frac{u[i]}{u[i^*]}$ of all remaining dimensions to between [0, 1] and set lower and higher bounds of the anchor dimension i^* to 1. The algorithm also keeps track of each midpoint q_i for every dimension in Line 9.

Example: An example of the tuple construction is given in Figure 2. In this example, the tuples with maximum RR and EE are (5, 22) and (1, 34), respectively. The tuple q_i is defined to be the midpoint of these tuples: (3, 28). The artificial tuples shown to the user (p_1 and p_2) are in the tolerably truthful field (the rectangle with (1, 20) at the lower left and q_i at the top right).

While there are remaining questions, the algorithm continues to narrow down the user's utility function in Lines 10 - 26. Since we can create any breakpoint we desire within the tolerably truthful field, we use each round of user interaction to narrow down the range of one of the $u[i]$'s by half (i.e., halve the width of $[L_i, H_i]$). To achieve this, we create pairs of tuples with breakpoint $b = \frac{L_i + H_i}{2}$ as follows. We first compute the desired slope $\hat{b} = -1/b$ (using the relationship from Section 3.4). We then create a pair of tuples that have this desired slope but that still remain in the tolerably truthful field by choosing the first (p_1) to be at the top left of the field and the second (p_2) to be along the bottom (Line 18) or right (Line 20) depending on the steepness of the slope. When the algorithm displays p_1 and p_2 to the user, it can correctly update the L_i or H_i bounds to narrow the range of u_i by a factor of two.

Example: Suppose the algorithm has so far narrowed down u_i/u_a to a range $L_i = 0.5$ and $H_i = 0.75$. To narrow down the range further, we set a breakpoint of $b = (0.5{+}0.75)/2 = 0.625$. This corresponds to a slope of $-1/0.625 = -1.6$. Continuing with the example in Figure 2, we will construct the tuple $p_2 = (3, 28 + 2 \times -1.6) = (3, 24.8)$ since $24.8 \geq 20$, the minimum EE value in the database. If this condition we were not true, we would have placed p_2 on the bottom of the tolerably truthful field.

Lastly, the algorithm prunes tuples that are $(1 + \epsilon)$-dominated by any other points in the database for all utility functions within the range $[L_i, H_i]$ in Lines 28-29, following the computationally efficient approach in [14], as with the Breakpoint algorithm.

5.1 Running Time

The running time of TT-Breakpoint is dominated by the computation of the tuples q_i^*, which takes $O(nd)$ time, where n is the total number of tuples and d is the number of dimensions. The running time for finding the anchor dimension i^* (Lines 2-8) is just $O(d)$ and the time for narrowing down the utility function (Lines 9-26) is $O(q)$, where q is the total number of questions asked. The final pruning step (Lines 27-29) takes $O(nd)$ time. Thus, the overall running time of the algorithm is $O(nd + q)$, which is only linear in the number of tuples, n.

5.2 Proofs of Correctness

Lemma 5.1. *After q questions, for each $1 \leq i \leq d$,*

$$H_i - L_i \leq 1/2^{\lfloor (q-d+1)/(d-1) \rfloor}.$$

Proof. In lines 2 - 7 of Algorithm 2, we ask the user $d - 1$ questions to determine i^*. Of the remaining $(q - d + 1)$ questions, each one is used to halve the width of one of the $H_i - L_i$ values. Since there are $(d - 1)$ such values and each one starts at 1, after q questions we will halve each of them at least $\lfloor (q-d+1)/(d-1) \rfloor$ times. Thus, after q questions we have each $H_i - L_i \leq 1/2^{\lfloor (q-d+1)/(d-1) \rfloor}$. □

Example: If $d = 3$, by asking just 16 questions we guarantee that each $H_i - L_i \leq 1/2^7 < 0.01$.

Theorem 5.2. *After q questions, TT-Breakpoint outputs a set that is a $O(d/2^{\lfloor (q-d+1)/(d-1) \rfloor})$-approximation of $\mathcal{I}$.*

Proof. Following the same reasoning as [14, Theorem 2], if we bound $H_i - L_i \leq \tau$ for each i, then we get a $\tau d(1+\epsilon)$-approximation for $\mathcal{I}$. The theorem follows by substituting $\tau = 1/2^{\lfloor (q-d+1)/(d-1) \rfloor}$ from Lemma 5.1. □

Example: If $d = 3$, by asking just 16 questions we guarantee a 0.05-approximation of $\mathcal{I}$.

Theorem 5.3. *TT-Breakpoint is a truthful algorithm.*

Proof. Recall that an algorithm is truthful if the tuples shown during interaction are no better than the user's eventual favorite tuple from the final output set. Since the optimal tuple in the entire

database is output in the approximation of $\mathcal{I}$, it suffices to show that any output tuples have lower value for the user than *some* tuple in the database.

In each round of TT-Breakpoint, we show tuples p_1 and p_2 from the tolerably truthful field to the user. Each of these tuples is chosen so as to be dominated by the tuple q_i since q_i is at the top right of the tolerably truthful field (see Figure 1). The utility of q_i on just dimensions i and i^* (call this function f) can be bounded

$$\begin{aligned} f(q_i) &= u[i^*]q_i[i^*] + u[i]q_i[i] \\ &= u[i^*]\left(\frac{q^*_{i^*}[i^*] + q^*_i[i^*]}{2}\right) + u[i]\left(\frac{q^*_{i^*}[i] + q^*_i[i]}{2}\right) \\ &= \left(u[i^*]\frac{q^*_{i^*}[i^*]}{2} + u[i]\frac{q^*_{i^*}[i]}{2}\right) + \left(u[i^*]\frac{q^*_i[i^*]}{2} + u[i]\frac{q^*_i[i]}{2}\right) \\ &= \frac{f(q^*_{i^*})}{2} + \frac{f(q^*_i)}{2} \\ &\leq \max(f(q^*_{i^*}), f(q^*_i)) \\ &\leq f(p^*), \end{aligned}$$

where p^* is the tuple with optimal utility in just dimensions i and i^*. The penultimate inequality follows from the fact that the mean of two numbers is at most the maximum of the two. The last line follows from the fact that the utility of the real tuples $q^*_{i^*}$ and q^*_i must be at most the utility of the optimal tuple p^*.

In conclusion, the utility of each displayed tuple is at most the utility of tuple q_i which in turn is at most the utility of the optimal tuple output by the TT-Breakpoint algorithm, so TT-Breakpoint is truthful. □

Theorem 5.4. *TT-Breakpoint is a tolerably truthful algorithm.*

Proof. We proved that TT-Breakpoint is a truthful algorithm in Theorem 5.3. We now show that each tuple p displayed by TT-Breakpoint has the property that, $1 \leq i \leq d$, $min_i \leq p_i \leq max_i$, where min_i and max_i are the minimum and maximum values in dimension i among all the tuples in the database.

The tuple $p_1 = (m_{i^*}, q_i[i])$ has the property since $min_{i^*} = m_{i^*} \leq max_{i^*}$ and $q_i[i]$ is the mean of the i dimensions of two tuples in the database and we know that the mean is always greater than the smaller of the two values and smaller than the greater of the two values. More precisely,

$$q_i[i] = \frac{q^*_{i^*}[i] + q^*_i[i]}{2} \geq \min\left(q^*_{i^*}[i], q^*_i[i]\right) \geq min_i,$$

and

$$q_i[i] = \frac{q^*_{i^*}[i] + q^*_i[i]}{2} \leq \max\left(q^*_{i^*}[i], q^*_i[i]\right) \leq max_i,$$

and similarly for $q_i[i^*]$.

For the tuple $p_2 = (m_{i^*} + a, q_i[i] - a)$ displayed in Line 5, we have that $min_{i^*} \leq min_{i^*} + a = m_{i^*} + a$ and $m_{i^*} + a \leq m_{i^*} + (q_i[i^*] - m_{i^*}) = q_i[i^*] \leq max_{i^*}$ (since $a = \min(q_i[i^*] - m_{i^*}, q_i[i] - m_i)$). Also, $q_i[i] - a \geq q_i[i] - (q_i[i] - m_i) = m_i = min_i$ (since $a = \min(q_i[i^*] - m_{i^*}, q_i[i] - m_i)$) and $q_i[i] - a \leq q_i[i] \leq max_i$. Thus p_2 satisfies the desired property.

For the tuple p_2 displayed on Line 21 of the algorithm there are two cases:

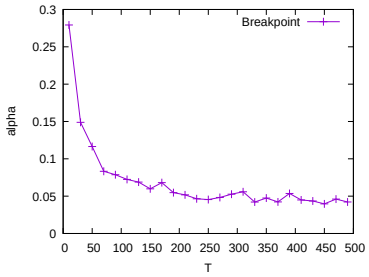


Figure 3: Results for Breakpoint when varying T on NBA ($q = 20, \epsilon = 0.05$)

Case 1: $q_i[i] + (width \cdot \hat{b}) < m_i$: Here, $p_2 = (m_{i^*} - \frac{height}{\hat{b}}, q_i[i] - height)$. We can bound $m_{i^*} - \frac{height}{\hat{b}} = m_{i^*} + height \cdot b \geq m_{i^*} = min_{i^*}$ and

$$\begin{aligned} m^*_i - \frac{height}{\hat{b}} &= m_{i^*} - \frac{q_i[i] - m_i}{\hat{b}} \\ &< m_{i^*} + width \\ &= m_{i^*} + (q_i[i] - m_{i^*}) \\ &= q_i[i] \\ &\leq max_i. \end{aligned}$$

Also, $q_i[i] - height = q_i[i] - (q_i[i] - m_i) = m_i = min_i$ and $q_i[i] - height \leq q_i[i] \leq max_i$. Thus, in this case p_2 preserves the property.

Case 2: $q_i[i] + (width \cdot \hat{b}) \geq m_i$: Here, $p_2 = (m_{i^*} + width, q_i[i] + width \cdot \hat{b})$. We can bound $m_{i^*} + width \geq m_{i^*} = min_{i^*}$ and $m_{i^*} + width = m_{i^*} + (q_i[i^*] - m_{i^*}) = q_i[i^*] \leq max_{i^*}$. Also, $q_i[i] + width \cdot \hat{b} \geq m_i = min_i$ and $q_i[i] + width \cdot \hat{b} \leq q_i[i] \leq max_i$ (as $\hat{b} < 0$). Again, p_2 preserves the property.

In conclusion, all tuples p displayed by TT-Breakpoint are such that $min_i \leq p[i] \leq max_i$. Combined with the truthfulness of the algorithm from Theorem 5.3, we have that TT-Breakpoint is tolerably truthful. □

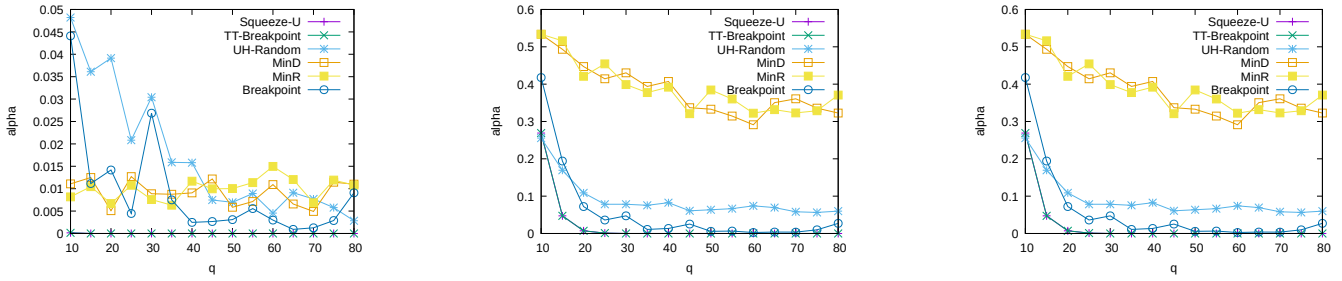
6 Experimental Evaluation

All the following experiments were conducted using a 16-core (2.5GHz) machine with 64GB RAM. All of the code was implemented in C++ and is available for download [1].

Datasets: For these experiments, both artificial and real datasets were used. For synthetic data, we used a data set generator [3] to create anti-correlated data sets as these have large skylines. We used three real datasets: "Island" which has 63383 points with 2D geographical coordinates, "NBA" which has 21961 4D points with records of players, and "House" which is a 6D dataset with 12793 housing utility values.

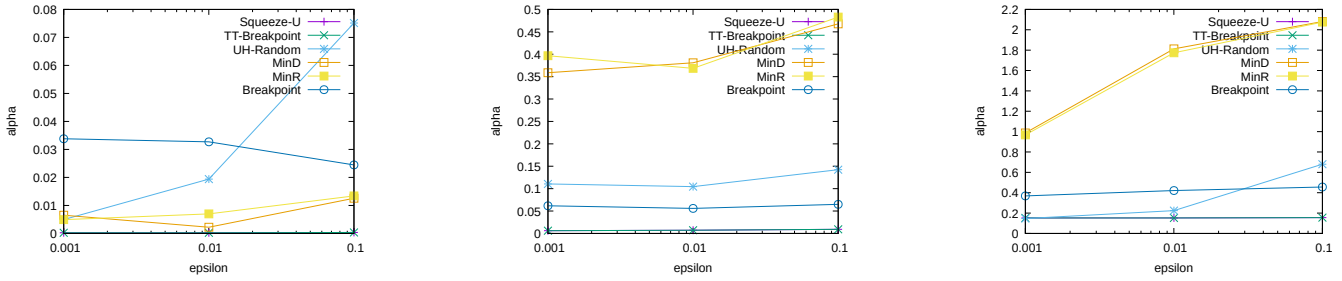
Algorithms: We evaluated our strongly truthful algorithm Breakpoint and our tolerable truthful algorithm TT-Breakpoint against Squeeze-u [14] (an algorithm that uses untruthful artificial points), and several strongly truthful algorithms: UH-Random [32], MinR, and MinD [14]. The algorithms Squeeze-u, MinR, and MinD are

[1] https://github.com/ashlall/Indistinguishability2



(a) Island **(b) NBA** **(c) House**

Figure 4: Varying number of questions, q ($\epsilon = 0.05$)



(a) Island **(b) NBA** **(c) House**

Figure 5: Varying ϵ ($q = 5d$), log-scale on x-axis

the only other algorithms designed specifically for the indistinguishability query. The UH-Random was another algorithm that was found to work well for the query in [14]. It is important to note that because the artificial algorithms points have more control over constructing points and therefore can more effectively narrow down the user's unknown utility function, it is not fair to directly compare the algorithms using artificial points with the strongly truthful algorithms. However, showing the algorithms side by side still allows us to compare their performances with this caveat in mind. Note that other non-interactive algorithms cannot be compared because it is not clear what tuples to show in the second round onwards.

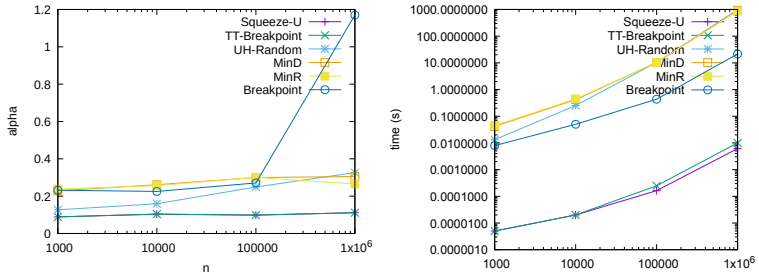
Parameter settings: In all of our experiments, we evaluated the algorithms on one hundred independently chosen random utility functions and reported their average performance. We measured the approximation value α (see Section 3.1) for each of the algorithms with varied parameters. As a reminder, α measures the worst case deviation from being ϵ-indistinguishable over all the points output by the algorithm, hence closer to zero is better. Unless otherwise stated, we used the default values $\epsilon = 0.05$ and $q = 5d$. For the parameter T in TT-Breakpoint (see Section 5), we varied T on the NBA data set (using 500 runs to reduce variance) and got the results shown in Figure 3. Since the performance starts to level off after $T = 100$, we used this in all our subsequent experiments.

6.1 Real Data

Firstly, we compared the performance of the algorithms as the number of questions per round asked to the user increased in Figure 4. In each experiment, we fixed the indistinguishability threshold $\epsilon = 0.05$, and varied q from 10 to 80 in increments of 10. We see that Squeeze-u and TT-Breakpoint had identical performance across all datasets, and they always outperform the strongly truthful algorithms. This result is expected, as both algorithms narrow down the utility ratio range of one pair of dimensions by half in a single round. When comparing the strongly truthful algorithms algorithms, we see Breakpoint usually had the lowest alpha values.

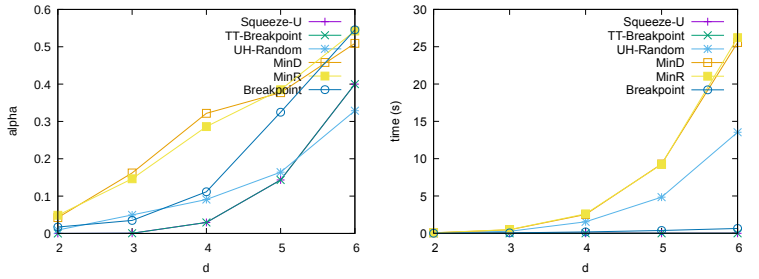
We next studied the effects of varying epsilon, the user's desired threshold for indistinguishability in Figure 5. In each experiment, we fixed the number of questions asked to the user in each round to be $q = 5d$, and $\epsilon = 0.05$. In our results, we again see that Squeeze-u and TT-Breakpoint had the best results. Among the strongly truthful algorithms, Breakpoint did not perform as well for the Island dataset, was the best in the case of NBA, and among the top two for House. This variation has to do with what can be learned from the real tuples in each dataset given how each algorithm attempts to narrow down the user's utility function.

Table 3 shows the runtimes for the algorithms on each of the real datasets. The two artificial tuple algorithms (Squeeze-u and TT-Breakpoint) ran very fast since most of their computation doesn't involve the actual tuples in the database. Among the strongly truthful algorithms, Breakpoint was consistently the fastest, taking less than one second to run in each instance.



(a) alpha **(b) time (s)**

Figure 6: Varying number of tuples in synthetic data with fixed dimensionality ($d = 3, q = 9, \epsilon = 0.05$)



(a) alpha **(b) time (s)**

Figure 7: Varying number of dimensions in synthetic data with fixed number of tuples ($n = 10000, q = 20, \epsilon = 0.05$)

Algorithm	Island	NBA	House
Squeeze-u	0.00	0.00	0.00
TT-Breakpoint	0.00	0.00	0.00
UH-Random	15.69	0.00	7.01
MinD	2.61	0.06	49.18
MinR	2.33	0.06	47.54
Breakpoint	0.39	0.00	0.29

Table 3: Running times (s.) ($\epsilon = 0.05, q = 5d$)

6.2 Scalability Test

We next studied the behavior of the algorithms when we increase the number of tuples in the database and the number of dimensions. Since the real datasets have static sizes, we generated anti-correlated data using a standard dataset generator [3].

In Figure 6 we show the results for varying the number of tuples n. As before, the artificial tuple algorithms perform the best and all the algorithms have fairly steady error, except that Breakpoint becomes large for the case of a million tuples (Figure 6a). The runtimes of the artificial tuple algorithms is the lowest and Breakpoint runs the fastest among the strongly truthful algorithms (Figure 6b).

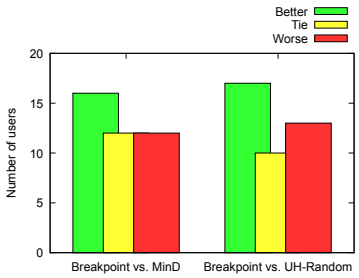
Figure 7 shows the corresponding result when we increase the dimensionality of the data d. We can see that all the algorithms have worse performance as the number of dimensions increases (Figure 7a). The runtime of Breakpoint, TT-Breakpoint, and Squeeze-u are the best, far less than the time used by the other strongly truthful algorithms (Figure 7b).

6.3 User Study

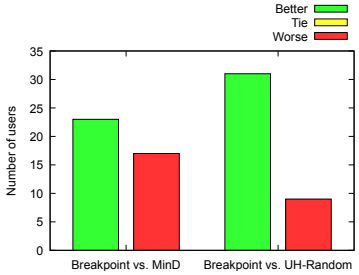
In order to test our hypothesis that showing the user two attributes at a time would make decision-making easier for users, we conducted a user study with 40 participants on a US college campus. We used a laptop database with 896 entries and five attributes: RAM, SSD, HDD, star rating, and number of ratings. We compared only strongly truthful algorithms, namely Breakpoint, UH-Random, and MinD. Each participant answered up to 10 rounds of questions for each algorithm and we asked the users to rate the amount of effort (from 1-5) and measured the time spent by users to answer all the questions. In Figure 8a we see that Breakpoint is rated as taking less effort moderately more often than the other two algorithms. When comparing which algorithms had faster response times (Figure 8b), Breakpoint fared better against both of the comparison algorithms.

6.4 Summary

In summary, we found the TT-Breakpoint algorithm to match the performance of the state-of-the-art artificial tuple algorithm, Squeeze-u, in both approximation and time while only showing tolerably truthful tuples. These algorithms greatly outperformed the strongly truthful algorithms in both these metrics as well. If the user is interested in strongly truthful algorithms that only show real tuples, then the Breakpoint is a strong option, outperforming the other strongly truthful algorithms in runtime and often being the best in terms of approximation. In our user study we found that Breakpoint takes less effort and user time than other algorithms.



(a) Effort Rating



(b) Response Time

Figure 8: Head-to-head comparison of Breakpoint algorithm with MinD and UH-Random

7 Conclusion

We propose an interactive framework for the Indistinguishability Query where users choose between two tuples with two attributes. We introduce a novel definition of truthfulness — tolerably truthful — in the context of interactive database queries. We develop a tolerably truthful and a strongly truthful algorithm, each effectively estimating the user's indistinguishability set with a close approximation guarantee for the former. Experimental results show that our tolerably truthful algorithm TT-Breakpoint has comparable performance to the state-of-the-art algorithm while only showing realistic tuples. Our strongly truthful algorithm Breakpoint is often better than other strongly truthful algorithms. Our user study showed that Breakpoint results in less effort and time for the user.

Several intriguing questions remain open for future research: (1) conducting further user studies to better understand the algorithms' performance with real users, (2) investigating the Breakpoint and TT-Breakpoint algorithms' performance in the presence of user errors, and (3) determining if we can get a better approximation bound while remaining tolerably truthful. We leave these open questions to future work.

References

[1] Abolfazl Asudeh, Azade Nazi, Nan Zhang, and Gautam Das. 2017. Efficient Computation of Regret-ratio Minimizing Set: A Compact Maxima Representative. In *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM, 821–834.

[2] A. Asudeh, A. Nazi, N. Zhang, and G. Das. 2017. Efficient Computation of Regret-ratio Minimizing Set: A Compact Maxima Representative. In *Proceedings of the ACM International Conference on Management of Data*.

[3] S. Borzsony, D. Kossmann, and K. Stocker. 2001. The skyline operator. In *Proceedings. 17th International Conference on Data Engineering*.

[4] C. Chan, H. Jagadish, K. Tan, A. Tung, and Z. Zhang. 2006. Finding k-dominant skylines in high dimensional space. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*.

[5] C. Chan, H. Jagadish, K. Tan, A. Tung, and Z. Zhang. 2006. On high dimensional skylines. In *Advances in Database Technology-EDBT 2006*. Springer, 478–495.

[6] Y. Chang, L. Bergman, V. Castelli, C. Li, M. Lo, and J. Smith. 2000. The Onion Technique: Indexing for Linear Optimization Queries. In *Proceedings of ACM SIGMOD International Conference on Management of Data*.

[7] S. Chester, A. Thomo, S. Venkatesh, and S. Whitesides. 2014. Computing k-Regret Minimizing Sets. In *Proceedings of the VLDB Endowment*.

[8] H. A. David. 1988. *The Method of Paired Comparisons*. New York: Oxford University Press.

[9] P. Fraternali, D. Martinenghi, and M. Tagliasacchi. 2012. Top-k bounded diversification. In *Proceedings of the 2012 International Conference on Management of Data*.

[10] M. Goncalves and M. Yidal. 2005. Top-k skyline: a unified approach. In *On the Move to Meaningful Internet System 2005*.

[11] Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman. 2008. A Survey of Top-k Query Processing Techniques in Relational Database Systems. *ACM Comput. Surv.* 40, 4, Article 11 (Oct. 2008), 58 pages.

[12] K. Jamieson and R. Nowak. 2011. Active ranking using pairwise comparisons. In *Advances in Neural Information Processing Systems*.

[13] Taylor Kessler Faulkner, Will Brackenbury, and Ashwin Lall. 2015. K-Regret Queries with Nonlinear Utilities. In *Proceedings of the VLDB Endowment*.

[14] Ashwin Lall. 2024. The Indistinguishability Query. In *Proceedings of the 2024 International Conference on Data Engeneering (ICDE 2024)* (Utrecht, Netherlands).

[15] J. Lee, G. won You, and S. won Hwang. 2009. Personalized top-k skyline queries in high-dimensional space. In *Information Systems*.

[16] X. Lian and L. Chen. 2009. Top-k dominating queries in uncertain databases. In *Proceedings of International Conference on Extending Database Technology: Advances in Database Technology*.

[17] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. 2007. Selecting stars: The k most representative skyline operator. In *Proceedings of International Conference on Data Engineering*.

[18] Denis Mindolin and Jan Chomicki. 2009. Discovering relative importance of skyline attributes. *Proceedings of the VLDB Endowment* 2, 1 (2009), 610–621.

[19] David M. Mount. 2012. CMSC 754: Computational Geometry Lecture Notes. Dept. of Computer Science, University of Maryland, College Park, MD, 20742. https://www.cs.cmu.edu/afs/cs/academic/class/15456-s14/Handouts/cmsc754-lects.pdf Pages 6-10.

[20] D. Nanongkai, A. Lall, A. Das Sarma, and K. Makino. 2012. Interactive Regret Minimization. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*.

[21] D. Nanongkai, A.D. Sarma, A. Lall, R.J. Lipton, and J. Xu. 2010. Regret-Minimizing Representative Databases. In *Proceedings of the VLDB Endowment*.

[22] D. Papadias, Y. Tao, G. Fu, and B. Seeger. 2005. Progressive skyline computation in database systems. In *ACM Transactions on Database Systems (TODS)*, Vol. 30. ACM, 41–82.

[23] P. Peng and R.C.W Wong. 2014. Geometry approach for k regret query. In *Proceedings of International Conference on Data Engineering*.

[24] L. Qian, J. Gao, and H.V. Jagadish. 2015. Learning user preferences by adaptive pairwise comparison. In *Proceedings of the VLDB Endowment*.

[25] L. Qin, J. Yu, and L. Chang. 2012. Diversifying top-k results. In *Proceedings of the VLDB Endowment*.

[26] X. Qin, C. Chai, and Y. et al. Luo. 2022. Interactively discovering and ranking desired tuples by data exploration. *VLDB Journal* (2022).

[27] M. Soliman, I. Ilyas, and K. Chen-Chuan Chang. 2007. Top-k query processing in uncertain databases. In *Proceedings of International Conference on Data Engineering*. IEEE, 896–905.

[28] Y. Tao, L. Ding, and J. Pei. 2009. Distance-based representative skyline. In *Proceedings of International Conference on Data Engineering*.

[29] Y. Tao, X. Xiao, and J. Pei. 2007. Efficient Skyline and Top-k Retrieval in Subspaces. In *TKDE*.

[30] Weicheng Wang, Raymond Chi-Wing Wong, and Min Xie. 2021. Interactive Search for One of the Top-k. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD/PODS '21)*. 1920–1932.

[31] T. Xia, D. Zhang, and Y. Tao. 2008. On skylineing with flexible dominance relation. In *Proceedings of International Conference on Data Engineering*.

[32] Min Xie, Raymond Chi-Wing Wong, and Ashwin Lall. 2019. Strongly Truthful Interactive Regret Minimization. In *Proceedings of the 2019 International Conference on Management of Data* (Amsterdam, Netherlands) *(SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 281–298.

[33] M. Xie, R. C.-W. Wong, J. Li, C. Long, and A. Lall. 2018. Efficient k-regret query algorithm with restriction-free bound for any dimensionality. In *Proceedings of the 2018 ACM International Conference on Management of Data*. ACM.