The Indistinguishability Query

Ashwin Lall
Department of Computer Science
Denison University
Granville, OH, US
lalla @ denison.edu

***Abstract*—We propose the indistinguishability query for identifying all of a user's near-optimal tuples. This query returns all the tuples that are at most a small fraction away from the optimal of the user's unknown utility function. This is motivated by the idea that users can have a hard time distinguishing very similar tuples and in fact even tuples that are slightly inferior in the identified criteria may have additional characteristics that make them more attractive to the user.**

In order to perform this query without knowledge of the user's utility function, we use a simple interactive framework that asks the user to perform a modest number of comparisons to narrow down their utility function. We show that the indistinguishability query cannot be approximated solely with real tuples in the database and thus our algorithms with provable bounds must present the user with artificial tuples. We also give heuristic algorithms that show the user only real tuples from the database.

Since the user may make errors while performing comparisons, we generalize our algorithms to account for user error as well. Experiments on synthetic and real data sets show that the indistinguishability query can be performed accurately while asking the user to compare a small number of tuples.

***Index Terms*—multi-criteria decision-making, user interaction, data analytics**

I. Introduction

A central question in multi-criteria decision-making is how to determine what subset of a large collection of tuples in a database should be shown to the user. The importance of the question stems from the fact that people do not have the time or attention to sift through potentially thousands or millions of items to determine their favorite option. On the other hand, it is difficult for the user to know how they value or rank items since they may not know their utility valuation for different attributes and it could be expensive to learn this function. In this paper, we propose an interactive framework for the user to identify all the tuples in a database that are not far from their optimal tuple while only considering a small number of tuples.

Suppose that Alice wants to purchase a car with high fuel efficiency (MPG) and safety rating (SR) but she doesn't know her relative valuation for these two criteria. Furthermore, a car database for her area has thousands of cars available, too many for her to go through. The standard approach in the literature is to assume that Alice has a utility function and to use one of a variety of methods to help her find her (near-)optimal car.

Some of the earliest work in multi-criteria decision-making focused on computing the *top-k* tuples in a large database (see [1] for a survey). These queries are predicated on the explicit knowledge of Alice's utility function and can thus return the k highest-valued tuples for Alice based on this function. Note that it is not automatically assumed that the single highest-valued item will be Alice's top choice as there may be other, less significant criteria Alice uses to make her final decision (color, brand name, etc.). The disadvantage of the top-k approach is that it is unlikely that Alice knows her exact utility function and it may be costly to learn it.

An alternative approach that makes no assumption about Alice's utility function is the *skyline operator* [2]. In this approach, the user is shown the skyline or Pareto-optimal tuples in the database. These tuples are precisely the ones that are not dominated by any other. A tuple is dominated by another if it is no better than the other in all of the attributes of interest and strictly worse in at least one. While this approach makes no assumption about Alice's utility function, it shows many tuples that are not of interest to Alice. Moreover, it does not show options that are slightly inferior to the optimal according to Alice's utility function.

Another recent approach has been to use *regret minimization* [3] queries to return a set of tuples among which at least one is guaranteed to be close to the optimal for Alice. Once again, this approach will show several uninteresting tuples and miss out on many tuples that are almost as good as the optimal. The regret query has also been generalized to the interactive case [4], [5] in which Alice makes several rounds of selections of her favorite from a small set of tuples in each round. Even these interactive approaches focus on finding a near-optimal tuple or even the optimal one, but do not show the full selection of near-optimal tuples.

This paper builds on these ideas to propose the indistinguishability query that aims to find *all* of Alice's near-optimal tuples. This is motivated by the fact that Alice may have a fear of missing out on tuples that were interesting to her that got pruned by the database query. The indistinguishability query can guarantee for her that only tuples that are significantly worse than her optimal (e.g., less than 90% of her optimal tuple's utility) would get pruned out. By using multiple rounds of interaction, the indistinguishability query is about to quickly narrow down Alice's set of viable tuples. Alice could then perform other filters (e.g., color or brand name) to make her final decision.

We say that a pair of tuples are ϵ-indistinguishable for Alice if her valuation of them with her (unknown) utility function is within a $(1+\epsilon)$-multiplicative factor of each other. (A

more formal definition will follow shortly.) The goal of the indistinguishability query then is to output the set of tuples that Alice finds ϵ-indistinguishable from the optimal for her utility function. Similar to the top-k approach, we want to show Alice a number of near-optimal options (including ones that are dominated by the optimal for her utility function) in case she prefers a less optimal tuple that has other attractive characteristics. For example, Alice might prefer a car with slightly worse MPG and SR if it is available in green. If the attributes identified by Alice (MPG and SR) contribute 95% of her overall utility, then identifying the ϵ-indistinguishable set with $\epsilon = (1/0.95 - 1) \approx 0.053$ guarantees that the overall optimal must be in the output set.

The above example gives the user guidance on how to select ϵ: if the user believes that the identified criteria cover at least c fraction of their total utility (e.g., $c = 0.9$ for 90% of their total utility), then they can select $\epsilon = 1/c - 1$. Alternatively, the user can select a large ϵ (e.g., $\epsilon = 0.5$) and can decrease ϵ repeatedly to rerun the query if the output set is too large. Importantly, it will be shown later in this paper that when decreasing ϵ the output set will always be a subset of the former and thus the work done so far does not need to be duplicated.

In order to perform the indistinguishability query, we narrow down Alice's utility function by asking her to choose her favorite among a small set of tuples. While it is unreasonable for Alice to specify her exact utility function, it is perfectly reasonable to expect that she can pick her favorite from a small set (e.g., two cars). Over a small number of rounds of interaction we show how to rapidly narrow down the tuples for the indistinguishability query. After a small number of rounds (e.g., six rounds for a total of twelve cars compared) we can output a subset that is guaranteed to contain all the cars that are indistinguishable from Alice's optimal. There may be extra cars that are included in the output, but we show that these are not far off from being in the desired output set. In the above scenario, Alice is asked to compare a small number of cars with the payoff that the system will then identify a selection of cars all of which will be near her optimal and thus of interest to her. She can then browse this set comfortable in the knowledge that she is not missing out on any cars that might be more interesting to her. The goal then is to minimize the set size ($s = 2$ in the example above) and the number of questions asked ($q = 6$ in the example) while outputting all of the cars Alice may be interested in and limiting how far off from being interesting the remaining cars are to her.

While Alice makes her interactive selections from a small pool of cars, she may encounter cars that are indistinguishable to her. In such a case, she might erroneously select a car that has slightly lower utility than another in the pool. We generalize our algorithms to also account for this source of error in the user selection of cars during the interactive phase of the algorithm.

When showing Alice cars during the interactive phase, we would ideally like to show her only cars that actually exist in the database. Unfortunately, we are able to show an impossibility result in which the output set is arbitrarily bad when constrained in this way. Because of this, we first give algorithms that show Alice artificially constructed tuples that can rigorously bound the approximation of the output set. We also propose two heuristic algorithms that show only real tuples from the database. Though the heuristics cannot have any guarantees, we show that they are effective on real world data sets.

In summary, the indistinguishability query is superior to many previously proposed multi-criteria decision-making queries because it retains *all* the candidate tuples that are indistinguishable from the optimal; much previous work was focused on finding a subset that guaranteed at most ϵ regret for some tuple in the subset. Also, unlike other prior work we cannot focus our search on the skyline of the set (potentially much fewer tuples) as there may be dominated tuples that are indistinguishable from the optimal. We show how to perform our query using light-weight interaction with the user, using both artificial and real tuples. Lastly, we account for user error in their selections.

Contributions: Our major contributions are:

- We define and motivate the indistinguishability query.
- We show that it is impossible to avoid many false positives when restricted to real tuples in the database.
- We give an algorithm for approximating the query using artificial tuples with provable bounds.
- We provide two heuristic algorithms for approximating the query with real tuples.
- We show how to generalize all the above algorithms to account for user error (i.e., inability of the user to distinguish between tuples that they value similarly).
- We present performance evaluation for all the proposed algorithms and show their efficacy on real and synthetic data sets.

Organization: We discuss the related work in the next section. In Section III we give several definitions, formally define the problem, and show an impossibility result for using real tuples. We present an algorithm for approximating the query using artificial tuples in Section IV. We next give two algorithms for approximating the query using real tuples from the database in Section V. In Section VI we show how to adapt our algorithms for the case that the user makes indistinguishability errors in her selection step of the interactive algorithms. We show the efficacy of our algorithms in Section VII and conclude in Section VIII.

II. Related Work

There is a large body of work focused on the topic of multi-criteria decision-making.

Top-k and skyline. As mentioned earlier, the top-k [6], [7], [8], [9], [10] query has been extensively studied. Since it is often infeasible to be aware of the user's utility function, the skyline query and a number of extensions has been studied in the literature [2], [11], [12], [13], [14], [15], [16], [9], [17], [18], [19]. While the skyline is guaranteed to display the optimal tuple for any utility function, it has two shortcomings.

One is that there is no way to keep it from showing tuples that are uninteresting to the user. The second is that the skyline necessarily removes tuples that are dominated by the user's optimal but that may still be of interest. There have also been a number of efforts to control the size of the output set [20], [21], [22]. Unlike these queries, the goal of the indisinguishability query is not to control the output size but to ensure that *all* the interesting tuples for the user are included in the output set.

The skyline query has been generalized in the past to account for approximate domination [23], [24]. In this setting, a tuple is on the skyline if it is undominated after being scaled up by a $(1+\epsilon)$ constant. This is connected to our query in that this definition implies ours, but the converse is not true. Moreover, we are interested in tuples that are specifically of interest to our user and discover them via user interaction.

Learning user preference. Previous work [25], [26] has attempted to learn the user's favorite by asking them to partition tuples into *desirable* and *undesirable* groups. This takes considerably more effort on the part of the user over simply picking their favorite from a small set. In [27] the skyline query is personalized to the user by asking about preferences between different attributes. Other work [28] attempts to learn preferences in a manner similar to ours, also using hyperplane pruning, by asking users their preferences between pairs of tuples. However, their goal is to find a preference order over all the tuples rather than to interactively find the near-optimal tuples for a given utility function.

Machine learning. Learning user preference has been studied in the machine learning community [29], including various bandit approaches [30], [31], [32]. These approaches however do not exploit the dominance structure of tuples and hence require a lot more user interaction.

Regret. Regret minimization [3], [33], [34], [35], [36], [37], [38], [39], [5] has been studied extensively to help the user find at least one tuple that it is close in value to their favorite, where closeness is measured by the maximum regret ratio—the worst case percentage loss in value over all utility functions. Similar to the skyline query, it is guaranteed to find a tuple close to the user's optimal while at the same time controlling the size of the output. However, it also ends up displaying many tuples that are not of interest to the user since it has to cover a broad swathe of possible utility functions.

Interactive methods. Interactive regret minimization methods present users with artificial [4] and real tuples already present in the database [5]. We follow these works in giving algorithms for both cases. Whereas in these works the goal is to find either the optimal tuple or at least one tuple that it is near-optimal (i.e., with low regret), the goal of this work is to interactively find *all* the tuples that are close to being near-optimal. Recently, [40] queries the user for pair-wise feedback on tuples and interactively discovers one of their top-k. Even more recently, [41] uses interactivity to discover and rank tuples. By way of contrast, we aim to find all of the user's top-k tuples within a predefined threshold.

Compared with existing methods, our solution aims to display *all* the near-optimal tuples for the user. The user can rest assured that all their near-favorite tuples are output and do not have to worry about missing out on some potential optimal. This is in contrast to skyline and regret approaches that only show *some* near-optimal tuple and that can focus on far smaller sets of tuples by pre-processing the input to just the skyline tuples. After the interactive stage of our algorithms, all the tuples output to the user are going to be of interest as they are near-optimal for the user's utility function. The cost of guaranteeing the above is a modest amount of user interaction. Unlike other interactive work, though, we guarantee that no interesting tuples are ever omitted in the output set.

III. Definitions

We represent the input set of tuples (sometimes referred to as points) by D where $|D| = n$ and each tuple has d attributes or dimensions. That is, D can be considered a subset of $\mathbb{R}_+^d$. The specific attributes are selected by the user and can be a subset of a much larger set in the database. For example, a car database may have dozens of attributes for the cars in it and Alice selects $d = 2$ attributes (e.g., MPG and safety rating) that she is interested in and is shown only these attributes when presented with tuples. We assume that Alice prefers attributes for which bigger is better; if there are ones in which smaller is better (e.g., price), our algorithm would invert that attribute by subtracting all values from the maximum, as is standard in the literature.

Similar to much of the previous work in this area [42], [18], [4], [33], [37], we assume that users have an unknown utility function $f : \mathbb{R}_+^d \to \mathbb{R}_+$ that evaluates the utility that the user places on a given tuple. We leave non-linear utilities and categorical variables to future work. We next present one of our central new definitions in this paper.

Definition 1 (ϵ-indistinguishability)**.** *Given a utility function f and some $\epsilon > 0$, we say that two points p_1 and p_2 are ϵ-indistinguishable if*

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

and

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

Intuitively, this means that for some small $\epsilon > 0$ the user is unable to differentiate two options that have utility within an ϵ fraction of each other. Our goal then is to compute the set of tuples that are indistinguishable from the user's optimal. We call this the indistinguishability query.

Definition 2 (Indistinguishability Query)**.** *For a database D, given some $\epsilon > 0$ and a utility function f for which the optimal tuple in D is $p^* = \text{argmax}_{p \in D} f(p)$, we define*

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

In other words, $\mathcal{I}_{f,\epsilon}$ is the set of tuples in the database that are ϵ-indistinguishable from the optimal for utility function f. Whenever f and ϵ are clear from the context, we abbreviate $\mathcal{I}_{f,\epsilon}$ with $\mathcal{I}$.

car	MPG	SR	MPG + 20SR
c_1	59	5	159
c_2	36	4	116
c_3	104	3	164
c_4	34	5	134
c_5	95	3	158

TABLE I: Example of the indistinguishability query. All the highlighted tuples are 0.05-indistinguishable from the optimal (c_3) for a user with the given utility function.

Example: Consider the table of cars c_1 to c_5 in Table I. If Alice is interested in the attributes of fuel efficiency (MPG) and safety rating (SR) and has (unknown to her) utility function $f(MPG, SR) = MPG + 20SR$, then her favorite car will be $p^* = p_3$ as it has the highest utility of 164. With an indistinguishability parameter of $\epsilon = 0.05$, Alice will still be interested in any car that has at least $1/(1+\epsilon) \approx 95.24\%$ of her maximum utility ($0.9524 \times 164 \approx 156.2$) and thus the indistinguishability query should return the cars $\{c_1, c_3, c_5\}$ (highlighted in the table). Note that tuple c_1 is very different from Alice's optimal c_3, but has similar *value* for Alice according to her utility function.

We assume that each attribute is normalized to the range $[0, 1]$ and that the largest value across all dimensions is 1. If this is not the case, we can easily divide each attribute by the largest value across all dimensions without affecting our results. We also assume that the user prefers larger values. In the case that the user would prefer a smaller value (e.g., mileage or price), we can simply invert that attribute by subtracting each value from 1.

There are a number of useful properties of $\mathcal{I}$ that connect it to previous queries.

Observation 1. *If $|\mathcal{I}| = k$ then the set $\mathcal{I}$ is the same as the result of the top-k query for the utility function.*

Observation 2. *The set $\mathcal{I}$ consists of the tuples that have at most ϵ regret ratio [3] compared with the optimal.*

Following a standard assumption made in the literature [42], [5], [33], [3], we assume in the rest of this paper that the utility function is a linear function that can be represented as $f(p) = u \cdot p$, where $u \in \mathbb{R}^d_+$. As argued in [28], attributes can be scaled non-linearly to capture a wider class of utility functions. We will later assume that $\max_{i=1}^{d} u_i = 1$ which can be achieved by normalizing the utility function by its largest value.

Since it may not be possible to always compute the set $\mathcal{I}_{f,\epsilon}$ exactly (to be explained why in Section III-B), or doing so will require excessive user interaction, we are satisfied with approximating the set.

Definition 3. *We define a set S to be an α-approximation for $\mathcal{I}$ if $\mathcal{I} \subseteq S$ and for each $p' \in S - \mathcal{I}$ we have that $p^* \cdot u - (1+\epsilon)p' \cdot u \leq \alpha$, where $f(p) = u \cdot p$ is the user's linear utility function and $p^* = \text{argmax}_{p \in D} p \cdot u$.*

While we allow a few tuples that are not ϵ-indistinguishable from the optimal to be output, the $\mathcal{I} \subseteq S$ condition means that we never omit 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 gets close to zero, we get closer to the set $\mathcal{I}$.

We next give a necessary condition for points in $\mathcal{I}$ that can be used to speed up computation.

Definition 4. *For any $c \geq 1$, we say that a tuple a c-dominates another tuple b if a dominates $cb = (cb_1, \ldots, cb_d)$.*

Definition 5. *The c-skyline of a set P is the set of points in P that are not c-dominated by any other tuples in P.*

We now use these definitions to make the following claim:

Observation 3. *If a tuple p' is not in the $(1+\epsilon)$-skyline, then it is not in $\mathcal{I}$.*

Proof. Let p' be a tuple not in the $(1+\epsilon)$-skyline, meaning that it is $(1+\epsilon)$-dominated by some $p \in D$. Thus, we have that for any user utility vector u, $(1+\epsilon)p' \cdot u < p \cdot u \leq \max_{x \in D} x \cdot u$. Thus, $p' \notin \mathcal{I}$. □

As a result, for all our algorithms, we perform a pre-processing step in which we remove all the $(1+\epsilon)$-dominated tuples and give each algorithm the $(1+\epsilon)$-skyline.

Since for any $\epsilon' < \epsilon$ it is clear that ϵ'-indistinguishability implies ϵ-indistinguishability, we have the following fact.

Observation 4. *For any utility f and $\epsilon' < \epsilon$, $\mathcal{I}_{f,\epsilon'} \subseteq \mathcal{I}_{f,\epsilon}$.*

Thus, if the query has to be re-computed for a smaller value of ϵ, we can always start with the previous solution and prune down from it.

A. User Interaction

To find the set of indistinguishably optimal points, we will need to ask the user questions to narrow down their utility function. Similar to [4], [5], we model this interaction in the form of rounds in which the user is shown s tuples for q rounds (sometimes called questions) and is asked for their favorite tuple in each round. Note that 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 small number of tuples and pick their favorite. We also later consider the possibility that they make errors in their selections.

B. Impossibility Result for Real Points

Unfortunately, it is possible to show that if we are restricted to only real tuples in the database, there are cases in which we cannot eliminate an arbitrary number of tuples in the output of the query.

Theorem 1. *For every integer $f > 1$ and $\epsilon > 0$, there exists a database such that any deterministic algorithm that shows only real tuples and outputs all tuples in $\mathcal{I}$ must also output f false positive tuples for some utility function.*

Proof. Fix any $f > 1$ and $\epsilon > 0$. Let $m = \lceil (1+\epsilon)f \rceil$ and define the database D of size $m+1$ as follows:

$$D = \{p_i \equiv (i/m, 1 - i/m) : 0 \leq i \leq m\}.$$

Consider a pair of users who have utility vectors $u = (1, 0)$ and $u' = \left(1, \frac{1}{1+\epsilon}\right)$. We first show that both users have identical relative valuation for any pair of tuples in D. For any i, j where $0 \leq i < j \leq m$, let $p_i = (i/m, 1 - i/m)$ and $p_j = (j/m, 1 - j/m)$. Then

$$u \cdot p_i = i/m < j/m = u \cdot p_j.$$

Similarly,

$$\begin{aligned} u' \cdot p_i &= \frac{i}{m} + \frac{1}{(1+\epsilon)}\left(1 - \frac{i}{m}\right) \\ &= \frac{i}{m}\left(1 - \frac{1}{1+\epsilon}\right) + \frac{1}{1+\epsilon} \\ &< \frac{j}{m}\left(1 - \frac{1}{1+\epsilon}\right) + \frac{1}{1+\epsilon} \\ &= u' \cdot p_j. \end{aligned}$$

Consequently, whenever the users with utility vectors u and u' are shown any set of tuples from D, they will always make the same selections and thus be shown the same set of output tuples by any deterministic algorithm.

We next show that the sets $\mathcal{I}_{u,\epsilon}$ (for user with vector u) and $\mathcal{I}_{u',\epsilon}$ (for user with vector u') are quite different. It is easy to verify that in both cases the optimal tuple is $p_m = (1, 0)$ with (in both cases) utility 1. For the utility function u, we know we can omit all the tuples $p_0, \ldots, p_{f-1}$ from the output set as for each $i, 0 \leq i < f$,

$$(1+\epsilon)u \cdot p_i = (1+\epsilon)i/m < (1+\epsilon)f/m \leq 1,$$

as $m \geq (1+\epsilon)f$. On the other hand, we cannot omit any tuples for the utility function u' as for each $0 \leq i \leq m$,

$$\begin{aligned} (1+\epsilon)u' \cdot p_i &= (1+\epsilon)\left(\frac{i}{m} + \left(1 - \frac{i}{m}\right)\left(\frac{1}{1+\epsilon}\right)\right) \\ &= \frac{(1+\epsilon)i}{m} + \left(1 - \frac{i}{m}\right) \\ &= \frac{\epsilon i}{m} + 1 \geq 1, \end{aligned}$$

so all tuples in D must be output.

In summary, any algorithm that shows only real tuples and outputs all tuples in $\mathcal{I}_{u',\epsilon}$ must also output f unnecessary (false positive) tuples for $\mathcal{I}_{u,\epsilon}$. □

C. User Error

Since our underlying assumption in this paper is that users may not be able to distinguish between some of their top choices, it stands to reason that they may also make errors when selecting their favorite choice from a list of options. In our model, we assume that users have an unknown utility function that captures their true valuation of each tuple, but they make small errors in their selections (e.g., due to making a hasty decision) and thus may not select their optimal tuple each time, negatively affecting the algorithms.

We account for user error by introducing a parameter $\delta > 0$ that denotes the amount of error that a user can have when making a selection. In other words, we will assume that the user may not be able to distinguish a pair of tuples p and q if they are δ-indistinguishable and as a result may even pick an option that is slightly worse than another according to their underlying utility function.

The parameter δ can be selected to be large in cases where the user doesn't want to spend a lot of time examining all the options and makes a lower-cost/higher-error selection. In most cases, though, to keep things simple, we will usually assume that $\delta = \epsilon$ to simplify the parameter selection step.

Notation	Meaning
D	The set of all points, $\|D\| = n$
d	The number of attributes
u	The user's optimal utility vector
p^*	The optimal point for u, i.e., $\arg\max_{p \in P} u \cdot p$
ϵ	The user's desired indistinguishability threshold
δ	The user's error parameter
$\mathcal{I}_{u,\epsilon}$ or $\mathcal{I}$	The ϵ-indistinguishable optimal points
s	The number of tuples displayed at each round
q	The number of rounds of questions asked
α	The approximation factor for a non-exact solution

TABLE II: Frequently used notation

D. Problem Definition

We are now ready to formally define the problem solved in this paper.

Problem: Given a database of tuples $D \subseteq \mathbb{R}_+^d$ and parameters $\epsilon > 0$ and $0 \leq \delta < 1$, approximately compute a user's ϵ-indistinguishable set $\mathcal{I}$ by interactively asking the user to pick their favorite point from a series of small sets, assuming that the user may err on δ-indistinguishable points.

For convenience, some of the most frequently used notation is given in Table II.

IV. Algorithms with Artificial Tuples (No Error)

For ease of presentation, we start with algorithms with no user error (i.e., $\delta = 0$). Our first algorithm, Squeeze-u (Algorithm 1), uses artificial tuples to approximate $\mathcal{I}$. It does this by narrowing down the parameters of the user's utility function similarly to the UtilityMax algorithm [4]. Then armed with bounds for the utility function, it eliminates all the tuples that cannot possibly be ϵ-indistinguishable from the optimal in such a way as to give a faithful approximation for $\mathcal{I}$.

More specifically, the first part of the algorithm (Lines 1-17) first determines $i^* = \arg\max_{i \in \{1,\ldots,d\}} u_i$, the largest coefficient in the utility vector u by asking $\lceil (d-1)/(s-1) \rceil$ questions of the user, where s is the number of points shown at a time. The attributes are chosen in such a way that the user will prefer tuple e_i over e_j precisely when $u_i \geq u_j$.

By normalizing to make the i^* attribute equal to one, we next bound the range of the remaining coefficients to the range $[0, 1]$. We then maintain the invariant $L_i \leq u_i \leq H_i$ for all

Algorithm 1 Squeeze-u(D, s, ϵ)

Input: A database of tuples, $D \subset [0,1]^d$; Number of tuples that can be showed to the user at a time, s; the indistinguishability parameter, ϵ.

Output: A subset of D.

1: Remove all points from D that are $(1+\epsilon)$-dominated by another point in D. ▷ Use Obs. 3 to prune some tuples
▷ Discover i^* such that $u_{i^*} = \max_{1 \le i \le d} u_i$
2: For each $1 \le i \le d$, let $m_i = \min_{p \in D} p_i$ and $M_i = \max_{p \in D} p_i$.
3: For each $1 \le i \le d$, define e_i as the point with $e_i[i] = m_i + (M_i - m_i)/2$ and $e_i[j] = m_j$ for all $j \neq i$.
4: Let $i^* = 1, i = 2$.
5: **while** $i < d$ **do**
6: Display $e_{i^*}, e_i, \ldots, e_{i+s-2}$ and say that the user chooses $e_{i'}$.
7: Let $i^* = i'$.
8: Let $i = i + s - 1$.
▷ Bound each u_j using the invariant $L_j \le u_j \le H_j$
9: For each $1 \le j \le d$, $j \neq i^*$, define $L_j = 0$ and $H_j = 1$. Define $L_{i^*} = H_{i^*} = 1$
10: Let $i = 1$.
11: **while** the user does not terminate **do**
12: **if** $i = i^*$ **then**
13: Let $i = (i \bmod d) + 1$.
14: For each $0 \le j \le s$, let $\chi_j = L_i + j(H_i - L_i)/s$.
15: Let $p_1, p_2, p_3, \ldots, p_s$ be defined as follows. For each $j \notin \{i, i^*\}$ and $1 \le k \le s$, let $p_k[j] = 0$. For each $1 \le k \le s$, let $p_k[i^*] = \sum_{l=k}^{s-1} \frac{\chi_l}{s}$ and let $p_k[i] = k/s$.
16: Display $p_1, p_2, p_3, \ldots, p_s$ and say that the user chooses p_c.
17: Update $L_i = \chi_{c-1}$ and $H_i = \chi_c$
18: Let $i = (i \bmod d) + 1$.
▷ Use the u_i bounds to prune out tuples not in $\mathcal{I}$
19: $C = D$
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 $[L_1, H_1] \times [L_2, H_2] \times \ldots [L_d, H_d]$.
22: Return C

$1 \le i \le d$, and reduce the difference of one of the $H_i - L_i$ by a factor of s with each question. In this way, we bound $H_i - L_i$ to within an exponentially small factor in terms of the number of questions asked to the user. We then eliminate the tuples whose utility is at least $(1+\epsilon)$ smaller than another tuple for all feasible utility functions that remain (Lines 18-20).

Before we can prove the guarantee of the algorithm, we need to prove the following lemma that shows that the algorithm approximates the true utility function very closely.

Lemma 1. *After answering q questions, Algorithm 1 guarantees a $\frac{1}{s^{(q-1)/(d-1)}}$ additive approximation for each u_i.*

Proof. We assume that the utility function u is such that

$$\max_{1 \le i \le d} u_i = 1.$$

If this is not the case, the utility vector can be scaled down by $\max_{1 \le i \le d} u_i$ without affecting the computation of $\mathcal{I}$.

The first part of the algorithm (Lines 1-6) first identifies one dimension i^* for which $u_{i^*} = 1$. This is done by creating artificial tuples such that the user will prefer e_i to e_j precisely when $u_i \ge u_j$ and then simply finding the maximum u_i value.

After this, we know that $u_{i^*} = 1$ and that $u_i \in [0,1]$ for all other i. The algorithm next takes turns iterating through each $i \neq i^*$ and narrowing down the value of u_i by a factor of s. It does so by displaying a set of very carefully crafted points, given in Line 14.

If the user chooses p_c in Line 15, we can infer that p_c is preferred to p_{c-1}, or that $u \cdot p_c \ge u \cdot p_{c-1}$. Now, p_c and p_{c-1} are identical for all indexes $j \notin \{i, i^*\}$. Thus, we have that

$$u_i \frac{c}{s} + u_{i^*} \sum_{l=c}^{s-1} \frac{\chi_l}{s} \ge u_i \frac{c-1}{s} + u_{i^*} \sum_{l=c-1}^{s-1} \frac{\chi_l}{s}$$

which is equivalent to $u_i \ge \chi_{c-1} = L_i + (c-1)(H_i - L_i)/s$ since $u_{i^*} = 1$.

Similarly, since the user chooses p_c over p_{c+1} we have that

$$u_i \frac{c}{s} + u_{i^*} \sum_{l=c}^{s-1} \frac{\chi_l}{s} \ge u_i \frac{c+1}{s} + u_{i^*} \sum_{l=c+1}^{s-1} \frac{\chi_l}{s}$$

which is equivalent to $u_i \le \chi_c = L_i + c(H_i - L_i)/s$ since $u_{i^*} = 1$.

Putting these together, we see that u_i has been narrowed down from $[L_i, H_i]$ to $[L_i + (c-1)(H_i - L_i)/s, L_i + c(H_i - L_i)/s]$. In other words, we have reduced the range of u_i by a factor of s.

After the user has been asked q rounds of questions, we have a tight bound on each u_i. Since we use the first $\lceil \frac{d-1}{s-1} \rceil$ questions to identify i^* and we can narrow down each of the other $d-1$ dimensions only once every $d-1$ iterations, we have that for each $1 \le i \le d$, $|H_i - L_i| \le \frac{1}{s^{\lfloor (q - \lceil \frac{d-1}{s-1} \rceil)/(d-1) \rfloor}} \approx \frac{1}{s^{(q-1)/(d-1)}}$. □

Example: If we have $d = 3$ and $s = 5$, then asking just 7 questions gives a $1/125$ additive approximation for each u_i.

Theorem 2. *Algorithm 1 returns a $O(d/s^{(q-1)/(d-1)})$ approximation of $\mathcal{I}$ after asking the user q questions.*

Proof. Let u be the user's true utility function and p^* be the user's favorite point in the database, i.e., $p^* = \arg\max_{p \in D} p \cdot u$. Let $\mathcal{R} = [L_1, H_1] \times \ldots \times [L_d, H_d]$. Say that for any $v_1, v_2 \in \mathcal{R}$, $|v_1[i] - v_2[i]| \le \tau$ for all $1 \le i \le d$. For any tuple p'

output by the algorithm, there must be some $v \in \mathcal{R}$ such that $(1+\epsilon)p' \cdot v \geq p^* \cdot v$. Thus,

$$\begin{aligned} p^* \cdot u &= \sum_{i=1}^{d} p_i^* u_i && (1)\\ &\leq \sum_{i=1}^{d} p_i^* v_i + \tau \sum_{i=1}^{d} p_i^* && (2)\\ &\leq (1+\epsilon)p' \cdot v + \tau \sum_{i=1}^{d} p_i^* && (3)\\ &\leq (1+\epsilon)p' \cdot u + (1+\epsilon)\tau \sum_{i=1}^{d} p_i' + \tau \sum_{i=1}^{d} p_i^* && (4)\\ &\leq (1+\epsilon)p' \cdot u + \tau d(2+\epsilon), && (5) \end{aligned}$$

where lines (2) and (4) follow from the fact that $|u[i]-v[i]| \leq \tau$ for all $1 \leq i \leq d$, line (3) follows from the definition of v above, and line (5) is due to the fact that $p_i' \leq 1$ and $p_i^* \leq 1$ for each $1 \leq i \leq d$.

The theorem follows by substituting $\tau = \frac{1}{s^{(q-1)/(d-1)}}$ from Lemma 1. □

Example: If we have $d = 3$, $s = 5$, and $\epsilon = 0.1$, then asking just 7 questions gives a 0.0504-approximation.

A. Running Time

The running time of the interactive part of Algorithm 1 (Lines 1-16) is negligible as it only needs to compute the points to be displayed. For pruning the points that are dominated by other points for all utility functions in the region (Lines 18-20), we need to do more computation.

In order to check if some point q is $(1+\epsilon)$-dominated by some point p for all utility functions in $\mathcal{R} = [L_1, H_1] \times [L_2, H_2] \times \ldots [L_d, H_d]$, we would have to check $(p - q(1+\epsilon)) \cdot v > 0$ for all $v \in \mathcal{R}$. Since $\mathcal{R}$ is a convex region, we can perform the check only on the boundary points $\{L_1, H_1\} \times \{L_2, H_2\} \times \ldots \{L_d, H_d\}$. Even so, this requires us to check 2^d conditions for all pairs of points p and q, resulting in $\Omega(2^d n^2)$ computation.

To get around this computational cost (particularly the quadratic dependency on n), we replace the above computation with a simpler one with similar efficacy. We first compute a lower bound for the utility of the user as $V = \max_{p \in D} p \cdot (L_1, \ldots, L_d)$. We then check, for each point, if after scaling it up by $(1+\epsilon)$ and evaluated on the upper bound for the utility function ranges it is still smaller than V. That is, we remove the points in the set $\{p \in D : (1+\epsilon)p \cdot (H_1, \ldots, H_d) < V\}$. Intuitively, if the utility of these points scaled up by $(1+\epsilon)$ is still less than the lower bound utility V even when using the upper bound for the utility function, there is no way that they can be in $\mathcal{I}$. We demonstrate in the evaluation section that this heuristic approach gives good performance with real and synthetic data sets. The running time of this approach is simply $O(n)$ since we can both compute V and prune points in linear time in n.

V. Algorithms with Real Tuples (No Error)

Since it is more realistic to show the user points that are actually in the database D during interaction, we study algorithms for the real point case as well. As noted in Section III-B, though, we cannot hope to show any worst case bounds in this setting.

To prune down the tuples in D to the set $\mathcal{I}$, we present the user with a series of options from D that allow us to narrow down the space of their utility function u. Following the techniques used in [5], we define the feasible region of the utility function by $\mathcal{R}_j \in \mathbb{R}^d$ after j queries have been made to the user. For this case, we normalize the vector u by the sum of its components without changing the result of the query. Thus, the region is initially $\mathcal{R}_0 = \{r \in \mathbb{R}_+^d : \sum_{i=1}^{d} r_i = 1\}$.

To update the feasible region, we use the utility hyperplane technique [5] to narrow it down after each question is asked of the user. For example, if on question j we determine that the user prefers tuple a to b, we then compute

$$\mathcal{R}_j = \{v \in \mathcal{R}_{j-1} : (a-b) \cdot v > 0\},$$

since we know that $a \cdot u > b \cdot u$.

We can use any of these $\mathcal{R}_j$ regions to prune down the set of tuples using the following lemma.

Lemma 2. *It is safe to prune a point b if there exists a point $a \in D$ such that $\mathcal{R}' \equiv \{v \in \mathcal{R}_j : (b(1+\epsilon) - a) \cdot v \geq 0\} = \emptyset$.*

Proof. Since $\mathcal{R}'$ only contains vectors for which $(b(1+\epsilon) - a) \cdot v \geq 0$ and $\mathcal{R}' = \emptyset$, it follows that $b(1+\epsilon) \cdot v < a \cdot v$ for each $v \in \mathcal{R}_j$ and thus $b(1+\epsilon) \cdot u < a \cdot u \leq p^* \cdot u$, where p^* is the optimal point for u in D. We can hence safely prune b as it is not ϵ-indistinguishable from p^*. □

We define a set of candidate points $\mathcal{C}$ that consists of tuples that are at most ϵ off from being the optimal for some utility function in $\mathcal{R}$. A crucial difference from [5] is that we want to maintain all such tuples—including non-skyline ones—and thus we have to do considerably more computation.

To decide which points to show the user at each iteration, we propose the following two heuristic techniques:

Minimize $\mathcal{R}$ width (MinR): Recall from the proof of Theorem 2 that the approximation factor is proportional to the maximum difference in range of feasible values of the u_i values. We call this quantity the *width* of $\mathcal{R}$ and our first algorithm attempts to minimize this value. More specifically, for any subset of s tuples, we consider the region $\mathcal{R}'$ formed if each of the points is chosen by the user and compute the average width of these regions. The goal is to then display a set that minimizes this quantity.

Minimize $\mathcal{R}$ diameter (MinD): Our second heuristic aims to minimize the diameter of the region $\mathcal{R}$ by preferring sets of points such that the average diameter is as small as possible over all possible choices made by the user.

For completeness, the full detail for these algorithms are given in Algorithm 2.

Algorithm 2 MinR / MinD(D, s, ϵ)

Input: A database of tuples, $D \subset [0,1]^d$; Number of tuples that can be showed to the user at a time, s; the indistinguishability parameter, ϵ.

Output: A subset of D.

Let $\mathcal{C}$ be all points from D that are not $(1+\epsilon)$-dominated by another point in D. ▷ Use Obs. 3 to prune some tuples
▷ Maintain a feasible region for the utility function
Set $\mathcal{R}_0 = \{u \in \mathbb{R}^d_+ : \sum_{i=1}^d u_i = 1\}$.
Set $j = 1$.
while the user does not terminate **do**
  repeat
    Randomly select a set S of s tuples from $\mathcal{C}$.
    (MinR) Compute the average width of $\mathcal{R}$ over all possible optimal points $p \in S$.
    (MinD) Compute the average diameter of $\mathcal{R}$ over all possible optimal tuples $p \in S$.
    Maintain the set S if it is the best so far.
  until T attempts have been made
  Display the best set S to the user and say that they choose tuple c.
  ▷ Narrow down feasible region based on user choice
  Update $\mathcal{R}_j = \bigcap_{p \in S - \{c\}} \{v \in \mathcal{R}_{j-1} : (c-p) \cdot v > 0\}$.
  Update $\mathcal{C}$ to be the tuples that are not pruned with the region $\mathcal{R}_j$.
  Set $j = j + 1$
Return $\mathcal{C}$

A. Running Time

For both the proposed heuristics (MinR and MinD), a greedy algorithm would have to try all possible sets of size s to find the one that minimizes the respective metric. Unfortunately, there are $\binom{m}{s}$ such sets (where m is the size of the $(1+\epsilon)$-skyline and s is the number of tuples shown at a time to the user) which makes the computation prohibitively expensive.

To avoid the above issues, instead of trying all possible sets of size s, we try several randomly selected sets and keep the one that optimizes the respective metric. If we try T such sets, then the running time for MinR and MinD width become $O(sT)$ per round of interaction for updating the region $\mathcal{R}$ for each of the s choices. We also prune tuples at each iteration in $O(n)$ time using an R-tree. Thus, the overall runtime of the algorithm is $O(q(sT+n))$, where q is the number of rounds of interaction (questions) with the user.

VI. User Error

Since our original motivation for this paper is that users may have a hard time choosing between tuples that have very similar utility values for them, we study what happens if it is possible for users to make errors in the interaction process. In this section, we show how to account for the inability of the user to correctly pick between tuples that have very similar utility values for them.

Algorithm 3 Squeeze-u2(D, s, ϵ, δ)

Input: A database of tuples, $D \subset [0,1]^d$; Number of tuples that can be showed to the user at a time, s; the desired indistinguishability parameter, ϵ; the user's indistinguishability parameter when comparing points, δ.

Output: A subset of D.

Remove all points from D that are $(1+\epsilon)$-dominated by another point in D. ▷ Use Obs. 3 to prune some tuples
▷ Discover i^* such that $u_{i^*} = \max_{1 \le i \le d} u_i$
For each $1 \le i \le d$, define e_i as the point with $e_i[i] = 1$ and $e_i[j] = 0$ for all $j \neq i$.
Let $i^* = 1, i = 2$.
while $i < d$ **do**
  Display $e_{i^*}, e_i, \ldots, e_{i+s-2}$ and say that the user chooses $e_{i'}$.
  Let $i^* = i'$.
  Let $i = i + s - 1$.
▷ Bound each u_j using the invariant $L_j \le u_j \le H_j$
For each $1 \le j \le d$, $j \neq i^*$, define $L_j = 0$ and $H_j = (1+\delta)^{\lceil (d-1)/(s-1) \rceil}$. Define $L_{i^*} = H_{i^*} = 1$.
Let $i = 1$.
while the user does not terminate **do**
  if $i = i^*$ **then**
    Let $i = (i \bmod d) + 1$.
  For each $0 \le j \le s$, let $\chi_j = L_i + j(H_i - L_i)/s$.
  Let $p_1, p_2, p_3, \ldots, p_s$ be defined as follows. For each $j \notin \{i, i^*\}$ and $1 \le k \le s$, let $p_k[j] = 0$. For each $1 \le k \le s$, let $p_k[i^*] = \sum_{l=k}^{s-1} \frac{\chi_l}{s}$ and let $p_k[i] = \frac{k}{s}$.
  Display $p_1, p_2, p_3, \ldots, p_s$ and say that the user chooses p_c.
  Update $L_i = \max(L_i, \frac{1}{1+c\delta}\left(\chi_{c-1} - \delta \sum_{j=c}^{s-1} \chi_j\right))$ and $H_i = \min(H_i, \frac{1}{1-c\delta}\left(\chi_c + \delta \sum_{j=c}^{s-1} \chi_j\right)$.
  Let $i = (i \bmod d) + 1$.
▷ Use the u_i bounds to prune out tuples not in $\mathcal{I}$
$C = D$
for all $p \in D$ **do**
  Remove all points from C that are $(1+\epsilon)$-dominated by p for all utility functions in $[L_1, H_1] \times [L_2, H_2] \times \ldots [L_d, H_d]$.
Return C

For generality, we assume that the user cannot tell apart tuples that are δ-indistinguishable. In practice, we may want to set $\delta = \epsilon$ for simplicity.

A. Artificial Tuples

We account for when the user cannot discern between δ-indistinguishable tuples by adapting Algorithm 1 to account for this in Algorithm 3. As before, we attempt to find the index i such that u_i is greatest. This time, however, because

of the user error, they may select a sub-optimal point each time. However, we are guaranteed that this sub-optimal point is at most $(1+\delta)$ factor short of the optimal. Thus, after the $\lceil (d-1)/(s-1) \rceil$ rounds of comparison (Lines 3-6) the selected i^* may be a $(1+\delta)^{\lceil (d-1)/(s-1) \rceil}$ factor short of the true optimal value. We account for this by still selecting $L_i^* = H_i^* = 1$ but now we set all the other upper bounds to $H_j = (1+\delta)^{\lceil (d-1)/(s-1) \rceil}$.

Since the user can make up to δ error, we cannot hope to compute each u_i to arbitrary precision any longer. We instead stop when each $H_i - L_i$ cannot be improved much more. We will show next that even with δ user error the algorithm still gives a good approximation.

Theorem 3. *Alg. 3 returns a $O(d\delta s)$-approximation for $\mathcal{I}$.*

Proof. If the user chooses p_c in Line 14, we know that $(1+\delta)p_c \cdot u$ is at least $p_{c-1} \cdot u$ (otherwise, the user would prefer p_{c-1} and be able to δ-distinguish it from p_c). In other words,

$$(1+\delta)\left(\frac{cu_i}{s} + u_{i^*}\sum_{l=c}^{s-1}\frac{\chi_l}{s}\right) \geq \frac{(c-1)u_i}{s} + u_{i^*}\sum_{l=c-1}^{s-1}\frac{\chi_l}{s}$$

or

$$\frac{(1+c\delta)u_i}{s} \geq u_{i^*}\left(\frac{\chi_{c-1}}{s} - \delta\sum_{l=c}^{s-1}\frac{\chi_l}{s}\right).$$

Substituting in $u_i^* = 1$, this gives us

$$\begin{aligned} u_i &\geq \frac{1}{1+c\delta}\left(\chi_{c-1} - \delta\sum_{l=c}^{s-1}\chi_l\right) \\ &\geq \frac{1}{1+s\delta}\left(\chi_{c-1} - \delta(s-1)\right) \\ &= \left(1 - s\delta + \Omega(\delta^2)\right)\left(\chi_{c-1} - \delta(s-1)\right) \\ &\geq \chi_{c-1} - \delta(s-1+s\chi_{c-1}) + \Omega(\delta^2) \\ &\geq \chi_{c-1} - \delta(2s-1) + \Theta(\delta^2). \end{aligned} \tag{6}$$

Similarly, since the user chooses p_c over p_{c+1} we have that

$$(1+\delta)\left(\frac{cu_i}{s} + u_{i^*}\sum_{l=c}^{s-1}\frac{\chi_l}{s}\right) \geq \frac{(c+1)u_i}{s} + u_{i^*}\sum_{l=c+1}^{s-1}\frac{\chi_l}{s}$$

or

$$\frac{u_i(1-c\delta)}{s} \leq u_{i^*}\left(\frac{\chi_c}{s} + \delta\sum_{l=c}^{s-1}\frac{\chi_l}{s}\right).$$

Again substituting $u_i^* = 1$, we get

$$\begin{aligned} u_i &\leq \frac{1}{1-c\delta}\left(\chi_c + \delta\sum_{l=c}^{s-1}\chi_l\right) \\ &\leq \frac{1}{1-s\delta}\left(\chi_c + \delta(s-1)\right) \\ &\leq \left(1 + s\delta + O(\delta^2)\right)\left(\chi_c + \delta(s-1)\right) \\ &\leq \chi_c + \delta(s-1+s\chi_c) + O(\delta^2) \\ &\leq \chi_c + \delta(2s-1) + \Theta(\delta^2). \end{aligned} \tag{7}$$

Since $\delta < 1$ and will typically be small (e.g., less than 0.1), we ignore the quadratic terms. Putting the above results together, we get that after each iteration we update the bounds of u_i from $[L_i, H_i]$ to the formulas given in Equations 6 and 7. We thus update L_i and H_i appropriately in Line 15 of Algorithm 3.

After a few iterations, the error of $\delta(4s-2)$ will start to dominate and we cannot narrow down the range of u_i any longer. Substituting $\tau = \delta(4s-2)$ into the bound from Theorem 2, we get that for the optimal point p^* and any tuple q output by the algorithm we have that $p^* \cdot u \leq (1+\epsilon)q \cdot u + d(2+\epsilon)\delta(4s-2)$. Thus, Algorithm 3 will give a $O(d\delta s)$-approximation for $\mathcal{I}$. □

Example: If $d = 3$, $s = 3$, $\epsilon = 0.1$, and $\delta = 0.001$, then we have a 0.063-approximation.

We show in the experimental section that the algorithm also works for slightly larger δ (e.g., 0.01) but has poor performance for larger values (e.g., 0.1).

B. Real Tuples

If the user indicates that they prefer tuple a to b, we then have that $a(1+\delta) \cdot u \geq b \cdot u$ and we can thus update the feasible region as follows:

$$\mathcal{R}_j = \{v \in \mathcal{R}_{j-1} : (a(1+\delta) - b) \cdot v > 0\},$$

which slightly weakens the update rule. All the previous results and algorithms follow.

VII. Experimental Evaluation

All the experiments in this section were performed on a machine with a quad-core 3.40GHz CPU with 16GB RAM. All code was implemented in C++ and is freely available[1].

Data sets: We use both synthetic and real data sets. For synthetic data, we used a data set generator [2] to create anti-correlated data sets which have large skyline sizes. The real data sets we used included an Island data set with 63383 geographical 2D coordinates, a 4D data set with 21961 NBA player/season records, and a 6D data set with 12793 housing utility values (House). For all data sets, we normalized them so that the largest value was always one.

Algorithms: We evaluated our algorithm for artificial points, Squeeze-u, together with the two algorithms that are constrained to using real points, MinR and MinD. We note that these algorithms are not directly comparable since the ones showing real points are at a disadvantage, but showing them together allows us to compare their performance nonetheless. We also included results for the state-of-the-art interactive regret minimization algorithm UH-Random [5] that shows the user randomly selected points from the database at each step and uses the user's selection to update the feasible region for the utility function $\mathcal{R}$ and produce an output set $\mathcal{C}$. We modified this algorithm to perform the same pruning computation performed by our algorithms to guarantee that all tuples in $\mathcal{I}$ would appear in the output and in the case when $\delta > 0$ the modified update rules were used. Note that other techniques such as skyline and regret-minimization cannot be

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

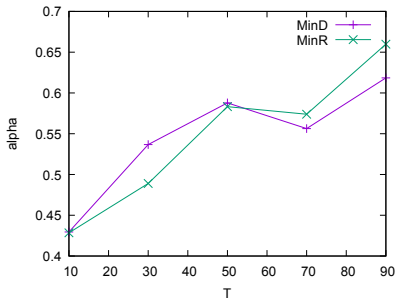


Fig. 1: Results for real points algorithms when varying T on NBA ($q = 3d, s = d, \epsilon = 0.05, \delta = 0.0$)

used here as there is no clear way in which to adaptively select the tuples shown in each subsequent round of interaction.

Parameter settings: In all our experiments, we evaluated the algorithms on ten independently random utility functions and reported the average performance. We studied the approximation value (α) for the different algorithms while varying various parameters. Recall that 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 default values of $\epsilon = \delta = 0.05$, s (number of tuples per round) equal to the number of dimensions d, and $q = 3d$. We selected s to be equal to the number of dimensions so that the first part of the Squeeze-u algorithm would complete in one round and $q = 3d$ so that each component of u would be updated at least 3 times by the Squeeze-u algorithm. For the MinR and MinD algorithms we also had a parameter T, the number of random repetitions to find a set that minimizes the width and diameter. We tried many values of T on the real data sets, with 100 independent runs each to reduce variance, and the results are shown in Figure 1 for the NBA dataset. As seen in this figure, the α value had increasing error with T, so we set $T = 10$ for all the remaining experiments.

A. No User Error

We first compared the performance of the algorithms when there was no user error (i.e., $\delta = 0$) to understand the behavior of the algorithms in this context. In Figure 2, we see the effect of increasing the number of questions on the value of α when fixing the number of points shown at each iteration to $s = d$. In all cases, the error for the Squeeze-u algorithm drops quickly with the number of questions and tends to zero. In contrast, the other algorithms, which are restricted to real points, have relatively high error and variability. This is due to the fact that real points may not allow us to eliminate many other points that are not ϵ-indistinguishable, as discussed in Section III-B. Note that the curves may not be monotonically decreasing with q because of small random variations in the randomly chosen utility functions.

We next studied the effect of varying the number of points shown in each round of user interaction, s. We fix the number of rounds to $q = 3d$ and vary s in Figure 3. Once again, the Squeeze-u algorithm is clearly the best and the others have a large amount of variability.

	Squeeze-u	UH-Random	MinD	MinR
Island	0.00	177.55	6.06	11.10
NBA	0.00	0.02	0.47	0.58
House	0.00	19.30	173.85	177.73

TABLE III: Running time (s.) ($\epsilon = 0.05, \delta = 0, s = d, q = 3d$)

	Squeeze-u	UH-Random	MinD	MinR
Island	0.00	557.61	8.54	8.60
NBA	0.00	0.02	0.57	0.66
House	0.00	54.56	149.92	144.33

TABLE IV: Running times (s.) ($\epsilon = \delta = 0.05, q = 3d, s = d$)

Figure 4 shows the performance of the algorithms when we vary ϵ from 0.001 to 0.1. (note log-scale on the x-axis). The performance of the Squeeze-u algorithm is fairly flat with this variation. This is in keeping with our expectations from theory since the theoretical bounds are independent of ϵ. The other, real point algorithms, do less well, especially when ϵ is larger.

We measured the running time of the algorithms in Table III ($s = d, q = 3d$ and $\epsilon = 0.05$). The Squeeze-u algorithm consistently took well under a hundredth of a second to execute. The MinD and MinR algorithms were very slow on House and UH-Random was very slow on Island.

B. With User Error

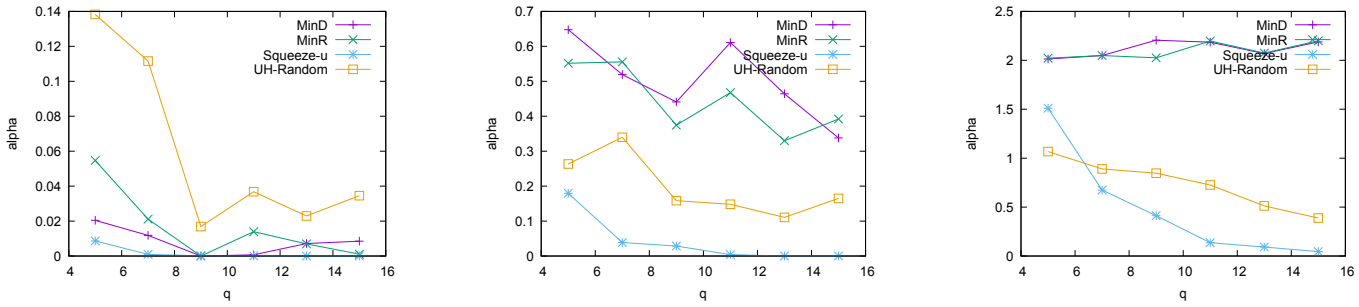
We next measured the performance of the algorithms with user error (i.e., $\delta > 0$). To maintain consistency with the previous figures, we label the results of the Squeeze-u2 algorithm (which accounts for up to δ error) as Squeeze-u. When allowing for δ error, it is possible for the user to mistakenly pick a sub-optimal tuple. For all the simulations, whenever the user was shown s tuples, we simulated user errors by collecting all the tuples that were δ-indistinguishable from the best among the s tuples and randomly picking one of them.

Figure 5 shows the performance of the algorithms when varying δ with $s = d, q = 3d$ and $\epsilon = 0.05$ (note logscale x-axis). All the algorithms show a degradation in performance when δ is increased from 1% to 10%, though Squeeze-u is still the overall best. The decrease in performance of Squeeze-u is in keeping with our theoretical expectations as we showed the bound for α to be proportional to δ in Section VI.

As can be seen in Table IV, the running times for most of the algorithms in this case are similar to the ones for when $\delta = 0$ (Table III). This is because the artificial tuple algorithms just show slightly different tuples and the real point algorithms have a weaker pruning condition, otherwise the algorithms are unchanged. Thus, we are able to account for user error without a significant run-time penalty.

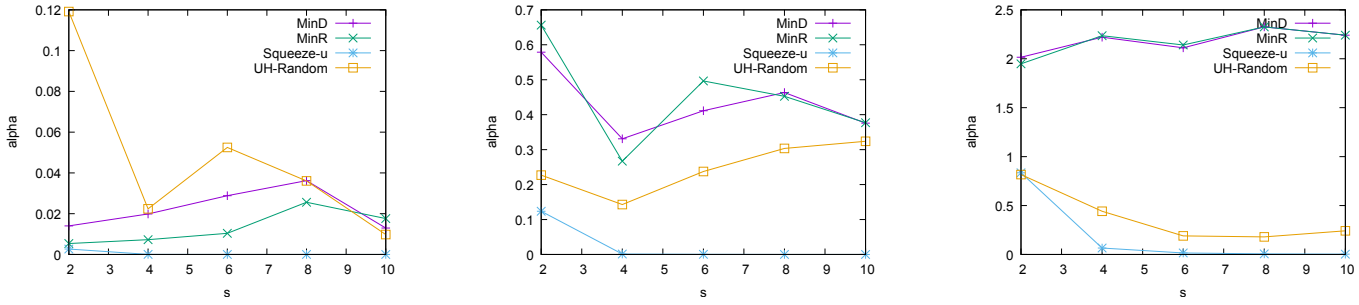
C. Scalability Test

We next performed a scalability test to examine the behavior of the algorithms when we increase the number of tuples or the number of dimensions. For these tests, we generated anti-correlated data sets of varying sizes using the data set generator mentioned previously [2]. We used anti-correlated



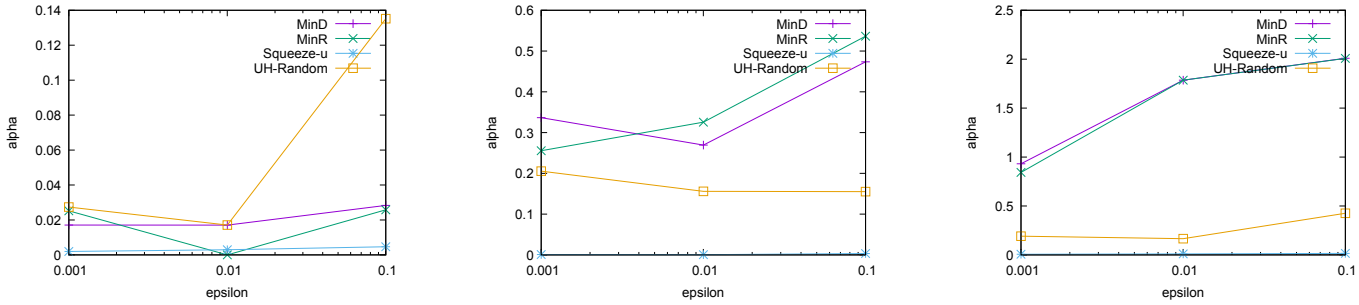
(a) Island (b) NBA (c) House

Fig. 2: Varying number of questions, q ($s = d, \epsilon = 0.05, \delta = 0$)



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

Fig. 3: Varying number of points shown per round, s ($q = 3d, \epsilon = 0.05, \delta = 0$)



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

Fig. 4: Varying ϵ ($s = d, q = 3d, \delta = 0$), log-scale on x-axis

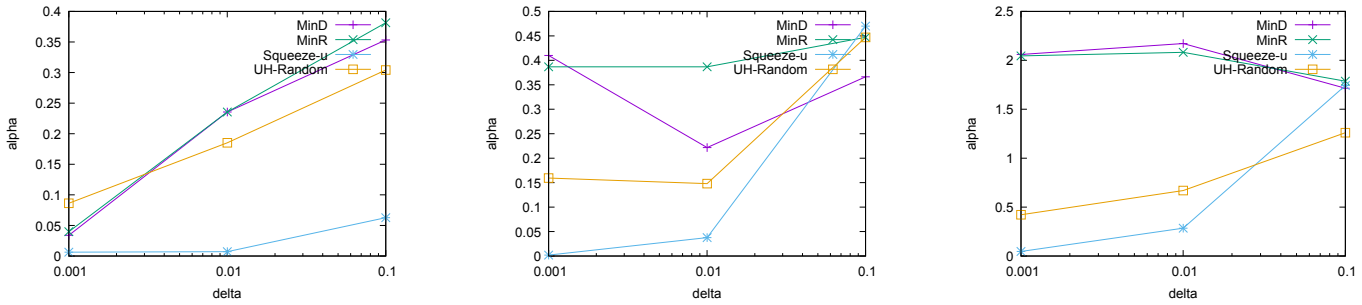
data so that the skyline size is large. In the interest of space, we only show the results with user error ($\delta > 0$) since this is the harder case.

We see in Figure 6a that for three-dimensional data with varying number of tuples, the performance of the algorithms stays fairly consistent even when we increase up to a million tuples (note that the x-axis is in logscale in Figure 6a). In particular, the Squeeze-u algorithm has very little degradation in approximation when the number of tuples is increased a thousandfold. The running time for all the algorithms do increase (note that both the x- and y-axes are in logscale in Figure 6b) as would be expected. The increase in running time for Squeeze-u is approximately linear (increasing by an order of magnitude with each increase in magnitude of number of tuples), as would be expected since we showed that the running time is linear in n in Section IV-A.

Figure 7a shows the performance of the algorithms when we fix the number of tuples to 10000 and vary the dimensionality. We now see instances where Squeeze-u does not have the best α value (Figure 7a). This is because the performance of the Squeeze-u algorithm seems to degrade for δ values greater than 0.01 (cf. Figure 5). The Squeeze-u algorithm's run-time continues to outperform the others (Figure 6b).

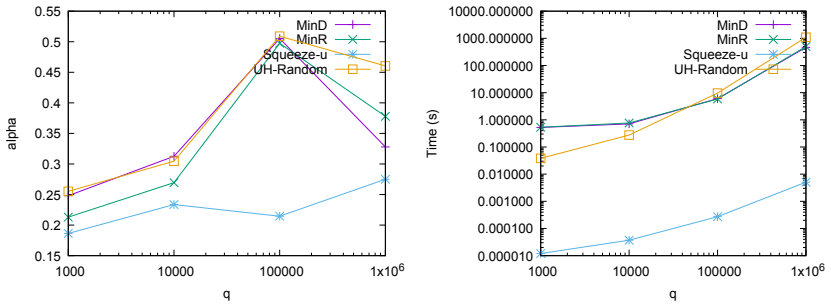
D. Summary

The experiments show that the Squeeze-u algorithm approximates the indistinguishability query very effectively. By having the user answer as few as 6 questions (when $d = 2$)



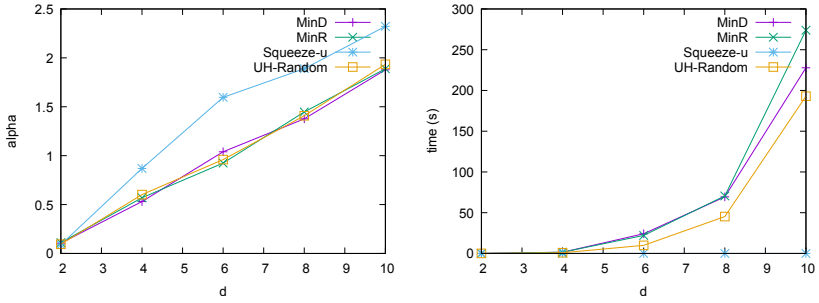
(a) Island (b) NBA (c) House

Fig. 5: Varying δ ($s = d, q = 3d, \epsilon = 0.05$), log-scale on x-axis



(a) alpha (b) time (s)

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



(a) alpha (b) time (s)

Fig. 7: Varying number of dimensions in synthetic data with fixed number of tuples ($n = 10000, s = 6, q = 18, \epsilon = 0.05, \delta = 0.05$)

and up to 18 (when $d = 6$) it bounds α to a small value (nearly zero) for almost all the data sets and parameter settings studied. The algorithms that are constrained to show only real points are not as consistent in their α or runtime performance. However, the other algorithms have a place in situations in which artificial tuples are undesirable.

In the case where $\delta > 0$, the Squeeze-u algorithm does well for small values of δ but doesn't perform as well when δ is larger than 1%. Future work might explore how to bound error better when there is a large amount of user error.

VIII. Conclusion

We present interactive algorithms for performing the indistinguishability query to identify all of the user's near-optimal tuples. We show that by having the user examine a few artificial tuples we can guarantee a close approximation of their optimal set with provable bounds and that using real tuples can also give good approximation in practice. Moreover, these algorithms generalize to the case in which the user can make small errors in their interactive selections. Experimental simulations validate the efficacy of our algorithms.

This novel query also raises a number of open questions. (1) Is it possible to get a closer approximation guarantee with artificial tuples? (2) What are the limits of the error that the user is allowed to make? (3) Can these results be generalized to non-linear [43] utility functions? We leave all these as open questions.

REFERENCES

[1] I. F. Ilyas, G. Beskales, and M. A. Soliman, "A survey of top-k query processing techniques in relational database systems," *ACM Comput. Surv.*, vol. 40, no. 4, Oct. 2008.

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

[3] D. Nanongkai, A. Sarma, A. Lall, R. Lipton, and J. Xu, "Regret-minimizing representative databases," in *Proceedings of the VLDB Endowment*, 2010.

[4] D. Nanongkai, A. Lall, A. D. Sarma, and K. Makino, "Interactive regret minimization," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, 2012.

[5] M. Xie, R. C.-W. Wong, and A. Lall, "Strongly truthful interactive regret minimization," in *Proceedings of the 2019 International Conference on Management of Data*, ser. SIGMOD '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 281–298.

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

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

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

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

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

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

[12] D. Mindolin and J. Chomicki, "Discovering relative importance of skyline attributes," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 610–621, 2009.

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

[14] ——, "On high dimensional skylines," in *Advances in Database Technology-EDBT 2006*. Springer, 2006, pp. 478–495.

[15] Y. Tao, X. Xiao, and J. Pei, "Efficient skyline and top-k retrieval in subspaces," in *TKDE*, 2007.

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

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

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

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

[20] P. Ciaccia and D. Martinenghi, "Reconciling skyline and ranking queries," vol. 10, no. 11. VLDB Endowment, aug 2017, p. 1454–1465. [Online]. Available: https://doi.org/10.14778/3137628.3137653

[21] K. Mouratidis and B. Tang, "Exact processing of uncertain top-k queries in multi-criteria settings," vol. 11, no. 8. VLDB Endowment, apr 2018, p. 866–879. [Online]. Available: https://doi.org/10.14778/3204028.3204031

[22] K. Mouratidis, K. Li, and B. Tang, "Marrying top-k with skyline queries: Relaxing the preference input while producing output of controllable size," in *Proceedings of the 2021 International Conference on Management of Data*, ser. SIGMOD/PODS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 1317–1330.

[23] C. Papadimitriou and M. Yannakakis, "On the approximability of trade-offs and optimal access of web sources," in *Proceedings 41st Annual Symposium on Foundations of Computer Science*, 2000.

[24] C. H. Papadimitriou and M. Yannakakis, "Multiobjective query optimization," in *Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, ser. PODS '01, 2001, p. 52–59.

[25] B. Jiang, J. Pei, X. Lin, D. W. Cheung, and J. Han, "Mining preferences from superior and inferior examples," in *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 390–398.

[26] D. Mindolin and J. Chomicki, "Discovering relative importance of skyline attributes," in *Proceedings of the VLDB Endowment*, 2009.

[27] J. Lee, G. won You, and S. won Hwang, "Personalized top-k skyline queries in high-dimensional space," *Information Systems*, vol. 34, no. 1, pp. 45–61, 2009.

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

[29] T. Liu *et al.*, "Learning to rank for information retrieval," *Foundations and Trends® in Information Retrieval*, vol. 3, no. 3, pp. 225–331, 2009.

[30] A. Bhargava, R. Ganti, and R. Nowak, "Bandit approaches to preference learning problems with multiple populations," *stat*, vol. 1050, p. 14, 2016.

[31] K. Ju, K. Jamieson, R. Nowak, and X. Zhu, "Top arm identification in multi-armed bandits with batch arm pulls." in *AISTATS*, 2016.

[32] K. Jamieson and R. Nowak, "Best-arm identification algorithms for multi-armed bandits in the fixed confidence setting," in *Annual Conference on Information Sciences and Systems (CISS)*, 2014.

[33] S. Chester, A. Thomo, S. Venkatesh, and S. Whitesides, "Computing k-regret minimizing sets," in *Proceedings of the VLDB Endowment*, 2014.

[34] W. Cao, J. Li, H. Wang, K. Wang, R. Wang, R. Wong, and W. Zhan, "k-regret minimizing set: Efficient algorithms and hardness," in *LIPIcs-Leibniz International Proceedings in Informatics*, vol. 68. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[35] P. K. Agarwal, N. Kumar, S. Sintos, and S. Suri, "Efficient algorithms for k-regret minimizing sets," in *arXiv:1702.01446v2, International Symposium on Experimental Algorithms (SEA)*, June 2017.

[36] A. Asudeh, A. Nazi, N. Zhang, and G. Das, "Efficient computation of regret-ratio minimizing set: A compact maxima representative," in *Proceedings of the ACM International Conference on Management of Data*, 2017.

[37] ——, "Efficient computation of regret-ratio minimizing set: A compact maxima representative," in *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM, 2017, pp. 821–834.

[38] M. Xie, R. C.-W. Wong, J. Li, C. Long, and A. Lall, "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, 2018.

[39] N. Kumar and S. Sintos, "Faster approximation algorithm for the k-regret minimizing set and related problems," in *2018 Proceedings of the Twentieth Workshop on Algorithm Engineering and Experiments (ALENEX)*. SIAM, 2018, pp. 62–74.

[40] W. Wang, R. C.-W. Wong, and M. Xie, "Interactive search for one of the top-k," in *Proceedings of the 2021 International Conference on Management of Data*, ser. SIGMOD/PODS '21, 2021, p. 1920–1932.

[41] X. Qin, C. Chai, and Y. e. a. Luo, "Interactively discovering and ranking desired tuples by data exploration," *VLDB Journal*, 2022.

[42] Y. Chang, L. Bergman, V. Castelli, C. Li, M. Lo, and J. Smith, "The onion technique: Indexing for linear optimization queries," in *Proceedings of ACM SIGMOD International Conference on Management of Data*, 2000.

[43] T. Kessler Faulkner, W. Brackenbury, and A. Lall, "K-regret queries with nonlinear utilities," in *Proceedings of the VLDB Endowment*, 2015.