User-Driven Refinement of Imprecise Queries

Bahar Qarabaqi
College of Computer and Information Science
Northeastern University, Boston, USA
Email: bahar@ccs.neu.edu

Mirek Riedewald
College of Computer and Information Science
Northeastern University, Boston, USA
Email: mirek@ccs.neu.edu

***Abstract*—We propose techniques for exploratory search in large databases. The goal is to provide new functionality that aids users in homing in on the right query conditions to find what they are looking for. Query refinement proceeds interactively by repeatedly consulting the user to manage query conditions. This process is characterized by three key challenges: (1) dealing with incomplete and imprecise user input, (2) keeping user effort low, and (3) guaranteeing interactive system response time. We address the first two challenges with a probability-based framework that guides the user to the most important query conditions. To recover from input errors, we introduce the notion of sensitivity and propose efficient algorithms for identifying the most sensitive user input, i.e., those inputs that had the greatest influence on the query results. For the third challenge, we develop techniques that can deliver estimates of the required probabilities within a given hard realtime limit and are able to adapt automatically as the interactive query refinement proceeds.**

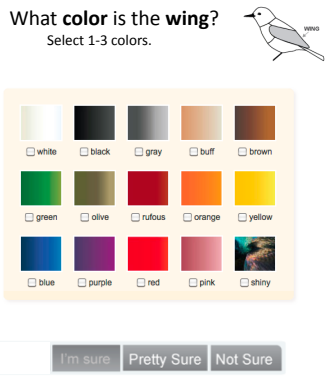

Fig. 1. Possible interface for specifying multiple wing color values

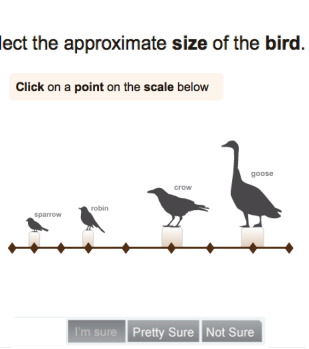

Fig. 2. Possible interface for specifying the bird size

I. Introduction

With Big Data comes a big responsibility to make databases more accessible to a broad spectrum of users, ranging from SQL experts to non-technical users querying them through form-based interfaces. While an expert with full SQL access will usually find her way around a database, less technical users working through limited interfaces often have difficulty finding what they are looking for. Hence, for databases to be relevant for these users, they have to provide functionality for *exploratory search*.

Consider a large crowd-sourced database of bird observations. Each record has attributes describing the properties of the bird (e.g., species, size) and the observation event (e.g., location, habitat features). An organization such as the Cornell Lab of Ornithology would like to leverage this database to help casual bird watchers identify the species of a bird they observed. To reach a broad audience, the Lab decides to make the database accessible through a simple interface where the user enters properties of the observed bird in a Web form (e.g., as shown in Figures 1 and 2). Now consider user Amy who wants to identify the species of a bird she observed. Amy will be sure about some attributes, e.g., her GPS device reliably recorded location. She will not know others, e.g., the bird's belly color. Then there will be many more attributes whose values she recalls with varying degrees of *uncertainty*. For instance, Amy thinks the bird's wing was mostly red, but maybe it was more of a brownish tone.

Assume Amy initially only fills in values she is certain about. Unfortunately, these are not selective enough and she receives an overwhelming list of possible matches, where none of the top-10 species displayed looks like the one she observed. Trying to narrow the search further, she has to decide which of the less certain attribute values to enter. Without help from the database, this is a tedious process of trial-and-error. If she selects only "red" for wing color, but the species is actually rufous, the correct answer might be eliminated. If she selects too many possible colors, only few of the irrelevant records might be eliminated. Or maybe the wing color selection does not even matter much: since the database is crowd-sourced, a large number of other users might have specified "red" instead of the correct "rufous" anyway.

Similar issues arise in many other important applications, including product search and online medical advice. For product search, suppose a user wants to leverage the wisdom of the crowd for deciding about a camera purchase. Crowd-sourced camera data will contain a mix of "objective" properties (e.g., megapixels and price) and subjective user evaluations (e.g., if the camera is good for sport photography). Now the user wants to find the right camera, relying on a mix of certain and uncertain values for the various properties. In the medical domain, a database of diseases, their symptoms, potential causes (e.g., family history and lifestyle choices), and possible remedies would similarly be consulted by people not feeling well. As sites like WebMD's symptom checker (http://symptoms.webmd.com) show, there is great interest in this kind of application.

In general, our work targets the challenge of *helping a database user fine-tune selection conditions* for exploratory analysis. In these applications the query is not pre-determined, but the goal is to find a query that returns a result that is "good enough" for some task.

We make several contributions aimed at improving support

for exploratory search in databases. First, exploratory search usually involves uncertainty; not only in the data (which motivated probabilistic database research [1]) but also in the query. To deal with uncertainty in user-provided query conditions, we propose a **probability-based framework** that makes this uncertainty explicit. Notice that for imprecise queries, the result is probabilistic even if the data is precise. Hence, **ranking of result records based on their probability** is inherent in exploratory search.

Our second contribution helps the user judge the potential risk of specifying a condition she is not very confident about, e.g., the wing color in the bird example. Specifying such a condition might provide useful information for improving the query result. However, if getting it just slightly wrong might significantly change the result, then it might be safer to not enter it at all. To provide this kind of risk-estimation functionality, we **introduce the novel notion of sensitivity of a condition** and **prove structural properties that allow its efficient computation**.

While sensitivity allows the user to quantify the *risk* of a query constraint, our third contribution aims at quantifying the *benefit* of a constraint by **identifying the best new conditions to be added** in order to improve result quality.

As a user-driven process, query refinement should be *interactive*. Unfortunately, computation time tends to be high when dealing with large databases and imprecise queries and data. We therefore propose **fast approximate estimation techniques that guarantee to deliver results within a given response time threshold**.

II. Data Model and Framework

We introduce the data model and propose a probability-based framework for exploratory search in databases.

A. Data Model

For simplicity and without loss of generality, we assume that the given database contains "certain" data, i.e., is not a probabilistic database. (We will point out non-trivial extensions required for probabilistic data where appropriate.) The database manages some entities of interest, each of them described by a set of attributes $\{X_1, X_2, \ldots, X_m\}$ and an entity-identifier Y. Again, for simplicity and without loss of generality, assume that data about these entities is stored in a relational view R with schema $\{X_1, X_2, \ldots, X_m, Y\}$. ($R$ could be a base table or the result of a complex SQL query.) Notice that even though Y uniquely identifies an entity, it does not need to be a key of R.

In the bird example, the entities of interest are the different bird species. Attribute Y is the species name. The X_i describe various properties of a bird and observation event, e.g., hasWingColorRed and obsLongitude. Data set R contains a record for each individual bird observed. Hence, there can be many records for the same species. Since not all individuals of a species look alike or are seen at the same location, the values of the X_i can differ even for records with the same entity ID Y. (For this reason Y is not necessarily a key of R.)

B. Probabilistic Query Framework

The user wants to find entities $y \in Y$ of interest (we slightly abuse notation and use Y to denote both the name of the attribute and its domain) and expresses her preferences by specifying constraints on some of the attributes X_i. This corresponds to query *SELECT Y FROM R WHERE condition($X_1, X_2, \ldots, X_m$)* in a relational database. To incorporate uncertainty, our framework supports constraints that are **probability distributions** in the condition. More precisely, for some attribute X_i, let A_i be the set of all possible probability distributions over the values in the domain of X_i. When interacting with the system, the user specifies some probability distribution $a_i \in A_i$. In the bird example, for binary attribute hasWingColorRed, the user might specify distribution $(0.8, 0.2)$ over domain ("yes", "no"), indicating that she is 80% confident that the observed bird's wing contained red.

Given such probability distributions for some of the X_i, we support exploratory search by doing the following:

1) Estimate for each entity $y \in Y$ the probability that it is what the user is looking for and present a list of the top-ranked entities based on these probabilities.
2) Estimate the sensitivity of each X_i for which the user specified a distribution a_i and present these X_i and their sensitivity scores in decreasing order.
3) Estimate how much each of the remaining unspecified X_j would help improve the query result quality if the user provided a distribution a_j for it. Present these attributes X_j and their quality-improvement score in decreasing order to the user.

For illustration, consider the bird-species identification example in Figure 3. It shows the information presented to the user after she specified conditions for attributes X_2 (size) and X_4 (main color). In the center is the list of the top-ranked species based on the user input so far. To the left is the ranked list of unspecified attributes, sorted by how much knowing their value would help improving the current result. On the right is the ranked list of specified attributes, sorted by their sensitivity score, i.e., how much changing their current condition would affect the current species ranking.

Assume the user decides to proceed by adding more query conditions in order to improve the result. Looking at the left table in Figure 3, she skips ShapeGroup and BillLength, because she did not observe these properties. Encountering WingColor next, she decides to enter the corresponding probability distribution a_{11}, possibly through an interface as shown in Figure 1. Alternatively, the high sensitivity score of X_2 might motivate the user to revisit her initial input, e.g., using an interface as shown in Figure 2, because she is not too confident about the observed bird's size.

Before discussing techniques for each of these functionalities, we formally introduce the probability of an entity to be what the user is looking for. Assuming the user specified distributions $a_1, \ldots, a_k$ for attributes $X_1, \ldots, X_k$, the desired probability for $y \in Y$ is

$$\Pr(Y = y \,|\, A_1 = a_1, \ldots, A_k = a_k, D),$$

written more compactly as $\Pr(Y \,|\, A_1, \ldots, A_k, D)$. Notice that this probability also depends on the database content D. In

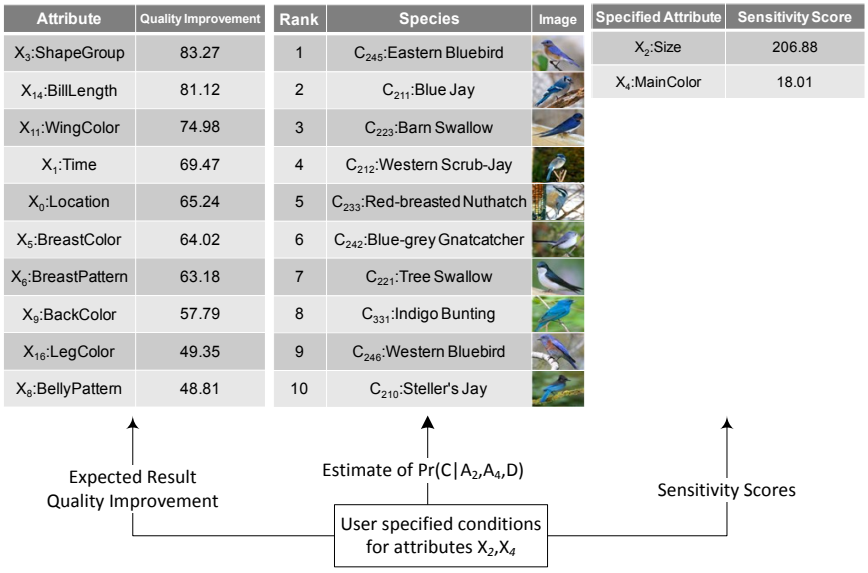

Fig. 3. Feedback to the user after she specified distributions for attributes X_2 and X_4

addition to specifying conditions for the X_i, we can also model explicit rejection of entities. E.g., after seeing a list of the most likely species, the user might choose to eliminate some. Let $\bar{y}_j$ denote that entity y_j was rejected and assume the user rejected entities $y_1, \ldots, y_l$. The entity probability then becomes

$$\Pr(Y \mid A_1, \ldots, A_k, \bar{y}_1, \ldots, \bar{y}_l, D).$$

We discuss in Section VI how to estimate this probability subject to hard realtime constraints on response time.

Whenever the user is looking for a single result, e.g., the correct bird species, the probability of Y as defined here directly captures the notion of result quality because random variable Y can take on the value of any *individual* entity. Furthermore, this probability is also meaningful when the user is looking for multiple result entities. Intuitively, the probability of entity y corresponds to the fraction of entity-y tuples (those whose Y-value equals y) among the tuples in R that satisfy the user-specified constraints. Stated differently, the entities with the highest probabilities are those that would occur most frequently in the result of *SELECT Y FROM R WHERE condition*$(X_1, X_2, \ldots, X_m)$ in a sufficiently large data set R. Furthermore, when looking for multiple results, the user would not only reject entities, but could also *accept* some and then continue the search process. The probabilities can be adjusted accordingly by essentially removing the accepted entities from consideration (as if they never existed in R).

III. Result Ranking

Given an imprecise query, we want to present a ranked result as shown in the central table in Figure 3. Due to the probabilistic nature of the query conditions, each entity $y \in Y$ is a query result with probability $\Pr(Y \mid A_1, \ldots, A_k, \bar{y}_1, \ldots, \bar{y}_l, D)$ as discussed above. Hence, we can present the most likely entities to the user first, i.e., **rank by entity probability**.

While this is a very natural ranking, we also propose another ranking that takes *user effort* into account. It is motivated by the fact that the user has to invest some time to look at the presented query result in order to decide which entities are of interest. In the example in Figure 3, the user would go over the list of birds in the center from top to bottom, determining if any looks similar to the bird she observed. Let ϕ_j denote the user effort required for deciding about the relevance of entity y_j.[1] Now consider two entities y_1 and y_2 with probabilities 0.51 and 0.49 and effort $\phi_1 = 10$ and $\phi_2 = 1$, respectively. If we rank y_1 before y_2, the user would have to invest an expected effort of $0.51 \cdot 10 + 0.49 \cdot (10 + 1) = 10.49$ when examining the ranked list top-down until the correct answer is found (assuming either y_1 or y_2 is correct, but not both). However, if y_2 was ranked first, expected user effort would drop to $0.49 \cdot 1 + 0.51 \cdot (1 + 10) = 6.1$.

Ranking by effort-adjusted entity probability. This example motivates an alternative ranking system based on effort-adjusted probabilities. The **effort-adjusted probability** of entity $y_i \in Y$ is defined as $\Pr(Y = y_i \mid A_1, \ldots, A_k, \bar{y}_1, \ldots, \bar{y}_l, D)/\phi_i$. It has the following property:

Lemma 1: Assume the user is looking for a single entity of interest by exploring the ranked list of entities one-by-one from top to bottom, until this entity is found. Expected user effort then is minimized if the entities are ranked in decreasing order of their effort-adjusted probability.

Proof sketch: Without loss of generality, assume the entities are ranked in order $y_1, y_2, \ldots, y_n$. If y_c is the correct result the user is looking for, she would have to go through entities

[1]Effort can vary, e.g., some bird species is easily recognizable from a picture, another requires reading a description. Effort can be measured based on the user's response time when interacting with our system.

y_1 to y_{c-1}, which are all ranked higher, until finding y_c. The corresponding effort is $\sum_{j=1}^{c} \phi_j$. Hence, we obtain the expected user effort as

$$\sum_{i=1}^{n} \Pr(Y = y_i \mid A_1, \ldots, A_k, \bar{y}_1, \ldots, \bar{y}_l, D) \cdot \sum_{j=1}^{i} \phi_j. \quad (1)$$

Now consider a pair (y_k, y_{k+1}) that is not correctly ranked by effort-adjusted probability, i.e., $p_k/\phi_k < p_{k+1}/\phi_{k+1}$. (To avoid clutter, we use p_i as a shorthand for $\Pr(Y = y_i \mid A_1, \ldots, A_k, \bar{y}_1, \ldots, \bar{y}_l, D)$.) Examining all affected terms in Eq. 1, we can see that swapping the positions of the two entities in the ranked list would change expected effort by $p_{k+1}\phi_k - p_k\phi_{k+1}$. This difference is positive because of the initial assumption $p_k/\phi_k < p_{k+1}/\phi_{k+1}$. Stated differently, swapping any two adjacent entities that are not correctly ordered by effort-adjusted probability will decrease expected user effort. This in turn implies that ranking in decreasing order of effort-adjusted probability minimizes expected user effort. (This step is the same as showing that Bubble Sort is a correct sort procedure.)

IV. Sensitivity Analysis

Our goal is to help the user judge the potential *risk* and *benefit* of conditions she considers for her exploratory search query. This section focuses on the risk aspects, while Section V deals with the benefit.

We propose the notion of *sensitivity* of a user-provided condition for an attribute X_i. Intuitively, user response a_i for attribute X_i has high sensitivity, if the current result ranking would change significantly if the user were to change her current answer "a little". To illustrate the use of sensitivity analysis, we return to the bird example. Recall that user Amy is not certain about the observed bird's size. She knows for sure that it was smaller than a crow, but cannot decide between the sizes in the range from sparrow to American Robin (which is significantly larger than its European cousin). Assume she entered probability distribution $(0, 0.3, 0.4, 0.3, 0, 0, 0, 0, 0)$ (see Figure 2 for bird size choices). Assume further that our system told her that any alternative input of the form $(0, q_1, q_2, q_3, 0, 0, 0, 0, 0)$, where $0 \leq q_1, q_2, q_3$ and $q_1 + q_2 + q_3 = 1$, would result in only minor changes of the species ranking. Then she can be confident about having entered the distribution information: It would eliminate very small and very large species from consideration without risking major re-shuffling of the remaining ones, even if her probability numbers are a bit "off".

On the other hand, if even minor changes, e.g., from $(0, 0.3, 0.4, 0.3, 0, 0, 0, 0, 0)$ to $(0, 0.33, 0.34, 0.33, 0, 0, 0, 0, 0)$ could dramatically change the species ranking, then adding this information is very risky. It would be better to initially avoid specifying bird size. However, as will become clearer after the formal definition of sensitivity is introduced, *an attribute with high sensitivity early on might have much lower sensitivity after more information about other attributes is entered*. Hence initially "risky" input can become useful at a later stage.

Sensitivity complements other approaches that try to detect problematic user input by determining if it differs significantly from what was expected based on historic searches and user responses so far. Those approaches rely on outlier detection techniques [2], [3] or include errors as variables in a prediction model [4].

A. Definition and Naive Algorithm

The sensitivity of user-provided condition a_i for attribute X_i depends on how much a change of a_i would affect the current result ranking. Let L_p and L_q be two entity rankings. For each entity $y \in Y$, let $\rho_p(y)$ and $\rho_q(y)$ denote y's rank in the first and second ranking, respectively. To measure how different the rank of each entity is between the two rankings, we use the popular **Minkowski distance**. For some constant $d > 0$ it is defined as

$$\mathrm{dst}(L_p, L_q) = \left(\sum_{y \in Y} |\rho_p(y) - \rho_q(y)|^d \right)^{1/d}.$$

Definition 1: Let L_p be the current entity ranking and X be an attribute for which the user has specified a condition. Furthermore, let $\mathcal{A}$ denote the set of all alternative conditions the user considers for attribute X. The *sensitivity* of the current ranking L_p to attribute X for a set of possible conditions $\mathcal{A}$ is defined as the maximum difference $\mathrm{dst}(L_p, L_q)$ between L_p and any other ranking L_q that could be obtained if the user were to change the current condition for attribute X to any other value in $\mathcal{A}$.

For brevity, we will refer to the "sensitivity of the current ranking L_p to attribute X for a set of possible conditions $\mathcal{A}$" simply as the "*sensitivity of attribute* X". By default the set $\mathcal{A}$ of possible alternative conditions includes any input the user might have given for X. If the user can exclude some answers with certainty, e.g., Amy knows that the bird was definitely smaller than a crow, then $\mathcal{A}$ could be limited accordingly. All results in this section apply to *any* such set $\mathcal{A}$.

Naive algorithm for computing sensitivity: Consider one-by-one each attribute for which the user specified a condition. When computing the sensitivity of attribute X_i, try every single condition in $\mathcal{A}_i$: compute the entity ranking based on this modified condition to find the ranking with the maximum distance from the current ranking L_p.

We will next identify an important structural property that allows us to dramatically reduce the space of conditions in $\mathcal{A}_i$ to be considered.

B. Structural Property of Ranking Distance

Consider an attribute X with three possible values x_1, x_2, and x_3. The user would specify some probability distribution over these three values. For instance, it might be $a = (0.4, 0.2, 0.4)$, indicating that the user believes X had value x_1 with probability 0.4 and so on. To compute the sensitivity of attribute X, we want to determine the greatest ranking difference that could be achieved by changing answer a to any other probability distribution considered, e.g., $(0.0, 0.4, 0.6)$. Since the third probability is determined by the other two (all have to add up to 1.0), we only need to consider a probability vector (p_1, p_2).

In the following, we prove that the maximum ranking difference can only be achieved by a distribution "on the

edge" of the space of possible alternative inputs considered. In particular, if all possible probability distributions are considered for X, then the maximum ranking difference is achieved for distribution (p_1, p_2) where either $p_1 = 0$ or $p_2 = 0$ or $p_1 + p_2 = 1$ (follows from $p_3 = 0$).

Consider three distributions for attribute X: probability vectors $p = (p_1, p_2)$, $q = (q_1, q_2)$, and $r = (r_1, r_2)$. Let the three vectors be located on a line, such that $q = p + \alpha \cdot (r - p)$ for some $0 < \alpha < 1$. Intuitively, q lies between p and r. We now show that the corresponding probabilities for each entity will have the same property, i.e., that the probability of entity y given distribution q will be between the probabilities obtained for distributions p and r. Hence we need to examine probabilities

$$\begin{aligned} P &= \Pr(Y \mid A = p, A_1, \ldots, A_k, \bar{y}_1, \ldots, \bar{y}_l, D), \\ Q &= \Pr(Y \mid A = q, A_1, \ldots, A_k, \bar{y}_1, \ldots, \bar{y}_l, D), \\ R &= \Pr(Y \mid A = r, A_1, \ldots, A_k, \bar{y}_1, \ldots, \bar{y}_l, D). \end{aligned}$$

Recall that each A_i is a distribution over the values of the corresponding attribute X_i. Hence the above probabilities are actually expectations over these combinations of X-values. Formally, P (and similarly Q and R) is defined as

$$\mathrm{E}_{X, X_1, \ldots, X_k}[\Pr(Y \mid X, X_1, \ldots, X_k, \bar{y}_1, \ldots, \bar{y}_l, D)].$$

Since expectations can be decomposed, we can equivalently write

$$\mathrm{E}_X\big[\mathrm{E}_{X_1, \ldots, X_k}[\Pr(Y \mid X, X_1, \ldots, X_k, \bar{y}_1, \ldots, \bar{y}_l, D)]\big].$$

Based on the definition of the expectation, we then obtain

$$P = \sum_{x \in X} \Pr(x) \cdot g(x),$$

where $g(x) = \mathrm{E}_{X_1, \ldots, X_k}[\Pr(Y \mid X = x, X_1, \ldots, X_k, \bar{y}_1, \ldots, \bar{y}_l, D)]$; similar for Q and R. For our 3-valued example attribute X we therefore have

$$P = p_1 \cdot g(x_1) + p_2 \cdot g(x_2) + (1 - p_1 - p_2) \cdot g(x_3), \quad (2)$$

$$Q = q_1 \cdot g(x_1) + q_2 \cdot g(x_2) + (1 - q_1 - q_2) \cdot g(x_3), \quad (3)$$

$$R = r_1 \cdot g(x_1) + r_2 \cdot g(x_2) + (1 - r_1 - r_2) \cdot g(x_3). \quad (4)$$

Since $q = p + \alpha \cdot (r - p)$, we can derive from Equation 3

$$\begin{aligned} Q =& (p_1 + \alpha(r_1 - p_1))g(x_1) + (p_2 + \alpha(r_2 - p_2))g(x_2) \\ &+ (1 - (p_1 + \alpha(r_1 - p_1)) - (p_2 + \alpha(r_2 - p_2)))g(x_3) \\ =& (p_1 g(x_1) + p_2 g(x_2) + (1 - p_1 - p_2)g(x_3)) \\ &+ \alpha((r_1 g(x_1) + r_2 g(x_2) + (1 - r_1 - r_2)g(x_3)) \\ &- (p_1 g(x_1) + p_2 g(x_2) + (1 - p_1 - p_2)g(x_3))). \end{aligned}$$

Together with Equations 2 and 4, we then obtain the desired result that $Q = P + \alpha \cdot (R - P)$.

The above analysis generalizes beyond 3-valued attributes to any discrete attribute type. Intuitively, we have shown the following powerful result: Assume entity y has a probability P of being the result based on distribution p for attribute X. Assume we also know that this probability will be R if the user changed her response from p to a different distribution r. Then each alternative response q that is a linear combination of p and r will result in a probability Q that is proportionally between P and R. Figure 4 illustrates this property for an example of five entities y_1 to y_5. For distribution p of attribute X, y_1 has the highest probability and y_5 the lowest. For a different distribution r those probabilities are almost reversed. For any distribution q between p and r, each entity's probability is proportionally between the corresponding probabilities for p and r. Using this result, we can now prove the following theorem:

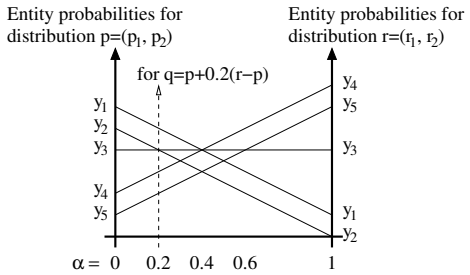

Fig. 4. Entity probabilities for collinear answers p, q, r

Theorem 1: Let p be the current user input for attribute X. Let q and r be two other possible responses from $\mathcal{A}$, such that $q = p + \alpha \cdot (r - p)$ for some $0 < \alpha < 1$. And let L_p, L_q, and L_r, respectively, denote the rankings obtained for these responses (while keeping all other responses constant). Then, if categories are ranked based on their probability or effort-adjusted probability (Section III), it holds that $\mathrm{dst}(L_p, L_q) \leq \mathrm{dst}(L_p, L_r)$.

Proof: Consider Figure 4 for illustration. As α is increased from 0 to 1, we obtain a series of different rankings between the entities due to their changing probabilities. In the example, for $0 < \alpha < 0.2$ the ranking of probabilities for q is identical to the ranking for p. At $\alpha = 0.2$, "adjacent" (in their ranking) entities y_2 and y_3 swap places. Then at $\alpha = 0.3$, adjacent y_2 and y_4 swap places, and so on. Notice that sometimes more than two entities might swap places "at the same time" when multiple lines intersect, e.g., for $\alpha = 0.4$ in the example. The result of this many-entity swap can always be equivalently expressed as a series of the corresponding binary swaps between adjacent entities. Consider again Figure 4. For $\alpha = 0.39$, the entity ranking is $(y_1, y_3, y_4, y_2, y_5)$; for $\alpha = 0.41$ it is $(y_4, y_3, y_1, y_5, y_2)$. The following sequence of binary swaps between adjacent entities transforms one ranking to the other: $y_2 \leftrightarrow y_5$, $y_3 \leftrightarrow y_4$, $y_1 \leftrightarrow y_4$, $y_1 \leftrightarrow y_3$.

Now consider the sequence of rankings $L_0 (= L_p), L_1, L_2, \ldots, L_{k-1}, L_k (= L_r)$ defined by gradually increasing α from 0 to 1. As we showed above, each ranking pair (L_i, L_{i+1}), $0 \leq i < k$, is identical except that two adjacent categories in L_i are swapped in L_{i+1}. We now show that this property implies that the distance from L_p to L_i cannot be greater than the distance from L_p to L_{i+1}. Let y and y' be the adjacent entities that swapped ranks between L_i and L_{i+1}. More precisely, both rankings are identical, except that $\rho_i(y) = \rho_i(y') - 1$, $\rho_{i+1}(y) = \rho_i(y')$, and $\rho_{i+1}(y') = \rho_i(y)$. Now consider all possible cases for the ranking of these entities in L_p:

Case 1: $\rho_p(y) > \rho_i(y)$ and $\rho_p(y') < \rho_i(y')$. This case is impossible, because it implies for the corresponding entity probabilities that $P' > P$, $Q' < Q$, and $R' > R$. Since we

showed above that $Q = P + \alpha(R - P) = (1-\alpha)P + \alpha R$ and similarly $Q' = (1-\alpha)P' + \alpha R'$, we obtain a contradiction.

Case 2: $\rho_p(y) \leq \rho_i(y)$ and $\rho_p(y') \geq \rho_i(y')$. Since all entities are at the same ranks in L_i and L_{i+1}, except for y and y', the only difference between $\mathrm{dst}(L_p, L_i)$ and $\mathrm{dst}(L_p, L_{i+1})$ is due to y and y'. More precisely, for any entity $\hat{y}$ different from y and y', the rank difference of $\hat{y}$ between L_p and L_i is the same as between L_p and L_{i+1}. Now consider the terms where the rank difference is different: $|\rho_p(y) - \rho_i(y)|^d + |\rho_p(y') - \rho_i(y')|^d$ for $\mathrm{dst}(L_p, L_i)$ versus $|\rho_p(y) - \rho_{i+1}(y)|^d + |\rho_p(y') - \rho_{i+1}(y')|^d$ for $\mathrm{dst}(L_p, L_{i+1})$. Because of the case condition and the fact that y and y' have swapped ranks between L_i and L_{i+1}, it follows that $|\rho_p(y) - \rho_{i+1}(y)|^d + |\rho_p(y') - \rho_{i+1}(y')|^d$ is equal to $(|\rho_p(y) - \rho_i(y)| + 1)^d + (|\rho_p(y') - \rho_i(y')| + 1)^d$, which is greater than $|\rho_p(y) - \rho_i(y)|^d + |\rho_p(y') - \rho_i(y')|^d$. Hence $\mathrm{dst}(L_p, L_i) < \mathrm{dst}(L_p, L_{i+1})$.

Case 3: $\rho_p(y) \leq \rho_i(y)$ and $\rho_p(y') < \rho_i(y')$. Observe that $|\rho_p(y) - \rho_i(y)| \geq |\rho_p(y') - \rho_q(y')|$. This is due to the facts that y' cannot precede y in L_p (proof like for case 1) and that both are ranked higher in L_p than in L_i. Therefore, similar to the analysis in case 2, we obtain the following for the rank-difference terms that are different between $\mathrm{dst}(L_p, L_i)$ and $\mathrm{dst}(L_p, L_{i+1})$:

$$
\begin{aligned}
&|\rho_p(y) - \rho_{i+1}(y)|^d + |\rho_p(y') - \rho_{i+1}(y')|^d \\
&= (\rho_{i+1}(y) - \rho_p(y))^d + (\rho_{i+1}(y') - \rho_p(y'))^d \\
&= (\rho_i(y) + 1 - \rho_p(y))^d + (\rho_i(y') - 1 - \rho_p(y'))^d \\
&= (\rho_i(y) - \rho_p(y))^d + d(\rho_i(y) - \rho_p(y))^{d-1} + \cdots + 1 \\
&+ (\rho_i(y') - \rho_p(y'))^d - d(\rho_i(y') - \rho_p(y'))^{d-1} + \cdots + (-1)^d \\
&\geq (\rho_i(y) - \rho_p(y))^d + (\rho_i(y') - \rho_p(y'))^d \\
&= |\rho_p(y) - \rho_i(y)|^d + |\rho_p(y') - \rho_i(y')|^d \\
&\Rightarrow \mathrm{dst}(L_p, L_i) \leq \mathrm{dst}(L_p, L_{i+1})
\end{aligned}
$$

Case 4: $\rho_p(y) > \rho_i(y)$ and $\rho_p(y') \geq \rho_i(y')$. The analysis is symmetric to case 3.

These cases cover all possible rankings for L_p. We can now inductively apply this argument, showing that $\mathrm{dst}(L_p, L_1) \leq \mathrm{dst}(L_p, L_2) \leq \cdots \leq \mathrm{dst}(L_p, L_k) = \mathrm{dst}(L_p, L_r)$. This completes the proof. ■

We can similarly prove that Theorem 1 also holds when the ranking distance is measured by the **number of inversions** between two rankings, instead of using a Minkowski distance.

While Theorem 1 guarantees that the maximum ranking difference can only be achieved by a distribution "on the edge" of the space of possible inputs in $\mathcal{A}$, one might wonder if an even stronger result could be shown. In particular, would the optimum have to be in one of the "vertices" of the space? Unfortunately, this does not hold in general, because the optimization problem is not convex. First, the set of alternative probability distributions, $\mathcal{A}$, might not be convex. Second, as Figure 6 shows, the objective function is monotonic, but not necessarily convex.

C. Efficient Sensitivity Algorithm

Recall that the sensitivity of attribute X is determined by the *maximal ranking difference* over all alternative probability distributions for X in $\mathcal{A}$. Based on Theorem 1, when searching for the distribution that maximizes the ranking distance to the current ranking, we only need to consider the "boundaries" of the data space. This often eliminates the vast majority of possible distributions in $\mathcal{A}$, dramatically reducing the cost of sensitivity analysis compared to the naive algorithm.

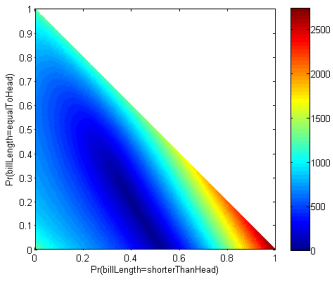

Fig. 5. Ranking distance for 3-valued attribute billLength and current distribution $(0.4, 0.2)$ when selecting other distributions (q_1, q_2) for the probability of the bill length being shorter than the head and equal to the head, respectively

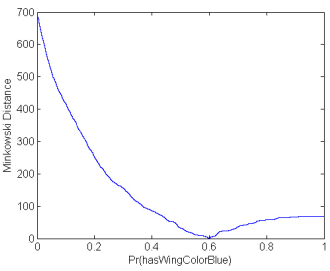

Fig. 6. Ranking distance for binary attribute hasWingColorBlue and current distribution (0.6) when selecting other distributions q_1 for the probability of the bird's wing containing blue

Consider again the 3-valued example attribute X with original user-response $(0.4, 0.2, 0.4)$. There is no need to explore distribution $(0.65, 0.15, 0.2)$, because we know from the theorem that since $(0.65, 0.15, 0.2) = (0.4, 0.2, 0.4) + 0.5 \cdot ((0.9, 0.1, 0.0) - (0.4, 0.2, 0.4))$, the ranking distance for input $(0.9, 0.1, 0.0)$ will be greater than or equal to the one obtained for $(0.65, 0.15, 0.2)$. Figure 5 illustrates this structural property for the 3-valued attribute billLength in our bird observation data set (discussed in Section VII). Starting with the original user response $p = (0.4, 0.2)$, we sampled 1 million alternative responses randomly and computed the distance in ranking to L_p. Consistent with Theorem 1, distances are indeed increasing along each line emanating from point $(0.4, 0.2)$. (There is some "color noise" in the graph, which are artifacts of the drawing process when interpolating between sample points.) In summary, to compute the sensitivity of attribute billLength, the naive algorithm would have to consider the entire colored triangle in Figure 5. Thanks to Theorem 1, we only need to consider the three edges of this triangle.

The implications of Theorem 1 are particularly powerful for binary attributes. Since the distribution of a binary attribute is represented by a single probability value, all possible inputs are collinear by definition. This means that to find the input that results in the greatest ranking difference, we only need to check the rankings for the lower and upper extreme of the range of possible probability values considered. Figure 6 shows this for binary attribute hasWingColorBlue in our bird observation data set. Starting with the original user response $p = 0.6$, we sampled 1 million alternative responses randomly and computed the distance in ranking to L_p.

D. Further Cost Reduction and Continuous Domains

Notice that our algorithm can exploit cases where $\mathcal{A}$, the set of possible probability distributions the user might specify for an attribute X, is explicitly constrained by the user. E.g., if Amy is sure that the bird was smaller than a crow, then

sensitivity analysis should only consider distributions of the form $(p_1, p_2, p_3, p_4, p_5, 0, 0, 0, 0)$ over the nine possible bird-size values. Theorem 1 obviously still applies and further reduces the space of combinations to be considered.

Our sensitivity algorithm as discussed so far would not work well for scenarios with an infinite (or very large) number of possible probability distributions in $\mathcal{A}$ and for attributes with infinite (or very large) domains.

Large $\mathcal{A}$. For the billLength example in Figure 5, we see that there can be an infinite number of probability distributions over the three domain values. Even after applying Theorem 1, the three edges of the colored triangle still contain an infinite number of possible combinations. Hence, instead of exhaustively exploring all of them, our algorithm in practice will draw a random sample and estimate the maximum ranking distance (and hence the sensitivity) from this sample. Theorem 1 is still crucially important. Instead of sampling from the entire colored triangle, we only have to sample from the edges.

Large attribute domain. For a continuous attribute with infinite domain, we cannot accurately represent the probability distribution over its values as a vector of probabilities. However, we can approximate the full distribution by discretizing it, e.g., using a histogram. For discrete attributes with very large domains, we use a coarser approximate representation.

Both sampling and approximation of distributions for attributes with large domains are perfectly acceptable in practice. Recall that in exploratory search the *user provides the distributions*. From the user's point of view, it makes virtually no sense to try and distinguish between distributions like $(0.8, 0.2)$ and $(0.81, 0.19)$. They both express that the user was highly confident about the attribute's value. An analogous argument applies to attributes with infinite and very large domains, where provided distribution input will always be approximate in nature anyway.

V. Recommending Additional Conditions

Section IV introduced sensitivity to quantify the risk of specifying imprecise conditions. We now turn to the somewhat "opposite" problem of recommending currently unspecified attributes that have the greatest potential for improving result quality (left table in Figure 3). Intuitively, we want to estimate how beneficial each currently unspecified attribute is for improving the query result quality. This is inherently difficult, because we do not know which entities the user wants to find and what conditions she would specify for these attributes.

Since the user's future input a for attribute X is not known, we model it as a random variable A. We can estimate the probability that A takes on a specific value a from all the information gathered so far, i.e.,

$$\Pr(A = a \,|\, A_1, \ldots, A_k, \bar{y}_1, \ldots, \bar{y}_l, D).$$

We discuss in Section VI how to estimate this probability in practice. For each value a, we can then compute

$$\Pr(Y \,|\, A = a, A_1, \ldots, A_k, \bar{y}_1, \ldots, \bar{y}_l, D)$$

and use it to create an entity ranking as discussed in Section III.

Assume we have a function F that returns by how much the entity ranking improves *with* the additional information $A = a$, compared to the current ranking *without* knowing A. Using this function F, we can then compute the **expected improvement** in entity ranking over random variable A. Ideally, F should measure how much the ranks improved for the entities the user is interested in. Since we do not know which entities the user is interested in, we have to find alternative measures that measure how well "winning" entities are separated from "losing" ones. More precisely, a ranking where the probability of being in the result is high only for a few entities and near-zero for all others, is desirable: likely answers and unlikely answers are well-separated and the few top-ranked entities shown to the user have a comparably high aggregate probability mass.

Entropy-based. Entropy directly captures this intuition. Using p_y as shorthand for $\Pr(Y = y \,|\, A_1, \ldots, A_k, \bar{y}_1, \ldots, \bar{y}_l, D)$, the entropy of a set of entities is defined as $-\sum_{y \in Y} p_y \cdot \log_2(p_y)$. In general, cases with few high-probability categories and many low-probability ones have low entropy, while those where many categories have similar probabilities have high entropy. For this reason entropy is widely used for selection of split attributes in decision trees, e.g., through information gain or gain ratio [5]. The goal of the tree is to separate different classes through the splits, which coincides with our search goal of separating likely result entities from unlikely ones. Hence, the expected improvement in the quality of entity ranking can be based on the expected entropy reduction after receiving the user response for attribute X. Instead of entropy, one could also use other common measures of "purity" for a probability distribution, including Gini [5].

Effort-based. Entropy and the other purity measures do not help the user decide if adding a condition for another attribute is *"worth the effort"*. If it costs the user an effort of ϕ_X to decide about and then specify a condition for attribute X, then this investment should result in savings of future effort of at least ϕ_X. We measure these future savings, denoted as Δ, in terms of the reduction of expected effort for going through the ranked list of entities as defined in Eq. 1. Then function F for measuring the ranking improvement after adding a condition for attribute X is defined as $\Delta - \phi_X$. A negative value signals that investing effort for specifying a condition for this attribute is on expectation not worth the small improvement in result quality.

VI. Probability Estimation

Our methods make use of $\Pr(Y = y \,|\, A_1, \ldots, A_k, \bar{y}_1, \ldots, \bar{y}_l, D)$ and $\Pr(A = a \,|\, A_1, \ldots, A_k, \bar{y}_1, \ldots, \bar{y}_l, D)$. Recall that $A_1, A_2, \ldots, A_k$ represent potentially imprecise conditions, i.e., distributions over the possible values of the corresponding attributes $X_1, \ldots, X_k$.

In principle, we can leverage the vast body of previous work on classification and prediction techniques to estimate these probabilities. Formulas of the type $\Pr(C \,|\, X_1, X_2, \ldots, X_k, D)$ define the *posterior probability* of a class C in Bayesian classification [6]. In classification problems in general the goal is to predict to which class a given input object belongs. This is done in two phases. First a classification model is constructed from a set of training objects, each of which has a combination of input values and

a known class label. Then this model is used to predict the class of an input with unknown class label [5]. It is easy to see the relationship to our problem: D can take on the role of the training set and attribute Y corresponds to the set of possible classes.

In addition to Bayesian classification techniques, it has been shown that virtually all popular classification methods such as SVMs, artificial neural networks, and decision tree ensembles can be modified to output such probabilities [7]. Based on this observation, we can in theory leverage almost any classification technique using the following basic approach for estimating $\Pr(Y \mid A_1, \ldots, A_k, \bar{y}_1, \ldots, \bar{y}_l, D)$:

- Using data set D, train a classification model $M(X_1, X_2, \ldots, X_k)$ that predicts the probability of each entity $y \in Y - \{y_1, \ldots, y_l\}$ for a given input vector $(x_1, x_2, \ldots, x_k) \in X_1 \times X_2 \times \cdots \times X_k$. (If D contains probabilistic data, the we can use classification techniques for uncertain data [8]. Alternatively, one can transform a probabilistic data set D to a data set without uncertainty by sampling multiple training records from each uncertain data record.)
- Sample a sufficiently large number of points from the joint distribution over $A_1, \ldots, A_k$. For each such vector $(x_1, x_2, \ldots, x_k)$, $M(x_1, x_2, \ldots, x_k)$ returns the desired probabilities for the entities.
- Return the average of these probabilities for each entity.

We can use the same approach also for estimating $\Pr(A \mid A_1, \ldots, A_k, \bar{y}_1, \ldots, \bar{y}_l, D)$. The only difference is that the model is trained to predict the possible user responses for unspecified attributes, not the probabilities of the entities.

A. Challenges

While conceptually straightforward, making the above probability-estimation algorithm work in practice is very difficult due to the requirement of guaranteeing *interactive* response time. Consider the status of the algorithm that recommends additional conditions after k attributes have been specified, i.e., only the remaining $m - k$ still need to be considered. For each of these candidate attributes X, we sample s_1 different possible user responses $a \in A$. To estimate $\Pr(A = a \mid A_1, \ldots, A_k, \bar{y}_1, \ldots, \bar{y}_l, D)$, the above algorithm samples s_2 vectors $(x_1, x_2, \ldots, x_k)$ and then computes $M(x_1, x_2, \ldots, x_k)$ at a cost c_M. Computation cost is dominated by the model evaluations, costing a total of

$$(m - k) \cdot s_1 \cdot s_2 \cdot c_M.$$

While s_1 and s_2 are easily tuneable, c_M is usually fixed or even hard to predict, depending on the model type used. We discuss in Section VI-C how to also make it tuneable so that we can achieve the best tradeoff between sample sizes and model quality for given hardware resources.

The cost formula above assumes that model M is readily available. Unfortunately, after the user provides additional information A_{k+1} for attribute X_{k+1}, we will need a model $M(X_1, \ldots, X_k, X_{k+1})$ in the next round. One option is to train this model on-the-fly, but this adds to the computation cost and hence increases response time. Also, for large data sets, state-of-the-art data mining models cannot be trained in a few seconds. To avoid on-the-fly model training, one could precompute these models ahead of time. However, this is tricky due to the exponential number of possible attribute subsets. We discuss our solution next.

B. Solution: Bagged Tree Ensembles

We propose using bagged decision tree ensembles [9] for probability estimation. A decision tree recursively partitions the data space, attempting to find partitions with high purity, i.e., where one class clearly dominates over all others. Each non-leaf node in the tree splits the data space on some attribute. Tree traversal for making a prediction starts at the root and proceeds like in a standard search tree. The leaf nodes contain predictions based on the distribution of the data records that fall into the corresponding region of the data space. Details can be found in any data mining textbook [5]. A bagged tree ensemble consists of many such trees, each trained on an independent bootstrap sample of the training data. To make a prediction for a given input, all trees are traversed and their individual outputs are averaged. We decided to use bagged trees for several reasons.

First, trees can handle any attribute type and missing values. Bagged trees are robust against noise and overfitting and have been shown to return excellent probabilities "out-of-the-box" [7]. This makes them ideal for our problem.

Second, due to their structure of splitting on an attribute at a time, trees can easily compute expectations like $\Pr(Y \mid A_1, \ldots, A_k, \bar{y}_1, \ldots, \bar{y}_l, D)$ in a *single pass* through the tree. At a split node whose attribute value is 100% certain, i.e., A_i is a distribution where a single value of the corresponding attribute X_i has probability 1.0, the entire "weight" follows the corresponding branch. If the attribute value was specified by the user, but is uncertain, the "weight" is partitioned according to the provided probability and each partial weight is sent down the corresponding branch. If the split attribute value has not been specified at all, partial weights are computed based on the training data distribution of the attribute's value in the corresponding region of the data space (which is stored in the node). It is easy to show that this is consistent with the desired computation of the expected entity probabilities when taking the expectation over all uncertain and missing input values. Hence there is no need to sample from $A_1, \ldots, A_k$, i.e., s_1 is effectively equal to 1.

Third, a tree trained for input $(X_1, \ldots, X_m)$ can be used to make predictions for any subset of these attributes. Here we exploit the tree's ability for dealing with missing input values during the prediction process by sending partial records, whose weights are determined by the training data distribution in the node, to the next lower level. (Details can be found in any standard data mining textbook [5].) This eliminates the problem of on-the-fly model training or pre-computing a large number of models for different attribute subsets.

And fourth, the comparably simple index-like structure makes tree cost predictable and tuneable, as we discuss below.

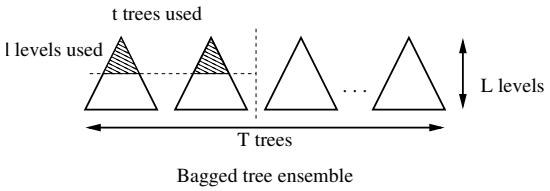

Fig. 7. Cost tuning for bagged trees

C. Controlling Tree Response Time

The greatest advantage of tree-based methods over other popular classification techniques is that their response time is very predictable and tuneable, which is crucial for guaranteeing interactive response time. We discuss this for the computation of $\Pr(A \mid A_1, \ldots, A_k, \bar{y}_1, \ldots, \bar{y}_l, D)$.

During a tree traversal for computing $\Pr(A \mid A_1, \ldots, A_k, \bar{y}_1, \ldots, \bar{y}_l, D)$ we can keep track of the number of tree nodes accessed. Due to the simple structure, tree traversal cost can be accurately predicted based on the number of nodes accessed. In addition to predictability, we can tune bagged tree cost very accurately as well. Assume the ensemble consists of T trees, each with at most L levels (see Figure 7). By using only $t < T$ of these trees, we can reduce cost proportional to the reduction in the number of nodes accessed. Similarly, cost can be reduced by limiting access to the top $l < L$ levels of each tree. To be able to do this, we also store the class distribution in each inner node of the trees, not just in the leaf level. As the user reveals more attribute value distributions and some tree branches are pruned due to zero probability mass, prediction time will drop over time. As that happens, the initial limits for t and l can be increased. Since this tree ensemble is capable of adapting to the time threshold, we call it *Adaptive Tree*. The term *Full Tree* refers to the full bagged ensemble with $t = T$ and $l = L$.

To be able to automatically adjust l and t, we need to estimate how many more tree nodes will be accessed if t increases to t' and l to l'. The former is fairly simple: since all trees are trained on bootstrap samples of the same size and similar data distribution, we can estimate the size of a newly added limited tree quite accurately as the average of the sizes of the already used t limited trees. The effect of the level increase is more difficult to estimate, because the sub-trees in the newly added levels are typically very skewed. However, since we already accessed all nodes up to level l, we know exactly how many children will be accessed at level $l + 1$. Hence as long as we increase l by at most 1 in each iteration, we can also accurately estimate the number of nodes accessed after a level-limit increase.

The only difficult case occurs when the user modifies or completely removes a previously specified condition, say on attribute X. After this modification, a child c of a node splitting on X that had zero probability mass before might now have non-zero probability and hence would be accessed. In some cases we will not know the number of tree nodes down to level l in the sub-tree rooted at this child c. To deal with this case, we introduce a "local" level limit l_c for c, where l_c denotes the level of c. This way in the next round, i.e., after the next user interaction, we only access c but do not traverse its sub-tree. After accessing c, we know the number of its children and increase the local level limit to $l_c + 1$. This process continues until the local limit reaches l. Using this approach we can always provide a hard upper bound on the number of tree nodes accessed in the next round.

We can then estimate the system cost for computing the ranked list of entities and the benefit of specifying a currently unspecified attribute, i.e., to generate the lists shown in the center and left table in Figure 3, as follows. (The analysis for sensitivity computation is similar and hence omitted due to space constraints.) Let n_y and n_a be the number of nodes accessed in the bagged tree models used to predict the entity probabilities and each attribute probability, respectively. Let θ denote the average time for accessing a single tree node and let $d = m - k$ be the number of remaining unspecified attributes. Then T_{old}, the system time when using the current (i.e., "old" tree model) is

$$T_{old} = n_y\theta + u + dn_a\theta + s_2 dn_y\theta + du + v.$$

This cost is obtained as follows. First the algorithm accesses the entity prediction model to compute the probabilities of all entities (cost: $n_y\theta$). Then it sorts the entities (cost: u). We use u to denote the constant time, i.e., independent of the tree model size, needed for sorting the entities. To compute the expected ranking quality improvement for each attribute, the corresponding attribute probability models are accessed (cost: $dn_a\theta$). For each of the s_2 samples from the obtained attribute-value distributions (see Section VI-A), the entity probabilities are computed (cost: $s_2 dn_y\theta$) and the hypothetical ranking is computed by sorting them (cost: du). Finally, v accounts for the constant, i.e., independent of tree model size, overhead for all other tasks including reading the user's response and updating the UI.

Similar to T_{old}, we can compute an upper bound on T_{new}, the cost of the next iteration. Instead of n_y and n_a, we use the corresponding upper bound on the number of tree nodes that will be accessed in the next iteration, computed as described above. (Also, note that d might have changed.) We can then choose any combination (l', t') for which T_{new} is below the response-time limit. Our experiments show that we indeed can successfully guarantee interactive response times.

VII. Experiments

The goal of the experiments is to provide a proof of concept for three important properties of our approach: (1) improved efficiency of sensitivity analysis due to Theorem 1, (2) quality of the recommended attributes for additional conditions, and (3) guarantee of interactive system response times. In all experiments, the effort-based function is used to compare the entity rankings.

A. Data

In our experiments we are using a real-world data set provided by the Cornell Lab of Ornithology. The bird data set contains examples of commonly observable bird species in North America. There are 372 different species, each described by 177 binary attributes, e.g., encoding the size of the bird, its color and the pattern of its different body parts, its behavior, and so on. There are 2,000,000 individual bird observations that we generated as follows. First, we randomly select an

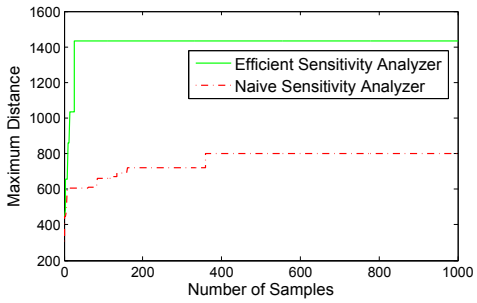

Fig. 8. Naive vs. efficient sensitivity estimation algorithm for a multi-valued attribute

observation from the *real* eBird data set [10], i.e., a data set containing actual observations reported by citizen scientists. Unfortunately, while eBird reports the species of the observed bird, it does not contain individual bird properties like color, size etc. We generate these individual bird features by sampling from a feature distribution that was carefully defined by the domain experts for every single species.

B. Sensitivity Analysis

In this experiment we explore how much difference it makes in practice to use the efficient algorithm for sensitivity estimation instead of the naive one. We consider for $\mathcal{A}$ the entire space of alternative responses. For each attribute, we randomly select 1,000 sample points and plot the current maximum distance versus the number of samples explored so far. The goal is to find a good estimation of the maximum distance with as few samples as possible. Figure 8 shows this experiment for attribute "size", which has 9 values. The naive algorithm chooses sample points randomly in the entire space of possible distributions for bird size, while the efficient algorithm only selects points on the boundary of the sample space. Figure 8 shows that after sampling only 27 points, the efficient algorithm has already reached a ceiling, while the naive one keeps improving with more samples. Still, even after 1000 samples for the naive algorithm, the maximum distance found by the efficient algorithm is 78% greater.

C. Additional Condition Recommendation

In this series of experiments we explore how well our condition recommendation framework improves the entity ranking quality. Applying the solution introduced in Section V, we compare the Full Tree (F) to the Adaptive Tree (A). This reflects the effect of potentially less accurate probability estimates in the Adaptive Tree due to the limited tree model. The other competitor is *Random* (R) which picks a random attribute.

In Tables I, II and III the first three columns report $\rho(y)$, the rank of the true entity (initially at rank 100) for the three approaches. The second and third sets of three columns report the probability of the true entity and the expected user effort, respectively. The first row shows the initial values based on the data set D only. The values are updated as conditions are added one by one in the next rows. The experiment is repeated for varying degrees of uncertainty in the user response. Uncertainty value x means that with probability x the user will specify an imprecise constraint. For imprecise

TABLE I. IMPROVEMENT DUE TO CONDITION RECOMMENDATION: UNCERTAINTY=0

$\rho(y)$			$\Pr(Y \mid A_1, \ldots, A_k, D)$			Expected Effort		
F	A	R	F	A	R	F	A	R
100	100	100	0.002	0.002	0.002	38.1	38.1	38.1
62	63	100	0.003	0.003	0.002	21.6	21.6	39.1
45	45	100	0.007	0.007	0.002	19.8	19.8	40.1
28	28	100	0.014	0.014	0.002	14.0	14.0	41.1
19	19	100	0.022	0.022	0.002	11.9	11.9	42.1
10	10	100	0.035	0.035	0.002	9.4	9.4	43.1
6	6	99	0.070	0.070	0.002	10.0	10.0	43.8
4	4	99	0.143	0.142	0.002	9.6	9.6	44.8
3	3	99	0.222	0.222	0.002	10.0	10.0	45.8
1	1	66	0.458	0.459	0.003	10.8	10.8	42.0
1	1	66	0.768	0.768	0.003	11.4	11.4	43.0

TABLE II. IMPROVEMENT DUE TO CONDITION RECOMMENDATION: UNCERTAINTY=0.5

$\rho(y)$			$\Pr(Y \mid A_1, \ldots, A_k, D)$			Expected Effort		
F	A	R	F	A	R	F	A	R
100	100	100	0.002	0.002	0.002	38.1	38.1	38.1
75	76	100	0.003	0.003	0.002	35.3	35.3	39.1
54	54	100	0.006	0.006	0.002	30.6	30.7	40.1
42	42	100	0.008	0.008	0.002	30.6	30.6	41.1
26	26	100	0.012	0.012	0.002	24.2	26.5	42.1
19	19	100	0.020	0.020	0.002	20.6	21.1	43.1
9	17	100	0.031	0.020	0.002	18.9	21.8	44.1
4	17	100	0.062	0.020	0.002	18.4	22.0	45.1
3	18	100	0.125	0.020	0.002	16.8	21.3	46.1
3	17	100	0.125	0.020	0.002	17.6	21.4	47.1
3	16	100	0.125	0.020	0.002	17.4	21.6	48.1

constraints, the probability distribution is $(0.25, 0.75)$ between "yes" and "no". To deal with the effect of randomness, both in the user's response and the intrinsic randomness of the Random method, we repeated each experiment many times. Due to the lack of space and similarity of results, we only report the result of a representative run. The effort is computed by assuming that answering any question incurs the same cost.

Tables I, II and III present the results when the uncertainty in the user's response is 0, 0.5 and 1 respectively. The Full and the Adaptive Trees clearly outperform the Random approach in all cases, providing a strong evidence that a wise selection of attributes has a great impact on the query refinement process. Although the Random approach asks the same number of questions, it can at best only slightly improve the initial rank.

Table I shows that with certain responses the results of the Adaptive and the Full Trees are very similar. The reason is that providing precise conditions for important attributes quickly shrinks the space of possible entities. It also prunes away zero-probability branches in the trees, letting the Adaptive Tree grow quickly toward full size. Also, Tables II and III show that the Full Tree is better than the Adaptive Tree for higher uncertainty values. This experiment and the timing experiment in Section VII-D indicate that there is a trade-off between the system response time and the quality of probability estimates.

D. Interactive Response Time

An important feature of our framework is to guarantee interactive response times, even for large data sets with many attributes. Our experiments show that this goal is achieved. We discuss representative results, comparing an Adaptive Tree to a Full Tree. Bagged tree ensembles (represented by the Full Tree results here) have been shown to provide excellent probability estimates, but might be too slow for large data sets. In this experiment the Full Tree ensemble consists of 100 trees. Each

TABLE III. IMPROVEMENT DUE TO CONDITION RECOMMENDATION: UNCERTAINTY=1

$\rho(y)$			$\Pr(Y \mid A_1, \ldots, A_k, D)$			Expected Effort		
F	A	R	F	A	R	F	A	R
100	100	100	0.002	0.002	0.002	38.1	38.1	38.1
75	76	100	0.003	0.003	0.002	35.3	35.3	39.1
63	63	100	0.004	0.004	0.002	37.5	37.5	40.1
49	49	100	0.006	0.006	0.002	37.4	37.2	41.1
45	45	100	0.007	0.007	0.002	37.0	38.1	42.1
36	36	100	0.008	0.008	0.002	38.5	38.2	43.1
29	34	100	0.010	0.008	0.002	39.6	38.5	44.1
20	36	100	0.015	0.008	0.002	41.4	39.9	45.1
15	36	100	0.022	0.008	0.002	42.5	41.2	46.1
13	40	102	0.022	0.008	0.002	43.6	42.7	48.1
13	39	102	0.022	0.008	0.002	44.8	44.0	49.1

is grown on an independent bootstrap sample of 2 million records, such that nodes with less than 1000 objects are not split any further.

We set the threshold for interactive user response time to 5 seconds. (Other thresholds showed similar results.) Adaptive Tree automatically selected an appropriate setting to guarantee this response time. We report results for the most expensive computation—recommending additional conditions.

Figures 9, 10, and 11 show how system response time varies as more user-provided conditions are added with varying degrees of uncertainty. The x-axis reports the number of attributes specified so far and the y-axis reports the system response time measured from the time the user submitted the new condition until the system responded.

Figure 9 shows results for a run where the user always provides 100% precise conditions. The Full Tree is very slow in the beginning, suffering from a response time of about 15 sec. As more conditions are added, Full Tree responds faster because more tree branches are pruned. Also, the more conditions are specified, the fewer attributes must be considered for recommending the next one. The Adaptive Tree holds response time below the 5 sec threshold. As user input results in pruned tree branches, it adapts number of trees and tree levels used, converging to the Full Tree size.

The basic picture is similar in Figures 10 and 11, as the uncertainty probability increases to 0.5 and 1, respectively. Recall that for uncertain responses, the probability distribution is (0.25, 0.75). Comparing the different Full Tree curves, it is evident that response time improves more slowly at higher uncertainty levels. This is due to the fact that an uncertain response does not result in pruning of tree branches as each child branch receives a non-zero weight. For uncertainty of 1, no branches in the tree can be pruned. Hence response time drops almost perfectly linearly due to the decreasing number of remaining attributes considered for future conditions.

The experiment shows that the Adaptive Tree virtually guarantees our system to respond within the set realtime limit. It initially tracks the 5 sec threshold and at some point mirrors the behavior of the Full Tree when it converged to full size as user input results in pruned tree branches.

VIII. RELATED WORK

Recent work on optimal questionnaire design explores the selection and ordering of questions on electronic forms to extract the most information from users [4]. Similarly, Visipedia [11] is designed to determine the category of an image by posing questions to the user based on visual properties. Questions are selected using information gain. Issues like sensitivity and interactive response time guarantees are not explored. The 20Q game uses a proprietary algorithm to select good questions in order to guess an object the user is thinking of [12]. It mixes ideas from artificial neural networks and binary search for selecting the best objects and the next question. Previous work on search in the presence of errors in the theory community is limited to scenarios where user responses are either correct or erroneous, but not uncertain [13]. Conditions for attribute values are not considered.

Abouzied et al. [14] propose techniques for learning quantified Boolean queries by asking users to label data objects as answers or non-answers. For human-assisted graph search, Parameswaran et al. [15] explore how to select an optimal set of graph nodes to minimize the number of reachability questions a human expert has to answer.

Classification is a well-studied problem in data mining and machine learning [5]. The goal is to learn a model to predict the class label for a given input. Issues related to interactive refinement of model input are not considered, but we can leverage traditional classification techniques for probability estimation. The notion of revealing information at a cost has been explored in active learning [16]. The most common scenario is as follows: Given a set of input vectors and a budget, choose some inputs to be labeled with their class value such that a model trained on the resulting labeled data set has the highest accuracy (or other measure of quality). Work on active feature acquisition and classification considers input attribute values to be revealed at a cost and tries to balance this cost with the benefit and cost of correctly or incorrectly classifying a given input. The goal is to find the best model and sequence of "tests", such that total cost is minimized [17], [18]. This different optimization goal results in very different solutions compared to exploratory search.

Querying of uncertain data has been studied for probabilistic databases [1], but the query is given and specified precisely. Our approach can also be applied to probabilistic data.

Search and ranking suggest a relationship to information retrieval [19], but our work deals with structured data. Previous work on top-k queries in database community [20] focuses on different goals not related to making a database user-friendly.

Crowd-sourcing refers to a variety of approaches that involve humans in the creation of data and the solution of problems that are difficult for computers [21], [22]. Our goal is to make it easier for non-technical users to access crowd-sourced databases.

IX. CONCLUSIONS

Databases will only gain wide acceptance if they support a broad spectrum of users, including non-technical ones, in finding the information they are looking for. We focused on scenarios where the user does not have a specific query in mind and conditions for some of the attributes might be imprecise. Our techniques help the user (1) quickly see which results are the best answers to her query, (2) evaluate the risk of

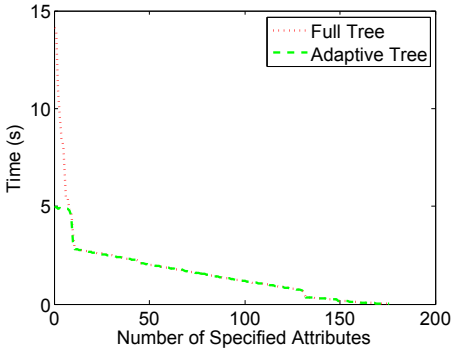

Fig. 9. Response time: Uncertainty=0

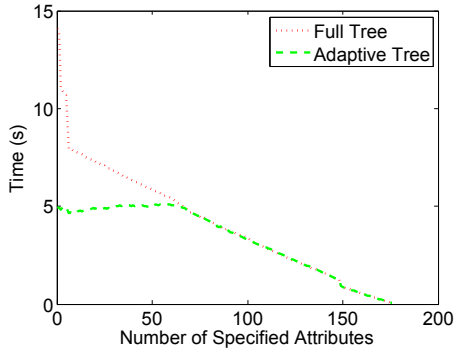

Fig. 10. Response time: Uncertainty=0.5

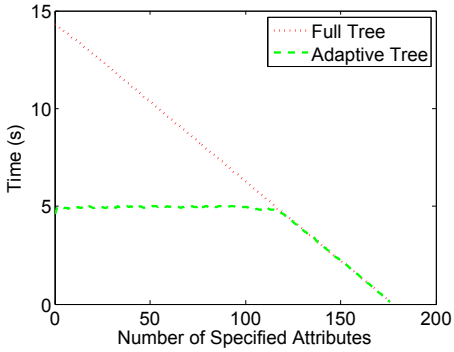

Fig. 11. Response time: Uncertainty=1

specifying conditions she is not certain about, and (3) decide which additional conditions to add.

This functionality relies on a probability-based framework, which makes uncertainty explicit, and on adaptive probability estimation techniques that can be tuned to deliver approximate estimates for a given hard realtime limit on response time. The main thrust of our future work is to identify other approaches for fast and accurate probability estimation.

ACKNOWLEDGMENT

This work was supported by the National Science Foundation under Grant Nos. IIS-1017793 and DRL-1010818. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors. We would like to thank the Merlin and eBird teams at the Cornell Lab of Ornithology, in particular Jessie Barry, Miyoko Chu, Scott Haber, Tim Levatich, and Syed Rehman, for providing the data and domain-related advice. We also thank the anonymous reviewers for their feedback.

REFERENCES

[1] D. Suciu, D. Olteanu, C. Re, and C. Koch, *Probabilistic Databases*. Morgan & Claypool, 2011.

[2] M. M. Breunig, H.-P. Kriegel, R. Ng, and J. Sander, "LOF: Identifying density-based local outliers," in *Proc. ACM SIGMOD*, 2000, pp. 93–104.

[3] E. Knorr, R. Ng, and V. Tucakov, "Distance-based outliers: Algorithms and applications," *VLDB Journal*, vol. 8, no. 3–4, pp. 237–253, 2000.

[4] K. Chen, H. Chen, N. Conway, J. M. Hellerstein, and T. S. Parikh, "Usher: Improving data quality with dynamic forms," *IEEE TKDE*, vol. 23, no. 8, pp. 1138–1153, 2011.

[5] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, 3rd ed. Morgan Kaufmann, 2011.

[6] T. Mitchell, *Machine Learning*. McGraw-Hill, 1997.

[7] A. Niculescu-Mizil and R. Caruana, "Predicting good probabilities with supervised learning," in *Proc. ICML*, 2005, pp. 625–632.

[8] S. Tsang, B. Kao, K. Yip, W.-S. Hoy, and S. D. Lee, "Decision trees for uncertain data," *IEEE TKDE*, vol. 23, no. 1, pp. 64–78, 2011.

[9] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, pp. 123–140, 1996.

[10] Cornell Lab of Ornthology and Audubon, "eBird project," ebird.org.

[11] S. Branson *et al.*, "Visual recognition with humans in the loop," in *Proc. Eur. Conf. on Computer Vision (ECCV)*, 2010.

[12] R. Burgener, "Artificial neural network guessing method and game," US Patent Application US 2006/0230008 A1, 2006, online game at www.20q.net.

[13] J. A. Aslam and A. Dhagat, "Searching in the presence of linearly bounded errors," in *Proc. STOC*, 1991, pp. 486–493.

[14] A. Abouzied, D. Angluin, C. Papadimitriou, J. M. Hellerstein, and A. Silberschatz, "Learning and verifying quantified boolean queries by example," in *Proc. PODS*, 2013, pp. 49–60.

[15] A. Parameswaran, A. D. Sarma, H. Garcia-Molina, N. Polyzotis, and J. Widom, "Human-assisted graph search: It's okay to ask questions," *Proc. VLDB Endowment*, vol. 4, no. 5, pp. 267–278, 2011.

[16] B. Settles, "Active learning literature survey," Univ. of Wisconsin–Madison, Tech. Rep. 1648, 2009.

[17] R. Greiner, A. J. Grove, and D. Roth, "Learning cost-sensitive active classifiers," *Artif. Intell.*, vol. 139, no. 2, pp. 137–174, 2002.

[18] S. Ji and L. Carin, "Cost-sensitive feature acquisition and classification," *Pattern Recognition*, vol. 40, no. 5, pp. 1474–1485, 2007.

[19] C. D. Manning, P. Raghavan, and H. Schutze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[20] M. Shmueli-Scheuer, C. Li, Y. Mass, H. Roitman, R. Schenkel, and G. Weikum, "Best-effort top-k query processing under budgetary constraints," in *Proc. ICDE*, 2009, pp. 928–939.

[21] A. Marcus, E. Wu, D. R. Karger, S. Madden, and R. C. Miller, "Crowdsourced databases: Query processing with people," in *Proc. CIDR*, 2011, pp. 211–214.

[22] J. Wang, G. Li, T. Kraska, M. J. Franklin, and J. Feng, "Leveraging transitive relations for crowdsourced joins," in *Proc. SIGMOD*, 2013, pp. 229–240.