Merlin: Exploratory Analysis with Imprecise Queries

Bahar Qarabaqi and Mirek Riedewald, *Member, IEEE*

Abstract—Merlin supports exploratory search in large databases. The user interacts with it by specifying probability distributions over attributes, which express imprecise conditions about the entities of interest. Merlin helps the user home in on the right query conditions by addressing three key challenges: (1) efficiently computing results for an imprecise query, (2) providing feedback about the sensitivity of the result to changes of individual conditions, and (3) suggesting new conditions. We formally introduce the notion of sensitivity and prove structural properties that enable efficient algorithms for quantifying the effect of uncertainty in user-specified conditions. To support interactive responses, we also develop techniques that can deliver probability estimates within a given realtime limit and are able to adapt automatically as interactive query refinement proceeds.

Index Terms—8.II.VIII.VIII. Interactive data exploration and discovery

1 Introduction

We propose Merlin, a new approach for *exploratory search* in large databases. It is designed to accommodate uncertainty and imprecision in user-provided query conditions through two major technical contributions: (1) a novel notion of sensitivity to quantify the impact of uncertainty on the query result, and (2) fast algorithms for probability estimation that can adapt to a user-specified realtime constraint on system response time.

To illustrate Merlin's functionality, consider the following example motivated by a collaboration with the Cornell Lab of Ornithology. Through hugely successful citizen science projects such as eBird (http://ebird.org), the Lab has collected more than 100 million reports of bird sightings, adding tens of millions annually. It wants to leverage this resource to help less experienced birders identify the species of a bird they observed. Assume each observation in the database specifies properties of the bird (e.g., species, size, color) and the observation event (e.g., location, weather, habitat features).[1] After observing an individual of an unknown species, the user can then search this database to find species that matched her description.

The main problem with this idea is that the user often is not certain about her query. Consider Amy, who just observed a "small to medium size bird that was mostly blue" in her garden. Through Web forms such as those shown in Figures 1 and 2, Amy could express such imprecise search conditions. For example, she might decide that the bird was most likely the size of a robin, but is not sure about it. The front-end turns the user input into a probability distribution over the different size values, so that Merlin can work with it. The lower the confidence, the wider the probability mass is distributed.[2]

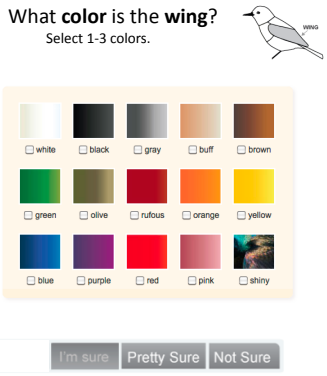

Fig. 1. Possible interface for specifying multiple wing color values

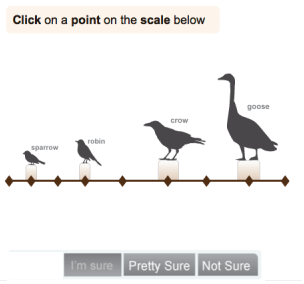

Fig. 2. Possible interface for specifying the bird size

After entering the initial search conditions, Merlin can present three types of feedback as shown in Figure 3. The table in the center shows the top-ranked species. Amy could go through the list and **explicitly eliminate** some species or **positively identify** a likely match. Note that Amy can continue the search even after positive identification, in order to find other likely matches. Alternatively, she can use the other two tables to refine her query. The table on the left tells her which other attributes would be most helpful in **narrowing the search**. Since the system does not know what species the user is looking for, an attribute's potential usefulness in improving result quality is measured based on the intuition that the most useful attributes are those that best separate "winners" from "losers". For example, breast

• *B. Qarabaqi and M. Riedewald are with the College of Computer and Information Science, Northeastern University, Boston, MA, USA. Email: {bahar, mirek}@ccs.neu.edu*

1. eBird currently does not collect individual bird features such as size and color. The Lab adds those features based on an expert-designed model as discussed in Section 7. In the future, successful bird identification could contribute additional records as users already entered the relevant attribute values during the search process.

2. Other interfaces could be used instead, e.g., allowing the user to specify a "score" for each bird size option. The design of the interface and the transformation into a probability distribution are beyond the scope of this article. Merlin works with any probability distribution provided.

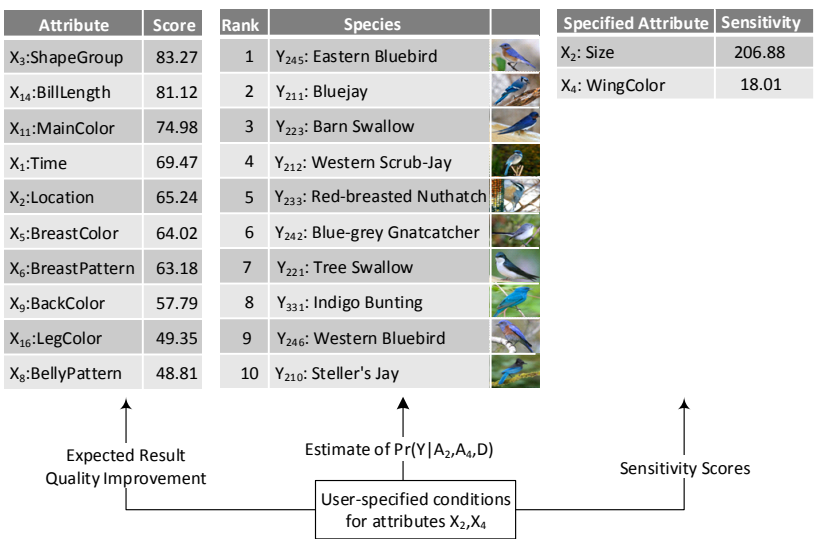

Fig. 3. Feedback to the user after she specified conditions for attributes X_2 and X_4

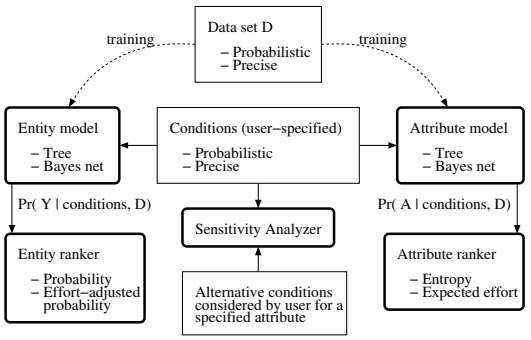

Fig. 4. Merlin overview. Entity and attribute models are trained from data set D. Any model type returning good probabilities can be used, e.g., bagged trees or Bayes nets. At query time, given user-specified conditions, the entity and attribute rankers produce ranked lists like those shown in the center and on the left in Fig. 3, respectively. Ranking can be based on different criteria as indicated in the boxes. Using the entity model, the sensitivity analyzer explores how sensitive the current result ranking is to changes the user considers for some attribute's condition.

color would receive a high score if among blue-winged species, some breast colors occur frequently on individuals of some, but not other species. On the other hand, if blue-winged species tend to have mostly the same breast color, then breast color would receive a low score, because it does not help distinguish the species.

The table on the right shows Merlin's novel **sensitivity score**, which is specifically designed to support exploratory analysis involving imprecise conditions. Recall Amy's uncertainty about the bird's size. Clearly, size is an important feature for distinguishing species. Hence Amy would like to enter it. However, what if she gets it slightly wrong and the correct species is eliminated? Our proposed notion of sensitivity helps Amy gauge the *risk* of entering a condition. It quantifies how much the query result (center table in Figure 3) could change if Amy were to modify the corresponding condition. Here Merlin could either consider all possible alternative inputs, or let the user specify a range of alternative inputs she considers. (For bird size, Amy could mark lower and upper end of the range of size options she considers in an interface similar to Figure 2.) A high sensitivity score might convince Amy to withdraw her input for this attribute and first enter others, e.g., location. She might then try size again at a later time. Since an attribute's sensitivity depends on the conditions on other attributes, the uncertainty on bird size might have less impact then.

For the system to interact effectively with a user, it needs to **respond quickly** after the user entered new information. Since the meaning of "interactive" varies depending on the user and application, Merlin lets the user choose her preferred threshold. It will treat it as a realtime constraint, producing a response within the time limit.

Exploratory search with imprecise conditions could benefit many other 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). In the medical domain, a database of diseases, their symptoms, potential causes (e.g., family history and lifestyle choices), and 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, Merlin's techniques can be applied to any relational database of interest, *helping a database user fine-tune imprecise conditions* for exploratory analysis.

We make several contributions aimed at improving support for exploratory search in databases. First, exploratory search usually involves uncertainty; not only in the data [1] but also in the query. To deal with it, we propose a **probability-based framework** in Section 2. 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. (Section 3)

Our second contribution helps the user judge the potential risk of specifying a condition she is not certain about. In particular, if getting it just slightly wrong might significantly change the result, then it might be safer to not enter it. 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** in Section 4.

Third, while sensitivity quantifies the *risk* of a condition, Section 5 discusses how to estimate the *benefit* by **identifying the best new conditions to be added** in order to improve result quality.

Fourth, as a user-driven process, query refinement should be *interactive*. Since computation time tends to be high when dealing with large data and imprecise queries, in Section 6 we propose **fast approximate estimation techniques that deliver results within a given response time threshold**. Experiments, related work, and conclusions are presented in Sections 7, 8, and 9, respectively.

TABLE 1
Important notation.

D: given relational table or view with schema $\{X_1, X_2, \ldots, X_m, Y\}$
X: data attribute, e.g., hasWingColorRed with domain $\{\mathrm{Y}, \mathrm{N}\}$, for which the user can specify a condition in the query
Y: data attribute identifying entities of interest, e.g., species of a bird
A: set of all possible probability distributions over the values in the domain of attribute X, e.g., $\{(p_1, p_2) \mid p_1, p_2 \geq 0, p_1 + p_2 = 1\}$ for hasWingColorRed
$a \in A$: specific probability distribution over the values in the domain of attribute X, e.g., $(0.2, 0.8)$
k: number of attributes for which the user has specified conditions
$\Pr(Y \mid A_1, \ldots, A_k, \bar{y}_1, \ldots, \bar{y}_l, D)$: entity probability, given distribution $a_i \in A_i$ for attribute X_i, $i = 1, \ldots, k$, and explicit rejection of entity y_j, $j = 1, \ldots, l$
$\Pr(A = a \mid A_1, \ldots, A_k, \bar{y}_1, \ldots, \bar{y}_l, D)$: probability that the user will specify condition a for attribute X, given distribution $a_i \in A_i$ for attribute X_i, $i = 1, \ldots, k$, and explicit rejection of entity y_j, $j = 1, \ldots, l$
ϕ_j: effort required from the user to decide if entity $y_j \in Y$ is of interest; e.g., measured as user response time after the entity is presented
L_p: ranked list of entities based on user-specified condition p
$\rho_p(y)$: rank of entity y in ranked list L_p
$\mathcal{A}_i \subseteq A_i$: alternative conditions the user considers for attribute X_i

2 Data Model and Framework

We introduce the data model and propose Merlin's probabilistic framework for exploratory search in databases. Important notation is summarized in Table 1.

2.1 Data Model

We are given a relational table or view D with schema $\{X_1, X_2, \ldots, X_m, Y\}$. Attribute Y takes on a special role, identifying entities of interest to the user. Depending on the problem, any attribute of D could take on this role. For instance, in the bird identification example, Y is the species name. When looking for geographical regions, Y would be the corresponding region identifier. Even though Y identifies entities of interest to the user, it does *not* need to be a key of D. The tuples in D could represent precise or probabilistic information, including crowd-sourced imprecise data.

In the bird example, the entities of interest are bird species. The X_i describe various properties of a bird and observation event, e.g., hasWingColorRed and obsLongitude. A species will be observed more than once, hence there will be multiple records for it. 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 Y-value. (For this reason Y is not necessarily a key of D.)

2.2 Probabilistic Query

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 conditions on some of the attributes X_i. In a traditional relational database setting, the user would then execute query

```
SELECT Y, COUNT(*) AS freq
FROM   D
WHERE  condition(X1) AND...AND condition(Xm)
  AND Y <> y1 AND ... AND Y <> yl
GROUP BY Y
ORDER BY freq DESC
```

to determine which of the entities are most frequently associated with attribute values satisfying the specified conditions. The second part of the WHERE clause reflects explicit rejection of entities $y_1, y_2, \ldots, y_l$. If we divide the freq attribute of each result tuple by the total count of tuples in D satisfying the WHERE clause, then we obtain for each $y \in Y$ the fraction it represents in the result. More generally, this query would approximate probability $\Pr(Y = y \mid \text{condition}(X_1), \ldots, \text{condition}(X_m), Y \neq y_1, \ldots, Y \neq y_l, D)$.

To accommodate uncertainty, the WHERE clause needs to support **probabilistic conditions**. 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, e.g., $\{(p_1, p_2) \mid p_1, p_2 \geq 0, p_1 + p_2 = 1\}$ for Boolean attribute hasWingColorRed with domain $\{\mathrm{Y}, \mathrm{N}\}$. When interacting with Merlin, the user specifies some probability distribution $a_i \in A_i$, e.g., $(0.2, 0.8)$ for hasWingColorRed.

Given user-specified conditions $a_1, \ldots, a_k$ for attributes $X_1, \ldots, X_k$, the probability of entity $y \in Y$ is defined as $\Pr(Y = y \mid A_1 = a_1, \ldots, A_k = a_k, Y \neq y_1, \ldots, Y \neq y_l, D)$, written more compactly as

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

Here $\bar{y}_j$ indicates that entity y_j was rejected by the user. 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 removing the accepted entities from consideration.

Probability distributions provide flexibility for expressing a mix of certain and uncertain conditions. Recall user Amy who is not certain about the observed bird's size. She might know for sure that the bird was larger than a sparrow and smaller than a crow, but cannot decide between the three size options around the American Robin (see Figure 2). She can express this by entering a distribution like $(0, 0, 0.2, 0.6, 0.2, 0, 0, 0, 0)$ for the nine possible values of the size attribute. The six zeros indicate her certainty about ruling out the overly small and overly large size options. Sensitivity analysis (Section 4) helps Amy determine if a small change of the non-zero values would have a major impact on the species ranking.

It is conceptually easy to extend Merlin to support inequality conditions such as $(\text{size} \geq 2 \text{ AND size} \leq 4)$. Instead of concrete distribution $(0, 0, 0.2, 0.6, 0.2, 0, 0, 0, 0)$, the user could specify a set of distributions $(0, 0, q_1, q_2, q_3, 0, 0, 0, 0)$, s.t. $0 \leq q_1, q_2, q_3$ and $q_1 + q_2 + q_3 = 1$. Then the formula for the probability of Y becomes an expectation over the different concrete distributions in this set. This is similar to the way we deal with possible future attribute conditions (Section 5) and not discussed here due to space constraints.

2.3 Dealing With Continuous Attributes

In the remainder of this article, we assume that all attributes of D have a discrete domain. In principle, Merlin could be extended to also support continuous domains. However, this additional complication does not provide any benefits for exploratory search. Recall that the *user provides the distributions* as conditions for her query. From the user's point of view, it makes virtually no sense to try and distinguish between distributions like $(0.8, 0.2)$ and $(0.801, 0.199)$. They both express that the user was highly confident about one

of the attribute's values. And as our analytical results for sensitivity show (Section 4), entity probabilities are "well-behaved" in the sense that a smaller change in distribution results in a smaller change in entity probability. Similarly, if an attribute has a very large domain, it can be compressed by representing a distribution over this domain with an appropriate histogram. In general, Merlin can approximate a continuous distribution as closely as desired.

2.4 Exploration Framework

Figure 4 shows the Merlin system components and their interactions. From the given data set D, two types of models are trained at "setup time", i.e., before the system is available for user queries. The entity model predicts $\Pr(Y \mid A_1, \ldots, A_k, \bar{y}_1, \ldots, \bar{y}_l, D)$, i.e., in the bird example the probability of each species. The attribute model predicts the probability of a distribution the user might enter if prompted by Merlin to provide a condition on another attribute such as head color. The probabilities output by these models are used at "query time" by the ranking functions to produce the lists in the center and on the left, respectively, in Figure 3. The sensitivity analyzer also relies on the entity model in order to determine how much a modification of one of the user-specified conditions would affect the entity ranking. Details are discussed in the following sections.

3 Result Ranking

Given user-specified conditions on the attributes of D, Merlin creates a ranked list of the entities $y \in Y$ as shown in the center of Figure 3. It could directly **rank by entity probability** $\Pr(Y \mid A_1, \ldots, A_k, \bar{y}_1, \ldots, \bar{y}_l, D)$.

We propose a measure that generalizes ranking by entity probability by taking *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 of the top-ranked entities might be of interest. In the example in Figure 3, the user scans the list of species in the center from top to bottom. Let ϕ_j denote the user effort required for deciding about the relevance of entity y_j.[3] 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). If y_2 was ranked first, expected effort would drop to $0.49 \cdot 1 + 0.51 \cdot (1 + 10) = 6.1$.

This example motivates **ranking by effort-adjusted entity probabilities**. The *effort-adjusted probability* of entity $y_i \in Y$ is defined as $\Pr(Y = y_i \mid A_1, \ldots, A_k, \bar{y}_1, \ldots, \bar{y}_l, D)/\phi_i$. Effort-adjusted probability has the following useful property (proof provided in the appendix):

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.

3. Effort can vary, e.g., some bird species is easily recognizable from a picture while another requires reading a description. In practice effort can be measured based on the user's response time when interacting with Merlin.

4 Sensitivity Analysis

When dealing with imprecise queries, it is essential to give the user feedback about the potential *risk* of a condition. We propose a very direct and natural measure that quantifies the *sensitivity* of the query result to changes in the condition. Intuitively, user-specified condition a for attribute X has high sensitivity if a "small" change of the condition would result in a "large" change of the entity ranking.

To illustrate the usage of sensitivity, we return to the bird identification example. Assume Amy entered condition $(0.4, 0.2, 0.4)$ for billLength, which is a 3-valued attribute with domain $\{1, 2, 3\}$. Since she is not really sure about the exact probabilities, she asks Merlin how much the current species ranking could possibly change if she were to enter any other probability distribution (p_1, p_2, p_3) instead. If Merlin determines that the resulting change could be "large", she might decide to remove her input for billLength. Alternatively, assume she is almost sure that the billLength value should be 1 or 3, but cannot decide how much probability mass exactly to assign. In particular, she considers all distributions (p_1, p_2, p_3) that satisfy $p_1 \geq 0.4 \wedge p_3 \geq 0.4$. If Merlin tells her that sensitivity in that case is "low", she knows that no matter if she enters $(0.4, 0.2, 0.4)$ or $(0.5, 0.1, 0.4)$, $(0.4, 0.15, 0.45)$ etc., the ranking would be similar. Notice that this does *not* mean that the condition on billLength is irrelevant. The ranking might change significantly if Amy chooses a condition that does not satisfy $p_1 \geq 0.4 \wedge p_3 \geq 0.4$. Low sensitivity simply gives her confidence that the exact choice of values does not matter much *in the range she considers*.

As will become clearer soon, sensitivity of an attribute depends on the conditions on *other* attributes. Hence an attribute with high sensitivity might have much lower sensitivity later on, after conditions on other attributes are modified.

4.1 Definition and Computation of Sensitivity

Definition 1. Let L be the entity ranking based on user-specified conditions $a_1, \ldots, a_k, \bar{y}_1, \ldots, \bar{y}_l$, and let $\mathcal{A}_i$ denote the set of all alternative conditions the user considers for some attribute X_i, $i \in \{1, \ldots, k\}$. The *sensitivity* of the current ranking L to attribute X_i for a set of possible conditions $\mathcal{A}_i$ is defined as the maximum difference $\mathrm{dst}(L, L')$ between L and any other ranking L' that could be obtained if the user were to change X_i's condition a_i to any other distribution $a_i' \in \mathcal{A}_i$. Function $\mathrm{dst}()$ is a distance measure between two entity rankings.

For brevity, we will refer to the "sensitivity of the current ranking L to attribute X_i for a set of possible conditions $\mathcal{A}_i$" simply as the *sensitivity of attribute* X_i. Note that the sensitivity of X_i is affected by the other conditions specified, because all conditions together determine the entity probabilities and hence the ranking. The choice of $\mathcal{A}_i$ also plays a crucial role. Constraints on alternative conditions considered can be specified by eliminating values from the domain of

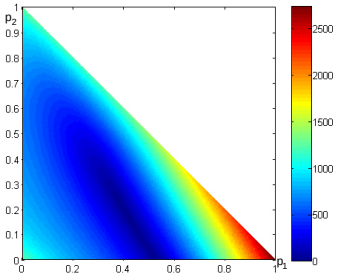

Fig. 5. Ranking distance for 3-valued attribute billLength when changing original condition $(0.4, 0.2)$ to another pair (p_1, p_2)

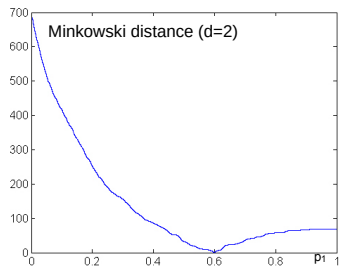

Fig. 6. Ranking distance for Boolean attribute hasWingColorBlue when changing original condition (0.6) to another value

attribute X_i or by limiting the range of probabilities considered for each value. The above example illustrated this for billLength. Similarly, distributions near the specified one can be considered by setting $\mathcal{A}_i$ to include an ε-neighborhood. For billLength with condition $(0.4, 0.2, 0.4)$, this could be defined as the set of probability distributions (p_1, p_2, p_3) that satisfy $|p_1 - 0.4| \leq \varepsilon \wedge |p_2 - 0.2| \leq \varepsilon \wedge |p_3 - 0.4| \leq \varepsilon$.

Figure 5 shows sensitivity for billLength measured in a bird observation data set (discussed in Section 7). Notice that any probability distribution (p_1, p_2, p_3) for a 3-valued attribute is uniquely determined by (p_1, p_2), because $p_3 = 1 - p_1 - p_2$. Hence we only need to consider distributions over the first two values. Starting with the original user-specified condition $p = (0.4, 0.2)$, we sampled 1 million alternative conditions randomly and computed the Minkowski distance (see Section 4.2) in ranking to L. As the colors indicate, distances increase as one "moves away" from $(0.4, 0.2)$. In this example, the greatest distance is obtained in the lower right corner, for distribution $(1.0, 0.0)$. However, near $(0.4, 0.2)$, distance is comparably low, indicating that if the user considers a small neighborhood around the original condition, then the ranking is stable. Figure 6 shows sensitivity for Boolean attribute hasWingColorBlue and initial condition 0.6. (To put the numbers into perspective, note that if each of the 367 species is 5 positions off, total ranking difference would be about 100.)

One can compute the sensitivity of attribute X_i using the following **naive algorithm**: Let L denote the entity ranking based on the user-specified conditions $a_1, \ldots, a_k, \bar{y}_1, \ldots, \bar{y}_l$. Repeat the following: Select a condition a_i' from $\mathcal{A}_i$ and compute the entity ranking L' based on the modified condition where a_i is replaced by a_i'. Keep track of the maximum distance between rankings L and L' and return it to the user.

For billLength, the naive algorithm has to sample from the entire colored triangular area shown in Figure 5. Fortunately, as we prove below, this is not necessary. Ranking distance increases monotonically as one conceptually moves on a line away from the initial condition, i.e., $(0.4, 0.2)$ in the example. (There is some "color noise" in the graph, which are artifacts of the drawing process when interpolating between sample points.) Because of this monotonicity property, it is guaranteed that the greatest distance will be found on the edge of the space of possible conditions, i.e., the three sides of the triangle in Figure 5. Hence instead of naively sampling from the entire triangle, we can use a more **efficient algorithm** that samples only from its sides, which dramatically reduces the search space. Similarly, if the user constrains the space of distributions, e.g., by "cutting off" the top corner of the triangle with constraint $p_2 \leq 0.3$, Merlin only has to sample from the edges of the resulting shape, not from its inner points. The implications are particularly powerful for Boolean attributes. Because ranking distance increases monotonically toward the extremes of the range (as Figure 6 shows for the bird data), to find the condition that results in the greatest ranking difference, one only needs to check the rankings for the lower and upper extreme of the range of possible probability values considered.

Next we will introduce ranking distance measures and prove the monotonicity property that implies that it is sufficient to sample from the edges. We also provide evidence that it is unlikely that a more efficient general algorithm exists by showing that the problem is not convex and that the optimal solution does not necessarily lie in a vertex.

4.2 Ranking Distance Measures

Merlin's sensitivity analyzer can work with any distance measure $\mathrm{dst}()$ between entity rankings. We introduce three important examples and show in Section 4.5 that they all admit an optimization algorithm that is much more efficient than the naive algorithm. The first two correspond to (generalizations of) the two most commonly used ranking correlation measures in information retrieval [2]. Let L_p and L_q be two rankings of the n entities in Y. For each entity $y \in Y$, let $\rho_p(y)$ and $\rho_q(y)$ denote y's rank in the corresponding ranking.

Minkowski distance: The distance of two rankings can be measured based on the difference in rank for each individual entity. For instance, the Spearman rank correlation coefficient, which is widely used for evaluating similarity between rankings in information retrieval, relies on the squared rank differences and is defined as $1 - \frac{6 \sum_{y \in Y} (\rho_p(y) - \rho_q(y))^2}{n(n^2-1)}$. We consider not only squared differences, but the *Minkowski distance* in general. For some constant $d > 0$ it is defined as

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

Kendall's τ distance: Kendall's τ is another commonly used measure of correlation between two rankings. It is defined as $\frac{C-D}{n(n-1)/2}$, where C and D denote the total number of entity pairs that are ranked in the same and opposite order, respectively, in the two rankings. Finding the maximal ranking distance is equivalent to finding the minimal correlation. Since n, the number of different entities, is a constant for a given problem, and $C + D = n(n+1)/2$, Kendall's τ increases linearly with $-D$. Hence we define a ranking distance measure based on Kendall's τ as

$$\mathrm{dst}_\tau(L_p, L_q) = D. \quad (2)$$

Intuitively, it measures the number of inversions between the two rankings.

Average Precision distance: Minkowski distance and Kendall's τ weigh ranking differences equally, no matter

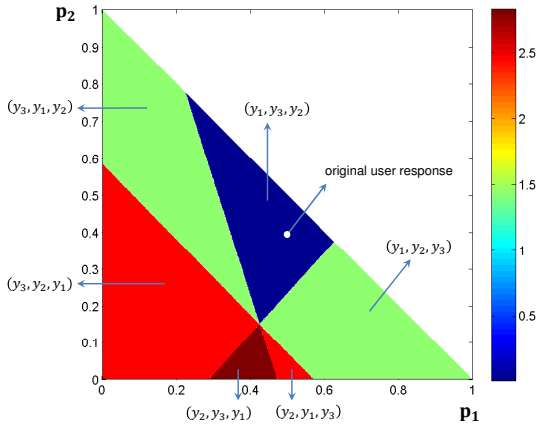

Fig. 7. For each point (p_1, p_2) the color indicates the Minkowski distance (for $d = 2$) between original ranking for condition $(0.5, 0.4)$ and the ranking obtained by changing it to (p_1, p_2). Rankings for each region are indicated.

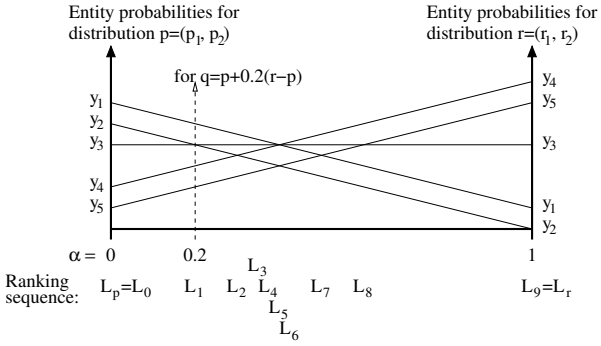

Fig. 8. Entity probabilities for collinear conditions p, q, r

if they occur near the top or near the bottom of the ranking. Recent work in the information retrieval community proposed the Average Precision (AP) correlation coefficient τ_{AP} [2] as a measure that weighs differences near the top higher, because those are the differences that more likely affect a user during Web search. It is based on $C_{p,q}(k)$, which measures for entity y at position k in ranking L_q, how many entities among those *above* it in L_q are also ranked above y in the other ranking L_p. Formally,

$$\tau_{\mathrm{AP}}(L_p, L_q) = \frac{2}{n-1} \sum_{k=2}^{n} \frac{C_{p,q}(k)}{k-1} - 1.$$

Finding the maximal ranking distance is equivalent to finding the minimal correlation, hence we use the negative AP correlation coefficient as a the corresponding distance measure

$$\mathrm{dst}_{\mathrm{AP}}(L_p, L_q) = -\tau_{\mathrm{AP}}(L_p, L_q). \tag{3}$$

4.3 Problem Hardness

We show that sensitivity computation in general is not a convex optimization problem and that the optimal solution might not lie in a vertex.

Convexity: If the set of alternative probability distributions, $\mathcal{A}_i$, is not convex, then the optimization problem is not convex. Even for convex $\mathcal{A}_i$, the objective function is not necessarily convex as shown by a simple counter-example.

Let X be an attribute with domain $\{x_1, x_2, x_3\}$ and let $a_1 = (1, 0, 0)$, $a_2 = (0, 1, 0)$ and $a_3 = (0, 0, 1)$ be probability distributions over this domain. Assume there are three entities y_1, y_2 and y_3 with the following entity probabilities (data D is omitted in the formulas for brevity): $\Pr(Y = y_1|A = a_1) = 0.5$, $\Pr(Y = y_1|A = a_2) = 0.4$, $\Pr(Y = y_1|A = a_3) = 0.1$, $\Pr(Y = y_2|A = a_1) = 0.4$, $\Pr(Y = y_2|A = a_2) = 0.1$, $\Pr(Y = y_2|A = a_3) = 0.4$, $\Pr(Y = y_3|A = a_1) = 0.1$, $\Pr(Y = y_3|A = a_2) = 0.5$, $\Pr(Y = y_3|A = a_3) = 0.5$.

Let the original condition for X be $a = (0.5, 0.4, 0.1)$, and let $\mathcal{A} = \{(p_1, p_2) \,|\, p_1, p_2 \geq 0, p_1 + p_2 \leq 1\}$. (This considers any probability distribution over the domain of X.) We randomly sampled 1.5 million distributions $a' = (p_1, p_2)$ from $\mathcal{A}$ and computed the Minkowski distance, using squared rank differences, of the resulting entity ranking to the ranking for the original distribution a. Figure 7 shows the result for all combinations of p_1 (x-axis) and p_2 (y-axis). (Recall that $p_3 = 1 - p_1 - p_2$.) The ranked list for original condition a is (y_1, y_3, y_2). In the bright red region, the distance is $\sqrt{6}$ and in the dark red region it is $\sqrt{8}$. Because of the discontinuity at the boundary between regions, it is easy to show that the objective function is neither convex nor concave.

Optimum in a vertex: If the maximum ranking distance could be shown to lie in one of the corners of the triangle, only those three rankings would have to be explored when computing sensitivity. Unfortunately, in the example the distance between the original ranking and those obtained in the three corners of the triangle are $\sqrt{2}$, $\sqrt{2}$ and $\sqrt{6}$, respectively. For $a' = (0.4, 0)$ in the dark red region, the ranked list is (y_2, y_3, y_1), resulting in a distance of $\sqrt{8}$ to the original ranking.

This counter-example can also be used to show the same negative results for the other two ranking distance measures, Kendall's τ distance and Average Precision distance. Fortunately, as discussed next, we are able to prove the "next best" structural property: that the optimum always lies on the "edge" of the set of possible conditions $\mathcal{A}$, i.e., the sides of the triangle in the example.

4.4 Collinearity of Probabilities and Impossibility of Repeated Entity Order Swaps

The proof that the optimum lies on the "edge" of the set of possible conditions $\mathcal{A}$ relies on two general properties, stated in Lemmas 2 and 3, which hold independent of the ranking distance measure. (Proofs are provided in the appendix.)

Lemma 2. Let p, r, and q be probability distributions for attribute X, and let the corresponding entity probabilities be $P = \Pr(Y \,|\, A = p, A_1, \ldots, A_k, \bar{y}_1, \ldots, \bar{y}_l, D)$, $Q = \Pr(Y \,|\, A = q, A_1, \ldots, A_k, \bar{y}_1, \ldots, \bar{y}_l, D)$, and $R = \Pr(Y \,|\, A = r, A_1, \ldots, A_k, \bar{y}_1, \ldots, \bar{y}_l, D)$. If $q = p + \alpha \cdot (r - p)$, $0 < \alpha < 1$, then $Q = P + \alpha \cdot (R - P)$.

Intuitively, the lemma states the following: Assume entity y has a probability P based on condition p for attribute X. Assume we also know that this probability will be R if the user changed the condition from p to a different distribution r. Then each alternative condition q that is a

linear combination of p and r will result in a probability Q that is proportionally between P and R. Figure 8 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 that is a linear combination of p and r, each entity's probability is proportionally between the corresponding probabilities for p and r.

Lemma 2 implies strong limitations on how entities can change their relative rankings. As illustrated by entity pairs (y_1, y_2) and (y_4, y_5) in Figure 8, if some entity y is ranked below another entity y' in both L_p (ranking on the left) and L_r (ranking on the right), then the rank order will be same for any L_q "in-between".

Lemma 3. Let p, r, and $q = p + \alpha \cdot (r - p)$, $0 < \alpha < 1$, be collinear conditions for attribute X. And let the entities be ranked by effort-adjusted probability. If entity y is ranked below entity y' in both L_p and L_r, then y is also ranked below y' in L_q.

4.5 Monotonicity of Ranking Distance

We can now prove the following result for sensitivity:

Theorem 1. Let p be the current query condition for attribute X and let $q, r \in \mathcal{A}$ be two alternative conditions considered by the user, such that $q = p + \alpha \cdot (r - p)$ for some $0 < \alpha < 1$. L_p, L_q, and L_r, respectively, denote the rankings obtained for these conditions, while keeping all other conditions unchanged. Then, if entities are ranked based on their probability or effort-adjusted probability (Section 3), it holds that $\mathrm{dst}(L_p, L_q) \leq \mathrm{dst}(L_p, L_r)$. The same property also holds for distance measures dst_τ and $\mathrm{dst}_{\mathrm{AP}}$.

Proof: We first prove the theorem for **Minkowski distance** dst. Consider Figure 8 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. 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 binary swaps between adjacent entities (using the Bubble-sort algorithm). Consider again Figure 8. 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: $y_1 \leftrightarrow y_3$, $y_1 \leftrightarrow y_4$, $y_2 \leftrightarrow y_5$, $y_3 \leftrightarrow y_4$.

Now consider the sequence of distinct 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. Each ranking pair (L_i, L_{i+1}), $0 \leq i < k$, is identical except that two adjacent entities in L_i are swapped in L_{i+1}. We show that this property implies that the Minkowski distance from L_p to L_i cannot be greater than the distance from L_p to L_{i+1}.

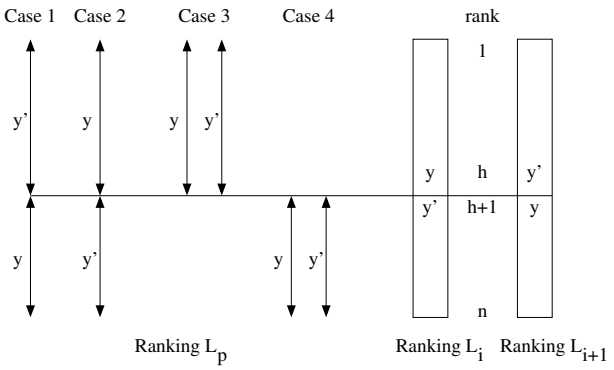

Fig. 9. Possible cases for the ranks of y and y' in L_p, relative to their position in L_i and L_{i+1}.

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 $\rho_i(y) = \rho_i(y') - 1$, $\rho_{i+1}(y) = \rho_{i+1}(y') + 1$, and $\rho_{i+1}(y') = \rho_i(y)$. Now consider all possible cases for their ranking in L_p (see Figure 9):

Case 1: $\rho_p(y) > \rho_i(y)$ and $\rho_p(y') < \rho_i(y')$. This case is impossible, because it implies $\rho_p(y) > \rho_p(y')$, which together with $\rho_i(y) < \rho_i(y')$ and $\rho_{i+1}(y) > \rho_{i+1}(y')$ violates Lemma 3.

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 $\mathrm{dst}(L_p, L_{i+1})$ are terms containing the ranks of y and y'. These terms are $|\rho_p(y) - \rho_i(y)|^d + |\rho_p(y') - \rho_i(y')|^d$ 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 constraint 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 (Lemma 3) 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

$\mathrm{dst}(L_p, L_1) \leq \mathrm{dst}(L_p, L_2) \leq \cdots \leq \mathrm{dst}(L_p, L_k) = \mathrm{dst}(L_p, L_r)$. □

We now prove Theorem 1 for **Kendall's τ distance** and **Average Precision distance**.

Proof: Consider the same rankings L_i and L_{i+1} as used in the proof for the Minkowski distance. Since L_i and L_{i+1} are identical except for the adjacent entities y and y', which swap places, the number of inversions compared to L_0 is also identical, except for the pair (y, y'). Note that y is ranked above y' in L_i, but below y' in L_{i+1}. Lemma 3 implies that in L_0, y must have been ranked above y'. Hence L_{i+1} has one additional inversion than L_i compared to L_0, completing the proof for $\mathrm{dst}_\tau(L_p, L_i) < \mathrm{dst}_\tau(L_p, L_{i+1})$.

For Average Precision distance, consider $\mathrm{dst}_{\mathrm{AP}}(L_0, L_{i+1}) - \mathrm{dst}_{\mathrm{AP}}(L_0, L_i) = \frac{2}{n-1}\sum_{k=2}^{n}(C_{0,i}(k) - C_{0,i+1}(k))/(k-1)$. For simplicity, let $h = \rho_i(y) = \rho_{i+1}(y')$ (and therefore $h+1 = \rho_i(y') = \rho_{i+1}(y)$). Since L_i and L_{i+1} are identical except for the swap of adjacent entities y and y', $C_{0,i}(k) = C_{0,i+1}(k)$ for all $k \neq h, (h+1)$. This implies $\mathrm{dst}_{\mathrm{AP}}(L_0, L_{i+1}) - \mathrm{dst}_{\mathrm{AP}}(L_0, L_i) = \frac{2}{n-1}T$, where

$$T = \frac{C_{0,i}(h)}{h-1} + \frac{C_{0,i}(h+1)}{h} - \frac{C_{0,i+1}(h)}{h-1} - \frac{C_{0,i+1}(h+1)}{h}.$$

To see that T is non-negative, note that $C_{0,i}(h) = C_{0,i+1}(h+1)$, because the entities ranked above y are the same in both L_i and L_{i+1}, except for y', which ranks above y only in L_{i+1}. However, we already showed that in L_0, y has to be ranked above y', therefore y' does not increase $C_{0,i+1}(h+1)$ compared to $C_{0,i}(h)$. A similar analysis shows that $C_{0,i}(h+1) = C_{0,i+1}(h) + 1$, because in L_{i+1}, y' is not ranked above y any more, reducing the number of agreements with the ranking in L_0 by one.

Using these results, and the fact that by definition $C_{0,i+1}(h+1) \leq h$ (there are only h entities total above rank $h+1$), it is easy to show that $T \geq 0$, and therefore $\mathrm{dst}_{\mathrm{AP}}(L_0, L_{i+1}) - \mathrm{dst}_{\mathrm{AP}}(L_0, L_i) \geq 0$. □

5 Recommending Additional Conditions

It is inherently difficult to determine how much any of the remaining attributes, i.e., those for which the user has not specified a condition yet, would improve result quality (left table in Figure 3). The usefulness of an attribute X depends on the condition $a \in A$ the user will enter for it—which is unknown. We address this problem by modeling the unknown future condition as a random variable. Merlin estimates the probability that the user will enter distribution $a \in A$ based on the information provided so far as

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

Merlin then estimates the entity probabilities given the predicted distribution a:

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

This leaves another challenge: Even knowing the likely new entity probabilities is not helpful unless one can determine which of them are the best. Doing so would be trivial if Merlin knew which entities the user is looking for—pick the ones where the entities of interest have the highest probabilities. Without that information, we can only rely on general quality measures that evaluate how well "winning" entities are separated from "losing" ones. More precisely, it is desirable to have a result where the probability is high for a few entities and near-zero for all others: 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 directly captures this intuition. Using p_y as shorthand for $\Pr(Y = y \mid A, A_1, \ldots, A_k, \bar{y}_1, \ldots, \bar{y}_l, D)$, entropy for the set of entities is defined as $-\sum_{y \in Y} p_y \cdot \log_2(p_y)$. In general, entropy is low if there are few high-probability entities and many low-probability ones. It is high if many entities have similar probability. (For this reason entropy is widely used for selection of split attributes in decision trees [3].) The expected improvement in the quality of the entity ranking is then measured as the expected entropy reduction for attribute X. Instead of entropy, one could also use other common measures of "purity" for classification problems, including Gini [3], or measures based on expected user effort [4].

6 Probability Estimation in Realtime

Given user-provided conditions $A_1, A_2, \ldots, A_k$ (which are distributions over the possible values of the corresponding attributes $X_1, \ldots, X_k$) and explicitly rejected entities $y_1, \ldots, y_l$, Merlin needs to estimate the probabilities of entities, $\Pr(Y \mid A_1, \ldots, A_k, \bar{y}_1, \ldots, \bar{y}_l, D)$, and of new conditions the user might specify, $\Pr(A \mid A_1, \ldots, A_k, \bar{y}_1, \ldots, \bar{y}_l, D)$. Formulas of the type $\Pr(C \mid X_1, X_2, \ldots, X_k, D)$ define the *posterior probability* of a class C in Bayesian classification [5]. 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 [6]. Based on this observation, we can in principle leverage almost any classification technique using the following 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 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, then Merlin can use classification techniques for uncertain data [7]. 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.)
- At query time, given user-provided conditions $a_1 \in A_1, \ldots, a_k \in A_k$, sample value x_i from distribution a_i for $i = 1, \ldots, k$. Then compute $M(x_1, x_2, \ldots, x_k)$, i.e., the probability for each entity, by running $(x_1, x_2, \ldots, x_k)$ through model M. Scale the probabilities of all entities $y \in Y - \{y_1, \ldots, y_l\}$ proportionally so that they sum up to 1.

We can similarly train and use a model $M_X(X_1, X_2, \ldots, X_k)$ to predict the user's input for an attribute X for which she has not yet provided a query condition. Given input $(x_1, x_2, \ldots, x_k)$ (i.e., values of some of the other attributes), it returns $\Pr(A \mid A_1, \ldots, A_k, \bar{y}_1, \ldots, \bar{y}_l, D)$, i.e., a probability distribution over possible user inputs for attribute X.

6.1 Challenges

While conceptually straightforward, the problem lies in guaranteeing *interactive* response time. Recall that user-specified conditions are distributions over the values of the corresponding attribute domain. Hence the probabilities of interest are actually expectations, which in practice are computed by sampling from the respective domains. Consider computing the entity probabilities $\Pr(Y \mid A_1, \ldots, A_k, \bar{y}_1, \ldots, \bar{y}_l, D)$ at query time as described above. If s_1 inputs $(x_1, x_2, \ldots, x_k)$ are sampled to compute the average probability of each entity, then total prediction cost is $s_1 \cdot c_M$, where c_M is the time it takes model M to make a single prediction. To recommend additional conditions (Section 5), entity probabilities have to be computed many more times: For each of the $(m-k)$ attributes X without user-specified condition, model M_X returns a distribution over possible user conditions $a \in A$; and for a random sample of s_2 conditions from this output of M_X, Merlin has to compute $\Pr(Y \mid A, A_1, \ldots, A_k, \bar{y}_1, \ldots, \bar{y}_l, D)$ using a model $M'(X, X_1, X_2, \ldots, X_k)$ similar to $M(X_1, X_2, \ldots, X_k)$. Hence the total prediction cost becomes

$$(m-k) \cdot s_1 \cdot s_2 \cdot c_{M'}. \tag{4}$$

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 6.3 how to make it tuneable.

In addition to prediction cost, one also has to consider model training time and storage cost. It often is not feasible to pre-compute and store models like M and M_X for all possible subsets of the set of attributes. On the other hand, training models on-the-fly can significantly increase system response time. (Note that for large data sets, state-of-the-art data mining models usually cannot be trained in a few seconds.) We discuss our solution next.

6.2 Solution: Bagged Tree Ensembles

We propose to use bagged decision tree ensembles [8] 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 [3]. 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 chose bagged trees for several reasons.

(1) Trees can handle any attribute type. Bagged trees are robust against noise and overfitting and have been shown to return excellent probabilities "out-of-the-box" [6]. Trees also can naturally deal with missing values, therefore one can use a model trained for input $(X_1, \ldots, X_m)$ to make predictions for any subset of these attributes. This eliminates the problem of on-the-fly model training or pre-computing a large number of models for different attribute subsets.

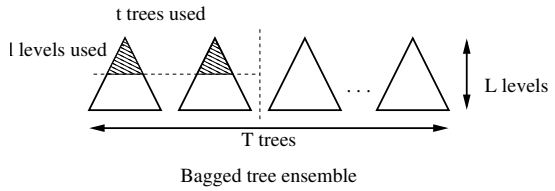

Fig. 10. Adaptive Tree restricted to $t \leq T$ models and $l \leq L$ levels

(2) Due to their structure of splitting on an attribute at a time, trees can compute expectations like $\Pr(Y \mid A_1, \ldots, A_k, \bar{y}_1, \ldots, \bar{y}_l, D)$ in a *single pass*. At a split node whose attribute value is 100% certain, i.e., the user specified a condition where a single value in the domain has probability 1.0, the entire "weight" follows the corresponding branch. For imprecise conditions, the "weight" is partitioned according to user-specified probability distribution and each partial weight is sent down the corresponding branch. If the user has not specified a condition for the split attribute, partial weights are determined by the training data distribution of the attribute 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 input values. Hence there is no need to sample from $A_1, \ldots, A_k$, i.e., s_1 in Eq. 4 is effectively equal to 1.

(3) The comparably simple index-like structure makes tree cost predictable and tuneable, as we discuss next.

6.3 Controlling Tree Response Time

A major advantage of tree-based methods over other popular classification techniques is that their response time is reasonably predictable and tuneable, because it is directly determined by the number of nodes accessed. This is crucial for guaranteeing interactive response time. Let *Full Tree* refer to a bagged ensemble consisting of T individual trees, each with at most L levels. By using only $t \leq T$ of these trees and limiting access to the top $l \leq L$ levels, we can reduce cost approximately proportionally with the total number of nodes accessed. To be able to stop at a non-leaf level, Merlin also stores the corresponding class distribution in each inner node of the trees. Our goal is to set (t, l) so that Merlin responds within a user-specified time. Notice that as the user enters query conditions, t and l might need to be changed adaptively. For instance, a condition assigning zero probability to some values of an attribute's domain effectively prunes the corresponding sub-tree(s), reducing the number of nodes accessed. When that happens, response time drops and 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*.

To determine if it is safe to increase l or t, Merlin needs 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 number of nodes accessed in a newly added limited tree quite accurately as the average of the already used t limited trees. The effect of a level increase is more difficult to estimate,

because the sub-trees in the newly added levels are typically not balanced. 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 Merlin increases l by at most 1 level at a time, it can accurately predict the number of nodes accessed after the 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. Since Merlin might not know how many nodes will be accessed in the corresponding sub-tree, it takes a conservative approach of adding one new level of the sub-tree at a time. It does so by adding a "local" level limit l_c for c, setting it initially to the level of c. This way in the next round, i.e., after the next user interaction, Merlin only accesses root node c of this sub-tree. After accessing c, Merlin knows the number of its children and hence can increase the local level limit to l_c+1. This process continues until the local limit reaches global limit l, at which point it is discarded.

Knowing the number of nodes accessed, Merlin can estimate system response time. The three major computations performed by Merlin in response to user input are (see also Figure 4) ranking of entities, ranking of attributes, and computing sensitivity. Let n_y denote the number of nodes accessed in the Adaptive Tree used to predict the entity probabilities, and let θ denote the average time for accessing a single tree node. Then the time for entity ranking is

$$T_{\text{eRank}} = n_y\theta + u.$$

This formula is obtained as follows. After the user enters new information, Merlin first has to accesses the entity prediction model to compute the probabilities of all entities (time: $n_y\theta$). Then it sorts the entities (time: u), where u is a constant independent of model size.

Let n_a denote the total number of nodes accessed in the Adaptive Trees for predicting future input for unspecified attributes. Then the time for attribute ranking is approximately

$$T_{\text{aRank}} = n_a\theta + (m-k)s_2n_y\theta.$$

This formula is obtained as follows. First, all models for unspecified attributes are accessed to obtain probability distributions for possible conditions the user might input for them (time: $n_a\theta$). For each of these $m-k$ attributes, s_2 conditions are sampled (see Section 6.1), and for each of them the corresponding new entity probabilities are computed (time: $(m-k)s_2n_y\theta$). The formula does not account for the cost of entropy computation (which is negligible compared to computation of the entity probabilities it needs as input) and the cost for sorting of the $m-k$ unspecified attributes (which is negligible compared to predicting the distribution and resulting entity probabilities for each of these attributes). In our current Merlin implementation, entity and attribute ranker are both executed after each user input. Hence a user-specified response-time threshold is applied to the total computation time

$$T = T_{\text{eRank}} + T_{\text{aRank}} + z, \tag{5}$$

where z accounts for the constant, i.e., independent of model size, overhead for all other tasks related to reading the user's response and updating the UI.

Sensitivity analysis is only executed on demand, and it has its own user-specified response time threshold. The time for sensitivity analysis is

$$T_{\text{sens}} = ks_3(v + n_y\theta + u + w) + z.$$

It is obtained as follows. For a specified attribute, an alternative condition is sampled from $\mathcal{A}$ (time: v), followed by the computation of the resulting entity probabilities (time: $n_y\theta$). This is followed by sorting of the entities (time: u) and computing the distance to the original ranking (time: w). This computation is performed for each of the specified k attributes and s_3 samples of alternative conditions for each of them. Both v and w, like u, are constant as they do not depend on the model.

All variables in the above formulas can be measured as Merlin interacts with the user. To estimate the response time for a different model parameter combination (l', t'), Merlin simply plugs in the corresponding new node counts n_y and n_a, estimated as discussed above. It can choose any combination (l', t') for which the estimated computation time is below the corresponding response-time limit. Our experiments show that we indeed can successfully guarantee interactive response times.

7 Experiments

The goal of the experiments is to provide a proof of concept for three important properties of Merlin: (1) improved efficiency of sensitivity analysis due to Theorem 1, (2) quality of the recommendations for additional conditions, and (3) guarantee of interactive system response times. In all experiments, the ranker for Y uses entity probabilities and the ranker for A uses entropy (see Figure 4). All algorithms were coded using Java. Our experiments were conducted on a low-end dual-core 2.13GHz Intel PC running Linux with 4GB of RAM and a 500GB IDE HDD.

7.1 Data and Models

Merlin is evaluated on a data set provided by the Cornell Lab of Ornithology. It reports sightings of 367 different bird species in North America, each described by 164 categorical attributes encoding multi-valued attributes such as beak length, bird size and shape, and Boolean attributes describing color and color pattern for different body parts. Each record reports an individual bird observation generated as follows. First, an observation record is randomly selected from the *real* eBird data set [9], containing actual observations reported by citizen scientists. While eBird lists the species of the observed bird, it does not contain individual bird properties such as color and size. Individual bird features were added to the eBird record by sampling from a feature distribution that was carefully defined by the domain experts for every single species. Notice that we removed location and time information from the records. (Based on location and time of observation, many bird species can be eliminated from consideration. Merlin discovers this automatically, recommending spatio-temporal attributes to

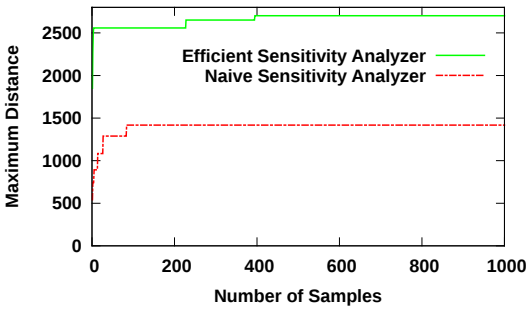

Fig. 11. Naive vs. efficient sensitivity estimation for a 9-valued attribute

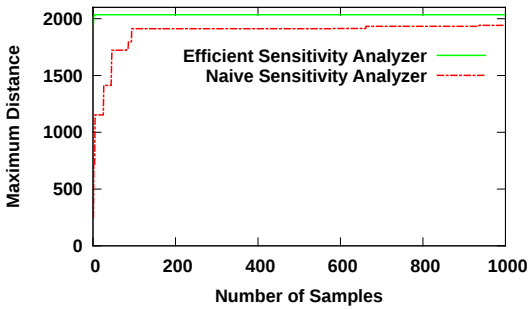

Fig. 12. Naive vs. efficient sensitivity estimation for a Boolean attribute

the user first. While this is desired in practice and leads to quicker identification, we wanted to make the problem more challenging to show that Merlin works well even when the most important attributes for bird identification are removed.) We sampled 3 million records, using 2/3 of them for model training and the other 1/3 for testing.

The models used in all experiments are bagged decision tree ensembles consisting of 100 individual trees, each grown so that nodes with fewer than 500 records are not split any further (i.e., they are leaves). *Full Tree* and *Adaptive Tree* refer to the complete bagged model and the proposed model with adaptive response time, respectively. Since there is a tradeoff between response time threshold and result quality for the Adaptive Tree, we study two cases: one with a generous 5 sec for the system to respond (A5) and the other with a tighter 1 sec system response time limit (A1). Because of these thresholds, A5 starts out with 18 (out of 100) individual trees, each restricted to 5 levels. A1 starts out with a smaller model, consisting of 12 trees and 3 levels.

In all experiments, user input is generated by selecting a record from the test data. (Recall that these records are generated from actual user-reported observations.) For a *precise* condition, the user provides the corresponding value from the test record with probability mass 1. For an *imprecise* condition, only 75% of the probability mass goes to the true value and the rest is divided between other values in the attribute's domain. The *Imprecision* parameter controls how many attributes have an imprecise condition. Setting Imprecision=p implies that with probability p the user provides an imprecise condition for an attribute. This decision is made independently for each attribute.

7.2 Sensitivity Analysis

We compare our algorithm that exploits Theorem 1 to the naive algorithm. The set of alternative conditions considered, $\mathcal{A}$, is set to the entire space of possible probability distributions for the attribute examined. Conditions on all attributes are imprecise, except the original condition for the attribute of interest is the uniform distribution to model a user completely unsure about the true value. We sample distributions randomly from $\mathcal{A}$. The naive algorithm uses the sampled values, while our proposed algorithm exploits Theorem 1 and instead uses the corresponding point on the edge, obtained where the ray connecting the original point to the sampled point intersects with the edge of $\mathcal{A}$. For computing the distance between rankings, we used Minkowski distance with $d = 2$. Figure 11 shows the maximum distance between the original ranking and the rankings obtained based on the sampled conditions for bird size, which has 9 values. After sampling only 4 points, the efficient algorithm discovered a distance within 5% of the final maximum distance it finds, while the naive one does not come close, even after 1000 samples. We observed similar results for the other attributes.

Figure 12 shows the same experiment for a Boolean attribute. Since the efficient algorithm reduces the infinite sample space to the two extreme points, it guarantees to find the maximum distance with two samples. The naive algorithm might be lucky to quickly sample a good point close to the boundary or might need to explore many points to get close to that maximum value.

7.3 Additional Condition Recommendation

We explore if Merlin can recommend attributes that lead to a quick result improvement (Section 5). The experiments compare the proposed Adaptive Tree algorithm (versions A5 and A1) against the Full Tree (F), which represents the "gold standard" in terms of producing calibrated probabilities without regard for computation cost. For each attribute, we set the number of samples s_2 to the attribute's domain size (see Section 6.1).

Table 2 shows a representative result for a random test record, reporting the observation of a less common, and hence difficult to identify, species initially ranked near position 100. Each row shows the rank and estimated probability of this species as a new condition is added for the k-th attribute selected by the corresponding algorithm.

All three tables show that Merlin is effective in recommending attributes that result in a quick improvement of the species' rank (without it knowing the correct result, of course!). As expected, the tighter response time constraint for A1 leads to lower-quality probability estimates, providing faster responses but slightly poorer results. Comparing the tables from left to right, it is also obvious that a larger fraction of imprecise conditions causes some result degradation. Still, even for Imprecision=1, the correct species quickly approaches the top-10. Note that the low probability of the target species for Imprecision=1 is not due to shortcomings of Merlin, but inherent to the problem. There are only few observations for the species, compared to those species initially at the top of the list. Since an imprecise condition gives 0.25 probability mass to the wrong attribute values,

TABLE 2
Comparison of Adaptive Tree (A5 and A1) and Full Tree (F). The first row shows the initial rank and probability of the species of interest. The next rows show its state after the user added a condition for the attribute recommended by the corresponding method.

Imprecision = 0						
k	$\rho(y)$			$\Pr(Y \mid A_1, \ldots, A_k, D)$		
	F	A5	A1	F	A5	A1
0	100	100	102	0.002	0.002	0.002
1	20	20	20	0.013	0.013	0.013
2	9	9	9	0.032	0.032	0.013
3	5	5	9	0.053	0.053	0.032
4	3	3	5	0.095	0.095	0.053
5	2	2	3	0.151	0.151	0.096
6	2	2	2	0.209	0.206	0.153
7	2	2	2	0.258	0.259	0.152
8	1	1	2	0.931	0.933	0.211
9	1	1	2	0.931	0.930	0.259
10	1	1	1	0.931	0.931	0.920

Imprecision = 0.5						
k	$\rho(y)$			$\Pr(Y \mid A_1, \ldots, A_k, D)$		
	F	A5	A1	F	A5	A1
0	100	100	102	0.002	0.002	0.002
1	20	20	20	0.013	0.013	0.013
2	9	9	9	0.024	0.024	0.024
3	6	6	6	0.030	0.030	0.030
4	3	3	3	0.053	0.054	0.054
5	3	3	3	0.074	0.073	0.054
6	2	2	2	0.117	0.118	0.086
7	2	2	2	0.145	0.146	0.122
8	1	1	2	0.415	0.422	0.146
9	1	1	2	0.415	0.422	0.146
10	1	1	2	0.413	0.423	0.146

Imprecision = 1						
k	$\rho(y)$			$\Pr(Y \mid A_1, \ldots, A_k, D)$		
	F	A5	A1	F	A5	A1
0	100	100	102	0.002	0.002	0.002
1	23	23	24	0.010	0.010	0.010
2	12	12	12	0.019	0.019	0.018
3	6	6	12	0.023	0.024	0.018
4	3	7	12	0.032	0.024	0.018
5	3	7	12	0.038	0.025	0.018
6	3	7	11	0.038	0.025	0.018
7	3	7	11	0.037	0.025	0.018
8	3	7	11	0.037	0.025	0.018
9	3	7	11	0.044	0.025	0.018
10	3	7	11	0.045	0.025	0.018

this means that other species with those properties receive a significant probability mass. To explore this further, we ran another experiment where we specified *all* 164 attributes with imprecise responses. The maximum probability that the correct species ever reached using the Full tree was just 0.11.

We also compared to a fast *Random* approach, which simply recommends a random attribute. Even after specifying precise conditions for 10 attributes, Random only achieved rank and probability of 67 and 0.002, respectively. For imprecise responses, Random showed no significant improvement in rank and probabilities. This shows that even a simple heuristic like entropy, which tries to select attributes that separate "winning" from "losing" species, is worth the extra computation time to help the user identify attributes that narrow the search.

7.4 Interactive Response Time

We explore if Merlin can indeed guarantee interactive response time. As before, the response time threshold for the Adaptive Tree is set to 5 and 1 sec for Eq. 5. Figures 13, 14, and 15 show system response time as the user adds more conditions. System response time is measured from the time the user submitted the new condition until the system responded.

Figure 13 shows results for a run where the user always provides precise conditions. With the Full Tree, response time initially is 544 sec, well above the threshold. (The plots cut off at 15 sec for readability.) As more conditions are added, response time decreases 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 threshold. As user input results in pruned tree branches, it automatically adapts the number of trees and tree levels used. In this experiment, the first attribute specified is bird size with a domain size of 9. With a precise response, about $8/9$ of the tree is pruned and therefore, the time drops to about $1/9$ of the threshold. In the next round, the tree is grown to an appropriate new size, so that its response time is just below the threshold again. The Adaptive Tree ultimately converges to the Full Tree size.

The basic picture is the same in Figures 14 and 15, for scenarios with Imprecision set to 0.5 and 1, respectively. Comparing the different Full Tree curves, it is evident that response time improves more slowly for imprecise conditions. This is due to the fact that an imprecise condition does not result in pruning of tree branches as each child branch receives a non-zero weight. For Imprecision=1, no branches in the tree can be pruned. Hence Full Tree response time drops only due to the decreasing number of remaining attributes considered for future conditions.

8 Related Work

Interactive exploratory search requires an acceptable response time which is not a universal constant, but depends on the user's inquiry, preferences, and environment [10]. To the best of our knowledge, Merlin is the only system for imprecise queries where the user can set system response time according to her preferences, gracefully trading accuracy of estimated probabilities for faster response time. BlinkDB [11] takes a different approach toward interactive query exploration in parallel systems. It trades off result accuracy for faster response time by running a (precise) query on an appropriate data sample.

Query steering [12] is a general framework for interactive data exploration. Merlin proposes concrete solutions for the interactive performance (Section 6) and navigation help (Section 5) components of this framework. Previous work on query steering and interactive query refinement requires the user to provide positive or negative examples of desired results. For example, in the AIDE data exploration framework [13] the user marks objects as relevant and irrelevant. The query refinement framework of Islam et al. [14] requires the user to specify missing and undesirable result tuples. And Abouzied et al. [15] propose techniques for learning quantified Boolean queries by asking users to label data objects as answers or non-answers. Given a set of missing result tuples, ConQueR [16] automatically modifies an SQL query so that it contains both the original result and the missing tuples, minimizing addition of other tuples. The crucial assumption underlying all these approaches is that the user can explicitly state desired result tuples. Hence they are not applicable to Merlin's target applications where the user cannot provide the results, but is searching for them by specifying (possibly imprecise) conditions.

Relaxation was explored for queries that return an empty result. Mottin et al. [17] propose an interactive approach where the user is guided towards a desirable relaxation.

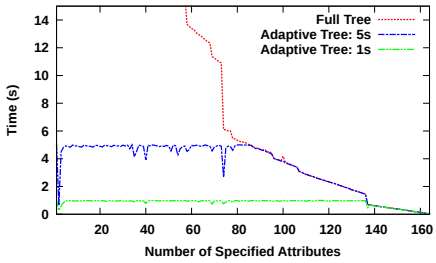

Fig. 13. Response time: Imprecision = 0

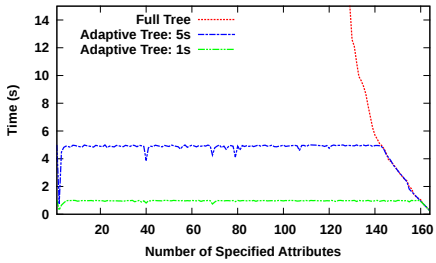

Fig. 14. Response time: Imprecision = 0.5

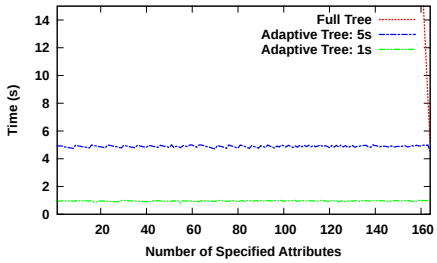

Fig. 15. Response time: Imprecision = 1

Similarly, Junker [18] explores the space of possible relaxations for interactive applications based on user preferences. For Select-Project-Join queries, the Stretch'n'Shrink technique [19] interactively finds relaxations and contractions based on user feedback in order for the query to reach a target output cardinality. All these approaches are concerned with navigating a huge space of possible query relaxations or contractions based on user preferences about which conditions are changeable. This is orthogonal to our work, where the goal is not to automatically modify a condition, but to efficiently process imprecise conditions and analyze their impact on the result.

The general idea of selecting good questions to efficiently extract information from a user has appeared in various contexts. The 20Q game [20] uses a proprietary algorithm to select good questions in order to guess an object the user is thinking of. Visipedia [21] determines the category of an image by posing questions to the user based on visual properties. A similar problem of selecting questions and their order was also explored in the context of questionnaire design [22]. The notion of revealing information at a cost has also been explored in active learning [23]. 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 such that a model trained on the resulting labeled data set has the highest accuracy. 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 of correctly classifying a given input [24], [25]. These approaches can in principle be incorporated into the Merlin framework for ranking of unspecified attributes. However, none of them considers tuneable response time. The general idea of reducing user-effort in interactive systems also appeared in other contexts, e.g., ontology matching [26] and constraint-based query systems for pattern discovery [27]. For human-assisted graph search, Parameswaran et al. [28] explore how to select an optimal set of graph nodes to minimize the number of reachability questions a human expert has to answer.

While also helpful for composing a query, query suggestion and auto-completion are orthogonal to our problem. They rely on repositories of historic queries to predict what query a user might be interested in [29], [30], even in the presence of typing errors [31].

Research on probabilistic databases [1] concentrated on dealing with imprecise *data*, while Merlin's design is centered around imprecise *queries*. The theory community explored search in the presence of errors, which considers correct and incorrect, but not imprecise user input [32]. Previous work on imprecise queries focused on finding result tuples "similar to" a desired one [33]. AIMQ [34] handles an imprecise query by turning it into a set of precise (equality) queries, then finds the most relevant tuples in a neighborhood based on provided or inferred distance measures. This notion of query imprecision is fundamentally different from ours. Our goal is to find the *precise* result the user is looking for. In the bird example, the user wants to identify the actual species, not others "like it". Furthermore, our framework is designed to let the user express uncertainty of the type "most likely the bird had blue on the wing, but there is a small chance it did not." These types of statements are easy to express with probability distributions, but cannot be captured by the notion of "similar" results, because it does not make sense to consider having wing color blue to be similar to not having wing color blue. However, it might be interesting to combine the two approaches so that the user can start with imprecise conditions (using Merlin), and then, after finding a match, expand the search to similar entities to double-check for possible mis-identification.

Agrawal et al. [35] are among the first to explore ranking of results for database queries; recent work includes [36]. Since all these approaches deal with precise queries, a tuple is either in the query result or not. Hence ranking is based on additional properties, e.g., frequency of attribute values or global importance of unspecified attributes. Uncertainty arises not from the query, but for example when the importance of an unspecified attribute is estimated based on specified ones. In Merlin, imprecise queries lead to a natural notion of ranking based on probability distributions of specified attributes. In addition to result ranking, for databases with many attributes the notion of attribute ranking was studied [37]. Attribute ranking could be added to Merlin for result presentation.

9 Conclusions

To gain wider acceptance for big data analysis, databases have to support a broad spectrum of users in finding the information they are looking for. We proposed Merlin for applications where the user is unsure about query conditions, allowing her to explicitly express the uncertainty through probabilities. Since dealing with probabilities and expectations is computationally expensive, Merlin can trade result accuracy for faster response time based on a user-controlled real-time threshold. To help the user evaluate the risk of specifying an imprecise condition, Merlin provides a

novel notion of sensitivity and a fast algorithm for estimating it.

As a next step, we plan to explore other approaches for fast and accurate probability estimation. It would also be interesting to explore suitable interfaces for users to express their uncertainty, and how to turn this input into the probability distributions Merlin is designed to work with.

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] E. Yilmaz, J. A. Aslam, and S. Robertson, "A new rank correlation coefficient for information retrieval," in *Proc. ACM SIGIR*, 2008, pp. 587–594.
[3] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, 3rd ed. Morgan Kaufmann, 2011.
[4] B. Qarabaqi and M. Riedewald, "User-driven refinement of imprecise queries," in *Proc. ICDE*, 2014, pp. 916–927.
[5] T. Mitchell, *Machine Learning*. McGraw-Hill, 1997.
[6] A. Niculescu-Mizil and R. Caruana, "Predicting good probabilities with supervised learning," in *Proc. ICML*, 2005, pp. 625–632.
[7] 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.
[8] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, pp. 123–140, 1996.
[9] Cornell Lab of Ornthology and Audubon, "eBird project," ebird.org.
[10] R. B. Miller, "Response time in man-computer conversational transactions," in *Proc. Fall Joint Computer Conference, Part I*, 1968, pp. 267–277.
[11] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica, "BlinkDB: Queries with bounded errors and bounded response times on very large data," in *ACM European Conference on Computer Systems (EuroSys)*, 2013, pp. 29–42.
[12] U. Cetintemel, M. Cherniack, J. DeBrabant, Y. Diao, K. Dimitriadou, A. Kalinin, O. Papaemmanouil, and S. B. Zdonik, "Query steering for interactive data exploration," in *Proc. CIDR*, 2013.
[13] K. Dimitriadou, O. Papaemmanouil, and Y. Diao, "Explore-by-example: An automatic query steering framework for interactive data exploration," in *Proc. ACM SIGMOD*, 2014, pp. 517–528.
[14] M. S. Islam, C. Liu, and R. Zhou, "A framework for query refinement with user feedback," *J. Syst. Softw.*, vol. 86, no. 6, pp. 1580–1595, 2013.
[15] 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.
[16] Q. T. Tran and C.-Y. Chan, "How to ConQueR why-not questions," in *Proc. ACM SIGMOD*, 2010, pp. 15–26.
[17] D. Mottin, A. Marascu, S. B. Roy, G. Das, T. Palpanas, and Y. Velegrakis, "A probabilistic optimization framework for the empty-answer problem," *Proc. VLDB Endow.*, vol. 6, no. 14, pp. 1762–1773, Sep. 2013.
[18] U. Junker, "QUICKXPLAIN: Preferred explanations and relaxations for over-constrained problems," in *Proc. AAAI*, 2004, pp. 167–172.
[19] C. Mishra and N. Koudas, "Interactive query refinement," in *Proc. EDBT*, 2009, pp. 862–873.
[20] R. Burgener, "Artificial neural network guessing method and game," US Patent Application US 2006/0230008 A1, 2006, online game at www.20q.net.
[21] S. Branson *et al.*, "Visual recognition with humans in the loop," in *Proc. Eur. Conf. on Computer Vision (ECCV)*, 2010.
[22] 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.
[23] B. Settles, "Active learning literature survey," Univ. of Wisconsin–Madison, Tech. Rep. 1648, 2009.
[24] R. Greiner, A. J. Grove, and D. Roth, "Learning cost-sensitive active classifiers," *Artif. Intell.*, vol. 139, no. 2, pp. 137–174, 2002.
[25] S. Ji and L. Carin, "Cost-sensitive feature acquisition and classification," *Pattern Recognition*, vol. 40, no. 5, pp. 1474–1485, 2007.
[26] I. F. Cruz, C. Stroe, and M. Palmonari, "Interactive user feedback in ontology matching using signature vectors," in *Proc. ICDE*, 2012, pp. 1321–1324.
[27] F. Bonchi, F. Giannotti, C. Lucchese, S. Orlando, R. Perego, and R. Trasarti, "On interactive pattern mining from relational databases," in *Proc. Int. Conf. on Knowledge Discovery in Inductive Databases (KDID)*, 2007, pp. 42–62.
[28] 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.
[29] Z. Bar-Yossef and N. Kraus, "Context-sensitive query auto-completion," in *Proc. WWW*, 2011, pp. 107–116.
[30] H. Cao, D. Jiang, J. Pei, Q. He, Z. Liao, E. Chen, and H. Li, "Context-aware query suggestion by mining click-through and session data," in *Proc. SIGKDD*, 2008, pp. 875–883.
[31] S. Chaudhuri and R. Kaushik, "Extending autocompletion to tolerate errors," in *Proc. ACM SIGMOD*, 2009, pp. 707–718.
[32] J. A. Aslam and A. Dhagat, "Searching in the presence of linearly bounded errors," in *Proc. STOC*, 1991, pp. 486–493.
[33] A. Motro, "VAGUE: A user interface to relational databases that permits vague queries," *ACM Trans. Inf. Syst.*, vol. 6, no. 3, pp. 187–214, 1988.
[34] U. Nambiar and S. Kambhampati, "Answering imprecise queries over autonomous web databases," in *Proc. ICDE*, 2006, pp. 45–54.
[35] S. Agrawal and S. Chaudhuri, "Automated ranking of database query results," in *Proc. CIDR*, 2003, pp. 888–899.
[36] S. Chaudhuri, G. Das, V. Hristidis, and G. Weikum, "Probabilistic information retrieval approach for ranking of database query results," *ACM Transactions on Database Systems (TODS*, vol. 31, no. 3, pp. 1134–1168, 2006.
[37] G. Das, V. Hristidis, N. Kapoor, and S. Sudarshan, "Ordering the attributes of query results," in *Proc. ACM SIGMOD*, 2006, pp. 395–406.

Bahar Qarabaqi received the PhD in Computer Science from the College of Computer and Information Science at Northeastern University in Boston, MA, USA. Her research interests include exploratory search in large databases and efficient analysis of massive observational data.

Mirek Riedewald received the PhD in Computer Science from the University of California, Santa Barbara. He is an Associate Professor in the College of Computer and Information Science at Northeastern University in Boston, MA, USA. His research interests include databases and data mining, with an emphasis on designing scalable parallel techniques for data-driven science. He has authored more than 35 articles published in peer-reviewed conferences and journals. He is a member of IEEE and ACM.

APPENDIX

PROOF OF LEMMA 1

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

$$\sum_{i=1}^{n} \Pr(Y = y_i \mid A_1, \ldots, A_k, \bar{y}_1, \ldots, \bar{y}_l, D) \cdot \sum_{j=1}^{i} \phi_j. \quad (6)$$

Assume for contradiction that ranking $y_1, y_2, \ldots, y_n$ minimizes expected user effort, but entities are *not* ranked in decreasing order of their effort-adjusted probabilities. Then there exists a pair of adjacent (in the ranking) entities (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 $\Pr(Y = y_i \mid A_1, \ldots, A_k, \bar{y}_1, \ldots, \bar{y}_l, D)$.) We show that swapping their positions will decrease expected effort, contradicting the assumption of minimality.

It is easy to see that swapping y_k and y_{k+1} in Eq. 6 changes expected user effort by $P_k\phi_{k+1} - P_{k+1}\phi_k$. This difference is negative because of $P_k/\phi_k < P_{k+1}/\phi_{k+1}$. Hence the initial ranking $y_1, y_2, \ldots, y_n$ did not minimize expected user effort, completing the proof of Lemma 1 by contradiction. □

PROOF OF LEMMA 2

Lemma. Let p, r, and q be conditions for attribute X, and let the corresponding entity probabilities obtained based on these conditions be $P = \Pr(Y \mid A = p, A_1, \ldots, A_k, \bar{y}_1, \ldots, \bar{y}_l, D)$, $Q = \Pr(Y \mid A = q, A_1, \ldots, A_k, \bar{y}_1, \ldots, \bar{y}_l, D)$, and $R = \Pr(Y \mid A = r, A_1, \ldots, A_k, \bar{y}_1, \ldots, \bar{y}_l, D)$. If $q = p + \alpha \cdot (r - p)$, $0 < \alpha < 1$, then $Q = P + \alpha \cdot (R - P)$.

Proof: For simplicity and without loss of generality, collinearity will be shown for an attribute X with three possible values x_1, x_2, and x_3. Hence a condition on X is a probability distribution $p \in \{(p_1, p_2, p_3) \mid p_1, p_2, p_3 \geq 0, p_1 + p_2 + p_3 = 1\}$. Since the third probability is determined by the other two (all have to add up to 1), we only need to consider a probability vector (p_1, p_2).

Consider three different conditions for attribute X, expressed as probability vectors $p = (p_1, p_2)$, $q = (q_1, q_2)$, and $r = (r_1, r_2)$. Let the three vectors satisfy $q = p + \alpha \cdot (r - p)$ for some $0 < \alpha < 1$, i.e., p, q, and r are **collinear conditions** for attribute X. (Intuitively, q lies on the line connecting p and r.) We show that the corresponding entity probabilities

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

have to be collinear as well, i.e., satisfy $Q = P + \alpha \cdot (R - P)$.

To see this, recall that each condition $a_i \in A_i$ 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

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

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

$$P = p_1 \cdot g(x_1) + p_2 \cdot g(x_2) + (1 - p_1 - p_2) \cdot g(x_3) \quad (7)$$

$$Q = q_1 \cdot g(x_1) + q_2 \cdot g(x_2) + (1 - q_1 - q_2) \cdot g(x_3) \quad (8)$$

$$R = r_1 \cdot g(x_1) + r_2 \cdot g(x_2) + (1 - r_1 - r_2) \cdot g(x_3) \quad (9)$$

Since $q = p + \alpha \cdot (r - p)$, we can derive from Equation 8

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

Together with Equations 7 and 9, we then obtain the desired result that $Q = P + \alpha \cdot (R - P)$. □

PROOF OF LEMMA 3

Lemma. Let p, r, and $q = p + \alpha \cdot (r - p)$, $0 < \alpha < 1$, be collinear conditions for attribute X. And let the entities be ranked by effort-adjusted probability. If entity y is ranked below entity y' in both L_p and L_r, then y is also ranked below y' in L_q.

Proof: We use ϕ and ϕ' to denote the user effort for y and y', respectively (see Section 3). As before, let P, Q, and R denote the probability of entity y in L_p, L_q, and L_r, respectively. We similarly define P', Q', and R' for y'.

As shown above, the collinearity of p, q, and r implies collinearity of P, Q, and R (similarly for P', Q', and R'). More precisely, $Q = P + \alpha \cdot (R - P) = (1 - \alpha)P + \alpha R$ and similarly $Q' = (1 - \alpha)P' + \alpha R'$. This implies $Q/\phi - Q'/\phi' = (1 - \alpha)(P/\phi - P'/\phi') + \alpha(R/\phi - R'/\phi')$. This difference is negative, because y' is ranked above y in both L_p and L_r and therefore $P/\phi < P'/\phi'$ and $R/\phi < R'/\phi'$. This in turn implies that y' is ranked above y in L_q as well. □