# Query Aware Determinization of Uncertain Objects

Jie Xu, Dmitri V. Kalashnikov, and Sharad Mehrotra, *Member, IEEE*

**Abstract**—This paper considers the problem of determinizing probabilistic data to enable such data to be stored in legacy systems that accept only deterministic input. Probabilistic data may be generated by automated data analysis/enrichment techniques such as entity resolution, information extraction, and speech processing. The legacy system may correspond to pre-existing web applications such as Flickr, Picasa, etc. The goal is to generate a deterministic representation of probabilistic data that optimizes the quality of the end-application built on deterministic data. We explore such a determinization problem in the context of two different data processing tasks – triggers and selection queries. We show that approaches such as thresholding or top-1 selection traditionally used for determinization lead to suboptimal performance for such applications. Instead, we develop a query-aware strategy and show its advantages over existing solutions through a comprehensive empirical evaluation over real and synthetic datasets.

**Index Terms**—Determinzation, uncertain data, data quality, query workload, branch and bound algorithm.

## 1 Introduction

With the advent of cloud computing and the proliferation of web-based applications, users often store their data in various existing web applications. Often, user data is generated automatically through a variety of signal processing, data analysis/enrichment techniques before being stored in the web applications. For example, modern cameras support vision analysis to generate tags such as indoors/outdoors, scenery, landscape/portrait, etc. Modern photo cameras often have microphones for users to speak out a descriptive sentence which is then processed by a speech recognizer to generate a set of tags to be associated with the photo [1]. The photo (along with the set of tags) can be streamed in real-time using wireless connectivity to Web applications such as Flickr.

Pushing such data into web applications introduces a challenge since such automatically generated content is often ambiguous and may result in objects with probabilistic attributes. For instance, vision analysis may result in tags with probabilities [2], [3], and, likewise, automatic speech recognizer (ASR) may produce an N-best list or a confusion network of utterances [1], [4]. Such probabilistic data must be "determinized" before being stored in legacy web applications. We refer to the problem of mapping probabilistic data into the corresponding deterministic representation as the *determinization* problem.

Many approaches to the *determinization* problem can be designed. Two basic strategies are the `Top-1` and `All` techniques, wherein we choose the most probable value / all the possible values of the attribute with non-zero probability, respectively. For instance, a speech recognition system that generates a single answer/tag for each utterance can be viewed as using a top-1 strategy. Another strategy might be to choose a threshold $\tau$ and include all the attribute values with a probability higher than $\tau$. However, such approaches being agnostic to the end-application often lead to suboptimal results as we will see later. A better approach is to design customized determinization strategies that select a determinized representation which optimizes the quality of the end-application.

Consider, for instance, an end application that supports triggers/alerts on automatically generated content. Examples of such an end-application includes publish/subscibe systems such as Google Alert, wherein users specify their subscriptions in the form of keywords (e.g., "California earthquake") and predicates over metadata (e.g., data type is video). Google Alert forwards all matching data items to user based on the subscriptions. Now consider a video about California Earthquake that is to be published on Youtube. The video contains a set of tags that were extracted using either automated vision processing and/or information extraction techniques applied over transcribed speech. Such automated tools may produce tags with probabilities (e.g, "California": 0.9, "earthquake": 0.6, "election": 0.7), while the true tags of the video could be "California" and "earthquake". The determinization process should associate the video with appropriate tags such that subscribers who are really interested in the video (i.e., whose subscription includes the words "California Earthquake" ) are notified while other are not overwhelmed by irrelevant data. Thus, in the example above, the determinization process should minimize metrics such as false positives and false negatives that result from a determinized representation of data.

Now consider a different application such as Flickr, to which photos are uploaded automatically from cameras along with tags that may be generated based on speech annotation or image analysis. Flickr supports effective retrieval based on photo tags. In such an application, users may be interested in choosing determinized representation that optimizes set-based quality metrics such as F-measure instead of minimizing false positives/negatives.

In this paper, we study the problem of deteminizing datasets with probabilistic attributes (possibly generated by automated

• *The authors are with the Department of Computer Science, Bren School of Information and Computer Science, University of California at Irvine, Irvine, CA 92617. E-mail: jiex, dvk, sharad@ics.uci.edu.*

data analyses/enrichment). Our approach exploits a workload of triggers/queries to choose the "best" deterministic representation for two types of applications – one, that supports triggers on generated content and another that supports effective retrieval.

Interestingly, the problem of determinization has not been explored extensively in the past. The most related research efforts are [5], [6], which explore how to give deterministic answers to a query (e.g. conjunctive selection query [5]) over probabilisitc database. Unlike the problem of determinizing an answer to a query, our goal is to determinize the data to enable it to be stored in legacy deterministic databases such that the determinized representation optimizes the expected performance of queries in the future. Solutions in [5], [6] can not be straightforwardly applied to such a determinization problem.[1]

Overall, the main contributions of this paper are:

- We introduce the problem of *determinizing* probabilistic data. Given a workload of triggers/queries, the main challenge is to find the deterministic representation of the data which would optimize certain quality metrics of the answer to these triggers/queries (Section 2).
- We propose a framework that solves the problem of determinization by minimizing the expected cost of the answer to queries. We develop a branch-and-bound algorithm that finds an approximate near-optimal solution to the resulting NP-hard problem (Section 3).
- We address the problem of determinizing a collection of objects to optimize set-based quality metrics, such as F-measure. We develop an efficient algorithm that reaches near-optimal quality (Section 4)
- We extend the solutions to handle a data model where mutual exclusion exists among tags. We show that correlations among tags can be leveraged in our solutions to get better results. We also demonstrate that our solutions are designed to handle various types of queries. (Section 5).
- We empirically demonstrate that the proposed algorithms are very efficient and reach high-quality results that are very close to those of the optimal solution. We also demonstrate that they are robust to small changes in the original query workload (Section 6).

## 2 Preliminary

Before we formally define the *determinization* problem, let us first introduce some important notation. The notation is summarized in Table 1. We will also illustrate it using the running example shown in Fig. 1.

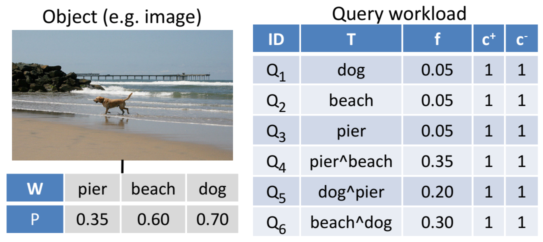
### 2.1 Notations

**Object.** An uncertain object $O$ is annotated with probabilistic attributes. It could be, for instance, an output of an entity resolution algorithm, or an image annotated by probabilistic

1. We will show that in terms of quality of query answer, our proposed determinization techniques are very close to those applied only over probabilistic databases, which are not supported by many legacy applications.

| Notation | Meaning |
|---|---|
| $O$ | Uncertain object |
| $W = \{w_1, w_2, \cdots, w_{\lvert W\rvert}\}$ | Set of uncertain tags for $O$ |
| $P = \{p_1, p_2, \cdots, p_{\lvert W\rvert}\}$ | Probability that $w_i$ is in the ground truth |
| $\mathcal{Q} = \{Q_1, Q_2, \cdots, Q_{\lvert \mathcal{Q}\rvert}\}$ | Query workload |
| $T_Q = \{q_1, q_2, \ldots, q_{\lvert T\rvert}\}$ | Set of query terms of query $Q$ |
| $f_Q$ | Weight, or frequency, of query $Q$ |
| $c_Q^+$ | Cost of a false positive for query $Q$ |
| $c_Q^-$ | Cost of a false negative for query $Q$ |
| $G_Q$ | Set of objects that satisfy $Q$ based on the ground truth |
| $A_Q$ | Set of objects that satisfy $Q$ according to the algorithm |

TABLE 1
Notations


Object (e.g. image)
Query workload

| W | pier | beach | dog |
|---|---|---|---|
| P | 0.35 | 0.60 | 0.70 |

| ID | T | f | c+ | c- |
|---|---|---|---|---|
| $Q_1$ | dog | 0.05 | 1 | 1 |
| $Q_2$ | beach | 0.05 | 1 | 1 |
| $Q_3$ | pier | 0.05 | 1 | 1 |
| $Q_4$ | pier^beach | 0.35 | 1 | 1 |
| $Q_5$ | dog^pier | 0.20 | 1 | 1 |
| $Q_6$ | beach^dog | 0.30 | 1 | 1 |

Fig. 1. Running Example.

tags generated by a speech recognizer from a provided speech annotation. Formally, an object $O$ has associated with it the following information:

1) $W = \{w_1, w_2, \cdots, w_{|W|}\}$, the set of uncertain tags provided by some data processing technique, e.g., speech recognition, or entity resolution.
2) $P = \{p_1, p_2, \cdots, p_{|W|}\}$, where each $p_i \in P$ is the probability that the corresponding uncertain tag $w_i \in W$ belongs to the ground truth tags $G$ that will be explained shortly. These probabilities are provided by the data processing techniques.

We call this object model the *tag model*. Fig. 1 illustrates our running example of an uncertain object $O$ represented in the *tag model*. The set of uncertain tags of $O$ is $W = \{pier, beach, dog\}$. The corresponding probability set is $P = \{0.35, 0.60, 0.70\}$. Initially, we make an assumption that tags are independent. Later we present an approach for handling correlation among tags and a more general *XOR* model where mutual exclusions exist among tags.

**Ground truth tags.** The set of ground truth tags $G$ include the real tags that should have been associated with the object, if the data processing technique outputted 100% accurate results. For instance, in the case where objects are speech annotated images, the ground truth tags correspond to the actual spoken words. The ground truth tags are *unknown* to the algorithm and are hence not part of the uncertain object. These tags are only used for describing the quality metrics we will define later. For the object in the running example, the ground truth tags could be $G = \{dog, beach\}$. Notice that due to the uncertainty in the results returned by the data processing techniques being used, the set of uncertain tags $W$ could be different from the set of ground truth tags $G$.

**Deterministic representation.** A determinization process for an object $O$ is to select a deterministic representation to represent $O$, such that it can be stored in a legacy system that does not support probabilistic input. We define a *deterministic representation* (or *answer set*) of an object $O$ as a set of deterministic tags $A = \{w_1, w_2, \cdots, w_{|A|}\}$. For example, one possible deterministic representation for the object in the running example is $A = \{pier, beach\}$.

When it is not clear from the context, we will use subscript '$O$' in variables $W$, $G$ and $A$ to indicate which object they refer to, e.g. $W_O$ refers to "$W$ of object $O$".

**Query workload.** A query workload is a set of queries $\mathcal{Q} = \{Q_1, Q_2, \cdots, Q_{|\mathcal{Q}|}\}$ that will be executed over the deterministic representation. For clarity of presentation, we will initially assume that each query $Q \in \mathcal{Q}$ is a conjunctive query. We will later discuss how to extend it to other types of queries. A conjunctive query $Q \in \mathcal{Q}$ has the following information associated with it:

1) $T = \{q_1, q_2, \ldots, q_{|T|}\}$ stores the set of terms associated with $Q = q_1 \wedge q_2 \wedge \cdots \wedge q_{|T|}$. Given a deterministic representation $A_O$ of an object $O$, the object satisfies a conjunctive query $Q$ iff $T \subseteq A_O$.
2) $f \in [0, 1]$ encodes the query weight that corresponds to the relative frequency of $Q$ in the workload.

We assume that, in general, such a workload is generated from a past query log, modeled based on the past querying patterns [7], or derived from frequent terms in past corpus of data [8]. We will use subscript '$Q$' for variables $T$ and $f$ to indicate which query they refer to, e.g. $f_Q$ refers to "weight of query $Q$". The running example in Fig. 1 demonstrates a workload of 6 conjunctive queries. The query terms and their weights are also shown.

## 2.2 Quality Metrics

Given deterministic representation for each uncertain object $O \in \mathcal{O}$, we can define quality metric $\mathcal{F}(\mathcal{O}, \mathcal{Q})$ of the deterministic dataset with regard to a given query workload $\mathcal{Q}$. The choice of the metric depends upon the end application.

**Cost-based metric.** For applications that support triggers/alerts, quality can be measured in terms of costs of false negatives and false positives. We will denote by $G_Q$ the set of objects that satisfy query $Q$, when the deterministic representation of each object in $\mathcal{O}$ consists of true/correct tags. We will denote by $A_Q$ the set of objects that satisfy query $Q$, based on the deterministic representations selected by an algorithm for objects in $\mathcal{O}$. Observe that for an object $O$, the case where $O \in A_Q$ but $O \notin G_Q$ corresponds to a false positive caused by the algorithm. In this case, the query will retrieve an irrelevant object. On the other hand, the case where $O \notin A_Q$ but $O \in G_Q$ corresponds to a false negative. In this case, the query will miss a relevant object.

In the application where the cost-based metric is used, each query is associated with two additional pieces of information: (1) $c^+$, the non-negative cost (penalty) of a false positive; (2) $c^-$, the non-negative cost (penalty) of a false negative. We will assume that the default costs of a false positive and a false negative are the same, e.g., $c^+ = c^- = 1$. For generality, however, we will keep $c^+$ and $c^-$ in the formulas in case they need to be different.[2] Again, we will use subscript '$Q$' in $c^+$, and $c^-$ to indicate which query they refer to.

If tags $A$ are chosen as the answer set of object $O$, whereas the ground truth tags are $G$, then the cost of query $Q$ with respect to object $O$ is defined as:

$$\mathit{cost}(O, Q) = \mathit{cost}(A_O, G_O, Q), \tag{1}$$

where

$$\mathit{cost}(A, G, Q) = \begin{cases} c_Q^+ & \text{if } O \in A_Q \wedge O \notin G_Q; \\ c_Q^- & \text{if } O \notin A_Q \wedge O \in G_Q; \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

The cost associated with the entire query workload $\mathcal{Q}$ with respect to object $O$ is defined as:

$$\mathit{cost}(O, \mathcal{Q}) = \sum_{Q \in \mathcal{Q}} f_Q \cdot \mathit{cost}(O, Q) \tag{3}$$

Then the cost-based metric is defined as:

$$\mathit{cost}(\mathcal{O}, \mathcal{Q}) = \sum_{O \in \mathcal{O}} \mathit{cost}(O, \mathcal{Q}). \tag{4}$$

For instance, assume that in Figure 1 the ground truth tags for the uncertain object are $G = \{dog, beach\}$. Suppose that the algorithm chooses $A = \{beach, pier\}$ as its answer set for $O$. Then, queries $Q_3$ and $Q_4$ in Figure 1 will cause two false positives, on the other hand, $Q_1$ and $Q_6$ will result in two false negatives. Thus, the cost with respect to $O$ as defined by Eq. (3) is $0.75$.

**Set-based quality metric.** Applications that support effective query-driven retrieval of objects often utilize set-based quality metrics, such as F-measure or Jaccard similarity. While we will focus on F-measure, our approaches can be naturally applied to many other set-based metrics. If $A_Q$ is the set of objects that satisfy query $Q$, based on the algorithm, the F-measure of answers to query $Q$, with respect to the set of objects $\mathcal{O}$ is defined as:

$$F_\alpha(\mathcal{O}, Q) = F_\alpha(A_Q, G_Q), \tag{5}$$

where

$$F_\alpha(A, G) = \frac{(1+\alpha) \cdot \mathit{Precision}(A, G) \cdot \mathit{Recall}(A, G)}{\alpha \cdot \mathit{Precision}(A, G) + \mathit{Recall}(A, G)} \tag{6}$$

where

$$\mathit{Precision}(A, G) = \frac{|A \bigcap G|}{|A|}, \tag{7}$$

2. For example, three-level low/medium(default)/high model can be used for costs. If a user of, say, a pub/sub system does not want to miss notifications about items that include certain tags, (s)he can set $c^-$ to "high" value. On the other hand, if (s)he does not want to be flooded with objects that are not relevant to his/her interest, (s)he can set $c^+$ to be "high" for the corresponding query.

$$Recall(A, G) = \frac{|A \bigcap G|}{|G|}. \tag{8}$$

The set-based metric is then defined as:

$$F_\alpha(\mathcal{O}, \mathcal{Q}) = \sum_{Q \in \mathcal{Q}} f_Q \cdot F_\alpha(\mathcal{O}, Q). \tag{9}$$

## 2.3 Determinization Problem

Having defined the notation, we now can define the determinization problem:

*Definition 1 (Determinization).* Given a set of uncertain objects $\mathcal{O} = \{O_1, O_2, \ldots, O_{|\mathcal{O}|}\}$, a query workload $\mathcal{Q}$ and a quality metric $\mathcal{F}$, the goal of the *deteriminization problem* is for each object $O \in \mathcal{O}$ to select from $W_O$ a set of tags $A_O \subseteq W_O$ as the deterministic representation of $O$, such that $\mathcal{F}(\mathcal{O}, \mathcal{Q})$ is optimized. □

The ground truth tags $G$ are not associated with uncertain objects and thus not known to any approach to the determinization problem. Therefore, such algorithms cannot directly measure the quality $\mathcal{F}(\mathcal{O}, \mathcal{Q})$ during their execution. Thus, in the following section, we will present our solution that is based on maximizing the *expected* quality measure $\mathbb{E}(\mathcal{F}(\mathcal{O}, \mathcal{Q}))$.

Before we describe our solution to the determinization problem, we note that a baseline algorithm for determinization is a *thresholding* (threshold-cut) approach. The thresholding approach employs a pre-defined threshold $\tau$. For each object $O$ it composes its answer set $A$ by choosing $w_i$ tags from $W$ such that $P(w_i) \geq \tau$. The advantage of this approach is that it is computationally efficient and potentially can be tuned to a particular dataset $\mathcal{O}$ and workload $\mathcal{Q}$ by changing $\tau$. The main drawback of this approach is that it is *unaware* of the query workload ("query-unaware") and thus does not necessarily optimize the given quality metrics, which leads to lower quality.

## 3 Determinization for the Cost-Based Metric

When the cost-based metric is used, $\mathcal{F}(\mathcal{O}, \mathcal{Q})$ is evaluated using $cost(\mathcal{O}, \mathcal{Q})$ measure described in the previous section. Thus, the goal of the determinization problem for the cost-based metric is for each object $O \in \mathcal{O}$ to choose tags $A_O \subseteq W_O$ as its deterministic representation, such that $cost(\mathcal{O}, \mathcal{Q})$ is minimized. It is easy to see that:

$$\min_{A_1, A_2, \ldots, A_{|\mathcal{O}|}} cost(\mathcal{O}, \mathcal{Q}) = \sum_{O \in \mathcal{O}} \min_{A_O} cost(O, \mathcal{Q}). \tag{10}$$

The task thus reduces to determining for each object $O \in \mathcal{O}$ its answer set $A_O$ *independently from other objects*, such that $cost(O, \mathcal{Q})$ specified by Eq. (3) is minimized.

### 3.1 Expected Cost

The challenge of minimizing $cost(O, \mathcal{Q})$ arises since the ground truth $G$ is not known and thus $cost(O, \mathcal{Q})$ cannot be computed directly. The idea of our solution is to choose the answer set based on *expected* cost. Namely, let $\mathcal{W}$ be the space of all possible ground truth sets for $O$, given the uncertain tags in $W$. Let $\mathcal{G} \in \mathcal{W}$ be an instance of a possible ground truth. Table 2 lists all the 8 possible deterministic representations of the uncertain object in our running example. A possible ground truth instance $\mathcal{G}$ can be any one of the 8 representations. Let $\mathbb{P}(\mathcal{G} = G_O)$ be the probability that the instance $\mathcal{G}$ is the actual ground truth of object $O$. Then, if we choose some deterministic representation $A_O$ for $O$, we can compute the expected cost of a query $Q$ with respect to $O$ as:

$$\mathbb{E}(cost(O, Q)) = \sum_{\mathcal{G} \in \mathcal{W}} cost(A_O, \mathcal{G}, Q) \cdot \mathbb{P}(\mathcal{G} = G_O) \tag{11}$$

Directly computing expected cost using Eq. (11) is exponential in the number of tags in $W$ and thus impractical. Let us define $P_{O \in G_Q} = \mathbb{P}(O \in G_Q)$ as the probability that object $O$ satisfies query $Q$, based on ground truth tags of $O$.[3] The following lemma provides an efficient way to compute the expected cost without enumerating every possible $\mathcal{G} \in \mathcal{W}$. The proof is given in Appendix A, which is available as supplemental online material.

*Lemma 1.* $\mathbb{E}(cost(O, Q))$ for a particular chosen answer set $A$ can be computed as:

$$\mathbb{E}(cost(O, Q)) = \mathbb{E}(cost(A, G, Q)) \tag{12}$$

where

$$\mathbb{E}(cost(A, G, Q)) = \begin{cases} c_Q^+ \cdot (1 - P_{O \in G_Q}) & \text{if } O \in A_Q; \\ c_Q^- \cdot P_{O \in G_Q} & \text{if } O \notin A_Q. \end{cases} \tag{13}$$

□

For query workload $\mathcal{Q}$ we have:

$$\mathbb{E}(cost(O, \mathcal{Q})) = \mathbb{E}(cost(A, G, \mathcal{Q})), \tag{14}$$

where

$$\mathbb{E}(cost(A, G, \mathcal{Q})) = \sum_{Q \in \mathcal{Q}} f_Q \cdot \mathbb{E}(cost(A, G, Q)). \tag{15}$$

Given Eq. (14), the optimal answer set, in the expected sense, can be determined as:

$$A_{opt} = \arg\min_{A \in \mathcal{W}} \mathbb{E}(cost(O, \mathcal{Q})). \tag{16}$$

While $A_{opt}$ is not necessarily equal to the ground truth $G$, it is the best answer set given that the ground truth can be any instance $\mathcal{G}$ from $\mathcal{W}$ with the corresponding probability of $\mathbb{P}(\mathcal{G} = G)$. Thus our goal reduces to being able to efficiently find $A_{opt}$. We refer to the problem of finding $A_{opt}$ that minimizes $\mathbb{E}(cost(O, \mathcal{Q}))$ as the ***Expected Determinization problem for the Cost-based Metric*** (***EDCM***). We prove in Appendix B that:

*Lemma 2.* *EDCM* is NP-hard. □

*EDCM* remains NP-hard even under some simplified special cases, such as (a) weights of all queries are the same (it is already shown in the proof in Appendix B); (b) each query

3. For example, under the assumption that tags in object $O$ are independent and query $Q$ is a conjunctive query, $P_{O \in G_Q} = \prod_{q_i \in T_Q} \mathbb{P}(q_i \in G_O)$.

| Answer set | Exp Cost | Answer set | Exp Cost |
|---|---|---|---|
| $\{\}$ | 0.331 | $\{dog, beach\}$ | 0.349 |
| $\{dog\}$ | 0.311 | $\{dog, pier\}$ | 0.428 |
| $\{beach\}$ | 0.321 | $\{beach, pier\}$ | 0.539 |
| $\{pier\}$ | 0.346 | $\{dog, beach, pier\}$ | 0.669 |

TABLE 2
Expected cost for all possible answer sets.

has the same (unit) cost of false positives and false negatives. This case can also be proven to be NP-hard using a similar idea to that of Appendix B, but by a reduction from the weighted MAX-SAT problem.[4]

To solve EDCM, we next will explain a naive enumeration-based algorithm which finds the exact solution to the problem. This naive algorithm is, however, exponential in the number of tags associated with the object. Therefore, it is infeasible when the number of tags is large, which is the case in multiple data sets we will use in Section 6. Hence, we develop a branch-and-bound algorithm to solve EDCM approximately. In Section 6, we empirically demonstrate that our approximate solution reaches the quality that is very close to the exact one, while being orders of magnitude more efficient.

## 3.2 Enumeration-Based Optimal Approach

The desired $A_{opt}$, as defined by Eq. (16), can be found by enumerating all possible answer sets and picking the one with the lowest expected cost. For each uncertain tag $w \in W$ we have two choices: either include it in the answer set or not and thus there are $2^{|W|}$ possible answer sets. Hence, while this algorithm is expected to produce high quality (low cost) results, it is exponential on $|W|$, and therefore it is impractical from the computational efficiency perspective.

For instance, Table 2 shows the expected costs with all possible answer sets for the running example. The enumeration-based approach will go over all the 8 possibilities and select $\{dog\}$, since it has the minimum expected cost of 0.311. For the thresholding approach, if we set $\tau = 0.5$ [5] as the threshold, it will output the answer set $\{dog, beach\}$, which ranks only the 5th among the eight alternatives.

## 3.3 Branch and Bound Algorithm

Instead of performing a brute-force enumeration, we can employ a faster branch and bound (BB) technique. The approach discovers answer sets in a greedy fashion so that answer sets with lower cost tend to be discovered first. Table 3 summarizes the notations we will use to illustrate the proposed BB algorithm.

### 3.3.1 Outline of the BB algorithm

The outline of the proposed BB algorithm is illustrated in Figure 2. The algorithm uses a branch and bound strategy to explore a search tree. Fig. 3 shows a complete search tree for the running example. Each node $v$ in the tree corresponds to a partial tag selection where decisions of whether to include certain tags in the answer set or not have been made. We capture the partial selection of tags using the concept of *sequence* defined as follow:

4. EDCM can be solved in polynomial time under very specific conditions such as all queries contain single distinct terms but such situations are unrealistic.

5. Intuitively, $\tau = 0.5$ is a very good choice of a threshold value, as it keeps tags that have at least 50% chance of being in the ground truth and removes the rest.

| Notation | Meaning |
|---|---|
| $N_{leaf}$ | Number of answer sets to be explored |
| $H$ | Priority queue for selecting the next best node |
| $\mathcal{Q}_w$ | Set of queries whose terms include tag $w$ |
| $S_v$ | Sequence of node $v$ |
| $m_v$ | (Unknown) Minimum cost reachable from $v$ |
| $l_v = \hat{c}_v + \hat{l}_v$ | Lower bound on $m_v$ |
| $h_v = \hat{c}_v + \hat{h}_v$ | Upper bound on $m_v$ |
| $\mathcal{Q}_v$ | Set of undecided/non-fixed queries for $v$ |
| $\hat{c}_v$ | Fixed cost of $v$ |
| $\hat{l}_v$ | Lower bound on non-fixed cost of $v$ |
| $\hat{h}_v$ | Upper bound on non-fixed cost of $v$ |
| $S^h_v$ | Upper bound sequence, used to compute $\hat{h}_v$ |

TABLE 3
Notation for Node $v$.

```
GET-ANSWER-BB($O, \mathcal{Q}, N_{leaf}$)
 1  $A^* \leftarrow \emptyset$          // best answer set observed so far
 2  $c^* \leftarrow +\infty$       // lowest cost observed so far
 3  initialize $H$ to be an empty priority queue
 4  add root node to $H$
 5  while $H$ is not empty do
 6      $v \leftarrow$ REMOVE-BEST-NODE($H$)
        // check if node $v$ can be pruned
 7      if $l_v > c^*$ continue
        // update lowest cost $c^*$ and best answer set $A^*$
 8      if $h_v < c^*$ then
 9          $c^* \leftarrow h_v$
10          $A^* \leftarrow A(S^h_v)$ // Answer set for $S^h_v$
11      if $N_{leaf} \leq 0$ break
12      if $l_v \neq h_v$ then      // check if node $v$ needs to branch
13          $w \leftarrow$ GET-NEXT-TAG($v$)
14          $N_{leaf} \leftarrow$ BRANCH($v, w, H, N_{leaf}$)
15  return $A^*$
```

Fig. 2. Branch and Bound Algorithm.

*Definition 2 (Sequence).* A *sequence* is defined as a mapping $S : W \rightarrow \{\texttt{yes}, \texttt{no}, \texttt{undecided}\}$, which maps each tag $w \in W$ into three categories:

1) $S[w] = \texttt{yes}$, the algorithm decides to put $w$ in $A$;
2) $S[w] = \texttt{no}$, the algorithm decides not to put $w$ in $A$;
3) $S[w] = \texttt{undecided}$, the algorithm has not yet decided whether to put $w$ into $A$ or not.

Sequence $S$ is an *answer sequence* if it does not contain undecided tags, and maps uniquely into one particular answer set $A = A(S)$, where for each $w \in W$ it holds $w \in A$ iff $S[w] = \texttt{yes}$ and $w \notin A$ iff $S[w] = \texttt{no}$. $\square$

We denote by $S_v$ the sequence that corresponds to node $v$. Note that the root of the tree corresponds to a sequence where all tags are undecided and each leaf node corresponds a possible answer sequence. The algorithm starts with a priority queue $H$ that consists of only the root node. At any step, the algorithm extracts from the priority queue the best node $v$,

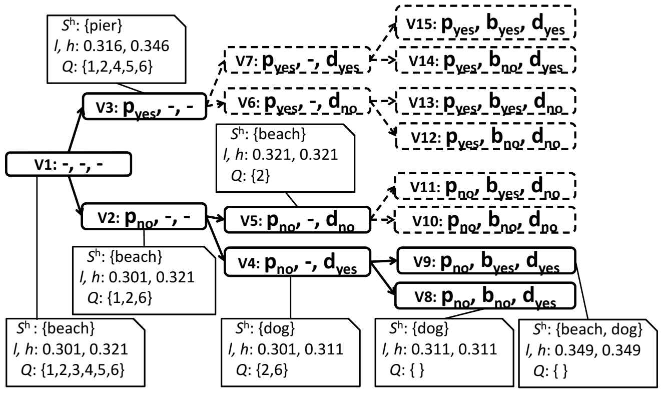


Fig. 3. Search Tree for the running example. For each node $v$ that is visited, three pieces of information are listed: (1) $S^h_v$, upper bound sequence; (2) $l_v$ and $h_v$, lower bound and upper bound; (3) $\mathcal{Q}_v$, undecided queries.

associated with a lower bound $l_v$ and an upper bound $h_v$ of the lowest cost achievable from expanding this node. If the lower bound $l_v$ is larger than the lowest cost $c^*$ found so far, the node can be pruned (Step 7). Otherwise, the algorithm selects an uncertain tag (Step 13), creates two branches corresponding to the the decision to map the tag to yes/no and computes lower and upper bounds of the two nodes generated during the branching procedure (Step 14). The execution time of the algorithm is controlled and bounded by using the $N_{leaf}$ parameter, where the algorithm is not allowed to explore more than $N_{leaf}$ answer sets (Step 11).

The performance of the algorithm depends upon the choice of the node (amongst all the non-leaf nodes) to branch, the computation of upper and lower bounds of a node, and the choice of the tag used to expand the node. We next describe these components of the algorithm in detail.

### 3.3.2 Node Selection

The algorithm maintains a priority queue $H$ for picking a node $v$ that contains the most promising sequence $S_v$ to continue with. Among the nodes in the priority queue, which one should we choose to branch next? Let us consider sequence $S_v$ that corresponds to a node $v \in H$. For $v$ we define $\mathcal{A}_v$ as the set of answer sequences that correspond to the leaf nodes derivable from node $v$ via the branching procedure described above. That is, $\mathcal{A}_v$ corresponds to the leaf nodes of the subtree rooted at node $v$. For example, for node $v_2$ of the tree in Figure 3, $\mathcal{A}_{v_2} = \{S_{v_8}, S_{v_9}, S_{v_{10}}, S_{v_{11}}\}$. Then for node $v$ let

$$m_v = \min_{A \in \mathcal{A}_v} \mathbb{E}(cost(A, G, \mathcal{Q})) \tag{17}$$

be the value of the minimum expected cost among these sequences. Notice that if we knew the exact value of $m_v$, it would have been an ideal key for the priority queue $H$, since it would lead to the quickest way to find the best sequences. The problem is that the exact value of $m_v$ is unknown when $v$ is branched, since the subtree rooted at $v$ is not yet constructed at that moment.

Even though $m_v$ is unknown it is possible to quickly determine good lower and upper bounds on its value $\ell_v \leq$

```
BRANCH($v, w, H, N_{leaf}$)
1  generate a copy $v_{yes}$ of node v
2  $N_{leaf} \leftarrow$ UPDATE-NODE($v_{yes}, w, yes, \mathcal{Q}_v, N_{leaf}$)
3  add $v_{yes}$ to priority queue $H$
4  generate a copy $v_{no}$ of node v
5  $N_{leaf} \leftarrow$ UPDATE-NODE($v_{no}, w, no, \mathcal{Q}_v, N_{leaf}$)
6  add $v_{no}$ to priority queue $H$
7  return $N_{leaf}$
```

Fig. 4. Branching Procedure.

$m_v \leq h_v$, without comparing scores of each sequence in $\mathcal{A}_v$. For any node $v$, if $S_v$ is an answer sequence then the bounds are equal to the cost of the sequence itself, otherwise the bounds are computed as explained next.

### 3.3.3 Upper Bound

To compute an upper bound on $m_v$, it is sufficient to pick one answer sequence $A$ from $\mathcal{A}_v$ and then set $h_v = \mathbb{E}(cost(A, G, \mathcal{Q}))$. We refer to such an answer sequence as the *upper bound sequence* $S^h_v$ for node $v$. The procedure for choosing $S^h_v$ determines the quality of the upper bound.

The proposed algorithm employs a greedy strategy to compute upper bound sequence, as illustrated in Figure 5. It traverses all $w \in S^h_v$ in a particular order that we will explain later. For each undecided tag $w$, it assigns to $S^h_v[w]$ the best yes/no decision that results in the minimum expected cost of query workload. We observe that the decision for one tag $w$ will affect only the results of those queries whose terms include $w$, which we refer to as $Q_w$. Thus, the greedy strategy makes best decision for each undecided tag $w \in S^h_v$ such that overall expected cost of queries in $Q_w$ is minimized.

Notice that once the algorithm determines the values of all $w \in S^h_v$, the procedure EXP-COST in Step 5 and 7 actually computes expected cost using Eq. 13. However, the challenge of computing expected cost arises since $S^h_v$ is not an answer sequence until the end of the greedy algorithm. Eq. 13 is not applicable in that case, because there might exist some undecided tag in $S^h_v$ and the algorithm can not yet determine whether $Q$ will be satisfied. To resolve this challenge, we rewrite Eq. 13 as follows:

$$\mathbb{E}(cost(S, G, Q)) = c^+_Q \cdot (1 - P_{O \in G_Q}) \cdot P_{O \in S_Q} + c^-_Q \cdot P_{O \in G_Q} \cdot (1 - P_{O \in S_Q}) \tag{18}$$

where $P_{O \in S_Q}$ is the probability that object $O$ satisfies query $Q$ based on the associated sequence $S$. Notice that if $S$ is an answer sequence, then $P_{O \in S_Q}$ will be either 1 or 0 and Eq. 18 is actually equivalent to Eq. 13. While $S$ is not yet an answer sequence, we make an assumption that for each undecided tag $w \in S$, $\mathbb{P}(w \in A)$, which denotes the probability that $w$ will be included in answer set $A$, is equal to $\mathbb{P}(w \in G)$. Thus, according to the value of $S[w]$, $\mathbb{P}(w \in A)$ is computed in the following three cases:

1) iff $S[w] =$ yes, then $\mathbb{P}(w \in A) = 1$.
2) iff $S[w] =$ no, then $\mathbb{P}(w \in A) = 0$.
3) iff $S[w] =$ undecided, then $\mathbb{P}(w \in A) = \mathbb{P}(w \in G)$.

Thus for a conjunctive query $Q$, $P_{O \in S_Q}$ can be computed as $P_{O \in S_Q} = \prod_{q_i \in T_Q} \mathbb{P}(q_i \in A)$.

```
COMPUTE-UPPER-BOUND-SEQUENCE($v$)
1   $S_v^h \leftarrow S_v$   // initialize upper bound sequence $S_v^h$
2   for each $w \in S_v^h$ do
        // make yes/no decision for an undecided tag $w$
3       if $S_v^h[w] = undecided$ then
4           $S_v^h[w] \leftarrow yes$
5           $c_{yes} \leftarrow \sum_{Q \in \mathcal{Q}_w}$ EXP-COST($S_v^h, G, Q$)   //Eq. 18
6           $S_v^h[w] \leftarrow no$
7           $c_{no} \leftarrow \sum_{Q \in \mathcal{Q}_w}$ EXP-COST($S_v^h, G, Q$)   //Eq. 18
            // choose the decision with the minimum cost
8           if $c_{yes} < c_{no}$ then
9               $S_v^h[w] \leftarrow yes$
10          else
11              $S_v^h[w] \leftarrow no$
```

Fig. 5. Procedure for computing upper bound sequence.

Observe that if $v$ is a leaf node, its sequence $S_v$ and upper bound sequence $S_v^h$ will be identical. Also notice that an upper bound sequence $S_v^h$ is an answer sequence even if the node $v$ is not a leaf node. Therefore every time upper bound sequence of a node is changed during branching procedure, we "observe" a new answer sequence. The algorithm stops after observing $N_{leaf}$ answer sequences and returns the best answer set observed thus far.

### 3.3.4 Lower Bound

As defined by Eq. (15), the expected cost $\mathbb{E}(cost(A, G, \mathcal{Q}))$ of answer set $A$ is computed over all the queries in query workload $\mathcal{Q}$. One answer set $A$ may be the best one for certain queries in $\mathcal{Q}$, but not for the others. To compute a lower bound on $m_v$, we find the answer set with lowest expected cost for each query *independently* and use the sum of these costs as the lower bound. That is, for a given node $v$

$$\sum_{Q \in \mathcal{Q}} f_Q \cdot \min_{A \in \mathcal{A}_v} \mathbb{E}(cost(A, G, Q)) \leq \min_{A \in \mathcal{A}_v} \mathbb{E}(cost(A, G, \mathcal{Q}))$$

Thus, $\sum_{Q \in \mathcal{Q}} f_Q \cdot \min_{A \in \mathcal{A}_v} \mathbb{E}(cost(A, G, Q))$ can be used as the lower bound on $m_v$, and the task becomes to compute $\min_{A \in \mathcal{A}_v} \mathbb{E}(cost(A, G, Q))$ for each $Q \in \mathcal{Q}$.

For a given node $v$ and query $Q$, there are three different cases for computing the lowest expected cost $\min_{A \in \mathcal{A}_v} \mathbb{E}(cost(A, G, Q))$, depending on $Q$ and the sequence $S_v$ that corresponds to $v$:

1) Query $Q$ will retrieve object $O$ based on $S_v$.[6] Thus only a false positive is possible and the expected cost can be already computed as $c_Q^+ \cdot (1 - P_{O \in G_Q})$, even if $S_v$ itself is not yet an answer sequence.
2) Query $Q$ will not retrieve object $O$ based on $S_v$.[7] Thus only a false negative is possible. Like Case 1, without knowing the decision for undecided tags in $S_v$, we can already compute the expected cost for $Q$ as $c_Q^- \cdot P_{O \in G_Q}$.
3) It is undecided whether $Q$ will retrieve object $O$ or not.[8] Thus, the best answer set derived from $S$ is the one that leads to the minimum cost. Therefore, the lowest expected cost is $\min(c_Q^+ \cdot (1 - P_{O \in G_Q}), c_Q^- \cdot P_{O \in G_Q})$.

6. For a conjunctive query, the test is $\forall q \in T_Q$, $S_v[q] = $ yes.
7. For a conjunctive query, the test is $\exists q \in T_Q$ s.t. $S_v[q] = $ no.
8. For a conjunctive query, the test is $\exists q \in T_Q$ s.t. $S_v[q] = $ undecided and $\nexists q \in T_Q$ s.t. $S_v[q] = $ no.

The computed values of lower and upper bounds can be used to estimate $m_v$. Such estimation can serve as the key in the priority queue $H$. The upper bound $h_v$ is the expected cost of the answer set determined by the greedy algorithm shown in Fig. 5. While not necessarily equal to $m_v$, it is very close to $m_v$ in our experiment. Therefore, we employ the upper bound as the key. The lower bound and upper bound are also utilized to prune the search space, as explained before.

### 3.3.5 Choosing a Tag to Branch Next

Once a node $v$ is chosen, we need to choose one `undecided` tag of the node to branch next. The intuition is that if tag $w$ we choose is a common term for the queries in workload, then making a decision for $w$ will help to quickly determine the expected cost of many queries. Hence, the algorithm sorts all tags of $S_v$ by frequency of occurrence. Every time it needs to choose an `undecided` tag to branch next, it chooses the one that appears in most of the queries in the workload. As a result, the greedy algorithm in Figure 5 also follows the same order for determining values of `undecided` tags.

## 3.4 Query-Level Optimization

The performance of the BB algorithm can be significantly improved further by employing query-level optimizations. For a given sequence $S_v$ of node $v$, we might be able to exactly determine the expected cost of certain queries (e.g., Cases 1 and 2 above), even if $S_v$ is not an answer sequence. In other words, for these queries we can get their cost without expanding $S_v$ into the corresponding answer sequences $A \in \mathcal{A}_v$. We refer to such cost as *fixed* cost of $v$, and to such queries as *fixed* or *decided* queries with respect to $O$. Thus, each node $v$ is also associated with the set of undecided queries $\mathcal{Q}_v$, its fixed cost $\hat{c}_v$, as well as its lower and upper bounds $\hat{l}_v$ and $\hat{h}_v$ on *non-fixed* cost. Note that $l_v = \hat{c}_v + \hat{l}_v$ and $h_v = \hat{c}_v + \hat{h}_v$. Fixed cost and fixed queries give us an opportunity to make the BB algorithm even more efficient by reducing the number of queries involved in computing the expected cost.

When the algorithm performs a branching at a node, as illustrated in Figure 4, it creates two new nodes $v_{yes}$ and $v_{no}$, by making `yes/no` decision for one undecided tag $w$. The procedure in Figure 6 is then used to compute the lower bound and upper bound of each new node. Since one additional tag is now decided for each new node $v$, it is possible that expected cost for some query $Q \in \mathcal{Q}_v$ can be computed (Case 1 and 2). If so, $Q$ becomes a *fixed* query for the new node and will not be considered in future computations derived from that node. Based on the relation between $S_v$ and $Q$, there are three different cases for updating $\hat{c}_v$, $\hat{l}_v$ and $\hat{h}_v$:

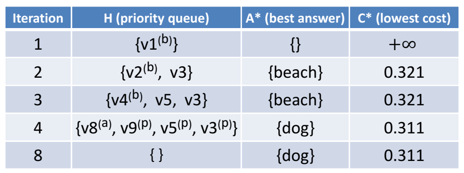
1) Query $Q$ will retrieve object $O$ based on $S_v$. The cost of $Q$ can only be caused by a false positive and $Q$ becomes a *fixed* query. Thus, the lower bound on non-fixed cost $\hat{l}_v$ and the fixed cost $\hat{c}_v$ will be updated as follow:

$$\hat{l}_v \leftarrow \hat{l}_v - f_Q \cdot \min(c_Q^+ \cdot (1 - P_{O \in G_Q}), c_Q^- \cdot P_{O \in G_Q}) \tag{19}$$

$$\hat{c}_v \leftarrow \hat{c}_v + f_Q \cdot c_Q^+ \cdot (1 - P_{O \in G_Q}) \tag{20}$$

```
UPDATE-NODE($v, w, value, \mathcal{Q}, N_{leaf}$)
1   $\mathcal{Q}_v \leftarrow \emptyset$   // initialize set of undecided queries
2   $S_v[w] \leftarrow value$ // store yes or no
3   for each $Q \in \mathcal{Q}$ do
4       if $Q$ will retrieve object $O$ then   // Case 1
5           update $\hat{c}_v$ and $\hat{l}_v$ according to Eq. 19 and 20
6       if $Q$ will not retrieve object $O$ then   // Case 2
7           update $\hat{c}_v$ and $\hat{l}_v$ according to Eq. 19 and 21
8       if $Q$ is not decided for $O$ then   // Case 3
9           $\mathcal{Q}_v \leftarrow \mathcal{Q}_v \cup \{Q\}$
10  if $S_v^h[w] \neq value$ then
11      COMPUTE-UPPER-BOUND-SEQUENCE($v$)
12      $N_{leaf} \leftarrow N_{leaf} - 1$ // new answer set observed
13  if $S_v^h[w] \neq value$ or $|\mathcal{Q}_v| < |\mathcal{Q}|$ then
        // update upper bound on non-fixed cost of node $v$
14      $\hat{h}_v \leftarrow$ EXP-COST($S_v^h, G, \mathcal{Q}_v$)
15  return $N_{leaf}$
```

Fig. 6. Procedure for updating node information, including lower and upper bounds.



| Iteration | H (priority queue) | A* (best answer) | C* (lowest cost) |
|---|---|---|---|
| 1 | {v1$^{(b)}$} | {} | $+\infty$ |
| 2 | {v2$^{(b)}$, v3} | {beach} | 0.321 |
| 3 | {v4$^{(b)}$, v5, v3} | {beach} | 0.321 |
| 4 | {v8$^{(a)}$, v9$^{(p)}$, v5$^{(p)}$, v3$^{(p)}$} | {dog} | 0.311 |
| 8 | { } | {dog} | 0.311 |

Fig. 7. Illustration of the BB algorithm shown in Fig. 2.

Since one query has become *fixed*, the upper bound on *non-fixed* cost $\hat{h}_v$ needs to be recomputed (Step 14).

2) Query $Q$ will not retrieve object $O$ based on $S_v$. The cost of $Q$ can only be caused by a false negative and $Q$ becomes a *fixed* query. In this case, $\hat{l}_v$ will be updated using Eq. 19 and $\hat{c}_v$ will be updated as follow:

$$\hat{c}_v \leftarrow \hat{c}_v + f_Q \cdot c_Q^- \cdot P_{O \in G_Q} \tag{21}$$

Like in case 1, $\hat{h}_v$ needs to recomputed as well.

3) $Q$ is still not *decided* for object $O$. In this case, neither $\hat{c}_v$ nor $\hat{l}_v$ needs to be updated.

This optimization technique trades memory consumption for computational efficiency, as each node $v$ in priority queue $H$ maintains the set of undecided queries $Q_v$. Thus, when the amount of memory is limited, this optimization technique should not be applied. However, it works well in practice and does not require much memory space. This is because typically few queries in $\mathcal{Q}$ are relevant to a particular objects $O$, e.g. only those that contain tags from $W$. Furthermore, each yes/no decision tend to convert lots of *undecided* queries into *decided* ones, thus $|\mathcal{Q}_v|$ reduces significantly as the algorithm moves toward the leaf nodes.

*Example 1.* Figures 3 and 7 illustrate the search tree and the execution of the proposed BB algorithm for the running example. Figure 7 lists the values of three important variables reached at the beginning of each iteration: (1) $H$, the priority queue; (2) $A^*$, the best answer set found so far; and (3) $c^*$, the lowest cost found so far. The super script 'b' means that the node is to be branched in this iteration. The superscript 'p' means that the node is to be pruned according to its lower bound and the lowest cost $c^*$ found so far. The superscript 'a' means that the node corresponds to an answer sequence and thus will not be branched anymore. Because of the priority queue, the algorithm quickly finds the best answer set $\{dog\}$ after 3 iterations. Due to the pruning strategy, only 3 out of 15 nodes have been explored and branched: $v_1$, $v_2$, and $v_4$.

## 3.5 Performance of Branch and Bound

Let $\mathcal{Q}_v$ be the set of queries in query workload and $T$ be the maximum number of query terms, $W$ be the number of uncertain tags in the object, $N_{leaf}$ be the limit on the number of answer sets observed. In Appendix C, we prove that the computational complexity of the BB algorithm is $O(N_{leaf}(|\mathcal{Q}_v| + \log N_{leaf}))$.

As we will see in Section 6, even small values of $N_{leaf}$, lead to nearly optimal quality of the BB algorithm on large datasets in practice. This results in orders of magnitude of improvement of BB over the optimal enumeration algorithm in terms of the efficiency.

## 3.6 Robust Workload

To make the solution resilient to small changes in the workload at runtime, the approach adds the *default workload* $\mathcal{Q}_{def}$ to the original workload $\mathcal{Q}_{orig}$ given to the algorithm, that is, $\mathcal{Q} = \mathcal{Q}_{orig} \cup \mathcal{Q}_{def}$. It does so by using a Lidstone's estimation-like technique that creates a uniform prior on unobserved queries by adding small values $\lambda$ to query frequencies [9]. Let $V$ be the vocabulary that contains all the distinct tags found in the provided corpus of objects $\mathcal{O}$. Then $\mathcal{Q}_{def}$ is constructed by creating a query $Q = w$ for each word $w \in V$ with $c^+ = c^- = 0.5$ and $f = \lambda$ for some small value of $\lambda$. This ensures that all the words found in the object corpus $\mathcal{O}$ are also covered in the query workload $\mathcal{Q}$, while at the same time the choice of tags for each objects $O \in \mathcal{O}$ will still be largely determined by the original workload $\mathcal{Q}_{orig}$ since $\lambda$ is small. We will see in Section 6 that this technique makes the overall approach very robust to small changes in the original query workload.

## 4 DETERMINIZATION FOR SET-BASED METRIC

Previous sections described a solution to the determinization problem for the cost-based metric. In this section we describe how to solve the problem for the set-based metrics.

## 4.1 Expected F-measure

Since the ground truth tags are not known, $F_\alpha(\mathcal{O}, \mathcal{Q})$ in Eq. 9 can not be computed directly. The idea is thus to determinize objects based on expected F-measure. That is, let $\mathcal{W}_\mathcal{O}$ be the space of all possible subsets of $\mathcal{O}$. Each $\mathcal{G} \in \mathcal{W}_\mathcal{O}$ could be a possible set of objects that satisfy a query $Q$ based on ground truth tags. Thus, the expected F-measure of a determinization result, with respect to query $Q$, is computed as:

$$\mathbb{E}(F_\alpha(\mathcal{O}, Q)) = \mathbb{E}(F_\alpha(A_Q, G_Q)) \tag{22}$$

where $\mathbb{E}(F_\alpha(A_Q, G_Q)) = \sum_{\mathcal{G} \in \mathcal{W}_\mathcal{O}} F_\alpha(A_Q, \mathcal{G}) \mathbb{P}(\mathcal{G} = G_Q)$. For a query workload $\mathcal{Q}$, we have

$$\mathbb{E}(F_\alpha(\mathcal{O}, \mathcal{Q})) = \sum_{Q \in \mathcal{Q}} f_Q \cdot \mathbb{E}(F_\alpha(\mathcal{O}, Q)). \tag{23}$$

```
DETERMINIZEF($\mathcal{O}, \mathcal{Q}, \tau$)
1   $count \leftarrow 0$
2   $prevF \leftarrow -\infty$
3   determinize $\mathcal{O}$ using query unaware algorithm
4   while $true$ do
5       $i \leftarrow$ PICKNEXT($\mathcal{O}$)
6       for each $Q \in \mathcal{Q}$ do
7           $F_Q^- \leftarrow \mathbb{E}(F_\alpha(A_Q \setminus \{O_i\}, G_Q))$
8           $F_Q^+ \leftarrow \mathbb{E}(F_\alpha(A_Q \bigcup \{O_i\}, G_Q))$
9       $currF \leftarrow$ BBF($O_i, \mathcal{Q}$)     // determinize $O_i$
10      $count \leftarrow (count + 1)\%|\mathcal{O}|$
11      if  $count = 1$
12          if $currF - prevF < \tau$
13              break
14          $prevF \leftarrow currF$
```

Fig. 8. Efficient Algorithm for set-based determinization

Thus, our goal reduces to determinizing all objects $O \in \mathcal{O}$, such that $\mathbb{E}(F_\alpha(\mathcal{O}, \mathcal{Q}))$ is maximized. It is easy to prove that this problem is also NP-hard.

## 4.2 Efficient approach for computing Expected F-measure

For a particular query $Q$, each object $O$ can either satisfy it or not, so there are $2^{|\mathcal{O}|}$ possible ground truth sets for $Q$. Simply enumerating over $\mathcal{W}_O$ to compute $\mathbb{E}(F_\alpha(A_Q, G_Q))$ is infeasible. One observation is that while computing expected F-measure with respect to a particular query $Q$, we need to consider only those objects that can possibly satisfy $Q$. We thus define candidate set as $\mathcal{O}_Q = \{O \in \mathcal{O} : P_{O \in G_Q} > 0\}$. In practice, $|\mathcal{O}_Q|$ is smaller than $|\mathcal{O}|$ and we can reduce the size of enumeration space from $2^{|\mathcal{O}|}$ to $2^{|\mathcal{O}_Q|}$.

However, the computation is still exponential. We take the approach proposed in [5], which uses Taylor series expansion to approximate $\mathbb{E}(F_\alpha(A_Q, G_Q))$. The approximation of F-measure for a particular $A_Q$ is computed as:

$$\hat{\mathbb{E}}(F_\alpha(A_Q, G_Q)) = \frac{(1+\alpha)\sum_{O \in A_Q} P_{O \in G_Q}}{|A_Q| + \alpha \sum_{O \in \mathcal{O}_Q} P_{O \in G_Q}} \tag{24}$$

In [5], it is shown that $\hat{\mathbb{E}}(F_\alpha(A_Q, G_Q))$ leads to highly accurate results, especially for large values of $|\mathcal{O}_Q|$. Thus we use a hybrid approach. When $|\mathcal{O}_Q|$ is smaller than a threshold $\tau_{enum}$, it employs enumeration to compute exact $\mathbb{E}(F_\alpha(A_Q, G_Q))$, otherwise, it takes the approximation $\hat{\mathbb{E}}(F_\alpha(A_Q, G_Q))$. We set $\tau_{enum}$ to be 6, as suggested by [5].

## 4.3 Iterative Algorithm

In this section, we propose an efficient *iterative* approach to the determinization problem for the set-based metric. The basic framework is outlined in Figure 8. It first determinize all objects (Step 3), using a query unaware algorithm, such as threshold-based or random algorithm, followed by an iterative procedure. In each iteration, the algorithm picks one object $O_i$ (Step 5). It then treats other objects $\mathcal{O} \setminus \{O_i\}$ as already determinized, and determinizes $O_i$ again such that the overall expected F-measure $\mathbb{E}(F_\alpha(\mathcal{O}, \mathcal{Q}))$ is maximized (Step 6-9). In this way, $\mathbb{E}(F_\alpha(\mathcal{O}, \mathcal{Q}))$ will either increase or remain the same in each iteration. For every $|\mathcal{O}|$ iterations, the algorithm checks the value of $\mathbb{E}(F_\alpha(\mathcal{O}, \mathcal{Q}))$, and stops if the increase of the value since last check-point is less than certain threshold. The main question is how to, in each iteration, determinize the chosen object $O_i$ such that the overall expected F-measure is maximized.

### 4.3.1 Negative and Positive F-measure

To answer the question, let us first define two values for each query $Q$: negative F-measure, referred as $F_Q^-$, and positive F-measure referred as $F_Q^+$. For a chosen object $O_i$, $F_Q^-$ is defined as the expected F-measure with respect to $Q$ if $O_i$ is excluded from $A_Q$, that is $F_Q^- := \mathbb{E}(F_\alpha(A_Q \setminus \{O_i\}, G_Q))$. Likewise, $F_Q^+$ is defined as the expected F-measure if $O_i$ is included in $A_Q$, that is $F_Q^+ := \mathbb{E}(F_\alpha(A_Q \bigcup \{O_i\}, G_Q))$.

*Example 2.* Assume the candidate set for a query $Q$ is $\mathcal{O}_Q = \{O_1, O_2, O_3, O_4, O_5, O_6, O_7\}$ and the corresponding probabilities of these objects satisfying $Q$ are $\{0.1, 0.5, 0.8, 0.3, 0.1, 0.1, 0.1\}$. Assume that in some iteration, according to the determinization results of objects in $\mathcal{O}_Q$, the set of objects satisfying $Q$ is $A_Q = \{O_1, O_2, O_3\}$. We can see that $\sum_{O \in A_Q} P_{O \in G_Q} = 1.4$, $\sum_{O \in \mathcal{O}_Q} P_{O \in G_Q} = 2$ and $|A_Q| = 3$. According to Eq. 24, the approximation of F1-measure (F-measure for $\alpha = 1$) is $2 \times 1.4/(3+2) = 0.56$. Now assume the algorithm needs to compute $F_Q^-$ and $F_Q^+$ for $O_1$. Observe that $A_Q \setminus \{O_1\} = \{O_2, O_3\}$, thus $|A_Q \setminus \{O_1\}| = 2$, $\sum_{O \in A_Q \setminus \{O_1\}} P_{O \in G_Q} = 1.3$ and the value of $F_Q^-$ is $2 \times 1.3/(2+2) = 0.65$. Since $A_Q \bigcup \{O_1\} = A_Q$, the value of $F_Q^+$ is already computed as 0.56.

### 4.3.2 Determinizing Individual Object

Having updated negative and positive F-measures for all queries, we are left with the problem of how to determinize the chosen object $O_i$ such that the overall expected F-measure of the query workload is maximized. This problem is virtually the same as the EDCM problem, where the goal is to determinize an object such that the overall expected cost of a query workload is minimized. Thus, we can employ the Branch and Bound algorithm in Section 3.3. More specifically, the BB algorithm can be applied with two small modifications:

1) If we look into Eq. 13, $c_Q^+ \cdot (1 - P_{O \in G_Q})$ is the expected cost of query $Q$ if object $O$ is included in $A_Q$ and $c_Q^- \cdot P_{O \in G_Q}$ is the expected cost if $O$ is excluded from $A_Q$. Likewise, $F_Q^+$ is the expected F-measure with respect to $Q$ if object $O$ is included in $A_Q$ and $F_Q^-$ is the expected F-measure if $O$ is excluded from $A_Q$. Therefore, while applying the BB algorithm, we just need to replace $c_Q^+ \cdot (1 - P_{O \in G_Q})$ and $c_Q^- \cdot P_{O \in G_Q}$ by $F_Q^+$ and $F_Q^-$ respectively.
2) Since the original BB algorithm is to find the minimum, while our task here is to find the maximum, the BB algorithm needs to be changed in a symmetric fashion (for example, exchanging the ways to compute lower bound and upper bound). The main structure of the algorithm stays unchanged.

We call the modified BB algorithm as BBF. As shown in Step 9 of Figure 8, the BBF procedure is used to determinize the chosen object $O_i$ to maximize the overall expected F-measure of query workload $\mathcal{Q}$ and return the resulting maximum expected F-measure.

### 4.3.3 Picking Next Object

Another question is how to pick next object to determinize. One strategy is for each object $O_i \in \mathcal{O}$ to look ahead the overall expected F-measure resulted from choosing this object. The object that leads to the maximum value is chosen as the object to determinize. This strategy, though ensuring maximum increase of the overall expected F-measure in each iteration, will add a linear factor to the overall complexity of the algorithm. Thus, it is not suitable for large datasets. Another strategy is to simply loop over the dataset or choose objects in a random order. Although this strategy is not necessarily the best one in terms of leading the algorithm towards convergence, it is a constant operation. We thus employ the second strategy.

### 4.3.4 Performance of Iterative Algorithm

The computational complexity of the iterative algorithm (Figure 8) is determined by the **while-loop**. Each iteration of the **while-loop** consists of computing $F_Q^-$ and $F_Q^+$ for each query $Q$ (Steps 6-8) and calling the **BBF** procedure (Step 9). We can prove the following lemma (Appendix D):

*Lemma 3.* $F_Q^-$ and $F_Q^+$ for one query $Q$ can be updated in constant time. □

Therefore, the complexity of each iteration is $O(|\mathcal{Q}| + BB_{time})$, where $BB_{time}$ is the complexity of the branch and bound algorithm described in Section 3.3. The overall complexity of the iterative algorithm is then $O(n(|\mathcal{Q}| + BB_{time}))$, where $n$ is the number of iterations before the algorithm stops.

## 4.4 Other Set-Based Metrics

While we illustrate the algorithm using F-measure, the iterative framework can also be used for optimizing other set-based metrics, such as Jaccard distance and Symmetric distance [6]. We can observe from Figure 8 that instead of computing $F_Q^-$ and $F_Q^+$ (Step 7, 8), the task is now to update expected Jaccard distance or Symmetric distance in the two cases where the chosen object $O_i$ is included in $A_Q$ and not. The remaining part of the algorithm can stay the same.

## 5 Extensions to Solutions

In developing the solution, we have made some assumptions both about the data and the query, many of which can be relaxed by appropriate extensions to the proposed scheme. We show how to handle correlation among tags and test the impact of leveraging the correlation in Appendix E. We show how to process non-conjuctive queries specified in the form of DNF/CNF in Appendix F. In this section, we focus on how to extend the proposed solution to handle mutual exclusion amongst tags.

Techniques like entity resolution and speech tagging may return probabilistic attributes that have mutually exclusive values/tags. Specifically, one such typical representation is when, one object $O$ is annotated with a set of probabilistic attributes $\{a_1, a_2, \cdots, a_K\}$. Each attribute $a_i$ is given as a random variable that takes mutually exclusive values $u_i \in U_{a_i}$ with probability $p_i \in P_{a_i}$, where:

1) $U_{a_i} = \{u_1, u_2, \cdots, u_{|U_{a_i}|}\}$ is an option tag set that consists of mutually exclusive possible tags for $a_i$.
2) $P_{a_i} = \{p_1, p_2, \cdots, p_{|U_{a_i}|}\}$ stores the corresponding probabilities, where $p_j$ is the probability that the ground truth value of attribute $a_i$ is $u_j$ and $\sum_j p_j = 1$.

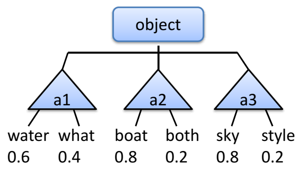


Fig. 9. Example of XOR model.

We call this representation as *mutual-exclusion (XOR) model*. Figure 9 shows an example of XOR model. The object has 3 attributes $\{a_1, a_2, a_3\}$. For instance, attribute $a_3$ can take two values, either `sky` with probability 0.8, or `style` with probability 0.2. Since the values are mutually exclusive, query $Q = \texttt{sky} \wedge \texttt{style}$ should not retrieve this object.

Observe that the tag model from Section 2 is actually a special case of the XOR model. Namely, each pair $(w_i, p_i)$ in the tag model corresponds to one attribute $a_i$ in the XOR model associated with an option set $U_i = \{w_i, null\}$ and a probability set $P_{a_i} = \{p_i, 1-p_i\}$, where $null$ means no option tag of the attribute is in the ground truth.

Based on above observation, both BB and iterative algorithms can be naturally extended to handle objects in XOR model as follows:

1) *Sequence.* Instead of mapping each attribute to `yes`, `no`, or `undecided`, a *sequence* now maps each attribute $a$ into either `undecided` or one option tag $u \in U_a$.

2) *Branching.* While branching on a tree node $v$, instead of making `yes/no` decisions, the algorithm will now create one new node $v_u$ for each $u \in U_a$, where $v_u$ is equal to $v$ except that the *sequence* of $v_u$ now maps attribute $a$ into $u$. As a result, the search tree is no longer binary in general.

3) *Computing* $P_{O \in G_Q}$. The way to compute $P_{O \in G_Q}$ will need to be adjusted accordingly. In general, there is known result [5] that if the number of the query terms $n$ in a conjunctive query $Q = q_1, q_2, \ldots, q_n$ is bounded by a constant, then for XOR model $P_{O \in G_Q}$ can be computed in $O(K)$ time where $K$ is the number of probabilistic attributes of object $O$.

## 6 Experimental Evaluation

In this section we empirically evaluate the proposed approaches in terms of both the quality and efficiency.

## 6.1 Experimental Setup

**Datasets.**

1. `RealSpeech` consists of 525 speech segments, each containing a sentence from broadcast news such as VOA. This dataset imitates speech annotation of uploaded videos. Ground

truth tags for this dataset are given, thus making it possible to test the quality of various approaches. The average number of words in one sentence is 20. We applied an ASR ensemble approach to combine two speech recognizers to get a *tag model* representation for each speech segment.

2. `PubData` consists of 11.7K publications and 14.6K authors derived from CiteSeer and HPSearch datasets. The objects in it are publications and attributes are references to authors in these publications, which can be ambiguous. The references are resolved using the entity resolution method from [10]. It generates the list of alternatives for each attribute as well as their probabilities. Notice that objects in this dataset are represented in *XOR model.*

3. `LargeImg` This semi-real dataset consists of $60,000$ real tagged Flickr images. Each image is tagged with 8 words on average. The probabilistic attributes for this dataset have been generated synthetically from these real tags as follows: For each ground truth tag $g \in G$, we generate an attribute with $N$ option tags. The probabilities $p_1, p_2, \cdots, p_N$ for $N$ option tags are assigned by generating $N$ random numbers drawn from uniform distribution in $(0,1]$ and normalizing them such that they add up to 1. The *interval method* in [5] is then used to decide which option tag $u_i$ is the ground truth value for the attribute, such that the chance of $u_i$ being the ground truth is $p_i$

**Query Workload.** We test our solution over several synthetic query workloads where parameters of workloads follow certain distributions, similar to [11], [12]. While we experiment with several query workloads (discussed later), most of the experiments were conducted on the main query workload $\mathcal{Q}_{orig}$ that consists of single- and multi-term queries. Similar to [5], to make queries more meaningful, we pick combinations of single and multiple terms whose co-occurrence count in the whole corpus of objects exceeds zero. Then for each query $Q \in \mathcal{Q}_{orig}$ we generate $f$, $c^+$ and $c^-$ by first generating two random numbers $r_1, r_2 \sim U[0,1]$ and then setting $f_Q = r_1$, $c_Q^+ = r_2$ and $c_Q^- = 1 - r_2$. We then generate the default workload $\mathcal{Q}_{def}$ as described in Section 3.6 to achieve robustness to changes in $\mathcal{Q}_{orig}$. The resulting workload is $\mathcal{Q} = \mathcal{Q}_{orig} \cup \mathcal{Q}_{def}$.

**Quality Metrics.** We report the true (real) quality metrics reached by various techniques as defined in Eq. (4) and (9). The expected quality metrics are used by the algorithms only internally and thus not reported. As common for these types of experiments, all techniques are first applied to the probabilistic data *without* giving them the knowledge of the ground truth tags. Then their output is compared against the ground truth tags, which are known for the dataset.

**Approaches.** We test the results of the following approaches.

1. `Branch and Bound (BB)`. Section 3.3.
2. `Iterative Algorithm`. Section 4.3.
3. `Enum`. Enumeration based approach.
4. `Thresholding`. This approach is described in Section 2.3. To set the threshold $\tau$ for the approach, we randomly set aside 2/5 of the data as validation dataset $D_{val}$ and 3/5 as testing dataset $D_{test}$. We pick the $\tau$ that leads to minimum cost on $D_{val}$. All methods are compared on $D_{test}$. In XOR model, we use top-1 method to pick the tag with maximum probability. For clarity of presentation, we still call it as `Thresholding`.
5. `Random guessing`. To have an idea of which score a mediocre approach would get, we use the following random strategy as another baseline. For each tag $w_i \in W$, the random strategy generates a random number $r_i \sim U[0,1]$ and includes $w_i$ into answer set $A$ iff $r_i > 0.5$.

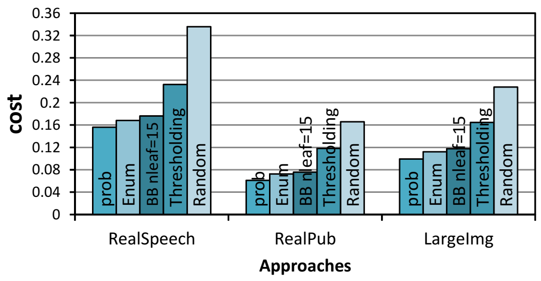


Fig. 10. Cost of different approaches.

## 6.2 Experiments

**Experiment 1 (Quality of various approaches).** This experiment compares all approaches in terms of the cost defined in Eq. (4). The lower the cost is, the better the quality is.

Figure 10 plots the cost of different approaches on three datasets. As expected, on all three datasets, we can observe that BB quickly achieves almost the same quality as that of `Enum` for small value of $N_{leaf}$. We can notice that both `Enum` and BB show about 30% improvement in cost over the baseline threshold-based approach.

**Experiment 2 (Comparing to the Probabilistic-DBMS solution).** In the setting of our problem, object tags *must* be stored as a *deterministic* representation. But a question arises of how the cost/quality will be affected if this was not a requirement. This experiment tests what would happen if the existing applications, such as Flickr, start to allow probabilistic input and also would be able to correctly interpret the probabilities internally.

We refer to this solution as *probabilistic solution* `Prob`. In this solution, there is no need to decide which tag should be in the answer set. The probabilistic representation of object $O$ is kept in a probabilistic DB. Thus, `Prob` can optimize over each individual query independently and choose deterministic representation that leads to minimum expected cost $\min(c_Q^+ \cdot (1 - P_{O \in G_Q}), c_Q^- \cdot P_{O \in G_Q})$. This approach provides us a sense of the best performance from the expected perspective.

Notice, however, even though `Prob` could reach high quality, naturally `Prob` *cannot* be used directly for the problem of determinzation studied in this paper. Hence the comparison is inherently unfair to determinization solutions, but we nevertheless plot the `Prob` solution in all experiments to provide a sense of how close / far the determinization solution is compared to the best possible performance as would be the case with the `Prob` approach.

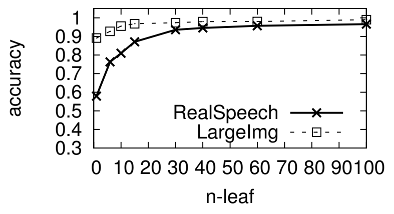


Fig. 11. Accuracy of BB.

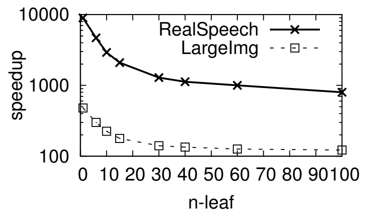


Fig. 12. Speedup of BB.

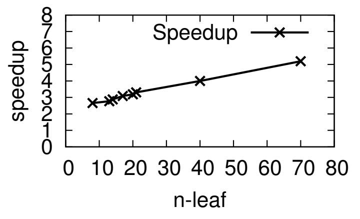


Fig. 13. Speedup over non-optimized BB.

Figure 10 compares BB and Enum to the probabilistic solution. One interesting observation is that Enum and the BB algorithm actually have similar quality to that of the probabilistic solution, which is supposed to achieve the best quality in terms of the expected cost.

*Thus, interestingly, for the studied problem setting, storing data in a deterministic DB by applying determinization procedure turns out to be a competitive solution (compared to using probabilistic DBs), especially given that deterministic DBs tend to be faster and more widespread.*

**Experiment 3 (Performance of the BB algorithm).** The goal of the Branch and Bound algorithm is to reduce the exponential search space of the problem, while maintaining almost the same quality. The BB algorithm stops after observing $N_{leaf}$ answer sets. This experiment compares Enum and the BB algorithm for different values of $N_{leaf}$ in terms of both efficiency and quality.

Figure 11 shows the accuracy of the BB algorithm on RealSpeech and LargeImg dataset, that is the fraction of objects where the cost of answer set $A$ generated by the BB algorithm is equal to that of $A_{opt}$ generated by Enum. The accuracy of 1 means the cost of $A$ always equal to the cost of $A$. The accuracy shown in the figure is averaged over 50 different instances of query workload, where queries in each instance are assigned random weights, false positive and false negative costs. Figure 12 shows the corresponding speedup by the BB algorithm over the Enum, which is computed as the time of Enum divided by the time of the BB algorithm.

On RealSpeech dataset, we can observe that after expanding only $N_{leaf} = 30$ leaf nodes, the accuracy is above 0.9, while BB is still about 1000 times faster than Enum and takes 0.06 second for the dataset. Similarly, on LargeImg dataset, where objects are represented using the *XOR model*, the BB algorithm quickly achieves the accuracy of 0.95 even for small $N_{leaf} = 15$. The difference is that the speedup is now $200\times$ and BB takes 1.43 second, while on the RealSpeech dataset the speedup is above $2000\times$ for $N_{leaf} = 15$. This, however, is as expected, since the average number of tags per object in LargeImg is less than that in RealSpeech. The advantage of BB is less in such cases.

Figure 13 shows the improvement achieved by using query-level optimization, as described in Section 3.4. When $N_{leaf}$ is small, the BB algorithm with query-level optimization shows a $3\times$ speedup over that without optimization, and the speedup increases as $N_{leaf}$ increases.

**Experiment 4 (Performance on various workloads).** This experiment tests different approaches on some special workloads: (a) *uniform weights*: weights of queries in the workload are the same. (b) *equal costs*: false positive cost is equal to false negative cost for each query. (c) *zipf's weights*: weights of queries in the workload follow zipf's distribution. As shown in figure 14, both Enum and BB outperforms Thresholding across all different workloads. The gap between BB and Thresholding in *equal costs* workload is relatively small. This is as expected, because Thresholding prefers consistent ratio of false positive and false negative costs across different queries, while BB can adapt well to different ratios.

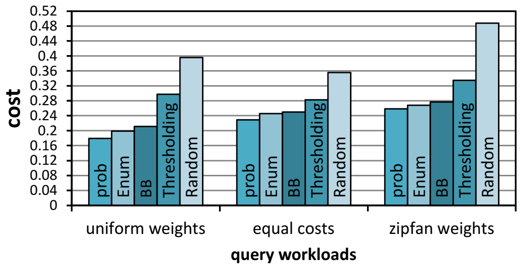


Fig. 14. Performance on special workloads.

**Experiment 5 (Adaptivity to change in query workload).** The query workload $\mathcal{Q}$ provided for generating the deterministic representation, and the actual query workload $\mathcal{Q}_{act}$ used during querying can be different in practice. This experiment tests how various approaches are affected as the original workload $\mathcal{Q}$ is gradually changed into a completely different workload $\mathcal{Q}_{rnd}$.

In this experiment, all the techniques are trained on $\mathcal{Q} = \mathcal{Q}_{orig} \cup \mathcal{Q}_{def}$ to determinize uncertain objects. We create another query workload $\mathcal{Q}_{rnd}$, which is the same as $\mathcal{Q}$, except the weights and costs of all queries in $\mathcal{Q}_{rnd}$ are generated as follows: for each query $Q \in \mathcal{Q}_{rnd}$, we first generate two random numbers $x_1, x_2 \sim U[0, 1]$ and then set $f_Q = x_1$, $c_Q^+ = x_2$ and $c_Q^- = 1 - x_2$. In this way, some queries such as those in $\mathcal{Q}_{def}$, which is not important in $\mathcal{Q}$, may become important in $\mathcal{Q}_{rnd}$. On the other hand, those queries with high weights in $\mathcal{Q}$ may become less important in $\mathcal{Q}_{rnd}$.

The actual workload $\mathcal{Q}_{act}$ is generated as a mix of $\mathcal{Q}$ and $\mathcal{Q}_{rnd}$ as follows. For each query $Q_1 \in \mathcal{Q}$ and its corresponding query $Q_2 \in \mathcal{Q}_{rnd}$, we construct query $Q' \in \mathcal{Q}_{act}$ as: (1) $T_{Q'} = T_{Q_1} = T_{Q_2}$; (2) $f_{Q'} = (1 - \alpha) f_{Q_1} + \alpha f_{Q_2}$; (3) $c_{Q'}^- = (1 - \alpha) c_{Q_1}^- + \alpha c_{Q_2}^-$; and (4) $c_{Q'}^+ = (1 - \alpha) c_{Q_1}^+ + \alpha c_{Q_2}^+$, where parameter $\alpha \in [0, 1]$ controls the relative contribution of the $\mathcal{Q}$ and $\mathcal{Q}_{rnd}$ workloads.

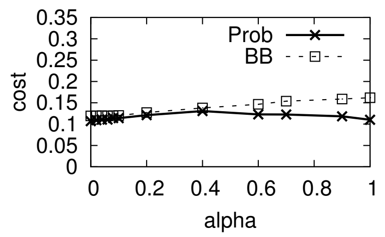


Fig. 15. Adaptiveness.

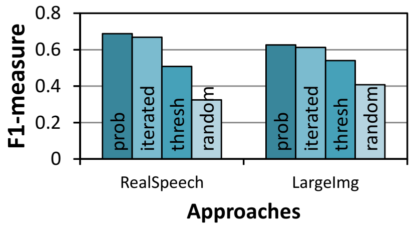


Fig. 16. F1 of various methods.

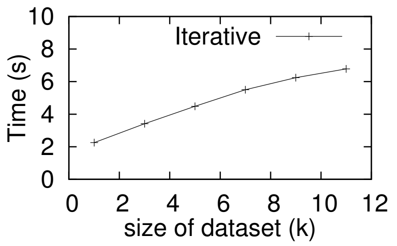


Fig. 17. Iterative algorithm efficiency.

Figure 15 shows the performance of the proposed `BB` algorithm and the probabilistic solution on `LargeImg` dataset. The actual query workload $\mathcal{Q}_{act}$ changes from the original workload $\mathcal{Q}$ ($\alpha = 0$) into a different workload $\mathcal{Q}_{rnd}$ ($\alpha = 1$), as $\alpha$ is changing from 0 to 1. The performance of `BB` remains very close to that of `Prob` for small values of $\alpha$ ($< 0.4$). Although not shown in the figure, `BB` is always better than `thresholding` until the query workload becomes completely different ($\alpha = 1$). This figure illustrates that the proposed BB approach is quite robust to small query changes.

**Experiment 6 (F1 measure of various approaches).** To test the proposed iterative algorithm, we compare it with other query unaware approaches, in terms of F1 measure. Figure 16 shows F1 measure of various approaches on both `RealSpeech` and `LargeImg` datasets. We use object selection techniques in [5] as the probabilistic solution. It can be seen that our `iterative` algorithm is better than threshold-based algorithm. `Enum` approach is not involved in comparison, because it is infeasible to test it even on a small dataset. However, the quality of `iterative` algorithm is very close to that of `prob`, which represents the upper bound on quality achieved by any determinization algorithm.

**Experiment 7 (Efficiency of iterative algorithm).** Figure 17 tests the effect of data size on the efficiency of `iterative` algorithm on `LargeImg` dataset. The number of objects in `LargeImg` is changed from $1,000$ to $11,000$. The figure illustrates that time spent by `iterative` is almost linear in the size of dataset.

## 7 Related Work

**Determinizing Probabilistic Data**. While we are not aware of any prior work that directly addresses the issue of determinizing probabilistic data as studied in this paper, the works that are very related to ours are [5], [6]. They explore how to determinize answers to a query over a *probabilistic* database. In contrast, we are interested in best deterministic representation of data (and not that of a answer to a query) so as to continue to use existing end-applications that take only deterministic input. The differences in the two problem settings lead to different challenges. Authors in [13] address a problem that chooses the set of uncertain objects to be cleaned, in order to achieve the best improvement in the quality of query answers. However, their goal is to improve quality of single query, while ours is to optimize quality of overall query workload. Also, they focus on how to select the best set of objects and each selected object is cleaned by human clarification, whereas we determinize all objects automatically. These differences essentially lead to different optimization challenges. Another related area is MAP inference in graphical model [14], [15], which aims to find the assignment to each variable that jointly maximizes the probability defined by the model. The determinization problem for the cost-based metric can be potentially viewed as an instance of MAP inference problem. If we view the problem that way, the challenge becomes that of developing fast and high-quality approximate algorithm for solving the corresponding NP-hard problem. Section 3.3 exactly provides such algorithms, heavily optimized and tuned to specifically our problem setting.

**Probabilistic Data Models**. A variety of advanced probabilistic data models [16], [17], [18] have been proposed in the past. Our focus however was determinizing probabilistic objects, such as image tags and speech output, for which the probabilistic attribute model suffices. We note that determining probabilistic data stored in more advanced probabilistic models such as And/Xor tree [6] might also be interesting. Extending our work to deal with data of such complexity remains an interesting future direction of work.

**Index term selection.** There are several related research efforts that deal with the problem of selecting terms to index document for document retrieval. A term-centric pruning method described in [19] retains top postings for each term according to the individual score impact that each posting would have if the term appeared in an ad-hoc search query. Authors in [20] propose a scalable term selection for text categorization, which is based on coverage of the terms. The focus of these research efforts is on relevance – that is, getting the right set of terms that are most relevant to document. In our problem, a set of possibly relevant terms and their relevance to the document are already given by other data processing techniques. Thus, our goal is not to explore the relevance of terms to documents, but to select keywords from the given set of terms to represent the document, such that the quality of answers to *triggers/queries* is optimized.

**Query intent disambiguation.** History data such as a query workload and click graph have been used in the research area of query intent disambiguation [21], [22], [23], [24]. Query logs in these works are used to predict more relevant terms for queries, or in other words more accurate intent of queries. However, our goal is not to predict appropriate terms, but to get the right keywords from the terms that are already generated by automated data generation tool. Query workload in our setting is not a source of feature for classification, but used to drive the optimization of the end performance of application.

Thus techniques in these works are not directly applicable in the context of the problem domain we are addressing.

**Query and tag recommendation.** Another related research area is that of query recommendation and tag recommendation [25], [26], [27], [28]. Based on a query-flow graph representation of query logs, authors in [25] develop a measure of semantic similarity between queries, which is used for the task of producing diverse and useful recommendations. Rae et al. [27] propose an extendable framework of tag recommendation, using co-occurrence analysis of tags used in both user specific context (personal, social contact, social group) and non user specific context (collective). The focus of these works is on how to model similarities and correlations between queries/tags and recommend queries/tags based on those information. However, our goal is not to measure similarity between object tags and queries, but to select tags from a given set of uncertain tags to optimize certain quality metric of answers to multiple queries.

## 8 Conclusions and future work

In this paper we have considered the problem of determinizing uncertain objects to enable such data to be stored in pre-existing systems, such as Flickr, that take only deterministic input. The goal is to generate a deterministic representation that optimize the quality of answers to queries/triggers that execute over the deterministic data representation. We have proposed efficient determinization algorithms that are orders of magnitude faster than the enumeration based optimal solution but achieves almost the same quality as the optimal solution. As future work, we plan to explore determinization techniques in the context of applications, wherein users are also interested in retrieving objects in a ranked order.

## References

[1] D. V. Kalashnikov, S. Mehrotra, J. Xu, and N. Venkatasubramanian, "A semantics-based approach for speech annotation of images," *TKDE'11*.

[2] J. Li and J. Wang, "Automatic linguistic indexing of pictures by a statistical modeling approach," *PAMI'03*.

[3] C. Wangand, F. Jing, L. Zhang, and H. Zhang, "Image annotation refinement using random walk with restarts," *ACM Multimedia'06*.

[4] B. Minescu, G. Damnati, F. Bechet, and R. de Mori, "Conditional use of word lattices, confusion networks and 1-best string hypotheses in a sequential interpretation strategy," *ICASSP'07*.

[5] R. Nuray-Turan, D. V. Kalashnikov, S. Mehrotra, and Y. Yu, "Attribute and object selection queries on objects with probabilistic attributes," *ACM TODS'11*.

[6] J. Li and A. Deshpande, "Consensus answers for queries over probabilistic databases," *PODS'09*.

[7] M. B. Ebarhimi and A. A. Ghorbani, "A novel approach for frequent phrase mining in web search engine query streams," *CNSR '07*.

[8] S. Bhatia, D. Majumdar, and P. Mitra, "Query suggestions in the absence of query logs," *SIGIR '11*.

[9] C. Manning and H. Schutze, *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[10] D. V. Kalashnikov and S. Mehrotra, "Domain-independent data cleaning via analysis of entity-relationship graph," *ACM TODS'06*.

[11] K. Schnaitter, S. Abiteboul, T. Milo, and N. Polyzotis, "On-line index selection for shifting workloads," *SMDB'07*.

[12] P. Unterbrunner, G. Giannikis, G. Alonso, D. Fauser, and D. Kossmann, "Predictable performance for unpredictable workloads," *VLDB'09*.

[13] R. Cheng, J. Chen, and X. Xie, "Cleaning uncertain data with quality guarantees," *PVLDB'08*.

[14] V. Jojic, S. Gould, and D. Koller, "Accelerated dual decomposition for map inference," *ICML '10*.

[15] D. Sontag, D. K. Choe, and Y. Li, "Efficiently searching for frustrated cycles in map inference," *UAI '12*.

[16] P. Andritsos, A. Fuxman, and R. J. Miller, "Clean answers over dirty databases: A probabilistic approach," *ICDE'06*.

[17] O. Benjelloun, A. D. Sarma, and A. H. J. Widom, "Uldbs: Databases with uncertainty and lineage," *VLDB'06*.

[18] L. Antova, T. Jansen, C. Koch, and D. Olteanu, "Fast and simple relational processing of uncertain data," *ICDE'08*.

[19] D. Carmel, D. Cohen, R. Fagin, E. Farchi, M. Herscovici, Y. S. Maarek, and A. Soffer, "Static index pruning for information retrieval systems," *SIGIR '01*.

[20] J. Li and M. Sun, "Scalable term selection for text categorization," *EMNLP-CoNLL'07*.

[21] X. Li, Y. Wang, and A. Acero, "Learning query intent from regularized click graphs," *SIGIR '08*.

[22] E. Pitler and K. Church, "Using word-sense disambiguation methods to classify web queries by intent," *EMNLP '09*.

[23] A. Ashkan, C. L. Clarke, E. Agichtein, and Q. Guo, "Classifying and characterizing query intent," *ECIR '09*.

[24] J. Teevan, S. Dumais, and D. Liebling, "To personalize or not to personalize: modeling queries with variation in user intent," *SIGIR '08*.

[25] I. Bordino, C. Castillo, D. Donato, and A. Gionis, "Query similarity by projecting the query-flow graph," *SIGIR '10*.

[26] A. Anagnostopoulos, L. Becchetti, C. Castillo, and A. Gionis, "An optimization framework for query recommendation," *WSDM '10*.

[27] A. Rae, B. Sigurbjörnsson, and R. V. Zwol, "Improving tag recommendation using social networks," *RIAO '10*.

[28] B. Sigurbjörnsson and R. V. Zwol, "Flickr tag recommendation based on collective knowledge," *WWW '08*.

**Jie Xu** received the BSc degree in computer science from Zhejiang University, China, in 2006 and the MSc degree in computer science from Zhejiang University, China, in 2008. He is currently a PhD candidate in the Computer Science Department of the University of California, Irvine. His research interests include information retrieval, machine learning, and computer vision.

**Dmitri V. Kalashnikov** is an Associate Adjunct Professor of Computer Science at the University of California, Irvine. He received his PhD degree in Computer Science from Purdue University in 2003. He received his diploma in Applied Mathematics and Computer Science from Moscow State University, Russia in 1999, graduating summa cum laude.

His general research interests include databases and data mining. Currently, he specializes in the areas of entity resolution & data quality, and real-time situational awareness. In the past, he has also contributed to the areas of spatial, moving-object, and probabilistic databases.

He has received several scholarships, awards, and honors, including an Intel Fellowship and Intel Scholarship. His work is supported by the NSF, DH&S, and DARPA.

**Sharad Mehrotra** received the PhD degree in computer science from the University of Texas at Austin in 1993. He is currently a professor in the Department of Computer Science at the University of California, Irvine (UCI) and the director of the Center for Emergency Response Technologies. Previously, he was a professor at the University of Illinois at Urbana-Champaign (UIUC). He has received numerous awards and honors, including SIGMOD Best Paper award 2001, DASFAA Best Paper award 2004, and CAREER award 1998 from the US National Science Foundation (NSF). His primary research interests are in the area of database management, distributed systems, and data analysis. He is a member of the IEEE.

## Appendix A
## Proof of Lemma 1 (Section 3.1)

We will prove Lemma 1 for two cases:

1) *if* $O \in A_Q$. In this case, we can see from Eq. (2) that *cost*$(A_O, \mathcal{G}, Q)$ will have non-zero value $c_Q^+$ only if $O \notin \mathcal{G}_Q$, that is, object $O$ does not satisfy query $Q$ given $\mathcal{G}$ as the ground truth tags of $O$. Therefore, we can rewrite Eq. (11) as follow:
$$\mathbb{E}(cost(O,Q)) = \sum_{\mathcal{G} \in \mathcal{W} \wedge O \notin \mathcal{G}_Q} \mathbb{P}(\mathcal{G}) \cdot c_Q^+ \qquad (25)$$
Notice that $c_Q^+$ is a constant value for a particular $Q$. Also, it is easy to see that $\sum_{\mathcal{G} \in \mathcal{W} \wedge O \notin \mathcal{G}_Q} \mathbb{P}(\mathcal{G}) = 1 - P_{O \in G_Q}$. We thus prove that $\mathbb{E}(cost(O,Q)) = c_Q^+ \cdot (1 - P_{O \in G_Q})$ in the case where $O \in A_Q$.
2) *if* $O \notin A_Q$. Similarly, in this case, *cost*$(A_O, \mathcal{G}, Q)$ will have non-zero value $c_Q^-$ only if $O \in \mathcal{G}_Q$. Therefore, we can rewrite Eq. (11) as follow:
$$\mathbb{E}(cost(O,Q)) = \sum_{\mathcal{G} \in \mathcal{W} \wedge O \in \mathcal{G}_Q} \mathbb{P}(\mathcal{G}) \cdot c_Q^- \qquad (26)$$
Since $\sum_{\mathcal{G} \in \mathcal{W} \wedge O \in \mathcal{G}_Q} \mathbb{P}(\mathcal{G}) = P_{O \in G_Q}$, we thus derive that $\mathbb{E}(cost(O,Q)) = c_Q^- \cdot P_{O \in G_Q}$ in the case where $O \notin A_Q$.

## Appendix B
## Proof of Lemma 2 (Section 3.1)

We will prove that the *Expected Determinization problem for the Cost-based Metric (EDCM)* is NP-Hard by reduction from MAX-SAT problem. Given a set of clauses $C = \{c_1, c_2, \ldots, c_m\}$, each of which is a disjunction of literals, MAX-SAT problem is to find an assignment of values to variables $V = \{v_1, v_2, \ldots, v_n\}$ that maximizes the number of satisfied clauses.

In *EDCM*, each object $O$ includes a set of tags $W$ and probabilities $P$. Each conjunctive query $Q_i$ from query workload $\mathcal{Q}$ is given weight $f_i$, costs $c_i^+$ and $c_i^-$, and a set of query terms $T_i = \{q_{i1}, q_{i2}, \ldots, q_{ik}\}$, where $T_i \subseteq W$. The goal is for the object to find answer set $A$ (a combination of its tags) that would minimize $\mathbb{E}(cost(O, \mathcal{Q}))$.

Any instance of MAX-SAT, characterized by $(V, C)$, is polynomially reducible to the following instance of *EDCM* with certain $(W, P, \mathcal{Q})$: For each variable $v_i \in V$, put two tags "$v_i$" and "$\neg v_i$" in $W$. All the probabilies in $P$ are 0.5. Queries in $\mathcal{Q}$ all have weight $f = 1$ and are constructed in two ways:

*a-queries*. For each disjunctive clause $c_i \in C$, put one conjunctive query $Q_i$ in $\mathcal{Q}$, which includes query term "$\neg v_i$" (or "$v_i$") if literal $v_i$ (or $\neg v_i$) is in $c_i$. For example, if $c_i$ is $a \vee \neg b \vee c$, then $T_i$ would be $\{$"$\neg a$", "$b$", "$\neg c$"$\}$. For query $Q_i$ set:

1) $c^+$ such that $c^+ \left(1 - P_{O \in G_Q}\right) = 1$
2) $c^- = 0$.

*b-queries*. For each $v_i \in V$, put three queries $\{$"$v_i$", "$\neg v_i$"$\}$, $\{$"$v_i$"$\}$ and $\{$"$\neg v_i$"$\}$ in $\mathcal{Q}$. For query $\{$"$v_i$", "$\neg v_i$"$\}$ set:

1) $c^+ = \infty$
2) $c^- = 0$.

For both $\{$"$v_i$"$\}$ and $\{$"$\neg v_i$"$\}$ set:

1) $c^+ = 0$
2) $c^- = n/0.5$, where $n$ is the number of a-queries.

Now we will show that the answer set $A$, that minimizes the expected total cost of queries in the constructed EDCM instance $(W, P, \mathcal{Q})$, corresponds to the assignment for $V$ that maximize the number of satisfied clauses in the MAX-SAT instance $(V, C)$. We observe that:

1) Including one tag "$v_i$" ( or "$\neg v_i$" ) in $A$ in the determinization instance corresponds to assigning true (or false) to the variable $v_i$ in the MAX-SAT instance.
2) If both tags "$v_i$" and "$\neg v_i$" constructed from one variable $v_i$ are included in answer set $A$ to the *EDCM* instance, the query $\{$"$v_i$", "$\neg v_i$"$\}$ will be satisfied, causing a false positive of infinity, and thus infinite expected total cost of all queries. On the other hand, excluding either one of the two tags from $A$ will cause an expected cost $n$, which is the maximum total cost that can be possibly caused by a-queries. To minimize the total expected, the answer set $A$ will include exact one of the two tags. This corresponds to the case in MAX-SAT instance that exact one of the two opposite literals $v_i$ and $\neg v_i$ is true.
3) Having shown that exact one of any two opposite tags "$v_i$" and "$\neg v_i$" will be selected in $A$, we know that the expected total cost of the *b-queries* is always $|V| \cdot n$, so minimizing expected total cost of all queries is the same as minimizing that of all *a-queries*. Since satisfying one *a-query* will cause a false positive cost of 1, while false negative costs of *a-queries* are 0, minimizing expected total cost of all *a-queries* is the same as maximizing the number of *a-queries* that are not satisfied. From the construction of *a-queries*, we can see that an unsatisfied *a-query* in the determinization instance corresponds to a satisfied clause $c_i$ in MAX-SAT instance $(V, C)$.

We can now conclude that the answer set $A$ that minimizes the expected total cost of queries in the *EDCM* instance, corresponds to the assignment for $V$ that maximizes the number of satisfied clauses in the MAX-SAT instance. Since MAX-SAT problem, known as NP-Hard, can be solved by mapping it to an instance of *EDCM*, we conclude *EDCM* is NP-Hard too.

## Appendix C
## Complexity of the BB Algorithm (Section 3.5

Let $\mathcal{Q}_v$ be the set of queries in query workload, $T$ be the maximum number of query terms, $W$ be the number of uncertain tags in the object, and $N_{leaf}$ be the limit on the number of answer sequence observed. Then we can prove: The computational complexity of BB is $O(N_{leaf}(|\mathcal{Q}_v| + \log N_{leaf}))$.

The complexity of the main BB procedure (Figure 2) is determined by the **while-loop**. Let us now make a conservative analysis of the **while-loop** part. First we will compute bounds on the number of iterations and the size of the priority queue $|H|$. Executing the last **if** makes $N_{leaf}$ decrease by one – since a call to **branch** will observe exactly one new answer set. Thus, the number of iterations is $O(N_{leaf})$. On each iteration, $|H|$ will first decease by 1 and then increase by 2 when the last **if** is executed. Thus, on each iteration $|H|$ is, very conservatively, $O(N_{leaf})$.

Each iteration of the **while-loop** consists of first calling the **remove-best-node** procedure – $O(\log |H|)$. Thus, the overall complexity of all call to **remove-best-node** is $O(N_{leaf} \log N_{leaf})$.

The **branch** procedure consists of copying node $v$ – $O(W + |\mathcal{Q}_v|)$, putting nodes in the priority queue – $O(\log N_{leaf})$, and calling **update-node**. The complexity of **update-node** comes from checking whether query $Q$ will retrieve the object base on $S_v$ in the loop – $O(|\mathcal{Q}_v|T)$, **compute-upper-bound-sequence** – $O(|\mathcal{Q}_v|WT)$ and computing the expected cost at the end of the procedure – $O(|\mathcal{Q}_v|)$, which makes its total complexity $O(|\mathcal{Q}_v|WT)$. Given that the number of iterations of the **while-loop** is $O(N_{leaf})$, the overall complexity of the last **if** is $O(N_{leaf} \cdot (W + |\mathcal{Q}_v| + \log N_{leaf} + |\mathcal{Q}_v| \cdot W \cdot T))$, which is also the complexity of the BB algorithm. Given that T and W are limited (by small numbers) in practice, the effective complexity of BB is $O(N_{leaf} \cdot (|\mathcal{Q}_v| + \log N_{leaf}))$.

## APPENDIX D PROOF OF LEMMA 3 (SECTION 4.3.4)

As we discussed in Section 4.2, when $|\mathcal{O}_Q|$ is larger than $\tau_{enum}$, we use $\hat{\mathbb{E}}(F_\alpha(A_Q, G_Q))$ as an approximation for $F_\alpha(A_Q, G_Q)$. If we look into Eq. 24, there are three terms:

1) $\alpha \sum_{O \in \mathcal{O}_Q} P_{O \in G_Q}$, this term does not involve $A_Q$, thus it can be pre-computed once for each query $Q$ and never changes between iterations.
2) $(1+\alpha) \sum_{O \in A_Q} P_{O \in G_Q}$ and $|A_Q|$. When $F_Q^-$ and $F_Q^+$ are updated in each iteration, only one object of $A_Q$ is involved. Therefore, these two terms can be pre-computed at the beginning of the algorithm and re-computed based on their values from last iteration in constant time.

Based on above observations, $F_Q^-$ and $F_Q^+$ can be updated in constant time, if $|\mathcal{O}| > \tau_{enum}$. In the case where $|\mathcal{O}_Q| \leq \tau_{enum}$, we use enumeration to compute exact $\mathbb{E}(F_\alpha(A_Q, G_Q))$. Since $\tau_{enum}$ is very small (in [5] it is set to be 6), $F_Q^-$ and $F_Q^+$ can still be updated in constant time.

## APPENDIX E HANDLING CORRELATION (SECTION 5)

The work can be extended to consider correlation of tags, for example by using the domain semantics as in [1]. Such techniques can be used to compute more accurate $P_{O \in G_Q}$ by leveraging known tag correlations. We demonstrate an example of how the proposed determinization framework, when provided with additional knowledge of correlation in data, can compute more accurate $P_{O \in G_Q}$ and thus further improve the quality of the result.

### E.1 Incorporating Correlation of Tags

The way to use correlations depends on the nature of the captured correlations and application domain. Similar to the one described in [5], we use a supervised learning framework to incorporate correlations in the form of co-occurrence between tags.

Given a feature vector for an object $O$ and a query $Q$, we train a classifier to estimate the probability that $O$ belongs to $G_Q$. The question now becomes how to choose features for $O$ and $Q$.

**2-term queries.** To understand the idea of the approach, let us consider an example of a 2-term query $Q = q_1 \wedge q_2$. An object $O$ can possibly satisfy $Q$ if there are two of its attributes $a_1$ and $a_2$, where $q_1$ and $q_2$ belong to option tags of $a_1$ and $a_2$ respectively. Under the assumption of tag independence, $P_{O \in G_Q} = p_{q1} \cdot p_{q2}$, where $p_{q1}$ and $p_{q2}$ are probabilities associated with the option tags $q_1$ and $q_2$. We call this probability $p_{ind}$. Though not accurate enough, $p_{ind}$ can be one of the features in determine whether $O$ will satisfy $Q$. The knowledge of correlations among tags in a past corpus of data set can help to compute $P_{O \in G_Q}$ more accurately. Let $\mathcal{D}_{train}$ be the past data set where values of all tags are known exactly and there is no uncertain attribute. For instance, in the scenario of speech annotation of images, $\mathcal{D}_{train}$ could be a large image collection where each image is manually annotated. Let $c(q_1, q_2)$ be the number of images in $\mathcal{D}_{train}$ annotated with both $q_1$ and $q_2$, $c(q_1, q_2)$ would be a strong feature in determine $P_{O \in G_Q}$. Let $q_{t1}$ and $q_{t2}$ be the top probabilities of option tags for $a_1$ and $a_2$, $p_{top} = q_{t1} \cdot q_{t2}$ is the highest joint probability of $a_1$ and $a_2$ under the assumption of independence of tags. As a measure of how far $p_{ind}$ is away from top probability, we use $p_{top} - p_{ind}$ as a third feature. Thus, the resulting feature vector is: $f = (p_{ind}, c(q_1, q_2), p_{top} - p_{ind})$. By training a classifier on a validation set $\mathcal{D}_{val}$ while using $\mathcal{D}_{train}$ for computing co-occurrence counts, we can apply the classifier on the test data set $\mathcal{D}_{test}$ to compute $P_{O \in G_Q}$.

**1-term queries.** A similar approach can be applied to 1-term queries in the form of $Q = q_1$. If one attribute $a_1$ contains $q_1$ as option tag. The feature vector is: $f = (p_{q_1}, c(q_1), p_{top} - p_{q_1})$, where $p_{q_1}$ is the probability associated with option tag $q_1$, $p_{top}$ is the highest probability of $a_1$, and $c(q_1)$ is the number of objects in $\mathcal{D}_{train}$ annotated with $q_1$

**m-term queries.** Given an m-term query $Q = q_1 \wedge q_2 \wedge \cdots \wedge q_m$, a similar approach can be applied. However, for larger $m$ it becomes a challenge to compute the co-occurrence count $c(q_1, q_2, \cdots, q_m)$. So we change $c(q_1, q_2, \cdots, q_m)$ into score $s(q_1, q_2, \cdots, q_m)$, which is computed as the sum of pairwise co-occurrence counts $c(q_i, q_j)$ for all distinct pairs of $q_i$ and $q_j$. Let $p_{ind} = p_{q1} \times p_{q2} \times \cdots \times p_{qm}$ be the probability that $O \in G_Q$ under the assumption of independence and $p_{top}$ be the probability of the top combination, the resulting feature vector is: $f = (p_{ind}, s(q_1, q_2, \cdots, q_m), p_{top} - p_{ind})$.

### E.2 Impact of leveraging correlation

This experiment tests the impact of leveraging correlations between tags. Figure 18 compares various approaches with and without correlations on `RealSpeech` and `LargeImg`

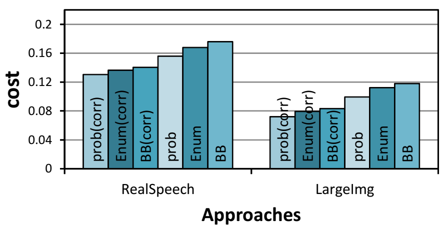


Fig. 18. Determinization quality improvement with correlation.

datasets. Introducing correlations among tags may result in more accurate $P_{O \in G_Q}$. The performance of the *probabilistic solution* serves as a good measure of the quality of this probability. The figure shows improved probability quality after leveraging correlations on both datasets. Also, with more accurate $P_{O \in G_Q}$, the quality of both `Enum` and `BB` can be further improved by about 20%

## APPENDIX F
## HANDLING OTHER QUERY TYPES (SECTION 5)

Thus far we have focused on conjunctive `AND` queries. However, our approaches can be easily generalized to other types of queries. We have formulated the formulas in Sections 2, 3.1 and 4 in such a fashion that, in addition to conjunctive queries, they still hold for other types of queries, as long as $P_{O \in G_Q}$ can be computed. Hence, to extend our approaches to handle other types of queries, one only need to specify a way for computing $P_{O \in G_Q}$.

Let us first consider `AND`/`NOT` (e.g. "University" `AND NOT` "California") as well as `OR`/`NOT` queries (e.g. "University" `OR NOT` "California"). First, we need to introduce some additional notation. Since now queries might contain negative terms, instead of using $T_Q$ to represent the set of query terms as shown in Table 1, we now use $T_Q^+$ to represent the set of positive query terms of $Q$ and $T_Q^-$ for the set of negative terms. Specifically,

1) if $Q$ is `AND`/`NOT` query

$$P_{O \in G_Q} = \prod_{q_i \in T_Q^+} \mathbb{P}(q_i \in G) \prod_{q_i \in T_Q^-} (1 - \mathbb{P}(q_i \in G)) \quad (27)$$

2) if $Q$ is `OR`/`NOT` query

$$P_{O \in G_Q} = 1 - \prod_{q_i \in T_Q^+} (1 - \mathbb{P}(q_i \in G)) \prod_{q_i \in T_Q^-} \mathbb{P}(q_i \in G) \quad (28)$$

In addition, a query in disjunctive normal form (DNF) can also be handled efficiently as long as there is no term that appears in more than one clause in one query. Let $C_Q$ be the set of conjunctive clauses in the query $Q$, each clause $c_I \in C_Q$ can be viewed as an `AND`/`NOT` query. We denote as $P_{c_i}$ the probability that $c_i$ is satisfied. This probability can be computed using Eq. (27). Thus,

$$P_{O \in G_Q} = 1 - \prod_{c_i \in C_Q} (1 - P_{c_i}) \quad (29)$$

It is easy to see that the probability for a CNF query can be handled in a similar way.

In case of generic DNF where one term can appear in more than one clause, the problem of estimating $P_{O \in G_Q}$ is NP-hard. However, in practice, number of terms in one query is limited, thus even an exponential algorithm is acceptable.