NYU Stern Research Working Paper
CeDER-08-02

A Quality-Aware Optimizer for Information Extraction

Panagiotis G. Ipeirotis
New York University
panos@nyu.edu

Alpa Jain
Columbia University
alpa@cs.columbia.edu

May 21, 2008

Abstract

Large amounts of structured information is buried in unstructured text. Information extraction systems can extract structured relations from the documents and enable sophisticated, SQL-like queries over unstructured text. Information extraction systems are not perfect and their output has imperfect precision and recall (i.e., contains spurious tuples and misses good tuples). Typically, an extraction system has a set of parameters that can be used as "knobs" and tune the system to be either precision- or recall-oriented. Furthermore, the choice of documents processed by the extraction system also affects the quality of the extracted relation. So far, estimating the output quality of an information extraction task was an ad-hoc procedure, based mainly on heuristics. In this paper, we show how to use receiver operating characteristic (ROC) curves to estimate the extraction quality in a statistically robust way and show how to use ROC analysis to select the extraction parameters in a principled manner. Furthermore, we present analytic models that reveal how different document retrieval strategies affect the quality of the extracted relation. Finally, we present our maximum likelihood approach for estimating—on the fly—the parameters required by our analytic models to predict the run time and the output quality of each execution plan. Our experimental evaluation demonstrates that our optimization approach predicts accurately the output quality and selects the fastest execution plan that satisfies the output quality restrictions.

1 Introduction

Unstructured text in large collections of text documents such as news paper articles, web pages, or email often embeds *structured* information that can be used for answering structured, relational queries. To extract the structured information from text documents, we can use an information extraction system, such as Snowball [3], Proteus [21], MinorThird [12], or KnowItAll [16], which take as input a text document and produce tuples of the target relation. Often, the extraction process relies on *extraction patterns* that can be used to extract instances of tuples.

Example 1 *An example of information extraction task is the construction of a table of company headquarters* Headquarters(Company, Location)*, from a newspaper archive. An information extraction system processes documents in the archive (such as the archive of* The New York Times *–see Figure 1) and may extract the tuple ⟨Army Research Laboratory, Adelphi⟩ from the news articles in the archive. The tuple ⟨Army Research Laboratory, Adelphi⟩ was extracted based on the pattern "⟨*ORGANIZATION in LOCATION*⟩", after identifying the organizations and locations in the given text using a* named-entity tagger.

Extracting structured information from unstructured text is inherently a noisy process, and the returned results do not have perfect "precision" and "recall" (i.e., they are neither perfect nor complete). The erroneous tuples may be extracted because of various problems, such as erroneous named-entity recognition or imprecise extraction patterns. Additionally, the extraction system may not extract all the valid tuples from the document, e.g., because the words in the document do not match any of the extraction patterns. To examine the quality of an extracted relation, we can measure the number of *good* and *bad* tuples in the output to study the two types of errors committed during the extraction: the "false negatives," i.e., the number of tuples missing from the extracted relation and the "false positives," i.e., the number of incorrect tuples that appear in the output.

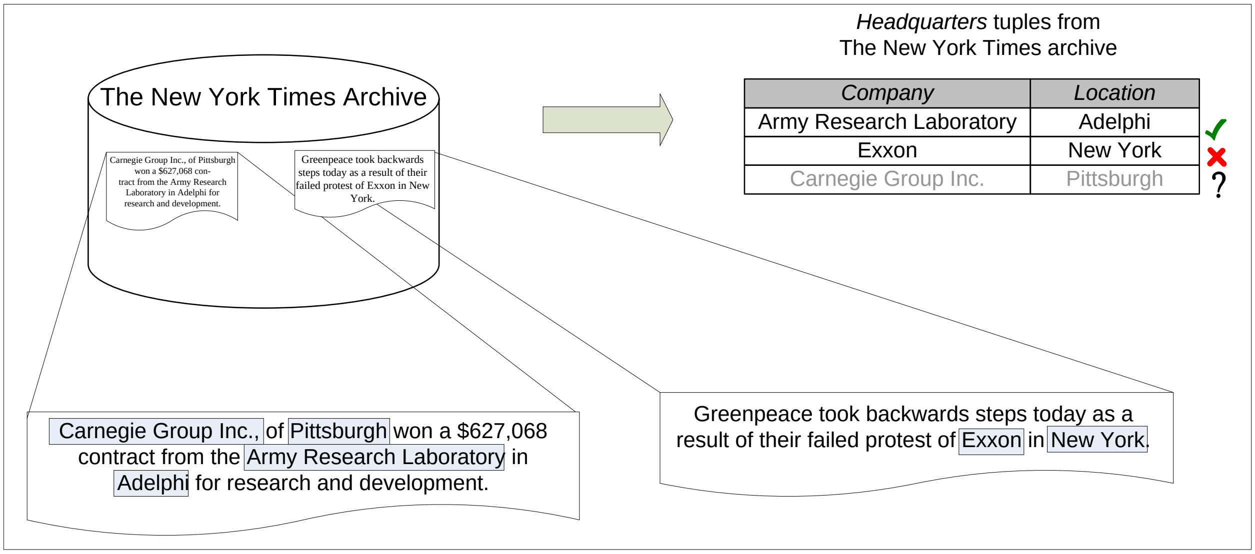


Figure 1: An example of an information extraction system extracting the relation *HeadQuarters(Company, Location)* from *The New York Times* archive, and extracting a correct tuple, an incorrect tuple, and missing a tuple that appears in the text.

Example 1 (continued.) *For the* HeadQuarters *relation, in Figure 1, the extraction pattern* '⟨ORGANIZATION in LOCATION⟩" *also generates the bad tuple ⟨Exxon, New York⟩. Figure 1 also shows a missing good tuple ⟨Carnegie Group Inc., Pittsburgh⟩ in the document, which was not identified, because the extraction system does not include a suitable pattern.*

To control the quality of the extracted relations, extraction systems often expose multiple tunable "knobs" that affect the proportion of good and bad tuples observed in the output. As an example of a simplistic knob, consider a decision threshold τ that defines the number of rules employed by the extraction system for the task of extracting the *Headquarters* relation. A small number of (precise) rules will generate a mostly correct tuples but may also miss many tuples that appear in the documents but do not match any of the (small number of) active rules. By adding more rules the system can capture more tuples (i.e., decrease the false negatives) but at the same time this also results in an increase in the incorrect tuples in the output (i.e., increase in the false positives). Other examples of knobs may be decision thresholds on the minimum confidence or minimum pattern support required before generating a tuple from the text. In a more extreme setting, we may even have multiple extraction systems for the same relation, each demonstrating different precision-recall tradeoffs.

A natural question that arises in a tunable extraction scenario is: How we can choose which extraction system to use and the appropriate parameter settings for an extraction task, *in a principled manner*? Unfortunately, this important task is currently performed empirically, or by following simple heuristics. In this paper, we approach the problem by analyzing a *set* of information extraction systems using *receiver operating characteristic (ROC)* curves. As we will see, this allows us to characterize IE systems *in a statistically robust manner*, and allows the natural modeling of parameters that have a non-monotonic impact on the false positives and false negatives in the output. We show how ROC analysis allows us to keep only the set of "Pareto optimal" configurations that cannot be fully dominated by other configurations. Furthermore, we demonstrate how we take into consideration other parameters, such as execution time and monetary cost, by using generalizing the basic ROC paradigm.

Beyond the choice of the extraction system and its settings, the quality characteristics of the extracted relation are also affected by the choice of documents processed by the extraction system. Processing documents that are not relevant to an extraction task may introduce many incorrect tuples, without adding any correct ones in the output. For instance, processing documents from the "Food" section of a newspaper for the *Headquarters* relation not only delays the overall extraction task, but also adds false tuples in the relation, such as ⟨*Crostini, Polenta*⟩, which are erroneously extracted from sentences like "*...enjoy this Polenta-based Crostini!*".

Until now, the choice of a document retrieval strategy was based only on the efficiency and the impact of this choice on the quality of the output was ignored. However, as argued above, considering the impact of the document retrieval strategy is also of critical importance. As an important contribution of this paper, we present a rigorous statistical analysis of multiple document retrieval strategies that show how the output quality—and, of course, execution time—is affected by the choice of document retrieval strategy. Our modeling approach results in a set of *quality curves* that predict the quality characteristics of the output over time, for different retrieval strategies and different settings of the extraction system.

The analytical models that we develop in this paper show predicting the execution time and output quality of an execution strategy requires knowledge of some database-specific parameters which are typically not known a priori. Using these analytical models, we show how we can estimate these database-specific parameters using a "*randomized maximum likelihood*" approach. Based on our analytical models and the parameter estimation methods, we then present an end-to-end *quality-aware* optimization approach that estimates the parameter values during execution and selects efficient execution strategies to meet user-specific quality constraints. Our quality-aware optimization approach quickly identifies whether the current execution plan is the best possible, or whether there are faster execution plans that can output a relation that satisfies the given quality constraints.

In summary, the contributions of this paper are organized as follows:

- In Section 2, we provide the necessary notation and background.
- In Section 3, we formally define the problem of estimating the quality of an extraction output, we show how to use ROC analysis for modeling an extraction system, and show how to select the Pareto-optimal set of configurations.
- In Section 4, we present our statistical modeling of multiple document retrieval strategies and examine their effect on output quality and execution time.
- In Section 5, we describe our maximum-likelihood approach that estimates on the fly the necessary parameters from the database, and in Section 6, we described a *quality-aware* optimizer that picks the fastest execution plan that satisfies given quality and time constraints.
- In Sections 7 and 8, we describe the settings and the results of our experimental evaluation, that includes multiple extraction systems and multiple real data sets.

Finally, Section 9 discusses related work and Section 10 concludes.

2 Notation and Background

We now introduce the necessary notation (Section 2.1) and briefly review various document retrieval strategies for information extraction (Section 2.2).

2.1 Basic Notation

In general, an information extraction system E processes documents from a text database D. The documents are retrieved from D using a document retrieval strategy, which is either query- or scan-based (see Section 2.2). The extraction system E, after processing a document d from D, extracts a set of tuples that are either *good* or *bad*.[1] Hence, the database documents—with respect to a set of information extraction systems—contain two disjoint set of tuples: the set T_{good} of *good* tuples and the set T_{bad} of *bad* tuples among the collective pool of tuples generated by the extraction systems.

The existence (or not) of good and bad tuples in a document, also separates the documents in D into three disjoint sets: the *good* documents D_g, the *bad* documents D_b, and the *empty* documents D_e. Documents in D_g contain at least one good tuple (and potentially bad tuples); documents in D_b do not contain any good tuples but contain at least one bad tuple; documents in D_e do not contain any tuples. Figure 2 illustrates this partitioning of database documents and tuples for an extraction task. Ideally we want to process only good documents; if we also process empty documents, the execution time increases but the quality remains unaffected; if we process bad documents, we increase not only the execution time but we worsen the quality of the output as well.

Finally, since a tuple t may be extracted from more than one document, we denote with $gd(t)$ and $bd(t)$ the number of distinct documents in D_g and D_b, respectively, that contain t. We summarize our notation in Table 1.

[1] The goodness of tuples is defined exogenously; for example ⟨*Microsoft*,*Redmond*⟩ is a good tuple, while ⟨*Microsoft*,*New York*⟩ is a bad one.

Table 1: Notation used in this paper

Symbol	Description
E	extraction system
S	retrieval strategy
D	database of text documents
D_g	*good* documents in D, i.e., documents that "contain" at least one good tuple
D_b	*bad* documents in D, i.e., documents with bad tuples and without good tuples
D_e	*empty* documents in D, i.e., documents with no good or bad tuples
D_r	documents retrieved from D
T_{good}	*good* tuples in the text database
T_{bad}	*bad* tuples in the text database
T_{retr}	tuples extracted from D_{proc} using E
$gd(t)$	number of distinct documents in D_g that contain t
$bd(t)$	number of distinct documents in D_b that contain t
$\boldsymbol{\theta}$	configuring parameter(s) of the extraction system E
$tp(\boldsymbol{\theta})$	true positive rate of E for configuring parameter $\boldsymbol{\theta}$
$fp(\boldsymbol{\theta})$	false positive rate of E for configuring parameter $\boldsymbol{\theta}$

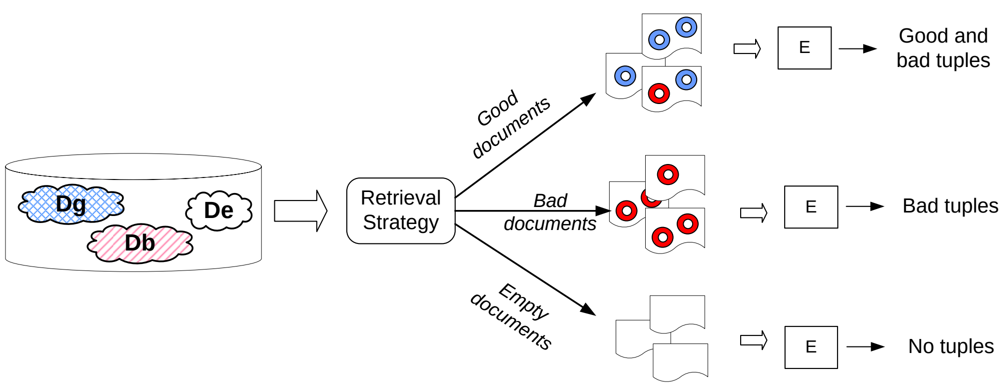


Figure 2: Partitioning database documents to analyze an extraction task.

2.2 Retrieval Strategies

In the previous section, we introduced the notion of *good* and *bad* tuples and the notion of *good*, *bad*, and *empty* documents. As mentioned, a good retrieval strategy does not retrieve from the database any *bad* or *empty* documents, and focuses on retrieving *good* documents that contain a large number of good tuples. Multiple retrieval strategies have been used in the past [4] for this task; below, we briefly review a set of representative strategies that we analyze further in Section 4:

- **Scan** is a scan-based strategy that retrieves and processes sequentially each document in the database D. While this strategy is guaranteed to process all *good* documents, it is inefficient, especially when the number of *bad* and *empty* documents is large. Furthermore, by processing a large number of *bad* documents, the *Scan* strategy may introduce many bad tuples in the output.

- **Filtered Scan** is a refinement of the basic *Scan* strategy. Instead of processing naively all the retrieved documents, *Filtered Scan* strategy [7, 21] uses a document classifier to decide whether a document is *good* or not. By avoiding processing bad documents, the *Filtered Scan* method is generally more efficient than *Scan*, and tends to have fewer bad tuples in the output. However, since the classifier may also erroneously reject *good* documents, *Filtered Scan* also demonstrates a higher number of false negatives.

- **Automatic Query Generation** is a query-based strategy that attempts to retrieve *good* documents from the database via querying. The *Automatic Query Generation* strategy sends queries to the database that are expected to retrieve *good* documents. These queries are learnt automatically, during a training stage, using a machine learning algorithm [4]. *Automatic Query Generation* tends to retrieve and process only a small subset of the database documents, and hence has a relatively large number of false negatives.

Ipeirotis et al. [24, 25] analyzed these strategies and showed how to compute the fraction of *all* tuples that each strategy retrieves over time. The analysis in Ipeirotis et al. [24, 25] implicitly assumed that the output of the extraction system is perfect, i.e., that the extraction system E extracts all the tuples from a processed document, and that all the extracted tuples are good. Unfortunately, this is rarely the case. In the rest of the paper, we show how to extend the work in [24, 25] to incorporate quality estimation techniques in an overall optimization framework.

2.3 Query Execution Strategy and Execution Time

We define the combination of a document retrieval strategy S and an information extraction system E, configured using a set of parameter settings $\boldsymbol{\theta}$, as an *execution strategy*. To compare the cost of alternative execution strategies, we define the execution time of an execution strategy, which is the total time required to generated the desired output quality. Specifically, we define the execution time for an execution strategy S over database D as:

$$\text{Time}(S, D) = \left(\sum_{d \in D_r} (t_R(d) + t_F(d)) + \sum_{d \in D_{proc}} t_E(d) + \sum_{q \in Q_{sent}} t_Q(q) \right) \tag{1}$$

where

- *D_r is the set of documents retrieved from D,*
- *$t_R(d)$ is the time to retrieve document d from D,*
- *$t_F(d)$ is the time to filter document d retrieved from D,*
- *D_{proc} is the set of documents processed using extraction system E with configuration θ,*
- *$t_E(d)$ is the time to process document d using extraction system E with configuration θ,*
- *Q_{sent} is the set of queries sent to D,*
- *$t_Q(q)$ is the time to process query q on D.*

We can simplify the above equation[2] by assuming that the time to retrieve, filter, or process a document is constant across documents (i.e., $t_R(d) = t_R$, $t_F(d) = t_F$, $t_E(d) = t_E$) and that the time to process a query is constant across queries (i.e., $t_Q(q) = t_Q$). So,

$$\text{Time}(S, D) = (|D_r| \cdot (t_R + t_F) + |D_{proc}| \cdot t_E + |Q_{sent}| \cdot t_Q) \tag{2}$$

2.4 Problem Statement

Given the definitions above, we can now describe the general form of our problem statement. Our goal is to analyze the quality characteristics of an extraction task when using *tunable* information extraction systems, coupled with various document retrieval strategies. More formally, we focus on the following problem:

Problem 2.1 *Consider a relation R along with a set of appropriately trained information extraction systems, each with its own set of possible parameter configurations, and a set of document retrieval strategies. Estimate the number $|T_{retr}^{good}|$ of* good *tuples and the number $|T_{retr}^{bad}|$ of* bad *tuples in the output, generated by each extraction system, under each possible configuration, for each retrieval strategy, and the associated execution time for each execution strategy.*

[2] Even though this simplification may seem naive, if we assume that the each of the times $t_R(d)$, $t_F(d)$, $t_E(d)$, and $t_Q(q)$ follows a distribution with finite variance, then we can show using the central limit theorem that our simplifying approximation is accurate if t_R, t_F, t_E, and t_Q are the mean values of these distributions.

This problem statement is very generic and can subsume a large number of query processing objectives. For example, in a typical problem setting, we try to minimize the execution time, while satisfying some quality constraints of the output (e.g., in terms of false positives and false negatives). Alternatively, we may try to maximize the quality of the output under the some constraint on the execution time. Yet another approach is to maximize recall, keeping the precision above a specific level, under a constraint in execution time. Many other problem specifications are possible. Nevertheless, given the number of good and bad tuples in the output along with the execution time required to generate that output, we can typically estimate everything that is required for alternative problem specifications.

3 Characterizing Output Quality

We begin our discussion by showing how to characterize, in a statistically robust way, the behavior of a *stand-alone* information extraction system. In Section 3.1, we explain why the traditional precision-recall curves are not well-suited for this purpose and describe the alternative notion of *receiver operating characteristics (ROC) curves*. Then, in Section 3.2, we show how to construct an ROC curve for an extraction system and how to use the ROC analysis to select only the Pareto optimal set of configurations across *a set* of extraction systems. Finally, in Section 3.3 we present our concept of *quality curves* that connect ROC curves and document retrieval strategies.

3.1 ROC Curves

One of the common ways to evaluate an extraction system E is to use a *test set* of documents, for which we already know the set T_{good} of correct tuples that appear in the documents. Then, by comparing the tuples T_{extr} extracted by E with the correct set of tuples T_{good}, we can compute the *precision* and *recall* of the output as:

$$precision = \frac{|T_{extr} \cap T_{good}|}{|T_{extr}|} \quad , \quad recall = \frac{|T_{extr} \cap T_{good}|}{|T_{good}|}$$

Typically, an extraction system E has a set of parameters $\boldsymbol{\theta}$ that can be used to make E precision- or recall-oriented, or anything in between. By varying the parameter values $\boldsymbol{\theta}$, we can generate configurations of E with different precision and recall settings, and generate a set of precision-recall points that, in turn, can generate the "best possible"[3] *precision-recall curve*. The curve demonstrates the tradeoffs between precision and recall for the given extraction system. Unfortunately, precision-recall curves are not statistically robust measures of performance, and depend heavily on the ratio of *good* and *bad* documents in the test set, as shown by the following example.

Example 2 *Consider an extraction system E that generates a table of companies and their headquarters locations,* Headquarters(Company, Location) *from news articles in* The New York Times *archive. To measure the performance of E, we test the system by processing a set of documents from the "*Business*" and the "*Sports*" section. The "*Business*" documents contain many tuples for the target relation, while "*Sports*" documents do not contain any. The information extraction system works well, but occasionally extracts spurious tuples from some documents, independently of their topic. If the test set contains a large number of "*Sports*" documents then the extraction system will also generate a large number of incorrect tuples from these "*bad*" documents, bringing down the precision of the output. Actually, the more "*Sports*" documents in the test set, the worse the reported precision, even though the underlying extraction system remains the same. Notice, though, that the recall is not affected by the document distribution in the test set and remains constant, independently of the number of "*Sports*" documents in the test set.*

The fact that precision depends on the distribution of *good* and *bad* documents in the test set is well-known in machine learning, from the task of classifier evaluation [32]. To evaluate classifiers, it is preferable to use ROC curves [14], which are independent of the class distribution in the test set. We review ROC curves next.

Receiver operating characteristic (ROC) curves were first introduced in the 1950's, where they were used to study the performance of radio receivers to capture a transmitted signal in the presence of noise. In a more general setting, ROC curves evaluate the ability of a decision-making process to discriminate true positives (signal) in the input, from true negatives (noise). An ROC model assumes that signal and noise follow some probability distributions across a decision variable x, which can be used to discriminate between the signal and noise. Figure 3 demonstrates a simple

[3] Since there is no guarantee that changes in $\boldsymbol{\theta}$ will have a monotonic effect in precision and recall, some settings may be strongly dominated by others and will not appear in the "best possible" precision-recall curve.

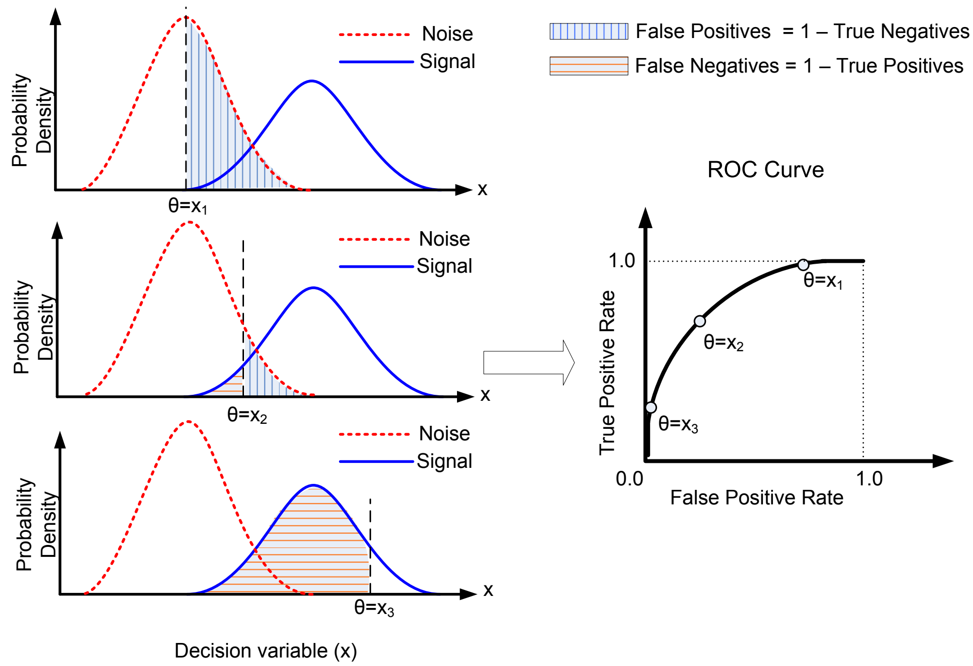


Figure 3: Characterizing decision-making task as a threshold picking process, for a simple scenario where changing the value of a configuring parameter results in a smooth tradeoff between true positives and false negatives.

decision-making process under this scenario, with a simple parameter. We classify an event as “noise” whenever the decision variable $x < \theta$ and as “signal” when $x \geq \theta$. By varying the value of decision threshold θ, the ability of detecting signal from noise varies. For instance, for $\theta = x_1$, the system does not classify any event as noise, and has a *high true positive rate*; at the same time, a significant fraction of the noise is classified incorrectly as signal, generating a *high false positive rate*. Analogously, for $\theta = x_3$, the system has *low false positive rate*, but also classifies significant fraction of the signal as noise, resulting in a system with *low true positive rate* as well.

The ROC curves summarize graphically the tradeoffs between the different types of errors. When characterizing a binary decision process with ROC curves, we plot the *true positive rate* tp (the fraction of positives correctly classified as positives, i.e., recall) as the ordinate, and the *false positive rate* fp (the fraction of negatives incorrectly classified as positives) as the abscissa. An ideal binary decision maker has $tp = 1$ and $fp = 0$; a random binary decision maker lies anywhere on the line $x = y$. The ROC curves have strong statistical properties and are widely adopted as performance evaluation metrics in a variety of areas, including machine learning [32], epidemiology [15], signal detection theory [14], and others.

3.2 Generating ROC Curves for an Information Extraction System

Given that ROC curves are more robust than precision-recall curves, it would be natural to use ROC curves for characterizing the performance and tradeoffs of different extraction systems. In principle, information extraction tasks can also be viewed as a decision-making task: each document contains a set of *good and bad tuples* and some parameter variable(s) are used to decide which of these tuples should appear in the output. However, there are some challenges that need to be addressed before using ROC curves for information extraction:

1. To define each ROC point, i.e., $tp(\boldsymbol{\theta})$ and $fp(\boldsymbol{\theta})$, we need to know the set of *all* possible good and bad tuples in each document. While we can conceivably locate all the good tuples in the document, the universe of bad tuples is in principle infinite.

Input: extraction system E, gold standard T_{gold}, test set D_t, range of values for θ
Output: ROC curve: $\{tp(\boldsymbol{\theta}), fp(\boldsymbol{\theta})\}$ for all values of θ
$\textbf{Result} = \emptyset$;
Retrieve documents D_t in the test set;
/* Identify all candidate tuples T */
$T = \emptyset$;
foreach *document d in D_t* **do**
    Extract tuples $t(d)$ from d, using E with the maximum-sensitivity setting;
    $T = T \cup t(d)$;
end
$T_{good} = T \cap T_{gold}$; $T_{bad} = T \setminus T_{gold}$;
/* Compute false positive and true positive rates for all values of θ */
foreach *value v of θ* **do**
    $T_r = \emptyset$;
    foreach *document d in D_t* **do**
        Extract tuples $t(d)$ from d, using E with $\theta = v$;
        $T_r = T_r \cup t(d)$;
    end
    $tp(v) = \frac{|T_r \cap T_{good}|}{|T_{good}|}$; $fp(v) = \frac{|T_r \cap T_{bad}|}{|T_{bad}|}$;
    $\textbf{Result}[v] = \{tp(v), fp(v)\}$;
end
return $\textbf{Result}$

Figure 4: Generating an ROC curve for an information extraction system.

2. Even after computing a point in the ROC curve, this is simply an instance of the performance in the test set, and does not reveal the confidence bounds for each of the $tp(\boldsymbol{\theta})$ and $fp(\boldsymbol{\theta})$ values.

3. Finally, an information extraction system offers multiple parameter knobs and the behavior of these knobs may be non-monotonic; thus, the simple decision process listed in Figure 3 does not describe the process anymore.

To solve the first problem, we need to find a way to measure the $fp(\boldsymbol{\theta})$ rate. We cannot measure the ratio of the bad tuples that appear in the output if we do not know the total number of bad tuples. To define each ROC point, i.e., $tp(\boldsymbol{\theta})$ and $fp(\boldsymbol{\theta})$, we need to know the set of *all* possible good and bad tuples that serve as normalizing factors for $tp(\boldsymbol{\theta})$ and $fp(\boldsymbol{\theta})$, respectively. For our work, we operationalize the definition of $tp(\boldsymbol{\theta})$ and $fp(\boldsymbol{\theta})$ using a *pooling*-based approach: we define the set of good and bad tuples as the set of tuples extracted by an extraction system across all possible configurations of the extraction system. In practice, we estimate the $tp(\boldsymbol{\theta})$ and $fp(\boldsymbol{\theta})$ values using a "test set" of documents and a set of "ground truth" tuples.[4]

Using the pooling-based approach, we can proceed to generate the ROC curve for an extraction system. We first need to generate the probability distributions for signal and noise across a decision variable θ of choice. Figure 4 describes the ROC construction algorithm. The first step is to use a test set of documents D_t, and a set of "gold standard" tuples T_{gold} that are correct and comprehensive (e.g., extracted by manually inspecting the documents in D_t). Then, to construct the ROC curve for an extraction system E, we begin with identifying the "maximum-sensitivity" setting of $\boldsymbol{\theta}$: this is the value(s) of $\boldsymbol{\theta}$ at which E extracts as many tuples (good and bad) as possible. Using the maximum-sensitivity setting of E, we extract all possible candidate tuples T (good and bad); by examining the intersection of T with T_{gold}, we identify all the good tuples T_{good} (signal) and the bad tuples T_{bad} (noise) that appear in D_t. The sets T_{good} and T_{bad} can then be used to estimate the true positive rate $tp(\boldsymbol{\theta})$ and the false positive rate $fp(\boldsymbol{\theta})$ for each $\boldsymbol{\theta}$ value: to achieve this, we simply examine how many of the T_{good} and T_{bad} tuples are kept in the output, for different $\boldsymbol{\theta}$ values. This leads us to the definition:

[4] An alternative solution would be to use the "*Free Response ROC (FROC) curves*," in which the ordinate remains un-normalized and corresponds to the average number of bad tuples generated by each document. However, we will see in Section 4 that the probabilistic interpretation of $tp(\boldsymbol{\theta})$ and $fp(\boldsymbol{\theta})$ in normal ROC curves is handy for our analysis.

Definition 3.1 [ROC Curve] A *receiver operating characteristic (ROC) curve* for an information extraction system E is a set of $\langle tp(\boldsymbol{\theta}), fp(\boldsymbol{\theta}), Time(\boldsymbol{\theta})\rangle$ values, where $tp(\boldsymbol{\theta})$ is the *true positive rate*, $fp(\boldsymbol{\theta})$ is the *false positive rate*, and $Time(\theta)$ is the mean time required to process a document when the configuring parameters of E are set to θ. We define as $tp(\boldsymbol{\theta})$ the probability of classifying a tuple $t \in T_{good}$ as good; similarly, we define as $fp(\boldsymbol{\theta})$ the probability of classifying a tuple $t \in T_{bad}$ as good. □

The $\langle tp(\boldsymbol{\theta}), fp(\boldsymbol{\theta})\rangle$ points of the ROC curve derived using the procedure above have one disadvantage: they do not offer any information about the robustness of the $tp(\boldsymbol{\theta})$ and $fp(\boldsymbol{\theta})$ estimates. Hence, they describe the performance of the extraction system E on the particular test set, used for the construction of the ROC curve, but do not reveal the robustness of these estimates. To provide confidence bounds for each $tp(\boldsymbol{\theta})$ and $fp(\boldsymbol{\theta})$ point, we use a 10-fold cross validation approach [17, 28]: When constructing the ROC curve, we split the test set into 10 partitions, and generate 10 different values for the $tp(\boldsymbol{\theta})$ and $fp(\boldsymbol{\theta})$ estimates for each setting $\boldsymbol{\theta}$. Using the set of these values we then generate the confidence bounds for each $\langle tp(\boldsymbol{\theta}), fp(\boldsymbol{\theta})\rangle$ point.

Finally, we need to address the issue of multiple parameters and of the non-monotonic behavior of some of these parameters. The definition of the ROC curve given above is rather agnostic to the behavior of each parameter. Using the algorithm of Figure 4, we generate an $\langle tp(\boldsymbol{\theta}), fp(\boldsymbol{\theta})\rangle$ point for each setting $\boldsymbol{\theta}$. Some of these points may be strongly dominated[5] by other points; since the strongly dominated points are guaranteed to generate a suboptimal execution, we simply ignore them and keep only the Pareto-optimal triplets $\langle tp(\boldsymbol{\theta}), fp(\boldsymbol{\theta}), Time(\boldsymbol{\theta})\rangle$ for the computation of the ROC curve. (This is similar to the construction of an ROC convex hull [32] but in our case we do not generate interpolated points between the Pareto optimal triplets.) Extending the Pareto optimal approach to multiple of extraction systems, we can easily generate a single ROC curve that contains only the non-dominated configurations across all systems.

An important characteristic of the ROC curves for our purpose is that, knowing the number of good and bad *documents* that E processes, we can compute the number of good and bad *tuples* in the output. (We will show that in more detail in Section 4.) Of course, the number of good and bad *documents* processed by E depends on the document retrieval strategy. We discuss this next.

3.3 Quality Curves for an Execution Strategy

In Sections 3.1 and 3.2, we discussed how an ROC curve can describe the behavior of an extraction system when extracting tuples from a single document. What we are interested in, though, is to summarize the behavior of an extraction system when coupled with a specific retrieval strategy. If the retrieval strategy retrieves many bad documents, then the extraction system also generates a large number of bad tuples, and similarly, the extraction system generates a large number of good tuples if the strategy retrieves many good documents. Thus, the output composition for an execution strategy at a given point in time depends on the choice of retrieval strategy and of the extraction system and its configuration $\boldsymbol{\theta}$.

To characterize the output of an execution strategy, we define the concept of a *quality curve*. A quality curve of an extraction system coupled with a retrieval strategy describes all possible compositions of the output at a given point in time, when the extraction system, configured at setting $\boldsymbol{\theta}$ processes documents retrieved by the associated retrieval strategy. Specifically, we define quality curves as:

Definition 3.2 [Quality Curve(E, R, P)] The *quality curve* of an extraction system E, characterized by the triplets $\langle tp(\boldsymbol{\theta}), fp(\boldsymbol{\theta}), Time(\boldsymbol{\theta})\rangle$, coupled with a retrieval strategy R is a plot of the number of good tuples as a function of number of bad tuples, at the point in time P, for all available extraction systems E and all possible values[6] of the parameter(s) $\boldsymbol{\theta}$. □

Figure 5 illustrates the concept of quality curves. The quality curves contains all the different possible outcomes that can be achieved at a given point in time, by picking different extraction systems E_i and different settings $\boldsymbol{\theta}_j$. For example, consider the point in time p_1. If we pick system E_1 and set it to its maximum sensitivity setting, we are able to retrieve approximately 1,800 good tuples and 2,500 bad tuples. Alternatively, under the most conservative setting for E_1, again at time p_1, we extract 1,100 good and 1,100 bad tuples. Another choice is to pick a different extraction system, E_2, which is much slower, but more accurate. In this case, at time p_1 the system extracts 400 good tuples, and only 100 bad tuples. The quality curve, demonstrates clearly the tradeoffs under the different settings. As time

[5] A triplet $\langle tp(\boldsymbol{\theta}), fp(\boldsymbol{\theta}), Time(\boldsymbol{\theta})\rangle$ strongly dominates a triplet $\langle tp(\boldsymbol{\theta}'), fp(\boldsymbol{\theta}'), Time(\boldsymbol{\theta}')\rangle$ *iff* $tp(\boldsymbol{\theta}) \geq tp(\boldsymbol{\theta}')$, $fp(\boldsymbol{\theta}) \leq fp(\boldsymbol{\theta}')$, and $Time(\boldsymbol{\theta}) \leq Time(\boldsymbol{\theta}')$.

[6] An alternative is to keep only the Pareto optimal set, as discussed in Section 3.2.

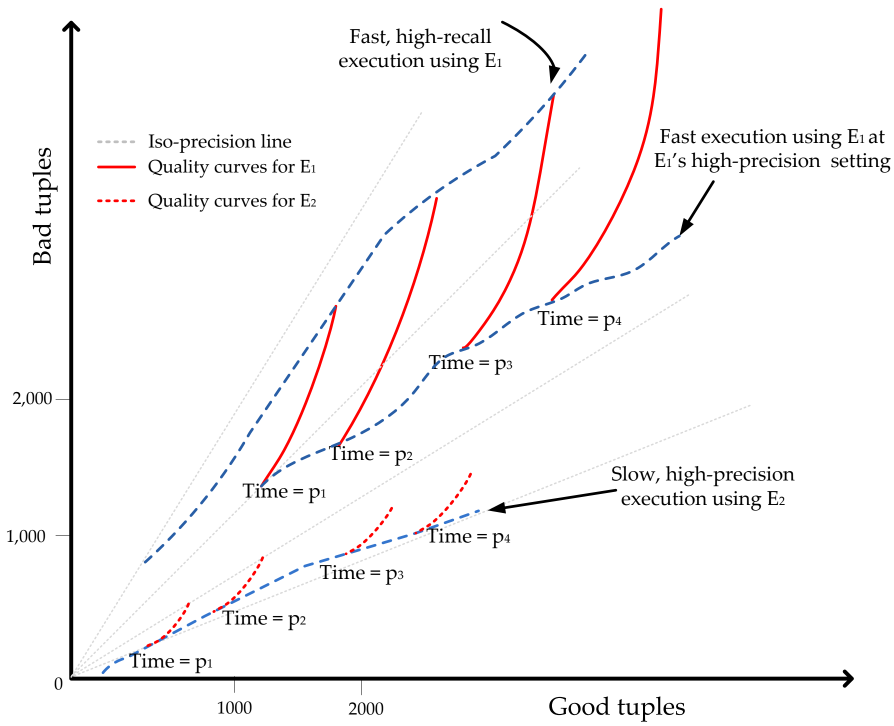


Figure 5: Quality curves for a retrieval strategy for different point in time, for two extraction systems.

progresses, and the extraction system processes more documents, the quality curve moves up and to the right, generating more bad and good tuples.

Our goal is to estimate the shape of the quality curves for each point in time, describing essentially the behavior of all possible execution strategies. Given the quality curves, we can then easily pick the appropriate strategy for a given extraction task, Next, we present our formal analysis of these execution strategies, and we show how to estimate the quality curves.

4 Estimating Output Quality

We begin our analysis by sketching a general model to study the output of an execution strategy in terms of the number of good and bad tuples generated (Section 4.1). We then examine how the choice of the parameters $\boldsymbol{\theta}$ affects the output composition (Section 4.2), and finally present our rigorous analysis of each retrieval strategy, namely, *Scan* (Section 4.3.1), *Filtered Scan* (Section 4.3.2), and *Automatic Query Expansion* (Section 4.3.3).

4.1 Analyzing An Execution Strategy: General Scheme

Consider an execution strategy with extraction system E, configured with parameters values $\boldsymbol{\theta}$, along with a document retrieval strategy S for a text database D. Our goal is to determine the number of good tuples $|T_{retr}^{good}|$ and bad tuples $|T_{retr}^{bad}|$ that this execution strategy will generate at any point in time. Based on these values, we can compute the quality curve for the combination of E with S.

We know that the database consists of *good* documents D_g, *bad* documents D_b, and *empty* documents D_e. During the information extraction task, the strategy S retrieves documents D_r from D that E subsequently processes. Specifically, E processes $|D_{gp}|$ *good* documents, $|D_{bp}|$ *bad* documents, and $|D_{ep}|$ *empty* documents.

In the first step of our analysis, we disentangle the effects of retrieval strategy from the effects of the extraction system. For the number of retrieved good tuples $|T_{retr}^{good}|$, we proceed as follows: The number of good tuples in the extracted relation depends only on the number of good documents $|D_{gp}|$ that are processed by E. The value $|D_{gp}|$ depends only on the retrieval strategy S. Given the number $|D_{gp}|$, the number of good tuples depends only on the settings of the extraction system. Assuming that we know the value of $|D_{gp}|$, we have:

$$E[|T_{retr}^{good}|] = \sum_{t \in T_{good}} Pr_g(t \big| |D_{gp}|) \tag{3}$$

where $Pr_g(t \big| |D_{gp}|)$ is the probability that we will see the *good* tuple t at least once in the extracted relation, after processing $|D_{gp}|$ good documents. The value $Pr_g(t \big| |D_{gp}|)$ depends only on the extraction system and in Section 4.2 we will analyze it further.

The analysis is similar for the number of retrieved bad tuples. In this case, since both *good* and *bad* documents contain bad tuples, the number of bad tuples in the extracted relation depends on the total number of good documents and bad documents $|D_{gp}| + |D_{bp}|$ processed by E. Specifically, we have:

$$E[|T_{retr}^{bad}|] = \sum_{t \in T_{bad}} Pr_b(t \big| |D_{gp}| + |D_{bp}|) \tag{4}$$

where $Pr_b(t \big| |D_{gp}| + |D_{bp}|)$ is the probability that we will see the *bad* tuple t at least once in the extracted relation, after processing a total of $|D_{gp}| + |D_{bp}|$ good and bad documents. The value $Pr_b(t \big| |D_{gp}| + |D_{bp}|)$ depends only on the extraction system and in Section 4.2 we will analyze it further.

Equations 3 and 4 rely on knowing the *exact* number of the good and bad documents retrieved using S and processed by E. In practice, however, we will only know the probability distribution of the good and bad documents in D_r, which is different for each retrieval strategy. Therefore, after modifying the Equations 3 and 4 to reflect this, we have:

$$E[|T_{retr}^{good}|] = \sum_{i}^{|T_{good}|} \cdot \sum_{j=0}^{|D_r|} Pr_g(t \big| |D_{gp}| = j) \cdot Pr(|D_{gp}| = j) \tag{5}$$

$$E[|T_{retr}^{bad}|] = \sum_{i}^{|T_{bad}|} \cdot \sum_{j=0}^{|D_r|} Pr_b(t \big| |D_{gp}| + |D_{bp}| = j) \cdot Pr(|D_{gp}| + |D_{bp}| = j) \tag{6}$$

The values of $E[|T_{retr}^{good}|]$ and $E[|T_{retr}^{bad}|]$ for different extraction strategies $\langle E(\boldsymbol{\theta}), S\rangle$ allow us to compute the quality curves for different number of retrieved documents, and hence for different points in time. Furthermore, we have disentangled the effect of the extraction system $E(\boldsymbol{\theta})$ from the effect of the document retrieval strategy S.

We now proceed, in Section 4.2, to analyze the factors $Pr_g(t|j)$ and $Pr_b(t|j)$ that depend only on the extraction system. Then, in Sections 4.3.1, 4.3.2, and 4.3.3 we show how to compute the factors $Pr(|D_{gp}| = j)$ and $Pr(|D_{gp}| + |D_{bp}| = j)$ for various document retrieval strategies.

4.2 Analyzing the Effect of the Information Extraction System

In this section, we examine the effect of the information extraction system on output quality. For our analysis, we assume that we know the values of $|D_{gp}|$ and $|D_{bp}|$. We will relax this assumption in the next sections.

The first step is to estimate the number of *distinct good tuples* that we extract. As we discussed in Section 2, we can extract good tuples only from good documents (see also Figure 2, page 4). To estimate the number of good tuples that are extracted from the retrieved documents, we model each retrieval strategy as *multiple sampling without replacement* processes running over the documents in D_g. Each process corresponds to a tuple $t \in T_{good}$, which we assume to be independently distributed across the D_g documents. If we retrieve and process $|D_{gp}|$ documents from D_g then the probability of retrieving k documents that contain a good tuple t that appears in $gd(t)$ good documents follows a hypergeometric distribution. Specifically, the probability of retrieving k documents with the tuple t is $Hyper(|D_g|, |D_{gp}|, gd(t), k)$ where $Hyper(D, S, g, k) = \binom{g}{k} \cdot \binom{D-g}{S-k} / \binom{D}{S}$ is the hypergeometric distribution.

Even if we retrieve k documents with tuple t from D_g, the extraction system E may still reject the tuple k times[7] with probability $(1 - tp(\boldsymbol{\theta}))^k$. In this case, the tuple t will *not* appear in the output. Therefore, the probability that

[7] We assume that the appearances of t in different documents are independent, e.g., they do not always follow the same pattern.

we will see a *good* tuple t, which appears in $gd(t)$ good documents in D, at least once in the extracted relation, after processing $|D_{gp}|$ good documents is equal to:

$$Pr_g(t \big| |D_{gp}|) = 1 - \sum_{k=0}^{gd(t)} \left(\mathit{Hyper}(|D_g|, |D_{gp}|, gd(t), k) \cdot (1 - \mathit{tp}(\boldsymbol{\theta}))^k \right)$$

To compute Pr_g, we need to know the value of $gd(t)$ for the good tuple, which is rarely known. However, the distribution $Pr(gd(t))$ tends to follow a power-law distribution [5, 24]. So, we can eliminate $gd(t)$ and have a general formula for all good tuples t:

$$Pr_g(t \big| |D_{gp}|) = 1 - \sum_{gd(t)=1}^{|D_g|} Pr(gd(t)) \cdot \sum_{k=0}^{gd(t)} \left(\mathit{Hyper}(|D_g|, |D_{gp}|, gd(t), k) \cdot (1 - \mathit{tp}(\boldsymbol{\theta}))^k \right) \quad (7)$$

The analysis is similar for the number of bad tuples. However, now both D_g and D_b contain bad tuples. By assuming that the level of noise is the same in D_g and D_b, and analogously to the case of good tuples, the probability that we will see at least once a bad tuple, which appears in $gd(t)$ good documents and in $bd(t)$ bad documents in D, is:

$$Pr_b(t \big| |D_{gp}| + |D_{bp}|) = 1 - \sum_{k=0}^{gd(t)+bd(t)} \left(H_b(gd(t) + bd(t), k) \cdot (1 - \mathit{fp}(\boldsymbol{\theta}))^k \right) \quad (8)$$

where $H_b(gd(t)+bd(t), k) = \mathit{Hyper}(|D_g|+|D_b|, |D_{gp}|+|D_{bp}|, gd(t)+bd(t), k)$ and $(1-\mathit{fp}(\boldsymbol{\theta}))^k$ is the probability of rejecting a bad tuple k times. Again, as in the case of good tuples, we can eliminate the dependency of $gd(t) + bd(t)$ by assuming that the frequency of bad tuples also follows a probability distribution. (As we will see in Section 7, the frequency of bad tuples also follows a power-law distribution.)

In this section, we have described how to compute the values Pr_g and Pr_b that are needed to estimate $E[|T_{retr}^{good}|]$ (Equation 5) and $E[|T_{retr}^{bad}|]$ (Equation 6). Next, we show how to compute the values for $Pr(|D_{gp}| = j)$ and $Pr(|D_{gp}| + |D_{bp}| = j)$ for each document retrieval strategy.

4.3 Analyzing the Effect of the Document Retrieval Strategy

4.3.1 Scan

Scan sequentially retrieves documents from D, in no specific order, and therefore, when *Scan* retrieves $|D_r|$ documents $|D_{gp}|$, $|D_{bp}|$, and $|D_{ep}|$ are random variables that follow the hypergeometric distribution. Specifically, the probability of processing exactly j good documents is:

$$Pr(|D_{gp}| = j) = \mathit{Hyper}(|D|, |D_r|, |D_g|, j) \quad (9)$$

Similarly, the probability of processing j good and bad documents is:

$$Pr(|D_{gp}| + |D_{bp}| = j) = \mathit{Hyper}(|D|, |D_r|, |D_g| + |D_b|, j) \quad (10)$$

Using the equations above in 5, we compute the expected number of good tuples in the extracted relation after *Scan* retrieves and processes $|D_r|$ documents from D.

4.3.2 Filtered Scan

The *Filtered Scan* retrieval strategy is similar to *Scan* with the exception of a document classifier that filters out documents that are not good candidates for containing good tuples. Instead, only documents that *survive* the classification step will be processed. Document classifiers are not perfect and they are usually characterized by their own *true positive rate* C_{tp} and *false positive rate* C_{fp}. Intuitively, given a classifier C, the true positive rate C_{tp} is the fraction of documents in D_g classified as good, and the false positive rate C_{fp} is the fraction of documents in D_b incorrectly classified as good. Therefore, the major difference with *Scan* is that now the probability of processing j *good* documents after retrieving $|D_r|$ documents from the database is:

$$Pr(|D_{gp}| = j) = \sum_{n=0}^{|D_r|} Hyper(|D|, |D_r|, |D_g|, n) \cdot Binom(n, j, C_{tp}) \tag{11}$$

where $Binom(n, k, p) = \binom{n}{k} \cdot p^k \cdot (1-p)^{n-k}$ is the binomial distribution. In Equation 11, n is the number of retrieved good documents. By definition, the remaining $|D_r| - n$ are bad or empty documents. So, by extending Equation 11 we can compute the probability of processing j *good* documents *and bad* documents after retrieving $|D_r|$ documents from the database:

$$Pr(|D_{gp}| + |D_{bp}| = j) = \sum_{i=0}^{j} \left(\sum_{n=0}^{|D_r|} H_g(n) \cdot Binom(n, i, C_{tp}) \cdot \sum_{m=0}^{|D_r|-n} H_b(m) \cdot Binom(m, j-i, C_{fp}) \right) \tag{12}$$

where $H_g(n) = Hyper(|D|, |D_r|, |D_g|, n)$ and $H_b(m) = Hyper(|D|, |D_r|, |D_b|, m)$.

By replacing the value of $Pr(|D_{gp}| = j)$ from Equation 12 to Equation 5, we can easily compute the expected number of distinct good tuples $E[|T_{retr}^{good}|]$ in the output. Similarly, for the bad tuples, we use Equation 12 to compute $Pr(|D_{gp}| + |D_{bp}| = j)$ and then replace this value in Equation 6 to compute the expected number of bad tuples in the output.

4.3.3 Automated Query Generation

The *Automated Query Generation* strategy retrieves documents from D by issuing queries, constructed offline and designed to retrieve mainly *good* documents [24]. The retrieved documents are then processed by E.

To estimate the number of *good* and *bad* documents retrieved, consider the case where *Automated Query Generation* has sent Q queries to the database. If the query q retrieves $g(q)$ documents and has precision $p_g(q)$ for *good* documents, i.e., expected fraction of *good* documents, then the probability for a *good* document to be retrieved by q is $\frac{p_g(q) \cdot g(q)}{|D_g|}$. The query q may also retrieve some bad documents. If the expected fraction of *bad* documents retrieved by q is $p_b(q)$, then the probability of a *bad* document to be retrieved by q is $\frac{p_b(q) \cdot g(q)}{|D_b|}$. Assuming that the queries sent by *Automated Query Generation* are only biased towards documents in D_g, the queries are conditionally independent within D_g. In this case, the probability that a *good* document d is retrieved by at least one of the Q queries is:

$$Pr_g(d) = 1 - \prod_{i=1}^{Q} \left(1 - \frac{p_g(q_i) \cdot g(q_i)}{|D_g|} \right)$$

Similarly, the probability that a *bad* document d is retrieved by at least one of the Q queries:

$$Pr_b(d) = 1 - \prod_{i=1}^{Q} \left(1 - \frac{p_b(q_i) \cdot g(q_i)}{|D_b|} \right)$$

To avoid having any dependencies on query-specific cardinalities $g(q)$ and precisions $p_g(q)$ and $p_b(q)$, we can compute the expected value for $Pr_g(d)$ and $Pr_b(d)$:

$$\begin{aligned} Pr_g(d) &= 1 - \left(1 - \frac{E[p_g(q)] \cdot E[g(q)]}{|D_g|} \right)^Q \\ Pr_b(d) &= 1 - \left(1 - \frac{E[p_b(q)] \cdot E[g(q)]}{|D_b|} \right)^Q \end{aligned} \tag{13}$$

where $E[p_g(q)]$ and $E[p_b(q)]$ are the average precisions of a query for *good* and *bad* documents, respectively, and $E[g(q)]$ is the average number of documents retrieved by a query.

Since each document is retrieved independently of each other, the number of good documents retrieved (and processed) follows a binomial distribution, with $|D_g|$ trials and $Pr_g(d)$ probability of success in each trial. (Similarly for the bad documents.)

$$Pr(|D_{gp}| = j) = Binom(|D_g|, j, Pr_g(d)) \tag{14}$$

$$Pr(|D_{bp}| = k) = Binom(|D_b|, k, Pr_b(d)) \tag{15}$$

Therefore,

$$Pr(|D_{gp}| + |D_{bp}| = j) = \sum_{i=0}^{j} Pr(|D_{gp}| = i) \cdot Pr(|D_{bp}| = j - i) \tag{16}$$

Similar to *Scan* and *Filtered Scan*, we can now estimate the values of $E[|T_{retr}^{good}|]$ and $E[|T_{retr}^{bad}|]$.

5 Estimating Model Parameters

In Section 4, we developed analytical models to derive quality curves for an extraction system, for different retrieval strategies. We now discuss the task of estimating parameters used by our analysis.

To estimate the quality curves, our analysis relies on two classes of parameters, namely the retrieval-strategy-specific parameters and the database-specific parameters. The retrieval-strategy-specific parameters include $E[p_g(q)], E[p_b(q)]$, and $E[h(q)]$ for the *Automatic Query Expansion* queries or the classifier properties C_{tp} and C_{fp} for *Filtered Scan*. The database-specific parameters include $|D_g|$, $|D_b|$, and $|D_e|$, $|T_{good}|$ and $|T_{bad}|$, and the frequency distribution of the good and bad tuples in the database. Of these two classes, the retrieval-strategy-specific parameters can be easily estimated in a pre-execution, offline step: the classifier properties and the query properties are typically estimated using a simple testing phase after their generation [24, 25]. On the other hand, estimating the database-specific parameters is a more challenging task.

Our parameter estimation process relies on the general principles of *maximum-likelihood estimation (MLE)* [18] along with the statistical models that we discussed earlier: in Section 4, we showed how to estimate the output *given various database parameters*, and now we will infer the values of the database parameters *by observing the output for a sample of database documents*. Specifically, we begin with retrieving and processing a sample D_r of documents from the database D. After processing the documents in D_r, we observe some tuples along with their frequencies in these retrieved documents. To this end, we identify the values for the database parameters that are *most likely* to generate these observations. Specifically, given a tuple t obtained from D_r, if we observe t in $s(t)$ documents in D_r, we are trying to find the parameters that maximize the likelihood function:

$$\mathcal{L}(parameters) = \prod_{t \in \textit{observed tuples}} Pr\{s(t)|parameters\} \tag{17}$$

To effectively estimate the database-specific parameters, we need to address one main challenge: our understanding of an execution strategy so far assumed that we know exactly whether a tuple is good or not (Section 4). However, in a typical execution, we do not have such knowledge; at best, we have a probabilistic estimate on whether a tuple is good or bad. In our parameter estimation framework, we decouple the issue of estimating parameters from the issue of determining whether an observed tuple is good or bad. Specifically, we present our estimation process by first assuming that we know whether a tuple is good or bad (Section 5.1). Then, we alleviate this (non-realistic) assumption and present two parameter estimation approaches. Our first approach, called *rejection-sampling-based MLE-partitioning*, randomly splits the tuples into good and bad following a *rejection-sampling* strategy, and then estimates the database parameters (Section 5.2). Our second approach, *preserves this uncertainty* about the "goodness" or "badness" of a tuple and simultaneously derives all the database parameters (Section 5.3).

5.1 Estimation Assuming Complete Knowledge

Our parameter estimation process begins with retrieving and processing documents using some execution strategy. After processing the retrieved documents D_r, we observe some tuples along with their document frequencies. Furthermore, for now we assume that we know for each observed tuple whether it is a *good* tuple or a *bad* tuple. Given this assumption, we show how to derive the parameters $|D_g|$, $|D_b|$, and $|D_e|$, and then we discuss how to estimate the tuples frequencies for the *good* and *bad* tuples, and the values $|T_{good}|$, and $|T_{bad}|$.

Estimating $|D_g|, |D_b|$, and $|D_e|$: We begin by first identifying the good, the bad, and the empty documents in D_r. For this, we process each document in D_r using the maximum-sensitivity setting of the extraction system E in the initial execution strategy.[8] Based on the type of tuples contained in each processed document, we can trivially compute the number of good documents $|D_{gp}|$, the number of bad documents $|D_{bp}|$, and the number of empty documents $|D_{ep}|$, in D_r. These values, together with our analysis in Section 4, can be used to derive the values for $|D_g|$, $|D_b|$, and $|D_e|$ in the entire database $|D|$. We now show how we derive $|D_g|$. (The derivation for $|D_b|$ and $|D_e|$ is analogous.) Using a maximum-likelihood approach, we find the value for $|D_g|$ that maximizes the probability of observing $|D_{gp}|$ good documents in D_r:

$$Pr\{|D_g| \big| |D_{gp}|\} = \frac{Pr\{|D_{gp}| \big| |D_g|\} \cdot Pr\{|D_g|\}}{Pr\{|D_{gp}|\}} \tag{18}$$

Since the value $Pr\{|D_{gp}|\}$ is constant across all possible values for $|D_g|$, we can ignore this factor for the purpose of maximization. From Section 4, we know how to derive the factor $Pr\{|D_{gp}| \big| |D_g|\}$ for each document retrieval strategy. (See Equations 9, 11, and 14.) Specifically, for *Scan*, we know that $Pr\{|D_{gp}| \big| |D_g|\} = Hyper(|D|, |D_r|, |D_g|, |D_{gp}|)$ (Eq. 9). Finally, for the factor $Pr\{|D_g|\}$ we assume a uniform distribution, i.e., no prior knowledge about the number of good and bad documents in the database. We can now derive the value for $|D_g|$ that maximizes Equation 18. For instance, for *Scan*, we derive $|D_g|$ as:

$$|D_g| = \underset{|D_g|}{argmax} \left\{ Hyper(|D|, |D_r|, |D_g|, |D_{gp}|) \right\} \tag{19}$$

Analytically,[9] the maximizing value of D_g is the solution for the equation $\digamma(D_g+1) + \digamma(D - D_g - D_r + D_{gp} + 1) = \digamma(D_g - D_{gp} + 1) + \digamma(D - D_g + 1)$, where $\digamma(x)$ is the digamma function. Practically, $\digamma(x) \approx \ln(x)$, and we have:

$$|D_g| \approx \left(\frac{|D| + 2}{|D_r|} \cdot |D_{gp}| \right) - 1 \tag{20}$$

Following a similar MLE-based approach, we can derive values for $|D_b|$ and $|D_e|$ using our analysis from Section 4 and the observed values $|D_{bp}|$ and $|D_{ep}|$.

Estimating β_g and β_b: The next task is to estimate the tuple-related parameters. One of the fundamental parameters required by our analysis is the frequency of each tuple in the database (e.g., $gd(t)$ and $bd(t)$ for a tuple t). Of course, we cannot know the frequency of the tuples before processing all documents in the database, but we may know the general family of their frequency distributions. Following such a parametric approach, our estimation task reduces to estimating a few parameters for these distributions. We rely on the fact that the tuple frequencies for both categories of tuples (i.e., good and bad) tend to follow a power-law distribution (see related discussion in Section 7). Intuitively, for both categories, a few tuples occur very frequently and most tuples occur rarely in the database.

For a random variable X that follows a power law distribution, the probability mass function for X is given as $Pr\{X = i\} = \frac{i^{\beta}}{\zeta(\beta)}$, where β is the exponent parameter of the distribution and $\zeta(\beta) = \sum_{n=1}^{\infty} n^{-\beta}$ is the Riemann zeta function [19]. Therefore, for the random variable $gd(t)$, which represents the frequency of a good tuple, and the random variable $gd(t) + bd(t)$, which represents the frequency of a bad tuple, we have:

$$Pr\{gd(t) = i\} = \frac{i^{\beta_g}}{\zeta(\beta_g)} \tag{21}$$

$$Pr\{gd(t) + bd(t) = i\} = \frac{i^{\beta_b}}{\zeta(\beta_b)} \tag{22}$$

where β_g and β_b are the exponents of the power-law distributions for the frequencies of good tuples and bad tuples, respectively. Now, we need to derive the values for the distribution parameters, namely, β_g and β_b. Below, we discuss our approach for estimating β_g; the estimation of β_b is analogous.

Uncertainty-preserving Maximum Likelihood: To derive β_g, we focus on the set T_{gr} of good tuples observed in D_r. For a good tuple t, we denote by $gs(t)$ the total number of documents that contain t in D_r. Our goal is to estimate the value of β_g that maximizes the likelihood of observing $gs(t)$ times each of the extracted tuples t, which is given as:

$$\mathcal{L}(\beta_g) = \prod_{t \in \textit{observed good tuples}} Pr\{gs(t) | \beta_g\} \tag{23}$$

[8] For our discussion, we assume that we have available only one extraction system, but our estimation process can be easily extended for a set of extraction systems.

[9] We set $\frac{d}{dD_g} Hyper(|D|, |D_r|, |D_g|, |D_{gp}|) = 0$ and use the fact that $\frac{d}{dn} n! = \digamma(n+1) \cdot n!$.

We have:

$$Pr\{gs(t)|\beta_g\} = \sum_{k=gs(t)}^{|D_g|} Pr\{gs(t)\big|k\} \cdot Pr\{gd(t) = k|\beta_g\} \tag{24}$$

We derive the factor $Pr\{gs(t)\big|k\}$ using our analysis from Section 4.2 by generalizing Equation 7. In Equation 7, we derived the probability of observing a good tuple *at least once in the output*, after processing $|D_{gp}|$ good documents. Now we are interested in deriving the probability of observing a good tuple $gs(t)$ *times* in the output after we have processed $|D_{gp}|$ good documents. Therefore,

$$Pr\{gs(t)\big|k\} = \sum_{m=0}^{k} (Hyper(|D_g|, |D_{gp}|, k, m) \cdot Binom(m, gs(t), tp(\boldsymbol{\theta}))) \tag{25}$$

For the factor $Pr\{gd(t) = k|\beta_g\}$, we use Equation 21. We can then estimate the value of β_g using Equations 23, 24, and 25.

Since it is difficult to derive an analytic solution for locating the value of β_g that maximized $\mathcal{L}(\beta_g)$, we proceed and compute $\mathcal{L}(\beta_g)$ for a set of values of β_g and pick the value that maximizes Equation 24. We refer to this estimation approach that exhaustively searches through the space of parameter values as *Exh*.

Iterative Maximum Likelihood: The exhaustive approach tends to be rather expensive computationally, since it examines all potential $gd(t)$ values for each tuple and then searches for the best possible value of β_g. Rather than searching through a space of parameter values, we also explored *iteratively refining* the estimated values for β_g. This alternative estimation approach iterates over the following two steps until the value for β_g has converged:

Step 0 **Initialize β_g:** We pick an initial value for β_g.

Step 1 **Estimate tuple frequencies, $gd(t)$:** In this step, for every good tuple t in D_r, we estimate its frequency in the *entire database*, i.e., we derive $gd(t)$ for t, based on its sample frequency $gs(t)$. In contrast to the uncertainty-preserving approach described above, we keep only a *single* value for $gd(t)$. Specifically, we identify the value for $gd(t)$ that maximizes the probability of observing the tuple frequency $gs(t)$ in the sample:

$$Pr\{gd(t)\big|gs(t)\} = \frac{Pr\{gs(t)\big|gd(t)\} \cdot Pr\{gd(t)\}}{Pr\{gs(t)\}} \tag{26}$$

We derive $Pr\{gs(t)\big|gd(t)\}$ and $Pr\{gd(t)\}$ as discussed above for the uncertainty-preserving approach. Notice that $Pr\{gd(t)\}$ depends on the value of β_g.

Step 3 **Estimate distribution parameter, β_g:** In this step, we estimate the most likely distribution parameter β_g that generates the tuple frequencies estimated in Step 2. We derive β_g by *fitting a power law*. We explore two methods to fit a power law: the maximum likelihood (MLE)-based approach [19, 30] and a less-principled (but extensively used) log regression-based (LogR) method [1, 30]. We refer to the iterative estimation method that uses MLE-based fitting as *Iter-MLE*, and we refer to the estimation method that uses log regression-based fitting as *Iter-LogR*.

Step 4 **Check for convergence of β_g:** If the β_g values computed in two iterations of the algorithm are close, then stop. Otherwise, repeat Step 2 and 3.

Estimating $|T_{good}|$ and $|T_{bad}|$: The final step in the parameter estimation process is to estimate $|T_{good}|$ and $|T_{bad}|$, for which we numerically solve Equations 5 and 6. Specifically, we rewrite Equations 5 and 6 as:

$$E[|T_{retr}^{good}|] = |T_{good}| \cdot \sum_{j=0}^{|D_r|} Pr_g(t_i\big||D_{gp}| = j) \cdot Pr(|D_{gp}| = j) \tag{27}$$

$$E[|T_{retr}^{bad}|] = |T_{bad}| \cdot \sum_{j=0}^{|D_r|} Pr_b(t_i\big||D_{gp}| + |D_{bp}| = j) \cdot Pr(|D_{gp}| + |D_{bp}| = j) \tag{28}$$

During the estimation process we know the number of good tuples observed after processing D_r; this is essentially $E[|T_{retr}^{good}|]$ in Equation 27. Furthermore, we can derive the probability Pr_g of observing a good tuple, after retrieving

Input: Tuple t, $\boldsymbol{\theta}_o$ setting used to generate t
R = generate a random number between 0 and 1
if $R < \frac{|T_{good}|}{|T_{good}|+|T_{bad}|} \cdot \frac{sig(\boldsymbol{\theta}_o)}{sig(\boldsymbol{\theta}_o)+nse(\boldsymbol{\theta}_o)}$ **then**
  mark t as good
else
  mark t as bad
end

Figure 6: Classifying an observed tuple t as a good or bad tuple.

D_r documents, using the estimated values for β_g and $|D_g|$. The only unknown in Equation 27 is $|T_{good}|$. So, we solve Equation 27 for $|T_{good}|$. We can derive $|T_{bad}|$ is the same manner using the observed bad tuples, i.e., $E[|T_{retr}^{bad}|]$, and Pr_b in Equation 28.

To summarize, we showed how we can estimate the various database-specific parameters used in our analysis. Our discussion so far assumed that we had complete knowledge of whether an observed tuple is good or not. In practice, though, we do not know this. We now relax that assumption and discuss two methods to address this issue.

5.2 Rejection-sampling-based MLE-Partitioning Approach

When estimating parameters of a database, we retrieve and process a document sample using some execution strategy. Upon observing a tuple in the output we do not know whether this tuple is good or bad. This assumption, however, is the basis for the analysis that we presented so far. In this section, we show how we can alleviate this assumption by using a technique based on the idea of *rejection sampling*. Intuitively, this technique randomly splits the set of extracted tuples into good and bad, by using the ROC analysis from Section 3.2 and then uses the randomized split to perform the analysis.

The basic idea behind this technique is that we do not necessarily need to know the absolutely correct classification for each tuple. If we have a roughly correct classification of the tuples into good and bad, then the analysis presented above should be able to handle the noise and still return good results.

Consider a tuple t generated, during the parameter estimation process, using an execution strategy consisting of an extraction system E tuned at setting $\boldsymbol{\theta}_o$. This tuple may be good or bad. This depends on two main factors: (a) the ability of E to differentiate between *good* tuples (signal) and *bad* tuples (noise), and (b) the prior distribution of the *good* and *bad* tuples that we feed to E. (Intuitively, the more E can correctly identify a good tuple, the higher the probability of an output tuple being a *good* tuple; similarly, the larger the number of *good* tuples that we feed to E the higher the number of observed tuples that will be good.) Instead of preserving the uncertainty about the tuple, for estimation purposes, we can make a decision and consider it either good or bad. To make this decision, we use the idea of rejection sampling and classify tuple t at random, using a *biased* coin.

An important part for generating a representative split of tuples into good and bad is to select properly the bias of the coin. This bias depends on the ability of the extraction system to distinguish signal from noise event. As discussed in Section 3.2, the first step to generate an ROC curve is to derive the probability distributions for signal and noise across all $\boldsymbol{\theta}$ settings. For each setting, we know sig($\boldsymbol{\theta}$), which is the fraction of all good tuples that are generated at $\boldsymbol{\theta}$ setting, and nse($\boldsymbol{\theta}$), which is the fraction of bad tuples generated at $\boldsymbol{\theta}$ setting. Therefore, for a tuple the probability that it is good (signal) is $\frac{\text{sig}(\boldsymbol{\theta})}{\text{sig}(\boldsymbol{\theta})+\text{nse}(\boldsymbol{\theta})} \cdot \frac{|T_{good}|}{|T_{good}|+|T_{bad}|}$ and we use that as a basis for the split. Figure 6 shows our overall process for classifying a tuple t observed at setting $\boldsymbol{\theta}_o$ using a biased coin.

One issue with this value is that we do not know the $|T_{good}|$ and $|T_{bad}|$ values during the estimation process. So, we begin with some initial value for $\frac{|T_{good}|}{|T_{good}|+|T_{bad}|}$ and split the observed tuples based on this initial value. Using these partitioned tuples, we proceed with the estimation process as discussed in Section 5.1 and derive values for $|T_{good}|$ and $|T_{bad}|$; then, we update our initial guess. As we retrieve and process more documents, we further refines this value.

Using the above partitioning approach, we generate a *deterministic* split of *all* the observed tuples, and we can now proceed with the estimation process as detailed in Section 5.1. In principle, our technique is similar to Monte Carlo techniques [18], but instead of trying multiple potential splits we simply take one; we observed that a single partition tends to work well in practice and is more efficient than multiple attempts.

5.3 Uncertainty-Preserving Approach

In the absence of any signal to noise ratio information or any other mechanism to partition the tuples, we can extend our general theory from Section 4 to estimate the parameters only based on the tuples observed in a document sample. Our second estimation approach preserves the uncertainty about the nature of a tuple, and estimates the desired parameters by exhaustively considering all possible scenarios involving each observed tuple.

Given a document d that "contains" $t_0(d)$ tuples that we observe using the maximum-sensitivity setting (see Section 5.1), we denote by $g(d)$ the total number of good tuples and by $b(d)$ the total number of bad tuples in d, such that $g(d) + b(d) = t_0(d)$. We do not know the exact values for $g(d)$ and $b(d)$, and so we examine an entire range of possible values for $g(d)$ and $b(d)$ given $t_0(d)$. Specifically, we consider all (x, y) pairs such that $(x, y) \in g(d) \times b(d)$. Our goal then is to identify the parameter values that maximize the probability of observing the tuples for all documents, for the given (x, y) pairs for each document. For efficiency, without loss of accuracy, we focus only on the most likely "breakdown" of the tuples observed for each document instead of all possible breakdowns.

Estimating $|D_g|, |D_b|, |D_e|$: We first identify the most likely breakdown of the observed tuples for each document. Consider a document d that contains $t_0(d)$ tuples, of which we have observed $s(d)$ tuples using the initial execution strategy. Our goal is to identify the most likely values for $g(d)$ and $b(d)$ that generated $s(d)$ tuples after processing d. Using an MLE-based approach, we identify $g(d)$ and $b(d)$ that maximize:

$$Pr\{g(d) = x, b(d) = y|s(d) = t, (x + y) = t_0(d)\} = \frac{Pr\{s(d)|g(d) = x, b(d) = y\} \cdot Pr\{g(d) = x\} \cdot Pr\{b(d) = y\}}{Pr\{s(d)\}} \quad (29)$$

We derive the first factor in the numerator, $Pr\{s(d)\big|g(d) = x, b(d) = y\}$, based on our discussion from Section 4. Specifically, we know that the number of tuples extracted by the extraction system at $\boldsymbol{\theta}$ setting follows a binomial distribution with the probability of success given as $tp(\boldsymbol{\theta})$. Similarly, the number of bad tuples extracted by the extraction system depends on $fp(\boldsymbol{\theta})$. So,

$$Pr\{s(d)|x, y\} = \sum_{g=0}^{s(d)} Binom\left(g, x, tp(\boldsymbol{\theta})\right) \cdot Binom\left((s(d) - g), y, fp(\boldsymbol{\theta})\right)$$

To compute the probability that a document d contains $g(d)$ good tuples, we rely on prior knowledge of the document frequency distribution, which tends to be power law. (We verified this experimentally.) If β_{gd} is the distribution parameter for the frequency of *good* tuples in a good document, we derive $Pr\{g(d) = x\}$ as:

$$Pr\{g(d) = x\} \quad = \quad \begin{cases} \frac{|D_b|}{|D|} + \frac{|D_e|}{|D|}, & x = 0 \\ \left(1 - \frac{|D_b|}{|D|} - \frac{|D_e|}{|D|}\right) \cdot \frac{x^{-\beta_{gd}}}{\zeta(\beta_{gd})}, & x > 0 \end{cases}$$

Similarly, to compute the probability that a document d contains $b(d)$ bad tuples, we also assume that the number of *bad* tuples in a document tend to follow a power law distribution. If β_{bd} is the distribution parameter for the frequency of *bad* tuples in a document, we derive $Pr\{b(d) = y\}$ as:

$$Pr\{b(d) = y\} \quad = \quad \begin{cases} \frac{|D_e|}{|D|}, & y = 0 \\ \left(1 - \frac{|D_e|}{|D|}\right) \cdot \frac{y^{-\beta_{bd}}}{\zeta(\beta_{bd})}, & y > 0 \end{cases}$$

Finally, we compute the factor $Pr\{s(d)\}$ in the denominator of Equation 29 based on the observed distribution for the number of tuples in a document, after processing D_r. Using the above analysis, we search through a space of possible values for $|D_g|, |D_b|, |D_e|, |\beta_{gd}|$, and $|\beta_{bd}|$ and identify the most likely parameter combination.

Estimating $|\beta_g|, |\beta_b|, |T_{good}|, |T_{bad}|$: For the next task, we begin with identifying the most likely frequency of each tuple t observed after processing D_r documents. As discussed in Section 5.1, the frequency of a tuple observed for D_r may not be final. Consider a tuple t that occurs in $s(t)$ documents among D_r. Our goal is to identify the most likely tuple frequency $d(t)$ of the tuple t in the entire database that maximizes the probability of observing t $s(t)$ times:

$$\begin{aligned} Pr\{d(t) = k\big|s(t)\} \quad &= \quad Pr\{t \in T_{good}|s(t)\} \cdot Pr\{gd(t) = k|s(t)\} \\ &+ \quad Pr\{t \in T_{bad}|s(t)\} \cdot Pr\{(gd(t) + bd(t)) = k|s(t)\} \end{aligned} \quad (30)$$

Symbol	Description
PT-Exh	Rejection-sampling-based approach that exhaustively searches through a range of values for the tuple frequency distribution parameters.
PT-Iter-MLE	Rejection-sampling-based approach that iteratively refines the tuple frequency distribution parameter values and uses MLE-based approach to fit the power law.
PT-Iter-LogR	Rejection-sampling-based approach that iteratively refines the tuple frequency distribution parameter values and uses log-based regression to fit the power law.
UP	Uncertainty-preserving approach that exhaustively considers all possible cases for each observed tuple.

Table 2: Techniques to estimate the database-specific parameters.

For brevity, we denote the first factor related to the case of good tuples by P_{og}, i.e., $P_{og} = Pr\{t \in T_{good}|s(t)\} \cdot Pr\{gd(t) = k|s(t)\}$; similarly we denote the second factor related to the case of bad tuples by P_{ob}. Using Bayes rule, we rewrite P_{og} in terms of values that we can derive using our analysis in Section 4. Specifically,

$$P_{og} = Pr\{s(t)|t \in T_{good}\} \cdot \frac{Pr\{t \in T_{good}\}}{Pr\{s(t)\}} \cdot Pr\{s(t)|gd(t) = k\} \cdot \frac{Pr\{gd(t) = k\}}{Pr\{s(t)\}} \tag{31}$$

The above equation consists of five distinct quantities of which we can derive two quantities using our earlier analysis. Specifically, we discussed how to derive $Pr\{s(t)|t \in T_{good}\}$ and $Pr\{s(t)|gd(t) = k\}$ by generalizing Equation 7 in Section 5.1. To compute $Pr\{t \in T_{good}\}$ in Equation 31, we use $\frac{|T_{good}|}{|T_{good}|+|T_{bad}|}$, and to compute $Pr\{gd(t) = k\}$, we follow Equation 21. Finally, to derive $Pr\{s(t)\}$ in the denominator, we rely on the observed frequency distribution after processing D_r documents.

So far, we discussed how to derive the quantity P_{og} for the case of good tuples. We proceed in a similar fashion for P_{ob} by generalizing our analysis from Section 4.2 to use $gd(t) + bd(t)$ as random variables and derive the probability of observing a bad tuple i times after processing D_r documents. Using the above derivations, we search through a range of values for β_g, β_b, $|T_{good}|$, and $|T_{bad}|$, and pick the combination that maximizes Equation 30. We can optimize this estimation process by focusing on typical values of β_g and β_b, which tend to be between 1 and 5; similarly, we can derive useful hints for the range of possible values for $|T_{good}|$ and $|T_{bad}|$ using the proportion of good and bad tuples observed when generating the ROC curves (Section 3.1).

To summarize, in this section we discussed our approach to estimate various database-specific parameters that are necessary for our analysis in Section 4, by exploring several ways for deriving these parameters. We summarize these methods in Table 2. These methods along with our analysis naturally lead to building a *quality-aware* optimization approach that can compare a family of execution strategies and effectively pick an execution strategy that meets given user-specified quality constraints. Next, we discuss our *quality-aware* optimizer, which builds on our analytical models.

6 Putting Everything Together

In Section 3, we introduced the concept of quality curves which characterize the output of an execution strategy (i.e., combination of retrieval strategy and an extraction system setting) over time. These curves allow us to compare different execution strategies, both in terms of speed and in terms of output composition, i.e., the number of good and bad tuples in the output. In Section 4, we showed how we can estimate the quality curves for an execution strategy, given a set of database parameters. Finally, in Section 5, we presented various methods that estimate the necessary parameters for our analysis given the output of the running execution strategy.

Using the analysis so far, we can outline the overall optimization strategy:

- Given the quality requirements, and in the absence of any real statistics about the database, pick an execution strategy, based on heuristics or based on some "educated guesses" for the parameter values.
- Run the execution strategy, observing the generated output.
- Use the algorithms of Section 5 to estimate the parameter values.

- Use the analysis of Section 4 to estimate the quality curves, examining whether there is a better execution strategy than the running one.
- Switch to a new execution strategy, or continue with the current one; go to Step 2.

In principle, the quality requirements of Step 1 depend on user preferences: sometimes users may be after "quick-and-dirty" results, while some other times users may be after high-quality answers that may take long time to produce. For this paper, as a concrete case of user-specified preferences, we focus on a "low-level" quality requirement where users specify the desired quality composition in terms of the minimum number τ_g of *good* tuples and the maximum number τ_b number of bad tuples that they are willing to tolerate. Even though, it may seem unrealistic to ask users to specify such values. However, several other cost functions can be designed on top of this "low-level" model: examples include minimum "precision," or minimum "recall" or even a goal to maximize a combination of the precision and recall within a pre-specified execution time budget.

Given the user-specified requirements, τ_g and τ_b, our quality-aware optimizer identifies execution strategies and execution times that have $E[|T_{retr}^{good}|] \geq \tau_g$ and $E[|T_{retr}^{bad}|] \leq \tau_b$. Then across the candidate strategies, the one with the minimum execution time is picked, following the general optimization outline that described above.

7 Experimental Settings

We now describe our experimental settings for the experiments in Section 8, focusing on the text collections, extraction systems, retrieval strategies, and baseline techniques used.

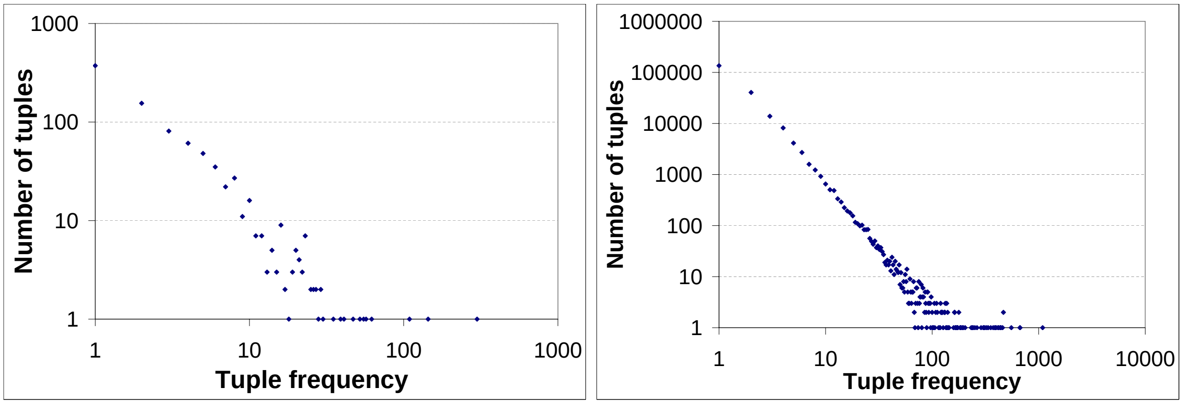
Information extraction systems: We used Snowball [3] and trained it for three relations: *Headquarters(Company, Location)*, from Section 1, *Executives(Company, CEO)*, and *Mergers(Company, MergedWith)*. For *Executives*, the extraction system generates tuple $\langle o, e\rangle$, where e is the CEO of the organization o, whereas for *Mergers*, the extraction system generates tuples $\langle o, m\rangle$, where organization o merged with the organization m. In our discussion, we focus only on the case of extracting *Headquarters* and *Executives*; our observations on *Mergers* largely agree with those for these two relations. We trained two instantiations of Snowball for each relation that differed in their extraction patterns. We refer to the extraction systems for *Executives* as E_1 and E_2, and to the extraction systems for *Headquarters* as H_1 and H_2. For θ, we picked *minSimilarity*, a tuning parameter exposed by Snowball, which is the threshold for the similarity between the terms in the context of a candidate tuple and terms in the extraction patterns learned for an extraction task.

Data set: We used three data sets for our experiments, namely, a collection of 135,438 newspaper articles from The New York Times from 1996 (*NYT96*), a collection of 50,269 documents from The New York Times from 1995 (*NYT95*), and a collection of 98,732 newspaper articles from The Wall Street Journal (*WSJ*). We used *NYT96* as the *training* set to learn extraction patterns, and train the retrieval strategies. For our experiments that test the quality-aware optimizer, we used *NYT95* and *WSJ*. Since the results for *WSJ* were largely similar with the results for *NYT95*, for brevity we report only the results for *NYT95*.

Retrieval strategies: To instantiate the retrieval strategies, we used a rule-based classifier (created using Ripper [11]) for *Filtered Scan*For *Automatic Query Expansion* we used QXtract [4] that uses machine learning techniques to automatically learn queries that match documents with at least one tuple. In our case, we train QXtract to only match *good* documents, avoiding at the same time the *bad* and *empty* ones (the original QXtract avoids only the *empty* documents).

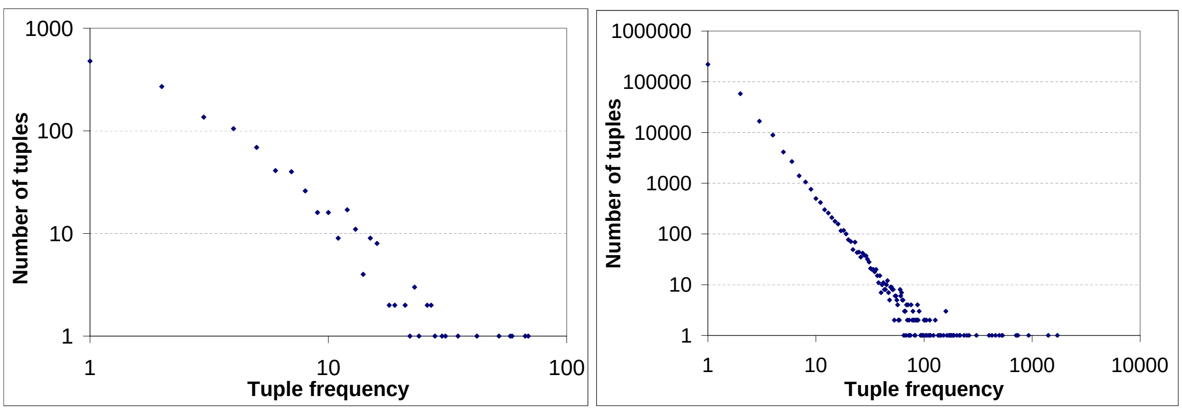
Tuple verification: Given the data sets and the retrieval strategies, we need to separate the tuples into *good* and *bad*. For this, we used *SDC Platinum*,[10], a paid service that provides authoritative information about financial governance and financial transactions. Furthermore, we retrieved additional data from *Wharton Research Data Services (WRDS)*[11] that also provides a comprehensive list of datasets that can be used to verify the correctness of the extracted tuples. For each relation and data set, we extracted all possible tuples and classified them into *good* and *bad* tuples, using the aforementioned resources. We observed that the tuple frequency distribution tends to follow a power-law for *both* good and bad tuples. Figure 7 shows the token degree distributions of both and good and bad tokens for *Headquarters* and similarly, Figure 8 shows the token frequency distributions for *Executives*.

[10] http://www.thomsonreuters.com/products_services/financial/sdc
[11] http://wrds.wharton.upenn.edu/



(a) (b)

Figure 7: (a) Good and (b) bad tuple frequency distribution for *Headquarters*.



(a) (b)

Figure 8: (a) Good and (b) bad tuple frequency distribution for *Executives*.

ROC curves: We generated the ROC curves for each extraction system, by varying the values for $minSimilarity$ from 0 to 1, using the methodology described in Section 3 to pick the Pareto-optimal points. We also used 10-fold cross-validatation to generate the confidence intervals [28] for each point. Figures 9(a) and 9(b) show the ROC curves of the extraction systems for *Headquarters* and *Executives* respectively, along with the associated confidence intervals.

Execution strategies: For a given relation, we generate execution strategies by first deriving variations of the associated extraction systems by varying values for *minSimilarity* and then combining each variation with each of the three document retrieval strategies. Overall, for each relation we have 2 extraction systems, 4 different values for *minSimilarity*, namely, 0.2, 0.4, 0.6, and 0.8, and 3 retrieval strategies, for a total of 24 different execution strategies per relation.

Baseline techniques: For our experiments, we refer to our *quality-aware* optimization approach as *Qawr*. We also generated two baseline techniques. Our first baseline uses existing work [25] that predicts the fastest execution strategy to reach a specified number of tuples. The optimizer in [25] assumes that the execution strategies only generate good tuples (see Section 2). Therefore, we give as input to this optimizer the *total tuples* needed, which is the sum of good and bad tuples, and select the fastest execution strategy using the method in [25]. We refer to this baseline as *Qign* (for "quality-ignorant").

Our second baseline technique relies on using *heuristics* from previously executed extraction tasks. Specifically, we use an extraction task as a training task, i.e., we run it first and see what execution strategies perform best for different types of quality requirements. Based on this information, we "learn" the most appropriate execution strategies for each quality requirement. Then, when faced with another extraction task, involving the extraction of a different relation, we use the same extraction strategies that performed well for the training task. We refer to this heuristics-based baseline as

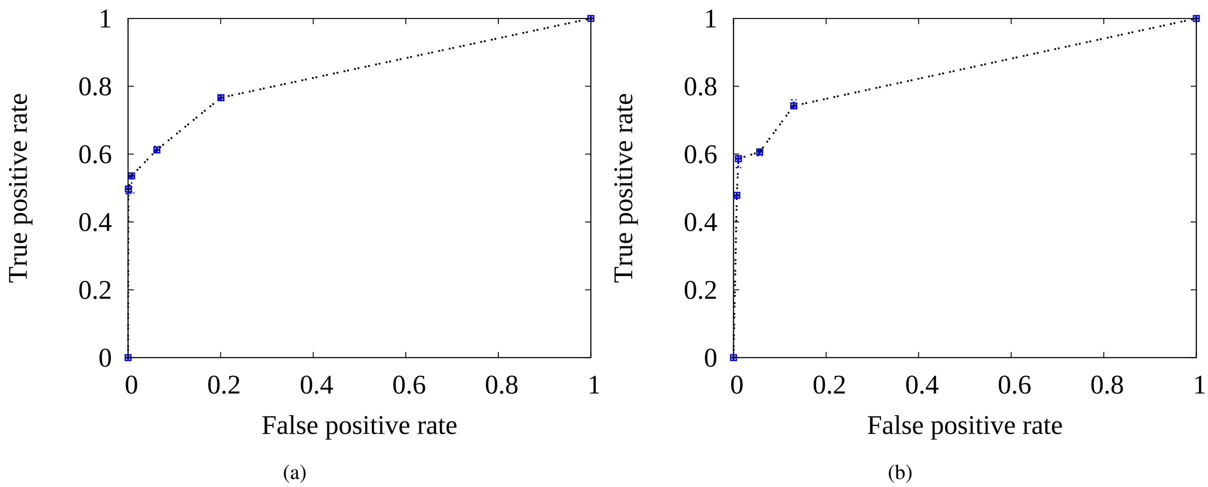


Figure 9: ROC curves for (a) *Headquarters* and (b) *Executives*.

Heur for ("heuristic").

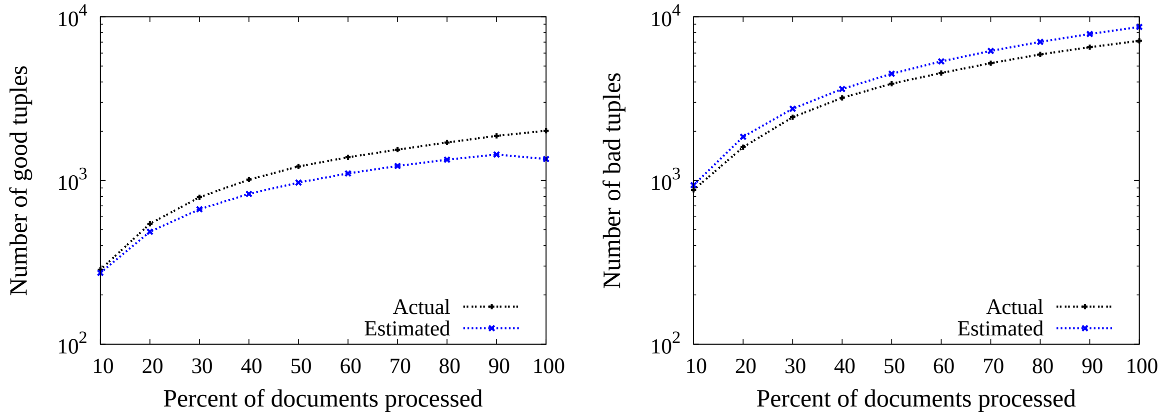
Combining manual and automated information extraction systems: To better illustrate some of the properties of our framework, we also ran experiments involving a generalized setting, where we have to make a decision between an automated extraction system and hiring people to read the documents and manually extract the target relations. Specifically, in addition to processing documents using an automated extraction systems such as Snowball, we could recruit human annotators using paid services such as Amazon Mechanical Turk. [12] In general, we expect manual extractions to be more quality-oriented than the automated extractions, but at the same time more expensive in terms of time and monetary cost. To build the "*manual extraction system*", we used the Mechanical Turk Service as follows: for any given document, we requested five annotators. The annotators had to read the entire document and identify tuple instances of *Headquarters* from the document (if any), with no limit on the maximum number of reported instances. We instructed the annotators to provide as answers only values that exist in the document, without any modifications to any entity (e.g., if a document mentions a company, say "Microsoft Corp.", the reported company name must be identical to this and not other possible variations, such as, Microsoft Corporation). We used the number of annotators that extracted a tuple as the $\boldsymbol{\theta}$: When $\boldsymbol{\theta} = 1$, we expect to see a high true positive rate but also a high false positive rate, as some annotators may erroneously extract some tuples; similarly, we expect to see a low true positive rate but also a low false positive rate when $\boldsymbol{\theta} = 5$.

Metrics: To compare the execution time of an execution plan chosen by our optimizer against a candidate plan, we measure the *relative difference in time* by normalizing the execution time of the candidate plan by that for the chosen plan. Specifically, we note the relative difference as $\frac{t_c}{t_o}$, where t_c is the execution time for a candidate plan and t_o is the execution time for the plan picked by our quality-aware optimizer.

8 Experimental Results

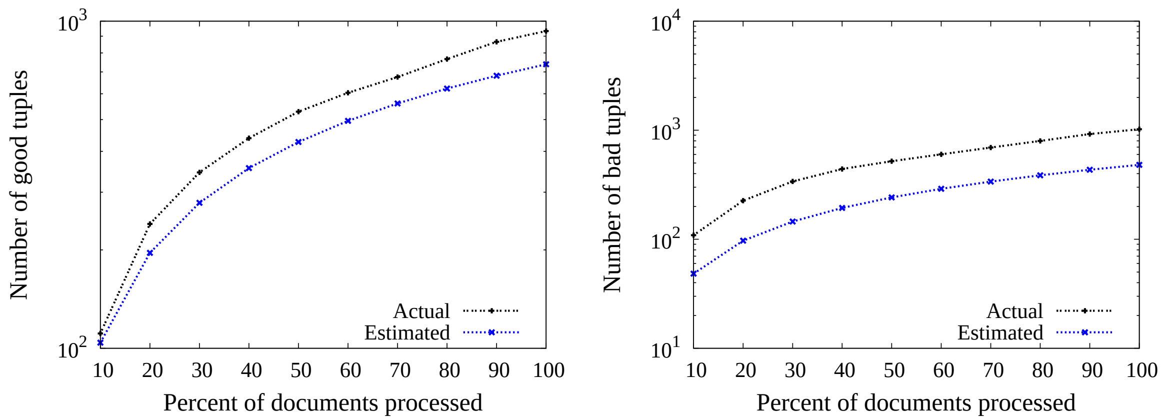
We now discuss our experimental results. Initially, we evaluate the accuracy of the models for predicting the output composition for an extraction system under different retrieval strategies, given complete information (Section 8.1). Then, we discuss the accuracy of our parameter estimation methods when we do not have information about the database parameters (Section 8.2). Subsequently, we evaluate the accuracy of our optimizer for selecting an execution plan for a desired output quality (Section 8.3) and, finally, we compare our approach against existing techniques for selecting an execution plan (Section 8.4).

[12] http://www.mturk.com



(a) (b)

Figure 10: Actual vs. estimated number of (a) good tuples and (b) bad tuples using *Scan* and H_1 with *minSimilarity* = 0.4, for *Headquarters*.



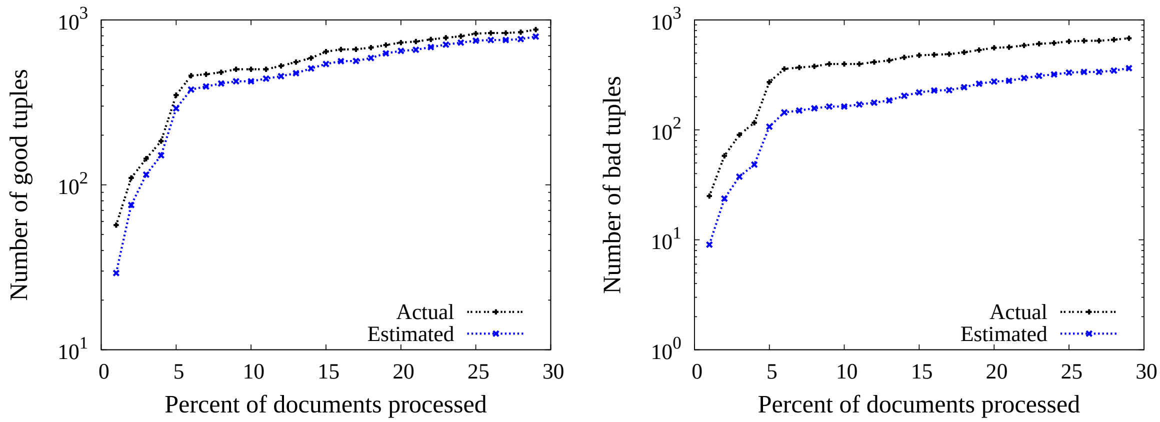
(a) (b)

Figure 11: Actual vs. estimated number of (a) good tuples and (b) bad tuples using *Filtered Scan* and H_1 with *minSimilarity* = 0.4, for *Headquarters*.

8.1 Accuracy of the Model

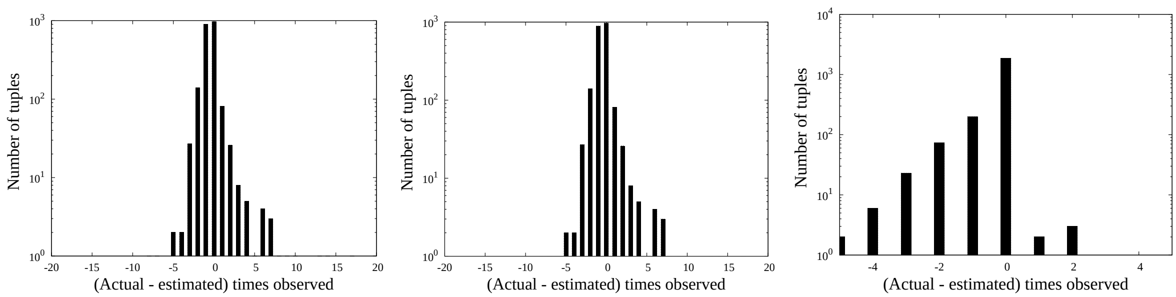
The first task of our evaluation examines the accuracy of the statistical models developed in Section 4. To verify the accuracy of our analysis, we initially assume complete knowledge of the various parameters used in the analysis. Specifically, we used the actual tuple degree distribution information along with the values for $|D_g|$, $|D_b|$, and $|D_e|$. Given a relation, for each associated execution plan we first estimate the output quality, i.e., $E[|T^{good}_{retr}|]$ and $E[|T^{bad}_{retr}|]$, using the analysis of Section 4, for varying values of $|D_r|$. Then, for each $|D_r|$ value, we measure the *actual* good and bad tuples extracted by each plan. Figure 10 shows the actual and estimated values for the good (Figure 10(a)) and bad (Figure 10(b)) tuples generated by the execution plan for *Headquarters* that uses *Scan* and H_1 with *minSimilarity* = 0.4. Figures 11 and 12 show the corresponding results for the *Automatic Query Expansion*and *Filtered Scan*retrieval strategies. In general, our estimated values are close to the actual ones, confirming the accuracy of our analysis. (The results are highly similar for other settings.)

For the analysis for the *bad* tuples for *Filtered Scan* (e.g., Figure 11(b)), our models underestimate the number of generated bad tuples, because of a modeling choice: we assume that the classifier output does not affect the probability distribution of the noise (see Section 3.1). However, this is not always true in reality. In fact, the bad documents that "survive" the classification step tend to contain bad tuples with noise distribution closer to the signal distribution; this results in higher false positive rates for the bad tuples coming from bad documents that pass the document classification



(a) (b)

Figure 12: Actual vs. estimated number of (a) good tuples and (b) bad tuples using *Automatic Query Expansion* and H_1 with *minSimilarity* = 0.4, for *Headquarters*.

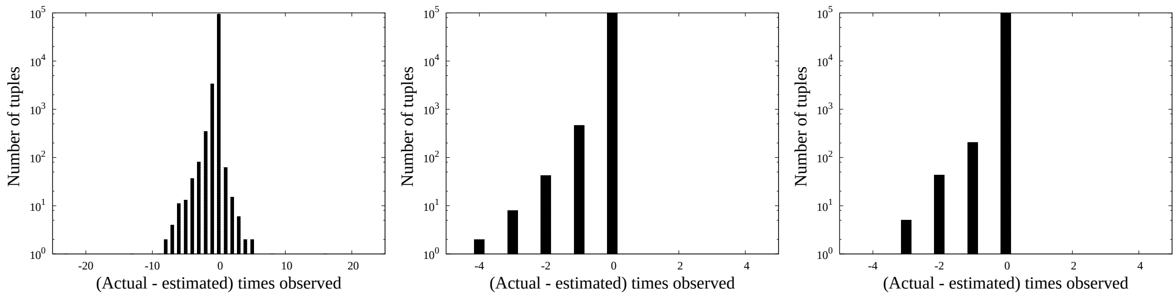


(a) (b) (c)

Figure 13: Distribution of the estimation error for *good* tuples using (a) *Scan*, (b) *Filtered Scan*, and (c) *Automatic Query Expansion*, for H_1 with *minSimilarity* = 0.4 when $|D_r| = |D|/2$ (log-scale).

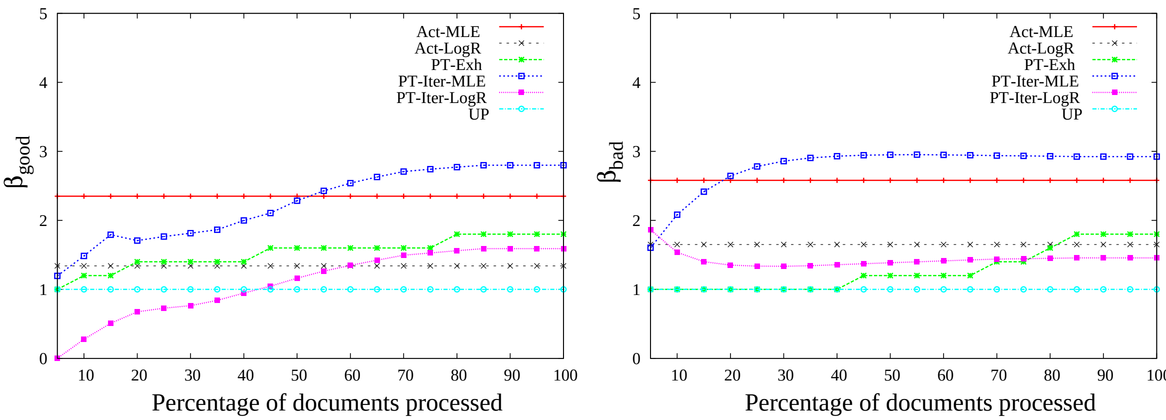
filter of *Filtered Scan*. For instance, we observed that, for a relatively small number of bad tuples in *Headquarters*, all documents that contain these tuples survived the classification step, thus resulting in higher-than-estimated values for the number of bad tuples.

As part of our experimental evaluation, we also study the estimated number of times that we observe a tuple after processing $|D_r|$ documents. Specifically, given $|D_r|$, we use our model along with the actual values for the tuple frequencies to derive, for each tuple, the expected number of times that we will observe it in the output after processing $|D_r|$ documents. For each tuple, we also derive the actual number of times we observe the tuple in the output. Given the estimated and the actual number of times we observe a tuple, we studied the distribution of the estimation error, computed as the number of actual observations minus the estimated number of observations, across all tuples. Figure 13 shows this distribution for *good* tuples for *Scan* (Figure 13(a)), *Filtered Scan* (Figure 13(b)), and *Automatic Query Expansion* (Figure 13(c)); Figure 14 shows the numbers for *bad* tuples for *Scan* (Figure 14(a)), *Filtered Scan* (Figure 14(b)), and *Automatic Query Expansion* (Figure 14(c)). For the case of *good* tuples, we observe that for about 99% of the tuples the estimation error is less than 1, meaning that our proposed analytical models fit well to a significant fraction of the database tuples. Furthermore, for each of the document retrieval strategies, we observed the estimation error to be approximately normally distributed around a mean of 0. This strengthens our previous observations: earlier, we showed that the estimated number of good tuples for varying number of database documents retrieved is close to the actual values. For the case of *bad* tuples we observe that for about 95% (for *Scan*) and about 99% (for *Filtered Scan* and *Automatic Query Expansion*) of the tuples, the estimation error is zero. For *Scan*, the estimation error is approximately normally distributed around the mean of 0. This is in line with our previous observations: Figure 10 suggested that our model accurately estimates the output composition. On the other hand, as observed earlier, the estimation error for



(a) (b) (c)

Figure 14: Distribution of the estimation error for *bad* tuples using (a) *Scan*, (b) *Filtered Scan*, and (c) *Automatic Query Expansion*, for H_1 with *minSimilarity* = 0.4 when $|D_r| = |D|/2$ (log-scale).



(a) (b)

Figure 15: Actual vs. estimated values of the frequency distribution parameters for (a) good tuples and (b) bad tuples.

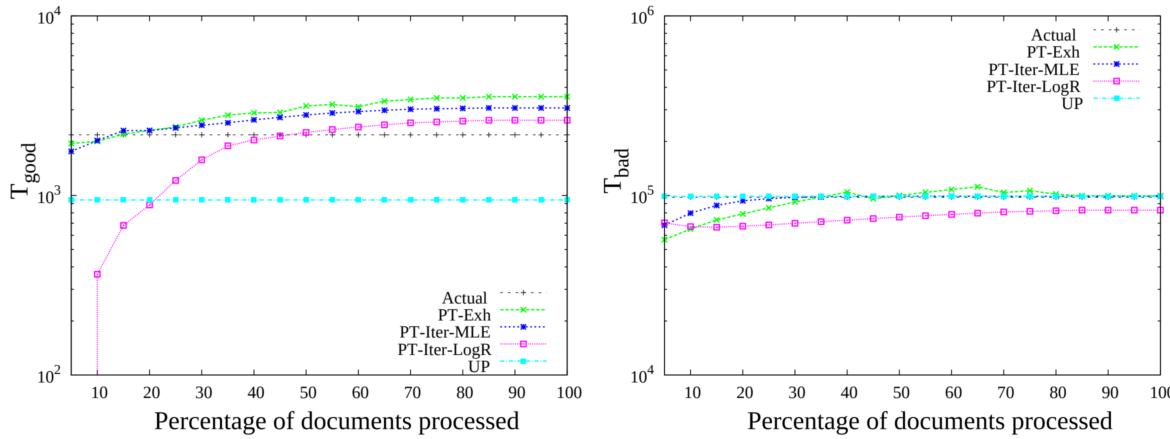
Filtered Scan and *Automatic Query Expansion* is skewed towards negative values, due to the reasons that we discussed earlier.

8.2 Accuracy of Parameter Estimation

In the evaluation presented above, we have seen that our techniques work well when they have access to the correct parameter values for a database. Now, we examine the accuracy of our parameter estimation algorithms presented in Section 5, which is critical to the accuracy of our optimizer. Specifically, we evaluate the performance of four estimation approaches from Section 5, namely, *PT-Exh*, *PT-Iter-MLE*, *PT-Iter-LogR*, and *UP* (see Table 2).

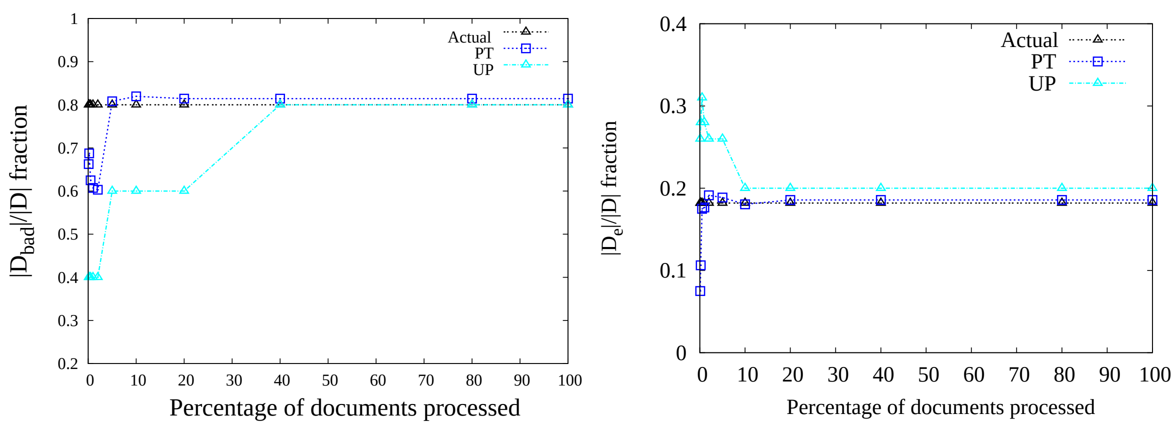
Figure 15 shows the estimated and actual values for the power law exponent for the good tuples (i.e., β_g, see Figure 15(a)), and for the bad tuples (i.e., β_b, see Figure 15(b)), as a function of the percentage of database documents processed. The figures also show the actual value for β_g and β_b by fitting a power law to the actual tuple frequency distribution using MLE and using log-based regression methods (Section 5.1). We refer to these actual values as *Act-MLE* and *Act-LogR*, respectively. Figure 16 shows the estimated and actual values for $|T_{good}|$ (Figure 16(a)) and $|T_{bad}|$ (Figure 16(b)), for varying percentage of the database documents processed. Finally, Figure 17 shows the estimated and actual values for $\frac{|D_b|}{|D|}$ and $\frac{|D_e|}{|D|}$.

The *UP* method tends to underestimate the parameter values associated with *good* tuples, i.e., the values for β_g (see Figure 15(a)) and $|T_{good}|$ (see Figure 16(a)). This is due to the fact that the overall number of good tuples in the database is relatively lower than the total number of bad tuples, which results in a small value for the fraction $\frac{|T_{good}|}{|T_{good}|+|T_{bad}|}$ used by the *UP* approach (Section 5.3). In effect, a small value for this fraction reduces the MLE approach's ability to *differentiate* between different values for $\frac{|T_{good}|}{|T_{good}|+|T_{bad}|}$ that we exhaustively plug-in, and thus *UP* picks a smaller-than-



(a) (b)

Figure 16: Actual vs. estimated values for (a) $|T_{good}|$ and (b) $|T_{bad}|$.



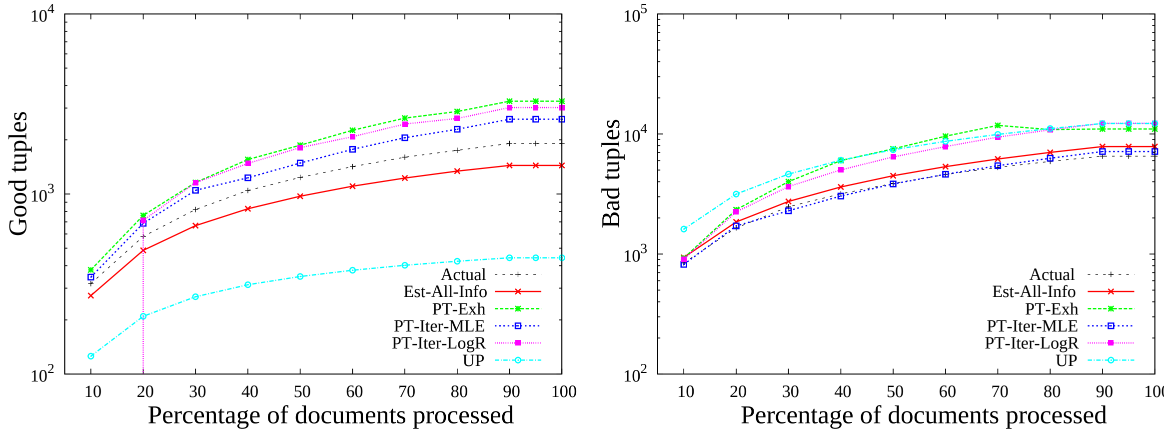
(a) (b)

Figure 17: Actual vs. estimated values for (a) $\frac{|D_b|}{|D|}$ and (b) $\frac{|D_e|}{|D|}$.

actual value for $|T_{good}|$. We can take this effect into consideration by using Bayesian priors [18, 19] on the parameter values to guide our estimation process, by assuming some prior distribution for the parameter values. Interestingly, though, the *UP* method converges quickly to a final value and is appealing for one-time parameter estimation scenarios.

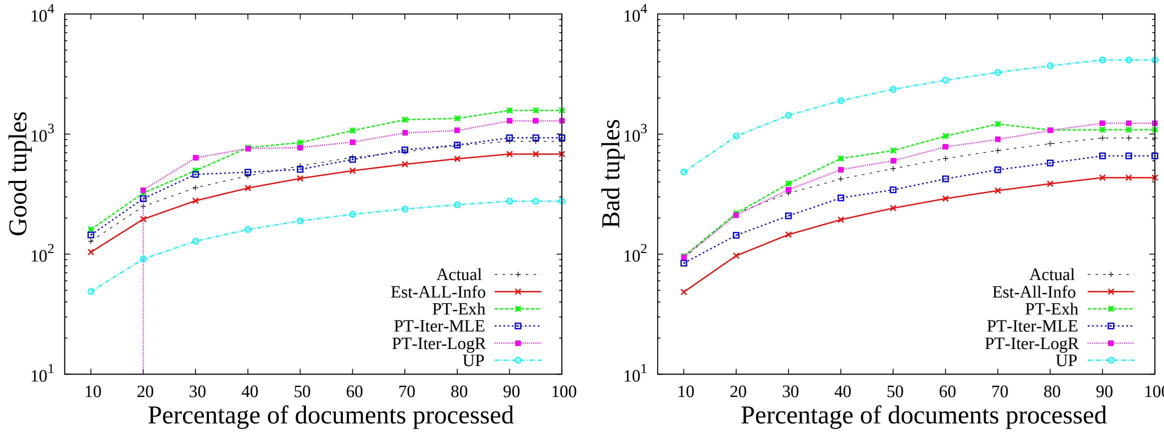
Among the three different partition-based estimation methods (see Section 5.2), an important case is for *PT-Iter-LogR* when estimating β_g and $|T_{good}|$ for small values for $|D_r|$. As seen in Figure 16, *PT-Iter-LogR* requires relatively larger database samples to generate an estimated value for $|T_{good}|$. This can be traced to one main reason: for small database samples, the observed tuple frequencies do not contain enough observations across different frequency values, i.e., the tuple frequency for the observed tuples is identical. Since regression-based techniques require at least two data points (i.e., we need to observe at least two different values of the tuple frequencies), *PT-Iter-LogR* fails to identify the estimated parameters for small database samples. *PT-Iter-LogR* converges to the actual values only after we have observed a good representative sample of the tuple frequency distributions.

To collectively examine the quality of the estimates generated by each technique, we compared the number of good and bad tuples estimated for different numbers of database documents retrieved and for various execution strategies. Figure 18 shows the estimated and actual number of good tuples (Figure 18(a)) and bad tuples (Figure 18(b)) for each estimation method, after processing different percentages of the database with *Scan*. For reference, we show the estimated values using actual tuple frequencies; see the lines labeled *Est-All-Info*. Our results show that our estimates of the quality composition are close to the actual values, with *PT-Iter-MLE* outperforming other techniques especially for



(a) (b)

Figure 18: Actual vs. estimated number of (a) good tuples and (b) bad tuples derived for H_1 with *minSimilarity* = 0.4 and *Scan*, using estimated parameters for *Headquarters*.



(a) (b)

Figure 19: Actual vs. estimated number of (a) good tuples and (b) bad tuples derived for H_1 with *minSimilarity* = 0.4 and *Filtered Scan*, using estimated parameters for *Headquarters*.

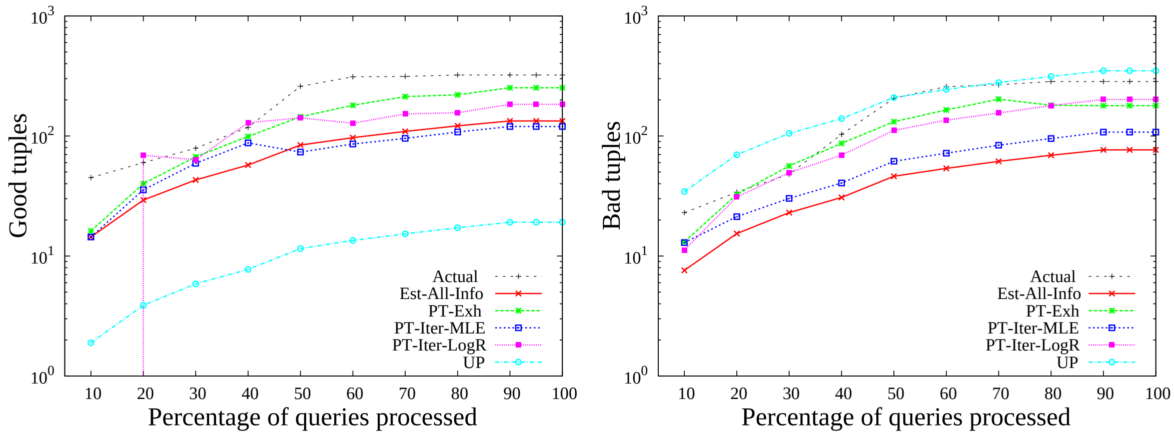
good tuples.

To summarize, our experiments established the accuracy of an important aspect of our optimization approach, namely, the parameter estimation step. As shown above, the MLE-based approaches outlined in Section 5 correctly converge to the actual values of the parameters. Furthermore, using these estimated values in our analytical model leads to correctly estimating the output compositions for various execution strategies.

8.3 Quality of Choice of Execution Strategies

After verifying the accuracy of the model and of the parameter estimation, we now study the accuracy of the optimizer choices. Specifically, we examine whether the optimizer picks the fastest execution strategy for given output-composition requirements. In particular, the optimizer takes as input two thresholds, τ_g and τ_b, specifying that the extraction relation must contain *at least* τ_g *good tuples* and *at most* τ_b *bad tuples*, i.e., $|T_{retr}^{good}| \geq \tau_g$ and $|T_{retr}^{bad}| \leq \tau_b$. (Alternatively, we can specify thresholds for precision and recall of the output.)

For these experiments, we use the *PT-Iter-MLE* estimation method from Section 5 and our analysis of Section 4 to derive the quality curves for each combination of retrieval strategy and extraction system. Given the output



(a) (b)

Figure 20: Actual vs. estimated number of (a) good tuples and (b) bad tuples derived for H_1 with *minSimilarity* = 0.4 and *Automatic Query Expansion*, using estimated parameters for *Headquarters*.

restrictions $|T_{retr}^{good}| \geq \tau_g$ and $T_{retr}^{bad} \leq \tau_b$, we identify the points on the quality curves for which $E[|T_{retr}^{good}|] \geq \tau_g$ and $E[|T_{retr}^{bad}|] \leq \tau_b$. Then, across these *qualifying candidate execution plans*, we pick the one with the fastest execution time.

To evaluate the choice of execution strategies for a query, we compare the execution time for the chosen plan S against that of the alternate executions plans that also meet the τ_g and τ_b output quality requirements. Tables 3 and 4 show the results of our experiments for *Headquarters* and *Executives*, respectively, for different values of τ_g and τ_b. For each choice of values for τ_g and τ_b, we show the number of candidate plans—among the total 24 plans considered (Section 7)—that meet the τ_g and τ_b output quality requirements. Furthermore, we show the number of candidate plans that result in faster executions than the plan chosen by our optimizer and the number of candidate plans that result in slower executions than the chosen plan. Finally, to highlight the difference between the execution time for the chosen execution strategy and other candidates, we compute the relative times for all plans as discussed in Section 7 and then show the minimum and maximum values as range indicators for both plans that are faster and slower than the chosen plan.

As shown in the results, our optimizer selects *Automatic Query Expansion* as the document retrieval strategy for lower values of τ_g and τ_b, and progresses towards selecting *Filtered Scan* and eventually picking *Scan* for higher values of τ_g. *Automatic Query Expansion* and *Filtered Scan* focus on the good documents and aim at generating relations with fewer bad tuples as compared to *Scan*. However, the maximum achievable number of good tuples are limited for *Automatic Query Expansion* and *Filtered Scan* (see Section 4) and thus, for higher τ_g values, the optimizer picks *Scan* as the retrieval strategy. In our experiments, we observed that execution plans that employ *Filtered Scan* result in higher execution times that those using *Automatic Query Expansion* and therefore, *Automatic Query Expansion* is picked over *Filtered Scan*, whenever possible. For most cases, the optimizer selects the fastest execution plan among the candidate plans as indicated by the low or zero values for the number of candidates with faster execution than the chosen plan. For cases, where the chosen plan is not the fastest option, the execution time of faster candidates is very close to the one of the chosen plan (e.g., relative time for faster plans is close to 1). On the other hand, the alternative slower plans, eliminated by our optimizer, have execution times that can be an order of magnitude larger.

In our next experiment, we used our quality-aware optimizer for generating *Headquarters*, while considering query execution strategies that involve both automated information extraction systems, such as Snowball, and manual information extraction systems, generated using the Mechanical Turk service (see Section 7). We observed that, in general, the manual extractions tend to be more quality-oriented than the automated extractions, but at the same time more expensive in terms of time and monetary cost.

Table 5 shows the choices of execution strategies picked by the optimizer for a random set of 2,000 documents. As seen in the table, using the automated extraction system (Au) results in "quick-and-dirty" executions, i.e., our optimizer selects Au when the user-specified requirement for τ_b is relatively high. On the other hand, using the manual extraction system (Ma) results in "slow-and-high-quality" executions and our optimizer appropriately selects Ma when users

Table 3: Statistics on the choice of execution strategies for different output quality requirements specified using τ_g and τ_b, and for *Headquarters*. (IE stands for information extraction system, and X for document retrieval strategy.)

Output Quality Requirements		**# Candidate Plans**	**Chosen Plan**			**# Faster Plans**	**# Slower Plans**	**Relative Time Range**			
								Faster Plans		**Slower Plans**	
τ_g	τ_b		IE	$\boldsymbol{\theta}$	X			*min*	*max*	*min*	*max*
8	1	9	H_1	0.8	*AQG*	0	4	-	-	7.87	49.75
8	4	13	H_1	0.6	*AQG*	0	9	-	-	7.83	51.19
8	8	15	H_1	0.6	*AQG*	0	10	-	-	7.83	51.19
8	16	20	H_1	0.4	*AQG*	0	14	-	-	7.83	51.19
16	3	12	H_1	0.6	*AQG*	0	6	-	-	19.33	74.71
16	8	15	H_1	0.6	*AQG*	0	10	-	-	17.73	76.88
16	16	20	H_1	0.4	*AQG*	0	12	-	-	11.33	76.88
16	32	21	H_1	0.4	*AQG*	0	14	-	-	11.33	76.88
16	80	24	H_1	0.4	*AQG*	0	16	-	-	11.33	76.88
32	6	12	H_1	0.6	*AQG*	0	6	-	-	29.79	129.27
32	16	15	H_1	0.6	*AQG*	0	10	-	-	28.70	129.27
32	32	20	H_1	0.4	*AQG*	0	14	-	-	26.94	129.27
32	64	21	H_1	0.4	*AQG*	0	14	-	-	26.94	129.27
32	160	24	H_1	0.4	*AQG*	0	16	-	-	26.94	129.27
64	12	11	H_1	0.6	*AQG*	0	7	-	-	33.25	134.86
64	32	15	H_1	0.6	*AQG*	0	10	-	-	31.52	134.86
64	64	19	H_1	0.4	*AQG*	0	13	-	-	25.92	134.86
64	128	23	H_1	0.2	*AQG*	0	20	-	-	1.64	221.73
64	320	24	H_1	0.2	*AQG*	0	22	-	-	1.64	221.73
128	25	11	H_1	0.6	*AQG*	0	7	-	-	51.38	167.21
128	64	15	H_1	0.6	*AQG*	0	10	-	-	47.16	167.21
128	128	18	H_1	0.4	*AQG*	0	16	-	-	1.46	244.31
128	256	23	H_1	0.2	*AQG*	0	18	-	-	1.46	244.31
128	640	24	H_1	0.2	*AQG*	0	20	-	-	1.46	244.31
256	51	12	H_1	0.6	*AQG*	0	8	-	-	75.11	329.66
256	128	17	H_1	0.4	*AQG*	0	10	-	-	72.54	329.66
256	256	19	H_1	0.4	*AQG*	0	13	-	-	64.01	329.66
256	512	21	H_1	0.2	*AQG*	0	18	-	-	1.17	384.85
256	1280	20	H_1	0.2	*AQG*	0	18	-	-	1.17	384.85
512	102	8	H_2	0.6	*FScan*	6	1	0.25	0.95	1.06	1.06
512	256	11	H_2	0.4	*FScan*	1	8	0.96	0.96	1.02	4.38
512	512	11	H_2	0.4	*FScan*	2	8	0.85	0.96	1.02	4.38
512	1024	14	H_2	0.2	*FScan*	1	12	0.94	0.94	1.07	4.87
512	2560	16	H_2	0.2	*Scan*	9	6	0.27	0.85	1.02	1.40
1024	204	3	H_2	0.6	*FScan*	0	2	-	-	3.61	4.04
1024	512	11	H_2	0.4	*FScan*	1	8	0.96	0.96	1.01	4.31
1024	1024	9	H_2	0.4	*FScan*	2	6	0.85	0.96	1.07	4.31
1024	2048	14	H_2	0.2	*FScan*	1	11	0.94	0.94	1.07	4.79
1024	5120	11	H_2	0.2	*Scan*	1	6	0.85	0.85	1.00	1.46
2048	20480	2	H_2	0.2	*Scan*	0	0	-	-	-	-

Table 4: Statistics on the choice of execution strategies for different output quality requirements specified using τ_g and τ_b, and for *Executives*. (IE stands for information extraction system, and X for document retrieval strategy.)

Output Quality Requirements		**# Candidate Plans**	**Chosen Plan**			**# Faster Plans**	**# Slower Plans**	**Relative Time Range**			
								Faster Plans		**Slower Plans**	
τ_g	τ_b		IE	$\boldsymbol{\theta}$	X			*min*	*max*	*min*	*max*
10	8	8	E_1	0.8	*AQG*	0	7	-	-	1.12	23.97
10	10	10	E_1	0.8	*AQG*	0	9	-	-	1.12	23.97
10	60	20	E_1	0.8	*AQG*	0	19	-	-	1.00	23.97
10	110	20	E_1	0.8	*AQG*	0	19	-	-	1.00	23.97
30	24	8	E_1	0.8	*AQG*	0	7	-	-	1.13	45.00
30	30	8	E_1	0.8	*AQG*	0	7	-	-	1.13	45.00
30	180	20	E_1	0.4	*AQG*	1	18	1.00	1.00	1.04	44.86
30	330	20	E_1	0.4	*AQG*	1	18	1.00	1.00	1.04	44.86
45	36	8	E_2	0.6	*AQG*	2	5	0.89	1.00	1.00	58.88
45	45	8	E_2	0.6	*AQG*	2	5	0.89	1.00	1.00	58.88
45	270	20	E_1	0.4	*AQG*	1	18	1.00	1.00	1.04	66.13
45	495	20	E_1	0.4	*AQG*	1	18	1.00	1.00	1.04	66.13
70	56	8	E_1	0.8	*AQG*	0	7	-	-	1.13	46.63
70	70	8	E_1	0.8	*AQG*	0	7	-	-	1.13	46.63
70	420	20	E_1	0.2	*AQG*	2	17	0.96	0.96	1.08	44.75
70	770	20	E_1	0.2	*AQG*	2	17	0.96	0.96	1.08	44.75
150	200	3	E_2	0.6	*FScan*	0	2	-	-	1.02	1.09
150	300	5	E_1	0.4	*FScan*	0	4	-	-	1.09	1.25
175	225	1	E_2	0.6	*FScan*	0	0	-	-	-	-
175	500	3	E_1	0.4	*FScan*	0	2	-	-	1.09	1.15
175	150	1	E_2	0.6	*FScan*	0	0	-	-	-	-
345	3795	3	E_2	0.6	*Scan*	0	2	-	-	1.00	1.13
345	2070	3	E_2	0.6	*Scan*	0	2	-	-	1.00	1.13
345	2520	3	E_2	0.6	*Scan*	0	2	-	-	1.00	1.13
375	3000	3	E_2	0.6	*Scan*	0	2	-	-	1.00	1.14
410	660	1	E_2	0.6	*Scan*	0	0	-	-	-	-
490	6660	2	E_2	0.2	*Scan*	0	1	-	-	1.14	1.14

desire high quality results.

8.4 Comparing with Baselines

Table 7 compares the performance of our optimization approach, *Qawr*, with the baseline *Qign* for different choices of values for the quality thresholds τ_g and τ_b. For each value for τ_g and τ_b, we show the choice of execution plan along with the actual quality and the execution time for both *Qawr* and *Qign*. As shown in Table 7, *Qign* fails to produce executions that meet the τ_g and τ_b requirements for all cases; on the other hand, *Qawr* produces execution plans that meet all but one of τ_g and τ_b requirements. The execution plans picked by *Qign* are generally faster than those picked by *Qawr*, as *Qign* largely overestimates the output quality and suggests retrieving fewer documents than necessary, but the *Qign* executions do not meet the output given quality requirements.

We compared *Qawr* with two variations of *Heur*. Specifically, for our first variation we used *Headquarters* as the "training" task and *Executives* as the target task; for our second variation we switched the training and target task relations. Table 8 shows the performance of *Heur* and *Qawr* for the task of extracting the *Executives* relation, for different choices of values for the quality thresholds. (To allow for a fair comparison, we used only one extraction system per relation.) As shown in Table 8, *Heur* sometimes fails to pick a suitable execution plan, even when such a plan exists. In other cases, when both techniques pick an execution plan, the chosen execution plans meet the quality requirements. However, the execution time of the plans chosen by *Heur* can be orders of magnitude higher than that for the *Qawr* plan. The analogous experiments for the task of generating *Headquarters* generated similar results. In this case, we observed that the *Heur* execution plans were faster than those picked by *Qawr*; but unfortunately, the *Heur* plans did not meet the quality requirements, unlike the *Qawr* plans.

To summarize, *Qawr* outperforms the two baselines, namely, *Qign* and *Heur*, and selects superior execution plans that efficiently meet the output quality requirements by taking into account the quality of the extraction systems and the associated retrieval strategies.

Evaluation conclusion: We demonstrated the efficiency and effectiveness of our quality-based optimizer for selecting efficient execution plans that meet the user-specified quality requirements. Furthermore, we compared with existing baselines (one based on [24] and one based on heuristics) and we demonstrated the superiority of our approach.

Table 5: Choice of execution strategies using query execution strategies that involve manual (Ma) and automated (Au) for different output quality requirements specifed using τ_g and τ_b, and for *Headquarters*. (IE stands for information extraction system.)

Output Quality Requirement		# Candidate Plans	Chosen Plan		# Faster Plans	# Slower Plans	Relative time range			
							Faster Plans		Slower Plans	
τ_g	τ_b		IE	θ			*min*	*max*	*min*	*max*
10	10	1	Ma	5	0	0	-	-	-	-
40	80	4	Ma	2	0	3	-	-	4.00	12.75
40	500	6	Au	0.6	0	5	-	-	80.30	665.01
60	2000	6	Au	0.2	0	5	-	-	43.50	93.50
200	200	2	Ma	4	0	1	-	-	6.07	6.07
300	320	3	Ma	3	0	2	-	-	4.85	7.27
400	300	1	Ma	4	0	0	-	-	-	-
400	1400	4	Ma	1	0	3	-	-	2.40	6.08

Table 6: Statistics on the choice of execution strategies using *Qign* and *Qawr* for different output quality requirements specified using τ_g and τ_b, and for *Headquarters*. (IE stands for information extraction system and X for document retrieval strategy.)

		Execution based on *Qign*						Execution based on *Qawr*				
Output Quality Requirement		Execution Plan			Output Quality		Relative Time	Execution Plan			Output Quality	
τ_g	τ_b	IE	θ	X	$\lvert T^{good}_{retr}\rvert$	$\lvert T^{bad}_{retr}\rvert$		IE	θ	X	$\lvert T^{good}_{retr}\rvert$	$\lvert T^{bad}_{retr}\rvert$
8	4	H_1	0.4	*Scan*	0	0	0.05	H_1	0.6	*AQG*	39	6
8	16	H_1	0.4	*Scan*	0	0	0.1	H_1	0.4	*AQG*	45	23
16	8	H_1	0.4	*Scan*	0	0	0.1	H_1	0.6	*AQG*	39	6
16	16	H_1	0.2	*Scan*	0	6	0.16	H_1	0.4	*AQG*	45	23
64	12	H_2	0.2	*Scan*	0	9	0.08	H_1	0.6	*AQG*	77	22
64	32	H_2	0.2	*Scan*	0	14	0.1	H_1	0.6	*AQG*	77	22
128	25	H_2	0.2	*Scan*	0	18	0.09	H_1	0.6	*AQG*	293	99
128	64	H_2	0.2	*Scan*	0	23	0.12	H_1	0.6	*AQG*	293	99
256	51	H_2	0.2	*Scan*	0	34	0.03	H_2	0.6	*FScan*	137	63
256	128	H_2	0.2	*Scan*	2	46	0.03	H_2	0.4	*FScan*	258	247
512	102	H_2	0.2	*Scan*	4	76	0.03	H_2	0.6	*FScan*	254	106
512	256	H_1	0.4	*AQG*	79	48	0.05	H_2	0.4	*FScan*	391	391
1024	512	H_1	0.4	*AQG*	309	256	0.02	H_2	0.6	*Scan*	1169	519

9 Related Work

Information extraction has received significant attention in the recent years (see [33, 16, 3, 29, 31] and references therein). A large family of existing solutions [33, 16, 3, 29, 31] focus on improving the extraction accuracy by directly manipulating the information extraction system for a given task. Another direction of work related to information extraction is that of representation: Gupta et al. [22], Caferalla et al. [9] presented approaches to use probabilistic databases to materialize extracted relations after appropriately deriving the probability of each tuple being correct, following the *Scan*strategy that we discussed in the paper.

Retrieval strategies for information extraction traditionally use the *Scan* strategy, where every document is processed by the information extraction system (e.g., [20, 34]). Some systems use the *Filtered Scan* strategy, where only the documents that match specific URL patterns (e.g., [7]) or regular expressions (e.g., [21]) are processed further. Agichtein and Gravano [4] presented query-based execution strategies. More recently, Etzioni et al. [16] used what could be viewed as an instance of *Automatic Query Generation* to query generic search engines for extracting information from the web. Cafarella and Etzioni [8] presented a complementary approach of constructing a special-purpose index for efficiently retrieving promising text passages for information extraction. Such document (and passage) retrieval improvements can be naturally integrated into our framework. These retrieval strategies, though, have resulted in relatively "static" pipelines for an extraction task. In this paper, we initiate the need to study—in a principled manner—and appropriately exploit the effects of available configuration parameters for extraction systems (as black-boxes) and various crawl- or query-based document retrieval strategies.

ROC curves have been long used to study the performance of radio receivers; in machine learning, ROC curves are preferred when evaluating the ability of binary decision-making process, such as classifiers, at discriminating signal from noise. ROC curves were so far mainly used to graphically summarize the performance of a decision-making

Table 7: Statistics on the choice of execution strategies using *Qign* and *Qawr* for differetn output quality requirements specified using τ_g and τ_b, and for *Executives*. (IE stands for information extraction system and X for document retrieval strategy .)

Output Quality Requirement		Execution based on *Qign* Execution Plan			Output Quality		Relative Time	Execution based on *Qawr* Execution Plan			Output Quality	
τ_g	τ_b	IE	θ	X	$\|T_{retr}^{good}\|$	$\|T_{retr}^{bad}\|$		IE	θ	X	$\|T_{retr}^{good}\|$	$\|T_{retr}^{bad}\|$
10	40	E_1	0.2	*Scan*	0	9	1.722	E_1	0.8	*AQG*	24	17
10	80	E_1	0.2	*Scan*	0	19	0.32	E_1	0.8	*AQG*	24	17
45	195	E_1	0.8	*AQG*	16	24	0.265	E_1	0.4	*AQG*	78	161
70	270	E_1	0.8	*AQG*	16	24	0.02	E_1	0.2	*AQG*	94	416
70	1120	E_1	0.8	*AQG*	35	65	0.01	E_1	0.2	*AQG*	94	416
115	150	E_1	0.8	*AQG*	16	24	0.04	E_2	0.4	*FScan*	110	356

Table 8: Statistics on the choice of execution strategies using *Heur* and *Qawr* for different output quality requirements specified by τ_g and τ_b, and for *Headquarters*. (X stands for document retrieval strategy.)

Output Quality Requirement		Execution based on *Qign* Execution Plan		Output Quality		Relative Time	Execution based on *Qawr* Execution Plan		Output Quality	
τ_g	τ_b	θ	X	$\|T_{retr}^{good}\|$	$\|T_{retr}^{bad}\|$		θ	X	$\|T_{retr}^{good}\|$	$\|T_{retr}^{bad}\|$
2	10	0.6	*AQG*	5	4	1.65	0.8	*AQG*	15	9
10	380	0.6	*Scan*	43	118	34.94	0.8	*AQG*	24	17
45	1495	0.6	*Scan*	225	764	104.2	0.6	*AQG*	73	76
60	2120	0.6	*Scan*	294	963	133.55	0.2	*AQG*	85	336
75	770	-	-	-	-	-	0.2	*AQG*	99	458
115	2050	-	-	-	-	-	0.4	*AQG*	143	328

process. In the context of information extraction systems, Hiyakumoto et al. [23] explored ROC curves but mainly to generate "rules" based on the visual representation of ROC curves. In our paper, we introduced the ROC generating process for an information extraction system and showed how it can be effectively utilized to build robust optimization techniques.

Our parameter estimation and optimization approach is conceptually related to adaptive query execution techniques developed for relational data (e.g., [26, 6]) and to database sampling techniques (e.g., [10]). The basic difference is that we assume a parametric retrieval model, which in turn allows us to use a maximum likelihood-based estimation model for parameter estimation.

The closest research effort related to this paper is the work by Ipeirotis et al. [25] that presents analytic models for predicting the execution time of various document retrieval strategies, with the goal of picking the strategy that reaches a target recall in the minimum amount of time. Our work builds on [25], and expands it to include the concept of quality estimation. In particular, we remove the (unrealistic) assumption that the extraction system is perfect, and we estimate the execution time and the quality of the output; we also pick the appropriate settings for the extraction system. Finally, we present an estimation framework that allow us to deal with unknown parameter values of the estimation framework. Our experimental comparison, in Section 8.4, shows that our techniques outperform the approach in [25].

There is also work on estimating the output quality for an extraction system, although existing research focuses on estimating the quality of extraction per se, and not the effect of document retrieval strategies on output quality. Agichtein [2] presented a heuristic-based approach on automatically tuning an extraction system's parameter. To identify a good configuration, Agichtein uses precision-recall curves, and thus suffers from being sensitive to the distribution of test set documents. In contrast, we decouple the effect of test set on performance measurement by using ROC curves to characterize an extraction system. Downey et al. [13] present a probabilistic model for deciding the confidence in a tuple, using evidence gathered from the text database and appropriately accounting for the strength of this evidence. The work in [13] estimates the probability that a tuple is good, based on its frequency on the set of extracted tuples. The technique, though, assumes a *Scan* retrieval strategy and will not work for other retrieval models.

Finally, Jain et al. [27] recently presented a query optimization approach for simple SQL queries over (structured data extracted from) text databases. Jain et al. consider multiple document retrieval strategies to process a SQL query, including *Scan*, *Automatic Query Generation*, and other query-based strategies. Unlike our setting, however, [27] focuses on extraction scenarios that involve multiple extraction systems, whose output might then need to be integrated and joined to answer a given SQL query. The SQL query optimization approach in [27] accounts for errors originating in the information extraction process, but relies mainly on heuristics and does not use the rigorous statistical models that we presented here, and hence cannot benefit from the MLE-based estimation to estimate the values of unknown database parameters.

10 Conclusion

We introduced a rigorous model for estimating the quality of the output of an information extraction system when paired with a document retrieval strategy. We showed how to generate an ROC curve can generate a statistically robust performance characterization of an extraction system, and then built statistical models that use the ROC curves concept to build the *quality curves* that predict the performance of coupling an extraction system with a retrieval strategy. Our analysis helps predict the execution time and output quality of an execution plan. Based on our analysis, we then show how to use these predictions to pick the fastest execution plan that generates output that satisfies the quality characteristics.

References

[1] ADAMIC, L. A., AND HUBERMAN, B. A. Zipf's law and the internet. vol. 3, pp. 143–150.

[2] AGICHTEIN, E. *Extracting Relations From Large Text Collections*. PhD thesis, Columbia University, 2005.

[3] AGICHTEIN, E., AND GRAVANO, L. Snowball: Extracting relations from large plain-text collections. In *Proceedings of the Fifth ACM Conference on Digital Libraries (DL 2000)* (2000).

[4] AGICHTEIN, E., AND GRAVANO, L. Querying text databases for efficient information extraction. In *Proceedings of the 19th IEEE International Conference on Data Engineering (ICDE 2003)* (2003).

[5] AGICHTEIN, E., IPEIROTIS, P. G., AND GRAVANO, L. Modeling query-based access to text databases. In *Proceedings of the Sixth International Workshop on the Web and Databases, WebDB 2003* (2003), pp. 87–92.

[6] AVNUR, R., AND HELLERSTEIN, J. M. Eddies: Continuously adaptive query processing. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data (SIGMOD 2000)* (2000), pp. 261–272.

[7] BRIN, S. Extracting patterns and relations from the world wide web. In *Proceedings of the First International Workshop on the Web and Databases, WebDB 1998* (1998), pp. 172–183.

[8] CAFARELLA, M. J., AND ETZIONI, O. A search engine for natural language applications. In *Proceedings of the 14th International World Wide Web Conference (WWW 2005)* (2005), pp. 442–452.

[9] CAFARELLA, M. J., RE, C., SUCIU, D., ETZIONI, O., AND BANKO, M. Structured querying of web text: A technical challenge. In *3rd Biennial Conference on Innovative Data Systems Research (CIDR'07)* (2007).

[10] CHAUDHURI, S., MOTWANI, R., AND NARASAYYA, V. R. Random sampling for histogram construction: How much is enough? In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data (SIGMOD'98)* (1998), pp. 436–447.

[11] COHEN, W. W. Fast effective rule induction. In *Proceedings of the 12th International Conference on Machine Learning (ICML'95)* (1995), pp. 115–123.

[12] COHEN, W. W. Minorthird: Methods for identifying names and ontological relations in text using heuristics for inducing regularities from data, 2004. Available at `http://minorthird.sourceforge.net`.

[13] DOWNEY, D., ETZIONI, O., AND SODERLAND, S. A probabilistic model of redundancy in information extraction. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05)* (2005), pp. 1034–1041.

[14] EGAN, J. P. *Signal Detection Theory and ROC Analysis*. Academic Press, 1975.

[15] ERDREICH, L. S., AND LEE, E. T. Use of relative operating characteristic analysis in epidemiology. *American Journal of Epidemiology 114*, 5 (1981), 649–662.

[16] ETZIONI, O., CAFARELLA, M. J., DOWNEY, D., KOK, S., POPESCU, A.-M., SHAKED, T., SODERLAND, S., WELD, D. S., AND YATES, A. Web-scale information extraction in KnowItAll (preliminary results). In *Proceedings of the 13th International World Wide Web Conference (WWW 2004)* (2004), pp. 100–110.

[17] FAWCETT, T. ROC graphs: Notes and practical considerations for data mining researchers. Tech. rep., Technical Report HPL-2003-4, HP Labs., 2003.

[18] GELMAN, A., CARLIN, J. B., STERN, H. S., AND RUBIN, D. B. *Bayesian Data Analysis*, 2nd ed. Chapman and Hall/CRC, 2003.

[19] GOLDSTEIN, M., MORRIS, S., AND YEN, G. G. Problems with fitting to the power-law distribution. *The European Physical Journal B - Condensed Matter and Complex Systems 41*, 2 (Sept. 2004), 255–258.

[20] GRISHMAN, R. Information extraction: Techniques and challenges. In *Information Extraction: A Multidisciplinary Approach to an Emerging Information Technology, International Summer School, (SCIE-97)* (1997), pp. 10–27.

[21] GRISHMAN, R., HUTTUNEN, S., AND YANGARBER, R. Information extraction for enhanced access to disease outbreak reports. *Journal of Biomedical Informatics 35*, 4 (Aug. 2002), 236–246.

[22] GUPTA, R., AND SARAWAGI, S. Curating probabilistic databases from information extraction models. In *Proceedings of the 32nd International Conference on Very Large Databases (VLDB 2006)* (2006).

[23] HIYAKUMOTO, L., LITA, L. V., AND NYBERG, E. Multi-strategy information extraction for question answering. In *FLAIRS* (2005), pp. 678–683.

[24] IPEIROTIS, P. G., AGICHTEIN, E., JAIN, P., AND GRAVANO, L. To search or to crawl? Towards a query optimizer for text-centric tasks. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data (SIGMOD 2006)* (2006), pp. 265–276.

[25] IPEIROTIS, P. G., AGICHTEIN, E., JAIN, P., AND GRAVANO, L. Towards a query optimizer for text-centric tasks. *ACM Transactions on Database Systems 32*, 4 (Dec. 2007).

[26] IVES, Z. G., FLORESCU, D., FRIEDMAN, M., LEVY, A. Y., AND WELD, D. S. An adaptive query execution system for data integration. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data (SIGMOD'99)* (1999), pp. 299–310.

[27] JAIN, A., DOAN, A., AND GRAVANO, L. Optimizing SQL queries over text databases. In *Proceedings of the 24th IEEE International Conference on Data Engineering (ICDE 2008)* (2007), pp. 636–645.

[28] MACSKASSY, S. A., PROVOST, F., AND ROSSET, S. Roc confidence bands: An empirical evaluation. In *Proceedings of the 22nd International Conference on Machine Learning (ICML 2005)* (2005), pp. 537–544.

[29] MANSURI, I., AND SARAWAGI, S. A system for integrating unstructured data into relational databases. In *ICDE* (2006).

[30] NEWMAN, M. E. J. Power laws, Pareto distributions and Zipf's law. *Contemporary Physics 46*, 5 (Sept. 2005), 323–351.

[31] PASCA, M., LIN, D., BIGHAM, J., LIFCHITS, A., AND JAIN, A. Names and similarities on the web: Fact extraction in the fast lane. In *ACL* (2006).

[32] PROVOST, F. J., AND FAWCETT, T. Robust classification for imprecise environments. *Machine Learning 42*, 3 (Mar. 2001), 203–231.

[33] RILOFF, E., AND JONES, R. Learning dictionaries for information extraction by multi-level bootstrapping. In *AAAI* (1999).

[34] YANGARBER, R., AND GRISHMAN, R. NYU: Description of the Proteus/PET system as used for MUC-7. In *Proceedings of the Seventh Message Understanding Conference (MUC-7)* (1998).