To Search or to Crawl? Towards a Query Optimizer for Text-Centric Tasks

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

Eugene Agichtein
Microsoft Research
eugeneag@microsoft.com

Pranay Jain
Columbia University
pranay.jain@columbia.edu

Luis Gravano
Columbia University
gravano@cs.columbia.edu

ABSTRACT

Text is ubiquitous and, not surprisingly, many important applications rely on textual data for a variety of tasks. As a notable example, information extraction applications derive structured relations from unstructured text; as another example, focused crawlers explore the web to locate pages about specific topics. Execution plans for text-centric tasks follow two general paradigms for processing a text database: either we can scan, or "crawl," the text database or, alternatively, we can exploit search engine indexes and retrieve the documents of interest via carefully crafted queries constructed in task-specific ways. The choice between crawl- and query-based execution plans can have a substantial impact on both execution time and output "completeness" (e.g., in terms of recall). Nevertheless, this choice is typically ad-hoc and based on heuristics or plain intuition. In this paper, we present fundamental building blocks to make the choice of execution plans for text-centric tasks in an informed, cost-based way. Towards this goal, we show how to analyze query- and crawl-based plans in terms of both execution time and output completeness. We adapt results from random-graph theory and statistics to develop a rigorous cost model for the execution plans. Our cost model reflects the fact that the performance of the plans depends on fundamental task-specific properties of the underlying text databases. We identify these properties and present efficient techniques for estimating the associated parameters of the cost model. Overall, our approach helps predict the most appropriate execution plans for a task, resulting in significant efficiency and output completeness benefits. We complement our results with a large-scale experimental evaluation for three important text-centric tasks and over multiple real-life data sets.

1. INTRODUCTION

Text is ubiquitous and, not surprisingly, many applications rely on textual data for a variety of tasks. For example, information extraction applications retrieve documents and extract structured relations from the unstructured text in the documents. Reputation management systems download web pages to track the "buzz" around companies and products. Comparative shopping agents locate e-commerce web sites and add the products offered in the pages to their own index.

To process a text-centric task over a text database (or the web), we can retrieve the relevant database documents in different ways. One approach is to *scan* or *crawl* the database to retrieve its documents and process them as required by the task. While this approach guarantees that we cover all documents that are potentially relevant for the task, it might be unnecessarily expensive in terms of execution time. For example, consider the task of extracting information on disease outbreaks (e.g., the name of the disease, the location and date of the outbreak, and the number of affected people) as reported in news articles. This task does not require that we scan and process, say, the articles about sports in a newspaper archive. In fact, only a small fraction of the archive is of relevance to the task. For tasks such as this one, a natural alternative to crawling is to exploit a search engine index on the database to retrieve –via careful querying– the useful documents. In our example, we can use keywords that are strongly associated with disease outbreaks (e.g., "World Health Organization," "case fatality rate") and turn these keywords into queries to find news articles that are appropriate for the task.

The choice between a crawl- and a query-based execution strategy for a text-centric task is analogous to the choice between a scan- and an index-based execution plan for a selection query over a relation. Just as in the relational model, the choice of execution strategy can substantially affect the execution time of the task. In contrast to the relational world, however, this choice might also affect the quality of the output that is produced: while a crawl-based execution of a text-centric task guarantees that all documents are processed, a query-based execution might miss some relevant documents, hence producing potentially incomplete output, with less-than-perfect *recall*. The choice between crawl- and query-based execution plans can then have a substantial impact on both execution time and output recall. Nevertheless, this important choice is typically left to simplistic heuristics or plain intuition.

In this paper, we introduce fundamental building blocks for the optimization of text-centric tasks. Towards this goal, we show how to rigorously analyze query- and crawl-based plans for a task in terms of both execution time and output recall. To analyze crawl-based plans, we apply techniques from statistics to model crawling as a document *sampling process*; to analyze query-based plans, we first abstract the querying process as a random walk on a *querying graph*, and then apply results for the theory of random graphs to discover relevant properties of the querying process. Our cost model reflects the fact that the performance of the execution plans depends on fundamental task-specific properties of the underlying text databases. We identify these properties and present efficient techniques for estimating the associated parameters of the cost model.

In brief, the contributions and content of the paper are as follows:

- A novel framework for analyzing crawl- and query-based execution plans for text-centric tasks in terms of execution time and output recall (Section 3).

SIGMOD 2006, June 27–29, 2006, Chicago, Illinois, USA.

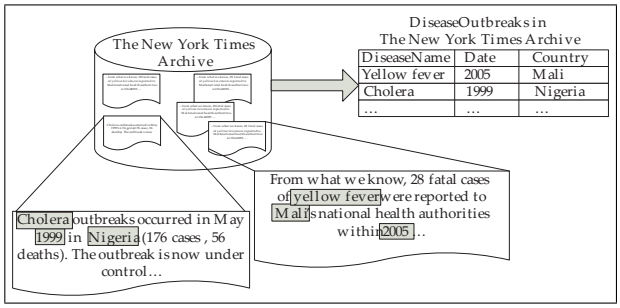

Figure 1: Extracting *DiseaseOutbreaks* tuples

- A description of four crawl- and query-based execution plans, which underlie the implementation of many existing text-centric tasks (Section 4).
- A rigorous analysis of each execution plan alternative in terms of execution time and recall; this analysis relies on fundamental task-specific properties of the underlying databases (Section 5).
- An optimization strategy that estimates the database properties that affect the execution time and recall of each plan and selects the best execution plan for the task description and target recall requirements (Section 6).
- An extensive experimental evaluation showing that our optimization strategy is accurate and results in significant performance gains. Our experiments include three important text-centric tasks and multiple real-life data sets (Sections 7 and 8).

Finally, Section 9 discusses related work, while Section 10 provides further discussion and concludes the paper.

2. EXAMPLES OF TEXT-CENTRIC TASKS

In this section, we briefly review three important text-centric tasks that we will use throughout the paper as running examples, to illustrate our framework and techniques.

2.1 Task 1: Information Extraction

Unstructured text (e.g., in newspaper articles) often embeds *structured* information that can be used for answering relational queries or for data mining. The first task that we consider is the *extraction of structured information from text databases*. An example of an information extraction task is the construction of a table *DiseaseOutbreaks(DiseaseName, Date, Country)* of reported disease outbreaks from a newspaper archive (see Figure 1). A tuple ⟨*yellow fever*, 2005, *Mali*⟩ might then be extracted from the news articles in Figure 1.

Information extraction systems typically rely on patterns —either manually created or learned from training examples— to extract the structured information from the documents in a database. The extraction process is usually time consuming, since information extraction systems might rely on a range of expensive text analysis functions, such as parsing or named-entity tagging (e.g., to identify all person names in a document). See [28] for an introductory survey on information extraction.

A straightforward execution strategy for an information extraction task is to retrieve and process every document in a database exhaustively. As a refinement, an alternative strategy might use *filters* and do the expensive processing of only "promising" documents; for example, the Proteus system [29] ignores database documents that do not include words such as "virus" and "vaccine" when extracting the *DiseaseOutbreaks* relation. As an alternative, query-based approaches such as QXtract [2] have been proposed to avoid retrieving all documents in a database; instead, these approaches retrieve appropriate documents via carefully crafted queries.

2.2 Task 2: Content Summary Construction

Many text databases have valuable contents "hidden" behind search interfaces and are hence ignored by search engines such as Google.

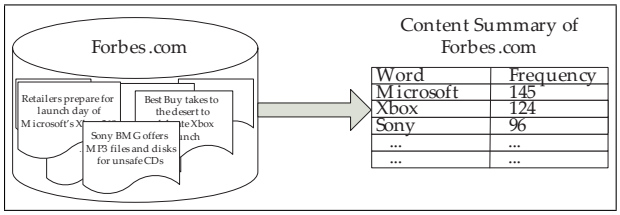

Figure 2: Content summary of Forbes.com

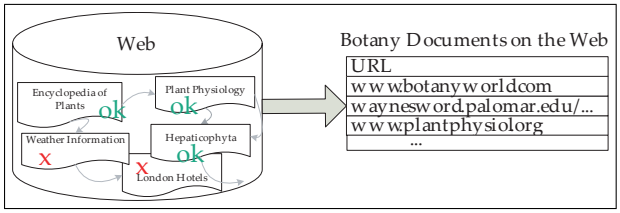

Figure 3: Focused resource discovery for *Botany* pages

Metasearchers are helpful tools for searching over many databases at once through a unified query interface. A critical step for a metasearcher to process a query efficiently and effectively is the selection of the most promising databases for the query. This step typically relies on statistical summaries of the database contents [11, 25]. The second task that we consider is the *construction of a content summary of a text database*. The content summary of a database generally lists each word that appears in the database, together with its frequency. For example, Figure 2 shows that the word "*xbox*" appears in 124 documents in the Forbes.com database. If we have access to the full contents of a database (e.g., via crawling), it is straightforward to derive these simple content summaries. If, in contrast, we only have access to the database contents via a limited search interface (e.g., as is the case for "hidden-web" databases [5]), then we need to resort to query-based approaches for content summary construction [9, 31].

2.3 Task 3: Focused Resource Discovery

Text databases often contain documents on a variety of topics. Over the years, a number of specialized search engines (as well as directories) that focus on a specific topic of interest have been proposed (e.g., FindLaw). The third task that we consider is the identification of the database documents that are about the topic of a specialized search engine, or *focused resource discovery*.

As an example of focused resource discovery, consider building a search engine that specializes in documents on botany from the web at large (see Figure 3). For this, an expensive strategy would crawl all documents on the web and apply a document classifier [39] to each crawled page to decide whether it is about botany (and hence should be indexed) or not (and hence should be ignored). As an alternative execution strategy, *focused crawlers* (e.g., [14, 13, 33]) concentrate their effort on documents and hyperlinks that are on-topic, or likely to lead to on-topic documents, as determined by a number of heuristics. Focused crawlers can then address the focused resource discovery task efficiently at the expense of potentially missing relevant documents. As yet another alternative, Cohen and Singer [20] propose a query-based approach for this task, where they exploit search engine indexes and use queries derived from a document classifier to quickly identify pages that are relevant to a given topic.

3. DESCRIBING TEXT-CENTRIC TASKS

While the text-centric examples of Section 2 might appear substantially different on the surface, they all operate over a database of text documents and also share other important underlying similarities.

Each task in Section 2 can be regarded as deriving "tokens" from a database, where a *token* is a unit of information that we define in a task-specific way. For *Task 1*, the tokens are the relation tuples that are

extracted from the documents. For *Task 2*, the tokens are the words in the database (accompanied by the associated word frequencies). For *Task 3*, the tokens are the documents (or web pages) in the database that are about the topic of focus.

The execution strategies for the tasks in Section 2 rely on task-specific *document processors* to derive the tokens associated with the task. For *Task 1*, the document processor is the information extraction system of choice (e.g., Proteus [29], DIPRE [7], Snowball [1]): given a document, the information extraction system extracts the tokens (i.e., the tuples) that are present in the document. For *Task 2*, the document processor extracts the tokens (i.e., the words) that are present in a given document, and the associated document frequencies are updated accordingly in the content summary. For *Task 3*, the document processor decides (e.g., via a document classifier such as Naive Bayes [23] or Support Vector Machines [40]) whether a given document is about the topic of focus; if the classifier deems the document relevant, the document is added as a token to the output and is discarded otherwise.

The alternate execution strategies for the Section 2 tasks differ in how they retrieve the input documents for the document processors, as we will discuss in Section 4. Some execution strategies fully process every available database document, thus guaranteeing the extraction of all the tokens that the underlying document processor can derive from the database. In contrast, other execution strategies focus, for efficiency, on a strict subset of the database documents, hence potentially missing tokens that would have been derived from unexplored documents. One subcategory applies a *filter* (e.g., derived in a training stage) to each document to decide whether to fully process it or not. Other strategies retrieve via querying the documents to be processed, where the queries can be derived in a number of ways that we will discuss. All these alternate execution strategies thus exhibit different tradeoffs between *execution time* and output *recall*.

DEFINITION 3.1. **[Execution Time]** *Consider a text-centric task, a database of text documents D, and an execution strategy $\mathcal{S}$ for the task, with an underlying document processor P. Then, we define the* execution time *of $\mathcal{S}$ over D, $Time(\mathcal{S}, D)$, as*

$$Time(\mathcal{S}, D) =$$

$$t_T(\mathcal{S}) + \sum_{q \in Q_{sent}} t_Q(q) + \sum_{d \in D_{retr}} \Big(t_R(d) + t_F(d) \Big) + \sum_{d \in D_{proc}} t_P(d)$$

where

- *Q_{sent} is the set of queries sent by $\mathcal{S}$,*
- *D_{retr} is the set of documents retrieved by $\mathcal{S}$ ($D_{retr} \subseteq D$),*
- *D_{proc} is the set of documents that $\mathcal{S}$ processes with document processor P ($D_{proc} \subseteq D$),*
- *$t_T(\mathcal{S})$ is the time for training the execution strategy $\mathcal{S}$,*
- *$t_Q(q)$ is the time for evaluating a query q,*
- *$t_R(d)$ is the time for retrieving a document d,*
- *$t_F(d)$ is the time for filtering a retrieved document d, and*
- *$t_P(d)$ is the time for processing a document d with P.*

Assuming that the time to evaluate a query is constant across queries (i.e., $t_Q = t_Q(q)$, for every $q \in Q_{sent}$) and that the time to retrieve, filter, or process a single document is constant across documents (i.e., $t_R = t_R(d)$, $t_F = t_F(d)$, $t_P = t_P(d)$, for every $d \in D$), we have:

$$Time(\mathcal{S}, D) = t_T(\mathcal{S}) + t_Q \cdot |Q_{sent}| + \big(t_R + t_F\big) \cdot |D_{retr}| + t_P \cdot |D_{proc}|$$

□

DEFINITION 3.2. **[Recall]** *Consider a text-centric task, a database of text documents D, and an execution strategy $\mathcal{S}$ for the task, with an underlying document processor P. Let D_{proc} be the set of documents from D that $\mathcal{S}$ processes with P. Then, we define the* recall *of $\mathcal{S}$ over D, $Recall(\mathcal{S}, D)$, as*

$$Recall(\mathcal{S}, D) = \frac{|Tokens(P, D_{proc})|}{|Tokens(P, D)|} \quad (1)$$

where $Tokens(P, \mathcal{D})$ is the set of tokens that the document processor P extracts from the set of documents $\mathcal{D}$. □

Input: database D, recall threshold τ, document processor P
Output: tokens $Tokens_{retr}$
$Tokens_{retr} = \emptyset$, $D_{retr} = \emptyset$, $recall = 0$
while $recall < \tau$ **do**
  Retrieve an unprocessed document d and add d to D_{retr}
  Process d using P and add extracted tokens to $Tokens_{retr}$
  $recall = |Tokens_{retr}|/|Tokens|$
end
return $Tokens_{retr}$

Figure 4: The *Scan* strategy

Our problem formulation is close, conceptually, to the evaluation of a selection predicate in an RDBMS. In relational databases, the query optimizer selects an access path (i.e., a sequential scan or a set of indexes) that is expected to lead to an efficient execution. We follow a similar structure in our work. In the next section, we describe the alternate evaluation methods that are at the core of the execution strategies for text-centric tasks that have been discussed in the literature.[1] Then, in subsequent sections, we analyze these strategies to see how their performance depends on the task and database characteristics.

4. EXECUTION STRATEGIES

In this section, we review the alternate execution plans that can be used for the text-centric tasks described above, and discuss how we can "instantiate" each generic plan for each task of Section 2. Our discussion assumes that each task has a target recall value τ, $0 < \tau \leq 1$, that needs to be achieved (see Definition 3.2), and that the execution can stop as soon as the target recall is reached.

4.1 Scan

The *Scan* ($\mathcal{SC}$) strategy is a crawl-based strategy that processes each document in a database D exhaustively until the number of tokens extracted satisfies the target recall τ (see Figure 4).

The *Scan* execution strategy does not need training and does not send any queries to the database. Hence, $t_T(\mathcal{SC}) = 0$ and $|Q_{sent}| = 0$. Furthermore, *Scan* does not apply any filtering, hence $t_F = 0$ and $|D_{proc}| = |D_{retr}|$. Therefore, the execution time of *Scan* is:

$$Time(\mathcal{SC}, D) = |D_{retr}| \cdot (t_R + t_P) \quad (2)$$

The *Scan* strategy is the basic evaluation strategy that many text-centric algorithms use when there are no efficiency issues, or when recall, which is guaranteed to be perfect according to Definition 3.2, is important. We should stress, though, that $|D_{retr}|$ for *Scan* is not necessarily equal to $|D|$: when the target recall τ is low, or when tokens appear redundantly in multiple documents, *Scan* may reach the target recall without processing all the documents in D. In Section 5, we show how to estimate the value of $|D_{retr}|$ that is needed by *Scan* to reach a target recall τ.

A basic version of *Scan* accesses documents in random order. Variations of *Scan* might impose a specific processing order and prioritize, say, "promising" documents that are estimated to contribute many new tokens. Another natural improvement of *Scan* is to avoid processing altogether documents expected not to contribute any tokens; this is the basic idea behind *Filtered Scan*, which we discuss next.

[1] While it is impossible to analyze all existing techniques within a single paper, we believe that we offer valuable insight on how to formally analyze many query- and crawl-based strategies, hence offering the ability to predict *a-priori* the expected performance of an algorithm.

Input: database D, recall threshold τ, classifier C, document processor P
Output: tokens $Tokens_{retr}$
$Tokens_{retr} = \emptyset$, $D_{retr} = \emptyset$, $recall = 0$
while $recall < \tau$ *and* $|D_{retr}| < |D|$ **do**
  Retrieve an unprocessed document d and add d to D_{retr}
  Use C to classify d as *useful* for the task or not
  if d *is useful* **then**
    Process d using P and add extracted tokens to $Tokens_{retr}$
  end
  $recall = |Tokens_{retr}|/|Tokens|$
end
return $Tokens_{retr}$

Figure 5: The *Filtered Scan* strategy

4.2 Filtered Scan

The *Filtered Scan* ($\mathcal{FS}$) strategy is a variation of the basic *Scan* strategy. While *Scan* indistinguishably processes all documents retrieved, *Filtered Scan* first uses a classifier C to decide whether a document d is *useful*, i.e., whether d contributes at least one token (see Figure 5). Given the potentially high cost of processing a document with the document processor P, a quick rejection of useless documents can speed up the overall execution considerably.

The training time $t_T(\mathcal{FS})$ for *Filtered Scan* is equal to the time required to build the classifier C for a specific task. Training represents a one-time cost for a task, so in a repeated execution of the task (i.e., over a new database) the classifier will be available with $t_T(\mathcal{FS}) = 0$. This is the case that we assume in the rest of the analysis. Since *Filtered Scan* does not send any queries, $|Q_{sent}| = 0$. While *Filtered Scan* retrieves and classifies $|D_{retr}|$ documents, it actually processes only $C_\sigma \cdot |D_{retr}|$ documents, where C_σ is the "selectivity" of the classifier C, defined as the fraction of database documents that C judges as useful. Therefore, according to Definition 3.1, the execution time of *Filtered Scan* is:

$$Time(\mathcal{FS}, D) = |D_{retr}| \cdot \left(t_R + t_F + C_\sigma \cdot t_P\right) \quad (3)$$

In Section 5, we show how to estimate the value of $|D_{retr}|$ that is needed for *Filtered Scan* to reach the target recall τ.

Filtered Scan is used when t_P is high and there are many database documents that do not contribute any tokens to the task at hand. For *Task 1*, *Filtered Scan* is used by Proteus [29], which uses a hand-built set of inexpensive rules to discard useless documents. For *Task 2*, the *Filtered Scan* strategy is typically not applicable, since all the documents are useful. For *Task 3*, the *Filtered Scan* strategy corresponds to a "hard" focused crawler [14] that prunes the search space by only considering documents that are pointed to by useful documents.

Both *Scan* and *Filtered Scan* are crawl-based strategies. Next, we describe two query-based strategies, *Iterative Set Expansion*, which emulates query-based strategies that rely on "bootstrapping" techniques, and *Automatic Query Generation*, which generates queries automatically, without using the database results.

4.3 Iterative Set Expansion

Iterative Set Expansion ($\mathcal{ISE}$) is a query-based strategy that queries a database with tokens as they are discovered, starting with a typically small number of user-provided *seed* tokens $Tokens_{seed}$. The intuition behind this strategy is that known tokens might lead to unseen tokens via documents that have both seen and unseen tokens (see Figure 6). Queries are derived from the tokens in a task-specific way. For example, a *Task 1* tuple $\langle Cholera, 1999, Nigeria\rangle$ for *DiseaseOutbreaks* might be turned into query [*Cholera AND Nigeria*]; this query, in turn, might help retrieve documents that report other disease outbreaks, such as $\langle Cholera, 2005, Senegal\rangle$ and $\langle Measles, 2004, Nigeria\rangle$.

Iterative Set Expansion has no training phase, hence $t_T(\mathcal{ISE}) = 0$. We assume that *Iterative Set Expansion* has to send $|Q_{sent}|$ queries to reach the target recall. In Section 5, we show how to estimate this value of $|Q_{sent}|$. Also, since *Iterative Set Expansion* processes all the documents that it retrieves, $t_F = 0$ and $|D_{proc}| = |D_{retr}|$. Then, according to Definition 3.1:

$$Time(\mathcal{ISE}, D) = |Q_{sent}| \cdot t_Q + |D_{retr}| \cdot \left(t_R + t_P\right) \quad (4)$$

Informally, we expect *Iterative Set Expansion* to be efficient when tokens tend to co-occur in the database documents. In this case, we can start from a few tokens and "reach" the remaining ones. (We define reachability formally in Section 5.4.) In contrast, this strategy might "stall" and lead to poor recall for scenarios when tokens occur in isolation, as was analyzed in [3].

Iterative Set Expansion has been successfully applied in many tasks. For *Task 1*, *Iterative Set Expansion* corresponds to the *Tuples* algorithm for information extraction [2], which was shown to outperform crawl-based strategies when $|D_{useful}| \ll |D|$, where D_{useful} is the set of documents in D that "contribute" at least one token for the task. For *Task 2*, *Iterative Set Expansion* corresponds to the query-based sampling algorithm by Callan et al. [10], which creates a content summary of a database from a document sample obtained via query words derived (randomly) from the already retrieved documents. For *Task 3*, *Iterative Set Expansion* is not directly applicable, since there is no notion of "co-occurrence." Instead, strategies that start with a set of topic-specific queries are preferable. Next, we describe such a query-based strategy.

Input: database D, recall threshold τ, tokens $Tokens_{seed}$, document processor P
Output: tokens $Tokens_{retr}$
$Tokens_{retr} = \emptyset$, $D_{retr} = \emptyset$, $recall = 0$
while $recall < \tau$ *and* $Tokens_{seed} \neq \emptyset$ **do**
  Remove a token t from $Tokens_{seed}$
  Transform t into a query q and issue q to D
  Retrieve up to $maxD$ documents matching q
  foreach *newly retrieved document* d **do**
    Add d to D_{retr}
    Process d using P and add newly extracted tokens to $Tokens_{retr}$ and $Tokens_{seed}$
    $recall = |Tokens_{retr}|/|Tokens|$
    if $recall \geq \tau$ **then**
      return $Tokens_{retr}$
    end
  end
end
return $Tokens_{retr}$

Figure 6: The *Iterative Set Expansion* strategy

Input: database D, recall threshold τ, document processor P, queries Q
Output: tokens $Tokens_{retr}$
$Tokens_{retr} = \emptyset$, $D_{retr} = \emptyset$, $recall = 0$
foreach *query* $q \in Q$ **do**
  Retrieve up to $maxD$ documents matching q
  foreach *newly retrieved document* d **do**
    Add d to D_{retr}
    Process d using P and add extracted tokens to $Tokens_{retr}$
    $recall = |Tokens_{retr}|/|Tokens|$
    if $recall \geq \tau$ **then**
      return $Tokens_{retr}$
    end
  end
end
return $Tokens_{retr}$

Figure 7: The *Automatic Query Generation* strategy

4.4 Automatic Query Generation

Automatic Query Generation ($\mathcal{AQG}$) is a query-based strategy for retrieving useful documents for a task. *Automatic Query Generation* works in two stages: query generation and execution. In the first stage, *Automatic Query Generation* trains a classifier to categorize documents as useful or not for the task; then, rule-extraction algorithms

derive queries from the classifier. In the execution stage, *Automatic Query Generation* searches a database using queries that are expected to retrieve useful documents. For example, for *Task 3* with *botany* as the topic, *Automatic Query Generation* generates queries such as [*plant AND phylogeny*] and [*phycology*]. (See Figure 7.)

The training time for *Automatic Query Generation* involves downloading a training set D_{train} of documents and processing them with P, incurring a cost of $|D_{train}| \cdot (t_R + t_P)$. Training time also includes the time for the actual training of the classifier. This time depends on the learning algorithm and is, typically, at least linear in the size of D_{train}. Training represents a one-time cost for a task, so in a repeated execution of the task (i.e., over a new database) the classifier will be available with $t_T(\mathcal{AQG}) = 0$. This is the case that we assume in the rest of the analysis. During execution, the *Automatic Query Generation* strategy sends $|Q_{sent}|$ queries and retrieves $|D_{retr}|$ documents, which are then all processed by P, without any filtering[2] (i.e., $|D_{proc}| = |D_{retr}|$). In Section 5, we show how to estimate the values of $|Q_{sent}|$ and $|D_{retr}|$ that are needed for *Automatic Query Generation* to reach a target recall τ. Then, according to Definition 3.1:

$$Time(\mathcal{AQG}, D) = |Q_{sent}| \cdot t_Q + |D_{retr}| \cdot \left(t_R + t_P\right) \quad (5)$$

The *Automatic Query Generation* strategy was proposed under the name *QXtract* for *Task 1* [2]; it was also used for *Task 2* in [31] and for *Task 3* in [20].

The description of the execution time has so far relied on parameters (e.g., $|D_{retr}|$) that are not known before executing the strategies. In the next section, we focus on the central issue of estimating these parameters. In the process, we show that the performance of each strategy depends heavily on task-specific properties of the underlying database; then, in Section 6 we show how to characterize the required database properties and select the best execution strategy for a task.

5. ESTIMATING EXECUTION PLAN COSTS

In the previous section, we presented four alternative execution plans and described the execution cost for each plan. Our description focused on describing the main factors of the actual execution time of each plan and did not provide any insight on how to estimate these costs: many of the parameters that appear in the cost equations are *outcomes* of the execution and cannot be used to estimate or predict the execution cost. In this section, we show that the cost equations described in Section 4 depend on a few fundamental task-specific properties of the underlying databases, such as the distribution of tokens across documents. Our analysis reveals the strengths and weaknesses of the execution plans and (most importantly) provides an easy way to estimate the cost of each technique for reaching a target recall τ. The rest of the section is structured as follows. First, Section 5.1 describes the notation and gives the necessary background. Then, Sections 5.2 and 5.3 analyze the two crawl-based techniques, *Scan* and *Filtered Scan*, respectively. Finally, Sections 5.4 and 5.5 analyze the two query-based techniques, *Iterative Set Expansion* and *Automatic Query Generation*, respectively.

5.1 Preliminaries

In our analysis, we use some task-specific properties of the underlying databases, such as the distribution of tokens across documents. We use $g(d)$ to represent the "degree" of a document d for a document processor P, which is defined as the number of distinct tokens extracted from d using P. Similarly, we use $g(t)$ to represent the "degree" of a token t in a database D, which is defined as the number of distinct documents that contain t in D. Finally, we use $g(q)$ to represent the "degree" of a query q in a database D, which is defined as the number of documents from D retrieved by query q.

[2]Note that we could also consider "filtered" versions of *Iterative Set Expansion* and *Automatic Query Generation*, just as we do for *Scan*. For brevity, we do not study such variations: filtering is less critical for the query-based strategies than for *Scan*, because queries generally retrieve a reasonably small fraction of the database documents.

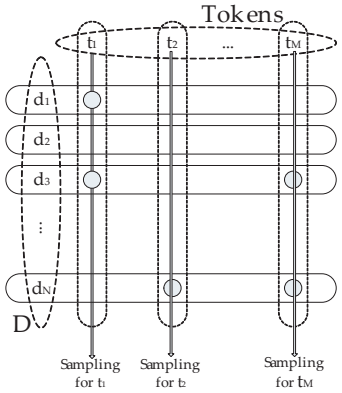

Figure 8: Modeling *Scan* as multiple sampling processes, one per token, running in parallel over D

In general, we do not know a-priori the exact distribution of the token, document, and query degrees for a given task and database. However, we typically know the distribution *family* for these degrees, and we just need to estimate a few parameters to identify the actual distribution for the task and database. For *Task 1*, the document and token degrees tend to follow a power-law distribution [3], as we will see in Section 7. For *Task 2*, token degrees follow a power-law distribution [43] and document degrees follow roughly a lognormal distribution [34]; we provide further evidence in Section 7. For *Task 3*, the document and token distributions are, by definition, uniform over D_{useful} with $g(t) = g(d) = 1$. In Section 6, we describe how to estimate the parameters of each distribution.

5.2 Cost of Scan

According to Equation 2, the cost of *Scan* is determined by the size of the set D_{retr}, which is the number of documents retrieved to achieve a target recall τ.[3] To compute $|D_{retr}|$, we base our analysis on the fact that *Scan* retrieves documents in no particular order and does not retrieve the same document twice. This process is equivalent to *sampling from a finite population* [38]. Conceptually, *Scan* samples for multiple tokens during execution. Therefore, we treat *Scan* as performing multiple "*sampling from a finite population*" processes, running in parallel over D (see Figure 8). Each sampling process corresponds to a token $t \in Tokens$. According to probability theory [38, page 56], the probability of observing a token t k times in a sample of size S follows the hypergeometric distribution. For $k = 0$, we get the probability that t does *not* appear in the sample, which is $\binom{|D|-g(t)}{S} / \binom{|D|}{S}$. The complement of this value is the probability that t appears in at least one document in the set of S retrieved documents. So, after processing S documents, the expected number of retrieved tokens for *Scan* is:

$$E[|Tokens_{retr}|] = \sum_{t \in Tokens} 1 - \frac{(|D| - g(t))!\,(|D| - S)!}{(|D| - g(t) - S)!|D|!} \quad (6)$$

Hence, we estimate the number of documents that *Scan* should retrieve to achieve a target recall τ as:

$$\widehat{|D_{retr}|} = \min\{S : E[|Tokens_{retr}|] \geq \tau |Tokens|\} \quad (7)$$

The number of documents $|D_{retr}|$ retrieved by *Scan* depends on the token degree distribution. For many databases, the distribution of $g(t)$ is highly skewed and follows a power-law distribution: a few tokens appear in many documents, while the majority of tokens can only be

[3]We assume that the values of t_R and t_P are known or that we can easily estimate them by repeatedly retrieving and processing a few sample documents.

extracted from only a few documents. For example, the *Task 1* tuple $\langle SARS, 2003, China\rangle$ can be extracted from hundreds of documents in the New York Times archive, while the tuple $\langle Diphtheria, 2003, Afghanistan\rangle$ appears only in a handful of documents. By estimating the parameters of the power-law distribution, we can then compute the expected values of $g(t)$ for the (unknown) tokens in D and use Equations 6 and 7 to derive the expected cost of *Scan*. In Section 6, we show how to perform such estimations on-the-fly.

The analysis above assumes a random retrieval of documents. If the documents are retrieved in a special order, which is unlikely for the task scenarios that we consider, then we should model *Scan* as "stratified" sampling without replacement: instead of assuming a single sampling pass, we decompose the analysis into multiple "strata" (i.e., into multiple sampling phases), each one with its own $g(\cdot)$ distribution. A simple instance of such technique is *Filtered Scan*, which (conceptually) samples *useful* documents first, as discussed next.

5.3 Cost of Filtered Scan

Filtered Scan is a variation of the basic *Scan* strategy, therefore the analysis of both strategies is similar. The key difference between these strategies is that *Filtered Scan* uses a classifier to filter documents, which *Scan* does not. The *Filtered Scan* classifier thus limits the number of documents processed by the document processor P. Two properties of the classifier C are of interest for our analysis:

- The classifier's selectivity C_σ: if D_{proc} is the set of documents in D deemed *useful* by the classifier (and then processed by P), then $C_\sigma = \frac{|D_{proc}|}{|D|}$.
- The classifier's recall C_r: this is the fraction of useful documents in D that are also classified as *useful* by the classifier. The value of C_r affects the *effective* token degree for each tuple t: now each token appears, on average, $C_r \cdot g(t)$ times[4] in D_{proc}, the set of documents actually processed by P.

Using these observations and following the methodology that we used for *Scan*, we have:

$$E[|Tokens_{retr}|] = \sum_{t \in Tokens} 1 - \frac{(C_\sigma \cdot |D| - C_r \cdot g(t))!\,(C_\sigma \cdot |D| - S)!}{(C_\sigma \cdot |D| - C_r \cdot g(t) - S)!\,(C_\sigma \cdot |D|)!} \quad (8)$$

Again, similar to *Scan*, we have:

$$\widehat{|D_{retr}|} = \frac{\widehat{|D_{proc}|}}{C_\sigma} = \frac{\min\{S : E[|Tokens_{retr}|] \geq \tau |Tokens|\}}{C_\sigma} \quad (9)$$

Equations 8 and 9 show the dependence of *Filtered Scan* on the performance of the classifier. When C_σ is high, almost all documents in D are processed by P, and the savings compared to *Scan* are minimal, if any. When a classifier has low recall C_r, then many *useful* documents are rejected and the effective token degree decreases, in turn increasing $|D_{retr}|$. We should also emphasize that if the recall of the classifier is low, then *Filtered Scan* is not guaranteed to reach the target recall τ. In this case, the maximum achievable recall might be less than one and $|D_{retr}| = |D|$.

5.4 Cost of Iterative Set Expansion

So far, we have analyzed two crawling-based strategies. Before moving to the analysis of the *Iterative Set Expansion* query-based strategy, we define "queries" more formally as well as a graph-based representation of the querying process, originally introduced in [3].

DEFINITION 5.1. **[Querying Graph]** *Consider a database D and a document processor P. We define the* querying graph $QG(D, P)$ *of D with respect to P as a bipartite graph containing the elements of Tokens and D as nodes, where Tokens is the set of tokens that P derives from D. A directed edge from a document node d to a token node t means that P extracts t from d. An edge from a token node t to document node d means that d is returned from D as a result to a query derived from the token t.* □

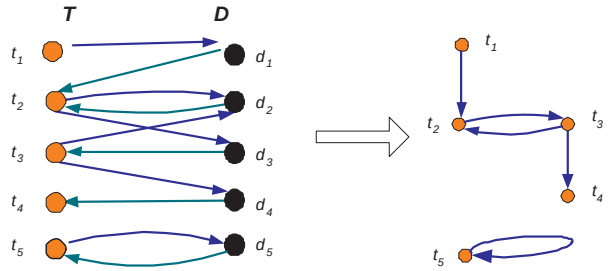

Figure 9: Portion of the querying and reachability graphs of a database

For example, suppose that token t_1, after being suitably converted into a query, retrieves a document d_1 and, in turn, that processor P extracts the token t_2 from d_1. Then, we insert an edge into QG from t_1 to d_1, and also an edge from d_1 to t_2. We consider an edge $d \rightarrow t$, originating from a document node d and pointing to a token node t, as a "*contains*" edge, and an edge $t \rightarrow d$, originating from a token node t and pointing to a document node d, as a "*retrieves*" edge.

Using the querying graph, we analyze the cost and recall of *Iterative Set Expansion*. As a simple example, consider the case where the initial $Tokens_{seed}$ set contains a single token, t_{seed}. We start by querying the database using the query derived by t_{seed}. The cost at this stage is a function of the number of documents retrieved by t_{seed}: this is the number of neighbors at distance one from t_{seed} in the querying graph QG. The recall of *Iterative Set Expansion*, at this stage, is determined by the number of tokens derived from the retrieved documents, which is equal to the number of neighbors at distance two from t_{seed}. Following the same principle, the cost in the next stage (after querying with the tokens in distance two) depends on the number of neighbors at distance three and recall is determined by the number of neighbors at distance four, and so on.

The previous example illustrates that the recall of *Iterative Set Expansion* is bounded by the number of tokens "reachable" from the $Tokens_{seed}$ tokens; the execution time is also bounded by the number of documents and tokens that are "reachable" from the $Tokens_{seed}$ tokens. The structure of the querying graph thus defines the performance of *Iterative Set Expansion*. To compute the interesting properties of the querying graph, we resort to the theory of random graphs: our approach is based on the methodology suggested by Newman et al. [35] and uses generating functions to describe the properties of the querying graph QG. We define the generating functions $Gd_0(x)$ and $Gt_0(x)$ to describe the degree distribution[5] of a *randomly chosen* document and token, respectively:

$$Gd_0(x) = \sum_k pd_k \cdot x^k, \qquad Gt_0(x) = \sum_k pt_k \cdot x^k \quad (10)$$

where pd_k is the probability that a randomly chosen document d contains k tokens (i.e., $pd_k = Pr\{g(d) = k\}$) and pt_k is the probability that a randomly chosen token t retrieves k documents (i.e., $pt_k = Pr\{g(t) = k\}$) when used as a query.

[4] We assume uniform recall across tokens, i.e., that the classifier's errors are not biased towards a specific set of tokens. This is a reasonable assumption for most classifiers. Nevertheless, we can easily extend the analysis and model any classifier bias by using a different classifier recall $C_r(t)$ for each token t.

[5] We use undirected graph theory despite the fact that our querying graph is directed. Using directed graph results would of course be preferable, but it would require knowledge of the *joint* distribution of incoming and outgoing degrees for all nodes of the querying graph, which would be challenging to estimate. So we rely on undirected graph theory, which requires only knowledge of the two marginal degree distributions, namely the token and document degree distributions.

In our setting, we are also interested in the degree distribution for a document (or token, respectively) chosen by *following a random edge*. Using the methodology of Newman et al. [35], we define the functions $Gd_1(x)$ and $Gt_1(x)$ that describe the degree distribution for a document and token, respectively, chosen by following a random edge:

$$Gd_1(x) = x\frac{Gd_0'(x)}{Gd_0'(1)}, \qquad Gt_1(x) = x\frac{Gt_0'(x)}{Gt_0'(1)} \tag{11}$$

where $Gd_0'(x)$ is the first derivative of $Gd_0(x)$ and $Gt_0'(x)$ is the first derivative of $Gt_0(x)$, respectively. (See [35] for the proof.)

For the rest of the analysis, we use the following useful properties of generating functions [41]:

- *Moments*: The i-th moment of the probability distribution generated by a function $G(x)$ is given by the i-th derivative of the generating function $G(x)$, evaluated at $x = 1$. We mainly use this property to compute efficiently the mean of the distribution described by $G(x)$.
- *Power*: If $X_1, \ldots, X_m$ are independent, identically distributed random variables generated by the generating function $G(x)$, then the sum of these variables, $S_m = \sum_{i=1}^{m} X_i$, has generating function $[G(x)]^m$.
- *Composition*: If $X_1, \ldots, X_m$ are independent, identically distributed random variables generated by the generating function $G(x)$, and m is also an independent random variable generated by the function $F(x)$, then the sum $S_m = \sum_{i=1}^{m} X_i$ has generating function $F(G(x))$.

Using these properties and Equations 10 and 11, we can proceed to analyze the cost of *Iterative Set Expansion*. Assume that we are in the stage where *Iterative Set Expansion* has sent a set Q of tokens as queries. These tokens were discovered by following random edges on the graph; therefore, the degree distribution of these tokens is described by $Gt_1(x)$ (Equation 11). Then, by the *Power* property, the distribution of the total number of retrieved *documents* (which are pointed to by these tokens) is given by the generating function:[6]

$$Gd_2(x) = [Gt_1(x)]^{|Q|} \tag{12}$$

Now, we know that D_{retr} in Equation 4 is a random variable and its distribution is given by $Gd_2(x)$. We also know that we retrieve documents by following random edges on the graph; therefore, the degree distribution of these documents is described by $Gd_1(x)$ (Equation 11). Then, by the *Composition* property[7], the distribution of the total number of *tokens* $|Tokens_{retr}|$ retrieved by the D_{retr} documents is given by the generating function:[8]

$$Gt_2(x) = Gd_2(Gd_1(x)) = [Gt_1(Gd_1(x))]^{|Q|} \tag{13}$$

Finally, we use the *Moments* property to compute the expected values for $|D_{retr}|$ and $|Tokens_{retr}|$, after *Iterative Set Expansion* sends Q queries.

$$E[|D_{retr}|] = \left[\frac{d}{dx}[Gt_1(x)]^{|Q|}\right]_{x=1} \tag{14}$$

$$E[|Tokens_{retr}|] = \left[\frac{d}{dx}[Gt_1(Gd_1(x))]^{|Q|}\right]_{x=1} \tag{15}$$

Hence, the number of queries $|Q_{sent}|$ sent by *Iterative Set Expansion* to reach the target recall τ is:

$$\widehat{|Q_{sent}|} = \min\{Q : E[|Tokens_{retr}|] \geq \tau |Tokens|\} \tag{16}$$

[6]This is the number of *non-distinct* documents. To compute the number of distinct documents, we use the *sieve* method. For details, see [41, page 110].

[7]We use the *Composition* property and not the *Power* property because $|D_{retr}|$ is a random variable.

[8]Again, this is the number of *non-distinct* tokens. To compute the number of distinct tokens, we use the *sieve* method. For details, see [41, page 110].

Our analysis, so far, did not account for the fact that the tokens in a database are not always "reachable" in the querying graph from the tokens in $Tokens_{seed}$. As we have briefly discussed, though, the ability to reach all the tokens is necessary for *Iterative Set Expansion* to achieve good recall. Before elaborating further on the subject, we describe the concept of the *reachability graph*, which we originally introduced in [3] and is fundamental for our analysis.

DEFINITION 5.2. **[Reachability Graph]** *Consider a database D, and an execution strategy $\mathcal{S}$ for a task with an underlying document processor P and querying strategy R. We define the* reachability graph *$RG(D, \mathcal{S})$ of D with respect to $\mathcal{S}$ as a graph whose nodes are the tokens that P derives from D, and whose edge set E is such that a directed edge $t_i \rightarrow t_j$ means that P derives t_j from a document that R retrieves using t_i.* □

Figure 9 shows the reachability graph derived from an underlying querying graph, illustrating how edges are added to the reachability graph. Since token t_2 retrieves document d_3 and d_3 contains token t_3, the reachability graph contains the edge $t_2 \rightarrow t_3$. Intuitively, a *path* in the reachability graph from a token t_i to a token t_j means that there is a set of queries that start with t_i and lead to the retrieval of a document that contains the token t_j. In the example in Figure 9, there is a path from t_2 to t_4, through t_3. This means that query t_2 can help discover token t_3, which in turn helps discover token t_4. The absence of a path from a token t_i to a token t_j in the reachability graph means that we cannot discover t_j starting from t_i. This is the case for the tokens t_2 and t_5 in Figure 9.

The reachability graph is a directed graph and its connectivity defines the maximum achievable recall of *Iterative Set Expansion*: the upper limit for the recall of *Iterative Set Expansion* is equal to the total size of the connected components that include tokens in $Tokens_{seed}$. In random graphs, typically we observe two scenarios: either the graph is disconnected and has a large number of disconnected components, or we observe a giant component and a set of small connected components. Chung and Lu [18] proved this for graphs with a power-law degree distribution, and also provided the formulas for the composition of the size of the components. Newman et al. [35] provide similar results for graphs with arbitrary degree distributions. Interestingly for our problem, the size of the connected components can be estimated for many degree distributions using only a small number of parameters (e.g., for power-law graphs we only need an *estimate* of the average node out-degree [18] to compute the size of the connected component; in Section 6 we explain how we obtain such estimates). By estimating only a small number of parameters, we can thus characterize the performance limits of the *Iterative Set Expansion* strategy.

As discussed, *Iterative Set Expansion* relies on the discovery of new tokens to derive new queries. Therefore, in sparse and "disconnected" databases, *Iterative Set Expansion* can exhaust the available queries and still miss a significant part of the database, leading to low recall. In such cases, if high recall is a requirement, different strategies are preferable. The alternative query-based strategy that we examine next, *Automatic Query Generation*, showcases a different querying approach: instead of deriving new queries during execution, *Automatic Query Generation* generates a set of queries offline and then queries the database without using query results as feedback.

5.5 Cost of Automatic Query Generation

Section 4.4 showed that the cost of *Automatic Query Generation* consists of two main components: the training cost and the querying cost. Training represents a one-time cost for a task, as discussed in Section 4.4, so we ignore it in our analysis. Therefore, the main component that remains to be analyzed is the querying cost.

To estimate the querying cost of *Automatic Query Generation*, we need to estimate recall after sending a set Q of queries and the number of retrieved documents $|D_{retr}|$ at that point. Each query q retrieves

$g(q)$ documents, and a fraction $p(q)$ of these documents is useful for the task at hand. Assuming that the queries are biased only towards retrieving *useful* documents and not towards any other particular set of documents, the queries are conditionally independent[9] within the set of documents D_{useful} and within the rest of the documents, $D_{useless}$. Therefore, the probability that a useful document is retrieved by a query q is $\frac{p(q) \cdot g(q)}{|D_{useful}|}$. Hence, the probability that a useful document d is retrieved by at least one query is:

$$1 - Pr\{\textit{d not retrieved by any query}\} = 1 - \prod_{i=1}^{|Q|} \left(1 - \frac{p(q_i) \cdot g(q_i)}{|D_{useful}|}\right)$$

So, given the values of $p(q_i)$ and $g(q_i)$, the expected number of *useful* documents that are retrieved is:

$$E[|D_{retr}^{useful}|] = |D_{useful}| \cdot \left(1 - \prod_{i=1}^{|Q|} \left(1 - \frac{p(q_i) \cdot g(q_i)}{|D_{useful}|}\right)\right) \quad (17)$$

and the number of useless documents retrieved is:

$$E[|D_{retr}^{useless}|] = |D_{useless}| \cdot \left(1 - \prod_{i=1}^{|Q|} \left(1 - \frac{(1 - p(q_i)) \cdot g(q_i)}{|D_{useless}|}\right)\right) \quad (18)$$

Assuming that the "precision" of a query q is independent of the number of documents that q retrieves,[10] we get a simpler expression:

$$E[|D_{retr}^{useful}|] = |D_{useful}| \cdot \left(1 - \left(1 - \frac{E[p(q)] \cdot E[g(q)]}{|D_{useful}|}\right)^{|Q|}\right) \quad (19)$$

where $E[p(q)]$ is the average precision of the queries and $E[g(q)]$ is the average number of retrieved documents per query. An analogous expression follows for $E[|D_{retr}^{useless}|]$. The expected number of retrieved documents is then:

$$E[|D_{retr}|] = E[|D_{retr}^{useful}|] + E[|D_{retr}^{useless}|] \quad (20)$$

To compute the recall of *Automatic Query Generation* after issuing Q queries, we use the same methodology that we used for *Filtered Scan*. Specifically, Equation 19 reveals the total number of useful documents retrieved, and these are the documents that contribute to recall. These documents belong to D_{useful}. Hence, similarly to *Scan* and *Filtered Scan*, we model *Automatic Query Generation* as *sampling without replacement*; the essential difference now is that the sampling is over the D_{useful} set. Therefore, we have an effective database size $|D_{useful}|$ and a sample size equal to $|D_{retr}^{useful}|$.[11] By modifying Equation 6 appropriately, we have:

$$E[|\,Tokens_{retr}|] = \sum_{t \in Tokens} 1 - \frac{(|D_{useful}| - g(t))!\,(|D_{useful}| - S)!}{(|D_{useful}| - g(t) - S)!|D_{useful}|!} \quad (21)$$

where $S = |D_{retr}^{useful}|$. A good approximation of the average value of $|\,Tokens_{retr}|$ can be derived by setting S to be the mean value of the $|D_{retr}^{useful}|$ distribution (Equation 19). Similarly to the analysis for *Iterative Set Expansion*, we have:

$$\widehat{|Q_{sent}|} = \min\{Q : E[|\,Tokens_{retr}|] \geq \tau |\,Tokens|\} \quad (22)$$

In this section, we analyzed four alternate execution plans and we showed how their execution time and recall depend on fundamental task-specific properties of the underlying text databases. Next, we show how to exploit the parameter estimation and our cost model to significantly speed-up the execution of text-centric tasks.

[9]The conditional independence assumption implies that the queries are only biased towards retrieving useful documents, and not towards any subset of useful documents.

[10]We observed this assumption to be true in practice.

[11]The documents $D_{retr}^{useless}$ increase the execution time but do not contribute towards recall and we ignore them for recall computation.

6. PUTTING IT ALL TOGETHER

In Section 5, we examined how we can estimate the execution time and the recall of each execution plan by using the values of a few parameters, including the target recall τ and the token, document, and query degree distributions. In this section, we summarize our overall optimization approach, and show how we estimate —on-the-fly— the parameters needed. As we will show in our experimental evaluation in Section 8, our optimization approach leads to efficient executions of the text-centric tasks for the target recall value.

Our cost model of Section 5 relies on a number of parameters. For example, the value of $|\,Tokens|$ (i.e., the number of tokens in the database) is generally unknown before executing a task, and we need it both (1) to decide when we reach the desired recall for the task, to stop execution; and (2) to provide an "educated" estimate to bootstrap our estimation techniques. A robust estimation method for a database and a task is to retrieve multiple document samples from the database and analyze the token overlap across the samples to determine a $|\,Tokens|$ estimate. Similar estimation methods have been proposed for *Task 2* [30] and for *Task 3* [12, page 276]. In our experiments, we do not rely on estimates but rather use the actual value of $|\,Tokens|$.

Some parameters of our cost model, such as classifier selectivity and recall (Section 5.3), can be estimated accurately over a relatively small sample of database documents. In fact, the classifier characteristics for *Filtered Scan* and query degree and precision for *Automatic Query Generation* can be easily estimated during classifier training using cross-validation [16]. To estimate the token and document distributions, we rely on the fact that, for many tasks, we know the general *family* of these distributions, as we discussed in Section 5.1. Hence, our estimation task reduces to estimating a few parameters of well-known distribution families,[12] which we discuss below.

To estimate the parameters of a distribution family for a concrete text-centric task and database, we could resort to a "preprocessing" estimation phase before we start executing the actual task. For this, we could follow Chaudhuri et al. [16], and continue to sample database documents until cross-validation indicates that the estimates are accurate enough. An interesting observation is that having a separate preprocessing estimation phase is not necessary in our scenario, since we can piggyback such estimation phase into the initial stages of an actual execution of the task. In other words, instead of having a preprocessing estimation phase, we can start processing the task and exploit the retrieved documents for "on-the-fly" parameter estimation. This estimation relies on an initial (manual) assignment of tasks to distribution families, which are often natural for the tasks or have been investigated in the literature, as we discussed in Section 5.1.

The basic challenge in this scenario is to guarantee that the parameter estimates that we obtain during execution are as accurate as the estimates that we would derive through random sampling. This is straightforward for *Scan*, since *Scan* effectively performs random sampling over the databases. *Automatic Query Generation* performs random sampling over the D_{useful} documents and is thus equivalent to random sampling for the token degree distribution. For the document degree distribution, *Automatic Query Generation* then underestimates pd_0, the probability that $g(d) = 0$, i.e., that a document d is useless (see Section 5.4), and overestimates pd_k for $k \geq 1$. *Filtered Scan* has a similar bias, introduced by the classifier: since the classifier is not perfect, the observed token and document degrees are typically underestimates of the real values. Fortunately, for both *Filtered Scan* and *Automatic Query Generation*, we can compensate for the introduced bias using a *confusion matrix adjustment* [27], which we use for our

[12]Our current optimization framework follows a parametric approach, by assuming that we know the form of the document and token degree distributions but not their exact parameters. Our framework can also be used in a completely non-parametric setting, in which we make no assumptions on the degree distributions; however, the estimation phase would be more expensive in such a setting. The development of an efficient, completely non-parametric framework is a topic for interesting future research.

Input: database D, recall threshold τ, alternate strategies $\mathcal{S}_1, \ldots \mathcal{S}_n$
Output: tokens $Tokens_{retr}$
$statistics = \emptyset$
while $recall < \tau$ *and* $|D_{retr}| < |D|$ **do**
  /* Locate best possible strategy */
  $estTime = +\infty$
  foreach *available execution strategy* $\mathcal{S}$ **do**
    Compute the $Time(\mathcal{S}, D)$ for reaching target recall τ using the available $statistics$
    if $estTime \geq Time(\mathcal{S}, D)$ **then**
      $strategy = \mathcal{S}$
      $estTime = Time(\mathcal{S}, D)$
    end
  end
  /* Execute strategy */
  Continue execution using $strategy$, for a batch of documents
  Update $statistics$ using D_{retr} and $Tokens_{retr}$
end
return $Tokens_{retr}$

Figure 10: Choosing execution strategies adaptively

experiments. (Due to space restrictions, we omit the details.) Finally, for *Iterative Set Expansion*, we should notice that the execution plan samples the distributions generated by the functions $Gt_1(x)$ and $Gd_1(x)$ (see Section 5.4), while unbiased random sampling samples from the distributions generated by the functions $Gt_0(x)$ and $Gd_0(x)$. Equations 10 and 11 show how to convert the observed estimates to the real ones. Again, due to space restrictions, we omit the details.

Using the observations above, we can now describe our overall optimization approach. The optimization starts by choosing one of the execution plans described in Section 4, based on some "prior knowledge" about the token and document distributions (e.g., that the token and document degrees follow a power-law distribution for *Task 1*). Then, during execution, the adaptive strategy keeps updating the estimates for the token and document distributions and checks for their robustness using cross-validation. At any point in time, if the estimated execution time for reaching the target recall, $Time(\mathcal{S}, D)$, of a competing strategy $\mathcal{S}$ is smaller than that of the current strategy, then the optimizer switches to executing the less expensive strategy, continuing from the execution point reached by the current strategy. Figure 10 summarizes this algorithm.

Next, our experimental evaluation shows that our optimization approach accurately predicts the cost of each execution strategy and –in many cases– manages to choose the strategy that reaches the target recall τ with the minimum execution time.

7. EXPERIMENTAL SETTING

We now describe the experimental setting for each text-centric task of Section 2, including the real-world data sets for the experiments. We also present interesting statistics about the task-specific distribution of tokens in the data sets.

7.1 Information Extraction

Document Processor: For this task, we use the Snowball information extraction system [1] as the document processor (see Section 3). We use two instantiations of Snowball: one for extracting a *DiseaseOutbreaks* relation (*Task 1a*) and one for extracting a *Headquarters* relation (*Task 1b*). For *Task 1a*, the goal is to extract all the tuples of the target relation *DiseaseOutbreaks (DiseaseName, Country)*, which we discussed throughout the paper. For *Task 1b*, the goal is to extract all the tuples of the target relation *Headquarters (Organization,Location)*, where a tuple $\langle o, l \rangle$ in *Headquarters* indicates that organization o has headquarters in location l. A *token* for these tasks is a single *tuple* of the target relation, and a document is a news article from the New York Times archive, which we describe next.

Data Set: We use a collection of newspaper articles from The New York Times, published in 1995 (NYT95) and 1996 (NYT96). We use the NYT95 documents for training and the NYT96 documents for evaluation of the alternative execution strategies. The NYT96 database contains 182,531 documents, with 16,921 tokens for *Task 1a* and 605 tokens for *Task 1b*. Figures 11 and 12 show the document and token degree distribution (Section 5) for *Task 1a*: both distributions follow a power-law, a common distribution for information extraction tasks. The distributions are similar for *Task 1b*.

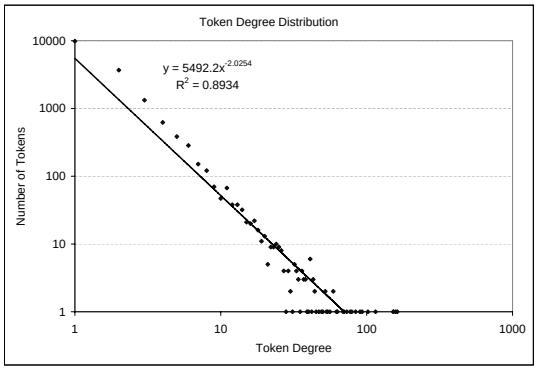

Figure 11: Token distribution for *Task 1*'s *DiseaseOutbreaks*

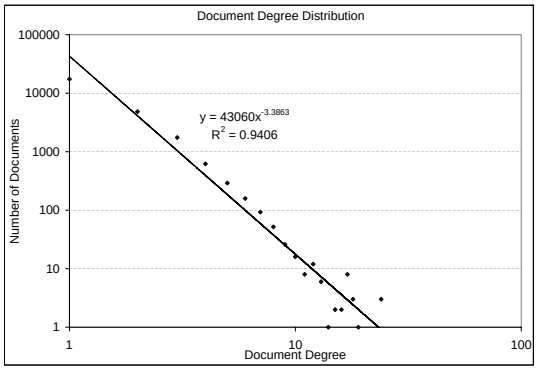

Figure 12: Document distribution for *Task 1*'s *DiseaseOutbreaks*

Execution Plan Instantiation: For *Filtered Scan* we use a rule-based classifier, created using RIPPER [19]. We train RIPPER using a set of 500 useful documents and 1,500 not useful documents from the NYT95 data set. We also use 2,000 documents from the NYT95 data set as a training set to create the queries required by *Automatic Query Generation*. Finally, for *Iterative Set Expansion*, we construct the queries using the conjunction of the attributes of each tuple (e.g., tuple $\langle typhus, Belize \rangle$ results in query [*typhus AND Belize*]).

7.2 Content Summary Construction

Document Processor: For this task, the document processor is a simple tokenizer that extracts the *words* that appear in the eligible documents, defined as a sequence of one or more alphanumeric characters and ignoring capitalization.

Data Set: We use the *20 Newsgroups* data set from the UCI KDD Archive [6]. This data set contains 20,000 messages from 20 Usenet newsgroups. We also randomly retrieve additional Usenet articles to create queries for *Automatic Query Generation*. Figures 13 and 14 show the document and token degree distribution (Section 5) for this task. The document degree follows a lognormal distribution [34] and the token degree follows, as expected [43], a power-law distribution.

Execution Plan Instantiation: For this task, *Filtered Scan* is not directly applicable, since all documents are "useful." For *Iterative Set Expansion*, the queries are constructed using words that appear in previously retrieved documents; this technique corresponds to the *Learned Resource Description* strategy for vocabulary extraction presented by Callan et al. [10]. Finally, for *Automatic Query Gener-*

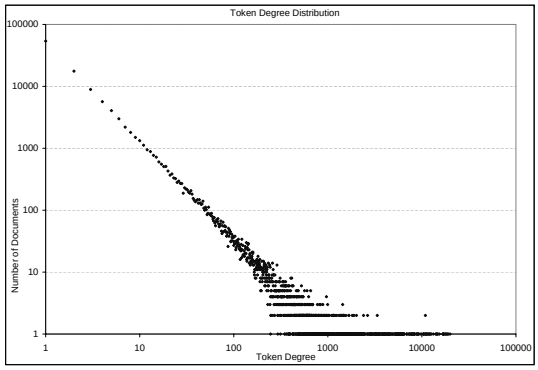

Figure 13: Token distribution for *Task 2*

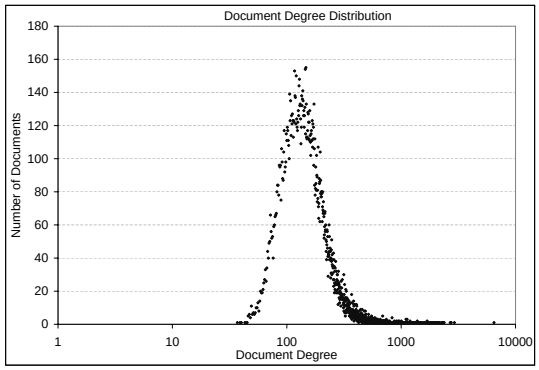

Figure 14: Document distribution for *Task 2*

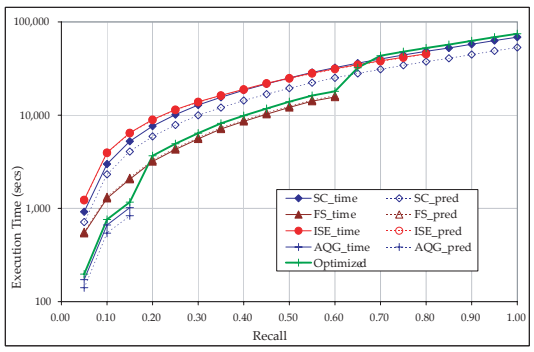

Figure 15: Actual vs. estimated execution times for *Task 1a*, as a function of the target recall τ

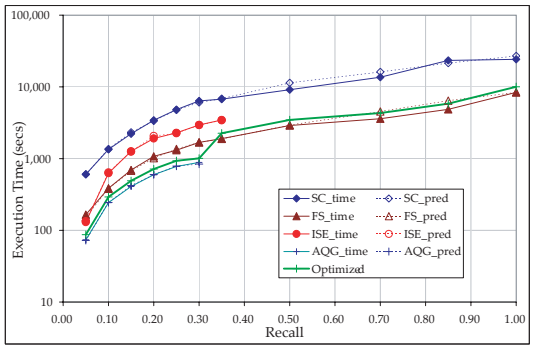

Figure 16: Actual vs. estimated execution times for *Task 1b*, as a function of the target recall τ

ation, we constructed the queries as follows: first, we separate the documents into topics according to the high-level name of the newsgroup (e.g., "comp", "sci", and so on); then, we train a rule-based classifier using RIPPER, which creates rules to assign documents into categories (e.g., *cpu AND ram* $\rightarrow$ *comp* means that a document containing the words "cpu" and "ram" is assigned to the "comp" category). The final queries for *Automatic Query Generation* contain the antecedents of the rules, across all categories. This technique corresponds to the *Focused Probing* strategy for vocabulary extraction presented by Ipeirotis and Gravano [31].

7.3 Focused Resource Discovery

Document Processor: For this task, the document processor is a multinomial Naive Bayes classifier, which detects the topic of a given web page [14]. The topic of choice for our experiments is "Botany."

Data Set: We retrieved 8,000 web pages listed in Open Directory[13] under the category "*Top* $\rightarrow$ *Science* $\rightarrow$ *Biology* $\rightarrow$ *Botany.*" We selected 1,000 out of the 8,000 documents as training documents, and created a multinomial Naive Bayes classifier that decides whether a web page is about Botany. Then, for each of the downloaded Botany pages, we used Google to retrieve all its "backlinks" (i.e., all the web pages that point to that page); again, we classified the retrieved pages and for each page classified as "Botany" we repeated the process of retrieving the backlinks, until none of the backlinks was classified under Botany. This process results in a data set with approximately 12,000 pages about Botany, pointed to by approximately 32,000 useless documents deemed irrelevant to the Botany topic. To augment the data set with additional useless documents, we picked 10 more random topics from the third level of the Open Directory hierarchy and we downloaded all the web pages listed under these topics, for a total of approximately 100,000 pages. After downloading the backlinks for these pages, our data set contained a total of approximately 800,000 pages, out of which 12,000 are relevant to Botany.

Execution Plan Instantiation: For this task, the *Scan* plan corresponds to an unfocused crawl, with a classifier deciding whether each of the retrieved pages belongs to the category of choice. As an instantiation of *Filtered Scan*, we use the "hard" version of the focused crawler described in [14]. The focused crawler starts from a few Botany web pages, and then visits a web page only when at least one of the documents that points to it is useful. Finally, to create queries for *Automatic Query Generation*, we train a RIPPER classifier using the training set, and create a set of rules that assign documents into the *Botany* category. We use these rules to query the data set and retrieve documents.

8. EXPERIMENTAL EVALUATION

In this section, we present our experimental results. Our experiments focus on the execution times of each alternate execution strategy (Section 4) for the tasks and settings described in Section 7. We compute the actual execution times and compare them against our estimates from Section 5. First, we compute our estimates with exact values for the various parameters on which they rely (e.g., token degree distribution). Then, we measure the execution time using our optimization strategy, which relies on approximations of these parameters, as described in Section 6.

Accuracy of Cost Model with Correct Information: The goal of the first set of experiments is to examine whether our cost model of Section 5 captures the real behavior of the alternate execution strategies of Section 4, when all the parameters of the cost model (e.g., token and document degree distributions, classifier characteristics) are known a-priori. For this, we first measure the *actual* execution time of the strategies, for varying values of the target recall τ. The lines *SC_time*, *FS_time*, *ISE_time*, *AQG_time* in Figures 15, 16, 17, and 18 show the actual execution time of the respective strategies for the tasks described in Section 7. Then, to predict the execution time of each

[13] http://www.dmoz.org

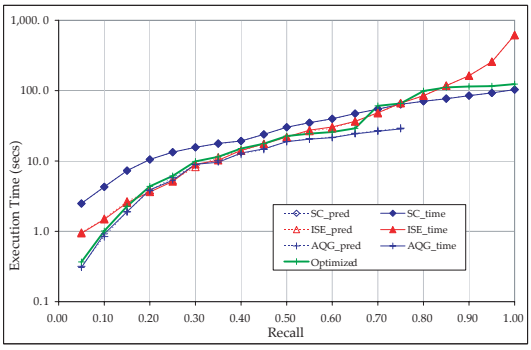

Figure 17: Actual vs. estimated execution times for *Task 2*, as a function of the target recall τ

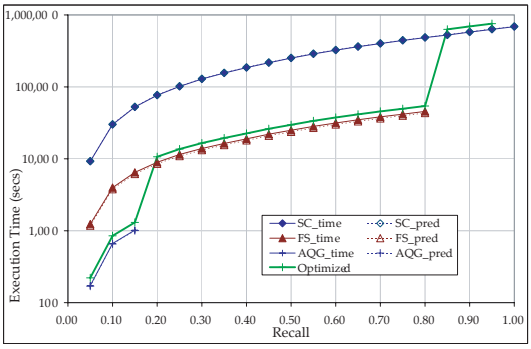

Figure 18: Actual vs. estimated execution times for *Task 3*, as a function of the target recall τ

strategy, we used our equations from Section 5. The lines *SC_pred*, *FS_pred*, *ISE_pred*, *AQG_pred* in Figures 15, 16, 17, and 18 show our execution time estimates for varying values of the target recall τ. The results were exceptionally accurate, confirming the accuracy of our theoretical modeling. The prediction error is typically less than 10% for all values of target recall τ.

Furthermore, our modeling captures well the limitations of each execution plan. For example, for *Task 1a* (Figure 15) *Automatic Query Generation* is the fastest execution plan when target recall $\tau < 0.15$. However, due to the limited number of queries generated during the training phase, *Automatic Query Generation* cannot reach high recall values. (We generated 72 queries for this task.) Our analysis correctly captures this limitation and shows that, for higher recall targets, other strategies are preferable. This limitation also appears for the *Iterative Set Expansion* strategy, confirming previously reported results [3]. The results are similar for *Task 3*: our analysis correctly predicts the execution time and the recall limitations of each strategy.

Quality of Choice of Execution Strategies: After confirming that our cost models accurately capture the actual execution time of the alternate execution strategies, we examine whether the cost model leads to the choice of the fastest plan for each value of target recall τ. We start executing each task by using the strategy that is deemed best for the target recall and the available statistics. These statistics are the expected distribution family of the token and document degrees for the task, with some "default" parameters, such as $\beta = -2$ for power-law distributions (see Section 7). Our experiments also assume knowledge of the actual value of $|Tokens|$, as discussed. During the actual execution, the available statistics are refined; if the acquired statistics show that an alternative strategy is preferable at some point in the execution, then we switch to the strategy that is deemed best (Figure 10).

The *Optimized* line in Figures 15, 16, 17, and 18 shows the actual execution time, for different recall thresholds, using our optimization approach. Typically, our optimizer finishes the task in the same time as the best possible strategy, resulting in execution times that can be up to 10 times faster than alternative plans that we might have picked based on plain intuition or heuristics. For example, consider *Task 1b* with recall target $\tau = 0.35$ (Figure 16): without our cost modeling, we might select *Iterative Set Expansion* or *Automatic Query Generation*, both reasonable choices given the relatively low target recall $\tau = 0.35$. However, *Automatic Query Generation* cannot achieve a recall of 0.35 and *Iterative Set Expansion* is more expensive than *Filtered Scan* for that task. Our optimizer, on the other hand, correctly predicts that *Filtered Scan* should be the algorithm of choice. In this example, our optimizer initially picked *Iterative Set Expansion*, but quickly revised its decision and switched to *Filtered Scan* after gathering statistics from only 1-2% of the database. In some cases, our prediction algorithm overestimates the achievable recall of a strategy (e.g., *Automatic Query Generation*). In such cases, our (incorrectly picked) strategy runs to completion; then, naturally, our technique picks the "next best" strategy and continues the execution from the point reached by the (incorrectly picked) strategy. In such cases, we sometimes even observed a small performance gain derived from this initial mistake, since the "incorrect" strategy outperforms the "correct" strategy for the first part of the execution. This result outlines an interesting future research direction: instead of picking a single strategy for a target recall, we could instead build multi-strategy executions explicitly, by choosing different strategies for different parts of the execution.

Conclusions: We demonstrated how our modeling approach can be used to create an optimizer for text-centric tasks. The presented approach allows for a better understanding of the behavior of query- and crawl-based strategies, in terms of both execution time and recall. Furthermore, our modeling works well even with on-the-fly estimation of the bulk of the required statistics, and results in close-to-optimal execution times. Our work provides fundamental building blocks towards a full query optimizer for text-centric tasks: given a specific target recall (e.g., "find 40% of all disease outbreaks mentioned in the news"), the query optimizer can automatically select the best execution strategy to achieve this recall.

9. RELATED WORK

In this paper, we analyzed and estimated the computational costs of text-centric tasks. We concentrated on three important tasks: information extraction (*Task 1*), text database content summary construction (*Task 2*), and focused resource discovery (*Task 3*).

Implementations of *Task 1* (Section 2.1) traditionally use the *Scan* strategy of Section 4.1, where every document is processed by the information extraction system (e.g., [28, 42]). Some systems use the *Filtered Scan* strategy of Section 4.2, where only the documents that match specific URL patterns (e.g., [7]) or regular expressions (e.g., [29]) are processed further. Agichtein and Gravano [2] presented query-based execution strategies for *Task 1*, corresponding to the *Iterative Set Expansion* strategy of Section 4.3 and *Automatic Query Generation* strategy of Section 4.4. More recently, Etzioni et al. [24] 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. For *Task 2*, the execution strategy in [10] can be cast as an instance of *Iterative Set Expansion*, as discussed in Section 4.3. Another strategy for the same task [31] can be considered an instance of *Automatic Query Generation* (Section 4.4). Interestingly, over large *crawlable* databases, where both query- and crawl-based strategies are possible, query-based strategies have been shown to outperform crawl-based approaches for a related database classification task [26], since small document samples can result in good categorization decisions at a fraction of the processing time required

by full database crawls. For *Task 3*, focused resource discovery systems typically use a variation of *Filtered Scan* [14, 13, 21, 33], where a classifier determines which links to follow for subsequent (expensive) retrieval and processing. Other strategies such as *Automatic Query Generation* may be more effective for some scenarios [20].

Other important text-centric tasks can be modeled in our framework. One such task is *text filtering* (i.e., selecting documents in a text database on a particular topic) [37], which can be executed following either *Filtered Scan*, or, if appropriate, *Automatic Query Generation*. Another task is the construction of comparative web shopping agents [22]. This task requires identifying appropriate web sites (e.g., by using an instance of *Automatic Query Generation*) and subsequently extracting product information from a subset of the retrieved pages (e.g., by using an implementation of *Filtered Scan*). As another example, web question answering systems [4] usually translate a natural language question into a set of web search queries to retrieve documents for a subsequent answer extraction step from a subset of the retrieved documents. This process can be viewed as a combination of *Automatic Query Generation* and *Filtered Scan*. Recently, Ntoulas et al. [36] presented query-based strategies for exhaustively "crawling" a hidden web database while issuing as few queries as possible.

Estimating the cost of a query execution plan requires estimating parameters of the cost model. We adapted common database sampling techniques (e.g., [16, 32]) for our problem, as we discussed in Section 6. Our work is similar in spirit to query optimization over structured relational databases, adapted to the intrinsic differences of executing text-centric tasks. Our work is complementary to previous research on optimizing query plans with user-defined predicates [17], in that we provide a robust way of estimating costs of complex text-centric "predicates". Our work can then be regarded as developing specialized, efficient techniques for important special-purpose "operators" (e.g., as was done for fuzzy matching [15]).

Closest to this paper, in [3] we presented results on modeling and estimating the achievable recall of *Iterative Set Expansion*, for *Task 1* (information extraction) and *Task 2* (database content summary construction). Our current work extends [3] in several ways. First, we develop rigorous cost models for *Iterative Set Expansion*, as well as for three additional general execution strategies, namely *Scan*, *Filtered Scan*, and *Automatic Query Generation*. We also present a principled, cost-based method for selecting the most efficient execution strategy automatically, whereas [3] only provided upper bounds on the possible recall that each strategy could achieve if run to completion. Finally, we thoroughly evaluated our cost estimation models and adaptive execution strategy over multiple tasks and multiple data sets.

10. CONCLUSION

In this paper, we introduced a rigorous cost model for several query- and crawl-based execution strategies that underlie the implementation of many text-centric tasks. We complement our model with a principled cost estimation approach. Our analysis helps predict the execution time and output completeness of important query- and crawl-based algorithms, which until now were only empirically evaluated, with limited theoretical justification. We demonstrated that our modeling can be successfully used to create an optimizer for text-centric tasks, and showed that the optimizer can adaptively select the best execution strategy to achieve a target recall, resulting in executions that can be orders of magnitude faster than alternate choices.

Our work can be extended in multiple directions. For example, the current framework assumes that the document processors have perfect "precision," in that they always produce accurate results. Relaxing this assumption and, correspondingly, predicting the precision of the output produced by different strategies is a natural next step. Another interesting direction is to apply our model to other text-centric tasks and also study how to minimize our reliance on task-specific prior knowledge of the token and document distributions for our analysis.

11. REFERENCES

[1] E. Agichtein and L. Gravano. Snowball: Extracting relations from large plain-text collections. In *DL*, 2000.

[2] E. Agichtein and L. Gravano. Querying text databases for efficient information extraction. In *ICDE*, 2003.

[3] E. Agichtein, P. G. Ipeirotis, and L. Gravano. Modeling query-based access to text databases. In *WebDB*, 2003.

[4] M. Banko, E. Brill, S. Dumais, and J. Lin. AskMSR: Question answering using the World-Wide Web. In *Symp. on Mining Answers from Texts and KBases*, 2002.

[5] M. K. Bergman. The Deep Web: Surfacing hidden value. *Journal of Electronic Publishing*, 7(1), Aug. 2001.

[6] C. L. Blake and C. J. Merz. UCI repository of machine learning databases. `http://www.ics.uci.edu/~mlearn/MLRepository.html`

[7] S. Brin. Extracting patterns and relations from the World Wide Web. In *WebDB*, 1998.

[8] M. J. Cafarella and O. Etzioni. A search engine for natural language applications. In *WWW*, 2005.

[9] J. P. Callan and M. Connell. Query-based sampling of text databases. *ACM TOIS*, 19(2):97–130, 2001.

[10] J. P. Callan, M. Connell, and A. Du. Automatic discovery of language models for text databases. In *SIGMOD*, 1999.

[11] J. P. Callan, Z. Lu, and W. B. Croft. Searching distributed collections with inference networks. In *SIGIR*, 1995.

[12] S. Chakrabarti. *Mining the Web*. Morgan Kaufmann, 2002.

[13] S. Chakrabarti, K. Punera, and M. Subramanyam. Accelerated focused crawling through online relevance feedback. In *WWW*, 2002.

[14] S. Chakrabarti, M. van den Berg, and B. Dom. Focused crawling: A new approach to topic-specific web resource discovery. *Computer Networks*, 31, 1999.

[15] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani. Robust and efficient fuzzy match for online data cleaning. In *SIGMOD*, 2003.

[16] S. Chaudhuri, R. Motwani, and V. R. Narasayya. Random sampling for histogram construction: How much is enough? In *SIGMOD*, 1998.

[17] S. Chaudhuri and K. Shim. Optimization of queries with user-defined predicates. *ACM TODS*, 24(2):177–228, 1999.

[18] F. Chung and L. Lu. Connected components in random graphs with given degree sequences. *Annals of Combinatorics*, 6:125–145, 2002.

[19] W. W. Cohen. Learning trees and rules with set-valued features. In *AAAI*, 1996.

[20] W. W. Cohen and Y. Singer. Learning to query the web. In *AAAI Workshop on Internet-Based Information Systems*, 1996.

[21] M. Diligenti, F. Coetzee, S. Lawrence, C. L. Giles, and M. Gori. Focused crawling using context graphs. In *VLDB*, 2000.

[22] R. B. Doorenbos, O. Etzioni, and D. S. Weld. A scalable comparison-shopping agent for the World-Wide Web. In *AGENTS'97*, 1997.

[23] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. Wiley, 1973.

[24] O. Etzioni, M. Cafarella, D. Downey, S. Kok, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates. Web-scale information extraction in KnowItAll. In *WWW*, 2004.

[25] L. Gravano, H. García-Molina, and A. Tomasic. *GlOSS*: Text-source discovery over the Internet. *ACM TODS*, 24(2):229–264, June 1999.

[26] L. Gravano, P. G. Ipeirotis, and M. Sahami. Query- vs. crawling-based classification of searchable web databases. *IEEE Data Eng. Bull.*, 25(1), 2002.

[27] L. Gravano, P. G. Ipeirotis, and M. Sahami. QProber: A system for automatic classification of hidden-web databases. *ACM TOIS*, 21(1):1–41, Jan. 2003.

[28] R. Grishman. Information extraction: Techniques and challenges. In *SCIE*, 1997.

[29] R. Grishman, S. Huttunen, and R. Yangarber. Information extraction for enhanced access to disease outbreak reports. *Journal of Biomedical Informatics*, 35(4), 2002.

[30] P. G. Ipeirotis. *Classifying and Searching Hidden-Web Text Databases*. Ph.D. thesis, Columbia University, 2004.

[31] P. G. Ipeirotis and L. Gravano. Distributed search over the hidden web: Hierarchical database sampling and selection. In *VLDB*, 2002.

[32] Y. Ling and W. Sun. An evaluation of sampling-based size estimation methods for selections in database systems. In *ICDE*, 1995.

[33] F. Menczer, G. Pant, and P. Srinivasan. Topical web crawlers: Evaluating adaptive algorithms. *ACM Transactions on Internet Technology*, 4(4):378–419, Nov. 2004.

[34] M. Mitzenmacher. Dynamic models for file sizes and double Pareto distributions. *Internet Mathematics*, 1(3):305–334, 2004.

[35] M. E. J. Newman, S. H. Strogatz, and D. J. Watts. Random graphs with arbitrary degree distributions and their applications. *Phys. Review E*, 64(2):1–17, 2001.

[36] A. Ntoulas, P. Zerfos, and J. Cho. Downloading textual hidden web content by keyword queries. In *JCDL*, 2005.

[37] D. W. Oard. The state of the art in text filtering. *UMUAI*, 7(3):141–178, 1997.

[38] S. M. Ross. *Introduction to Probability Models*. Academic Press, 8th ed., 2002.

[39] F. Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47, Mar. 2002.

[40] V. N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, Sept. 1998.

[41] H. S. Wilf. *Generatingfunctionology*. Academic Press Professional, Inc., 1990.

[42] R. Yangarber and R. Grishman. NYU: Description of the Proteus/PET system as used for MUC-7. In *MUC-7*, 1998.

[43] G. K. Zipf. *Human Behavior and the Principle of Least Effort*. 1949.