Towards a Query Optimizer for Text-Centric Tasks

PANAGIOTIS G. IPEIROTIS
New York University
EUGENE AGICHTEIN
Emory University
and
PRANAY JAIN and LUIS GRAVANO
Columbia University

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 article, 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

This material is based upon work supported by the National Science Foundation under Grants No. IIS-97-33880, IIS-98-17434, and IIS-0643846. The work of Panagiotis G. Ipeirotis is also supported by a Microsoft Live Labs Search Award and a Microsoft Virtual Earth Award. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or of the Microsoft Corporation.
Authors' addresses: P. G. Ipeirotis, Department of Information, Operations, and Management Sciences, New York University, 44 West Fourth Street, Suite 8-84, New York, NY 10012-1126; email: panos@stern.nyu.edu; E. Agichtein, Department of Mathematics and Computer Science, Emory University, Mathematics and Science Center, 400 Dowman Drive Suite W401, Atlanta, GA 30322; email: eugene@mathcs.emory.edu; P. Jain and L. Gravano, Computer Science Department, Columbia University, 1214 Amsterdam Avenue, New York, NY 10027-7003; email: {pranay.jain, gravano}@cs.columbia.edu.

efficient techniques for estimating the associated parameters of the cost model. We also present two optimization approaches for text-centric tasks that rely on the cost-model parameters and select efficient execution plans. Overall, our optimization approaches help build efficient 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.

Categories and Subject Descriptors: H.2.4 [**Database Management**]: Systems—*Textual databases, distributed databases*

General Terms: Algorithms, Experimentation, Measurement, Performance

Additional Key Words and Phrases: Metasearching, text database selection, distributed information retrieval, information extraction, focused crawling

ACM Reference Format:

Ipeirotis, P. G., Agichtein, E., Jain, P., and Gravano, L. 2007. Towards a query optimizer for text-centric tasks. ACM Trans. Datab. Syst. 32, 4, Article 21 (November 2007), 46 pages. DOI = 10.1145/1292609.1292611 http://doi.acm.org/10.1145/1292609.1292611

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 such an approach guarantees that we cover all documents that are potentially relevant for the task, this method 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 article, 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 from 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 article 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).

—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).

—Two optimization approaches that estimate "on-the-fly" the database properties that affect the execution time and recall of each plan. The first alternative follows a "global" optimization approach, to identify a single execution plan that is capable of reaching the target recall for the task. The second alternative partitions the optimization task into "local" chunks; this approach potentially switches between execution strategies by picking the best strategy for retrieving the "next-k" tokens at each execution stage (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 Sections 10 and 11 provide further discussion and conclude the article, respectively. This article expands on earlier work by the same authors [Ipeirotis et al. 2006; Agichtein et al. 2003], as discussed in Section 9.

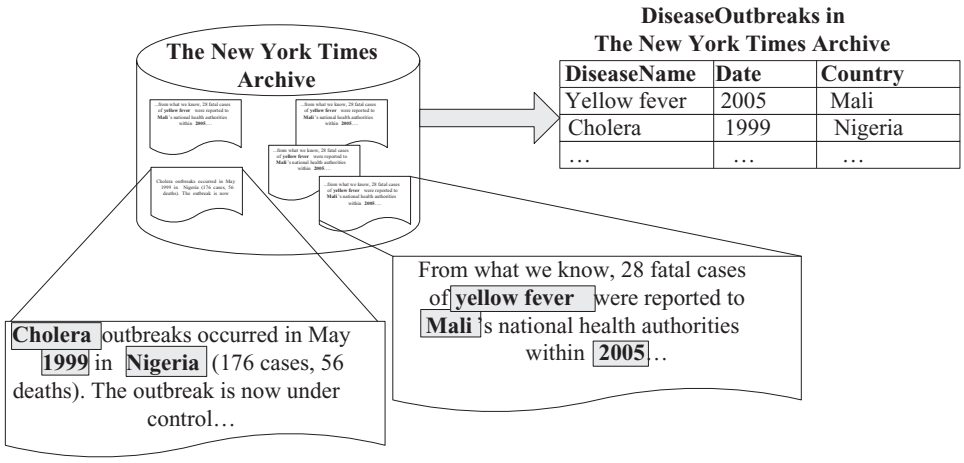

Fig. 1. Extracting *DiseaseOutbreaks* tuples.

2. EXAMPLES OF TEXT-CENTRIC TASKS

In this section, we briefly review three important text-centric tasks that we will use throughout the article 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 Grishman [1997], McCallum [2005], and Cunningham [2006] for introductory surveys 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 [Grishman et al. 2002] 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 [Agichtein and Gravano 2003] have been proposed to avoid retrieving all documents in a database;

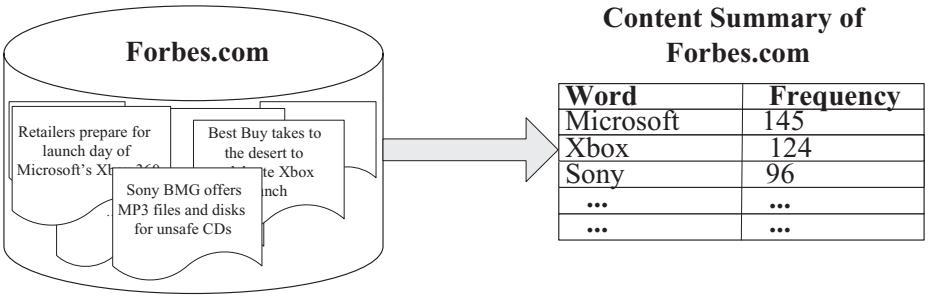

Fig. 2. Content summary of `Forbes.com`.

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. 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 [Callan et al. 1995; Gravano et al. 1999]. 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 [Bergman 2001]), then we need to resort to query-based approaches for content summary construction [Callan and Connell 2001; Ipeirotis and Gravano 2002].

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 [Sebastiani 2002] 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*

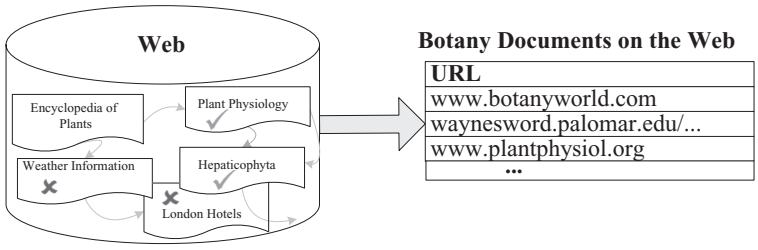

Fig. 3. Focused resource discovery for *Botany* pages.

(e.g., Chakrabarti et al. [1999, 2002], Menczer et al. [2004]) 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 [1996] 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 [Grishman et al. 2002], DIPRE [Brin 1998], Snowball [Agichtein and Gravano 2000], GATE/ANNIE,[1] MinorThird[2]): 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 [Duda et al. 2000] or Support Vector Machines [Vapnik 1998]) whether a given URL is a page about the topic of focus; if the classifier deems the document relevant, the URL is added as a token to the output and is discarded otherwise. Table I summarizes these abstractions.

[1]`http://gate.ac.uk/ie/annie.html`.
[2]`http://minorthird.sourceforge.net/`.

Table I. The Three Example Tasks Within Our Framework

Task	Document	Doc. Processor	Token
Information Extraction	News article	Information extraction system	Relation tuple
Content Summary Construction	Text document	Word tokenizer	Word
Focused Resource Discovery	Web page	Web page classifier	URL of page on topic of focus

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

$$Time(\mathcal{S}, D) = t_T(\mathcal{S}) + \sum_{q \in Q_{sent}} t_Q(q) + \sum_{d \in D_{retr}} (t_R(d) + t_F(d)) + \sum_{d \in D_{proc}} t_P(d), \quad (1)$$

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[3] (i.e., $t_Q = t_Q(q)$, for every $q \in Q_{sent}$) and that the time to retrieve, filter, or process

[3]This is a simplifying assumption, and does not hold for all queries. However, the tasks that we examine in this article typically involve keyword queries of only moderate length and result size, which, in turn, is reflected in little variance in the execution time of the queries.

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:

$$Time(\mathcal{S}, D) = t_T(\mathcal{S}) + t_Q \cdot |Q_{sent}| + \left(t_R + t_F\right) \cdot |D_{retr}| + t_P \cdot |D_{proc}| \qquad (2)$$

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

$$Recall(\mathcal{S}, D) = \frac{|Tokens(P, D_{proc})|}{|Tokens(P, D)|}, \qquad (3)$$

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

Based on the definitions of *Execution Time* and *Recall*, we now define our problem formally:

PROBLEM 3.1. *Consider a text-centric task, a database of text documents D, a document processor P for the task, and a set of alternative execution strategies $\mathcal{S}_1, \ldots, \mathcal{S}_n$ for the task. Given a target recall value τ, the goal is to identify an execution strategy $\mathcal{S}$ among $\mathcal{S}_1, \ldots, \mathcal{S}_n$ such that*

—*$Recall(\mathcal{S}, D) \geq \tau$ and*
—*$Time(\mathcal{S}, D) \leq Time(\mathcal{S}_j, D)$ if $Recall(\mathcal{S}_j, D) \geq \tau$.*

In other words, the goal is to identify an execution strategy $\mathcal{S}$ that is the fastest across the alternative strategies that reach the recall target τ for the task.

In a dual formulation of the problem, the goal is to identify the execution strategy $\mathcal{S}$ that can reach the maximum recall within a prespecified time threshold τ_{time}. In this article, we focus on the problem formulation stated in Problem 3.1. However, it is easy to adapt our techniques to work for the dual problem definition.

Our problem formulation implicitly assumes that we process documents sequentially. Our model could be easily expanded, however, to include parallel execution strategies as long as we consider only a relatively low degree of parallelism (i.e., a small number of parallel processors relative to the number of documents). In this case, the execution times are (roughly) divided by the degree of parallelism. If, in contrast, we assume a high degree of parallelism, then we could trivially process the complete database in a short time but at the expense of a significant waste of resources (many useless documents are processed).

Our problem 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

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

Fig. 4. The *Scan* strategy.

literature.[4] 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. Also, we define *Tokens* as the set of tokens that the document processor at hand can extract from all the database documents collectively.

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:

$$Time(\mathcal{SC}, D) = |D_{retr}| \cdot (t_R + t_P). \tag{4}$$

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

[4]While it is impossible to analyze all existing techniques within a single article, 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, estimate $|\widehat{Tokens}|$ of $|Tokens|$
Output: tokens $Tokens_{retr}$
$Tokens_{retr} = \emptyset$, $D_{retr} = \emptyset$, $recall = 0$
while $recall < \tau$ *and* $|D_{retr}| \leq |D|$ **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
  $recall = |Tokens_{retr}|/|\widehat{Tokens}|$
end
return $Tokens_{retr}$

Fig. 5. The *Filtered Scan* strategy.

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.

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*, that is, 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 2, the execution time of *Filtered Scan* is:

$$Time(\mathcal{FS}, D) = |D_{retr}| \cdot (t_R + t_F + C_\sigma \cdot t_P). \tag{5}$$

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 [Grishman et al. 2002], 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 [Chakrabarti et al. 1999] 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

Input: database D, recall threshold τ, tokens $Tokens_{seed}$, document processor P, estimate $|\widehat{Tokens}|$ of $|Tokens|$
Output: tokens $Tokens_{retr}$
$Tokens_{retr} = \emptyset$, $D_{retr} = \emptyset$, $recall = 0$
while $Tokens_{seed} \neq \emptyset$ **do**
  Remove a token t from $Tokens_{seed}$
  Transform t into a query q and issue q to D
  /* $maxD$ is the maximum number of results that D returns for a query */
  if q *matches fewer than* $maxD$ *documents* **then**
    Retrieve all documents matching q
  else
    Retrieve all $maxD$ documents matching q
  end
  foreach *newly retrieved document* d **do**
    Add d to D_{retr}
    Process d using P and add newly extracted tokens to $Tokens_{retr}$ and $Tokens_{seed}$
    $recall = |Tokens_{retr}|/|\widehat{Tokens}|$
    if $recall \geq \tau$ **then**
      return $Tokens_{retr}$
    end
  end
end
return $Tokens_{retr}$

Fig. 6. The *Iterative Set Expansion* strategy.

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 set 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:

$$Time(\mathcal{ISE}, D) = |Q_{sent}| \cdot t_Q + |D_{retr}| \cdot (t_R + t_P). \quad (6)$$

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 by Agichtein et al. [2003].

```
Input: database D, recall threshold τ, document processor P, queries Q, estimate |Tokens^| of |Tokens|
Output: tokens Tokens_retr
Tokens_retr = ∅, D_retr = ∅, recall = 0
foreach query q ∈ Q do
    /* maxD is the maximum number of results that D returns for a query */
    if q matches fewer than maxD documents then
        Retrieve all documents matching q
    else
        Retrieve all maxD documents matching q
    end
    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 ≥ τ then
            return Tokens_retr
        end
    end
end
return Tokens_retr
```

Fig. 7. The *Automatic Query Generation* strategy.

Iterative Set Expansion has been successfully applied in many tasks. For *Task 1*, *Iterative Set Expansion* corresponds to the *Tuples* algorithm for information extraction [Agichtein and Gravano 2003], 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. [1999], 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.

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[5] (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:

$$Time(\mathcal{AQG}, D) = |Q_{sent}| \cdot t_Q + |D_{retr}| \cdot (t_R + t_P). \quad (7)$$

The *Automatic Query Generation* strategy was proposed under the name *QXtract* for *Task 1* [Agichtein and Gravano 2003]; it was also used for *Task 2* by Ipeirotis and Gravano [2002] and for *Task 3* by Cohen and Singer [1996].

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

[5]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.

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. Using the document degree, we also separate the database documents into two sets: the set of *useful* documents D_{useful}, which contains documents with $g(d) \geq 1$, and the set of *useless* documents $D_{useless}$, which contains documents with $g(d) = 0$. Analogously to the document case, 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 in D from which processor P can extract t. 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.

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 [Agichtein et al. 2003], as we will see in Section 7. For *Task 2*, token degrees follow a power-law distribution [Zipf 1949; Baayen 2006] and document degrees follow roughly a lognormal distribution [Mitzenmacher 2004]; we provide further evidence in Section 7. For *Task 3*, the document and token distributions are, by definition, uniform over D_{useful} with $g(t) = g(d) = 1$, and we have $g(d) = 0$ for all documents in $D_{useless}$. In Section 6, we describe how to estimate the parameters of each distribution.

5.2 Cost of Scan

According to Equation (4), 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.[6] 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* [Ross 2002]. 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 [Ross 2002, 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:

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

Typically, we do not know the exact $g(t)$ for each token $t \in Tokens$. However, as discussed in Section 5.1, we have some knowledge about the form of the degree

[6]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.

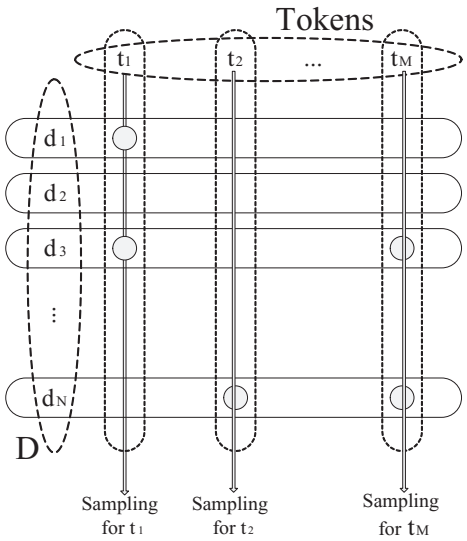

Fig. 8. Modeling *Scan* as multiple sampling processes, one per token, running in parallel over D.

distribution. Therefore, we can estimate $E[|Tokens_{retr}|]$ without knowing the $g(t)$ values but rather using estimates for the probabilities $Pr\{g(t) = i\}$, which are common across all tokens. In this case we have:

$$E[|Tokens_{retr}|] = |Tokens| \cdot \sum_{k=1}^{\infty} Pr\{g(t) = k\} \cdot \left(1 - \frac{(|D| - k)!\,(|D| - S)!}{(|D| - k - S)!|D|!}\right). \quad (9)$$

Hence, we estimate[7] the number of documents that *Scan* should retrieve to achieve a target recall τ as:

$$\widehat{|D_{retr}|} = \min\{S : E[|Tokens_{retr}|] \geq \tau |Tokens|\}. \quad (10)$$

The number of documents $|D_{retr}|$ retrieved by *Scan* depends on the token degree distribution. In Figure 9, we show the expected recall of *Scan* as a function of the number of retrieved documents, when $g(t)$ is uniform for all tokens. 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 extracted from only a few documents. For example, the *Task 1* tuple ⟨*SARS*, 2003, *China*⟩ can be extracted from hundreds of documents in *The New York Times* archive, while the tuple ⟨*Diphtheria*, 2003, *Afghanistan*⟩ appears only in a handful of documents. The recall of *Scan* for a given sample size S is lower over a database with a power-law token degree distribution compared to the recall over a database with uniform token degree distribution, when the token degree distributions have the same mean value (see Figure 10).

[7]To avoid numeric overflows during the computation of the factorials, we first take the logarithm of the ratio $\frac{(|D|-k)!(|D|-S)!}{(|D|-k-S)!|D|!}$ and then use the Stirling approximation $\ln x! \approx x \ln x - x + \frac{\ln x}{2} + \frac{1}{2} \ln 2\pi$ to efficiently compute the logarithm of each factorial. After computing the value of the logarithm of the ratio, we simply compute the exponential of the logarithm to estimate the original value of the ratio.

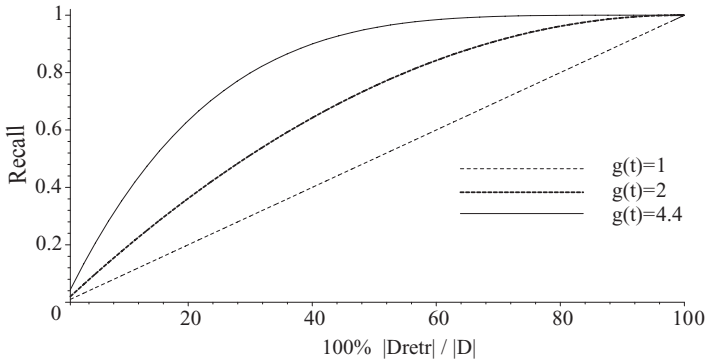

Fig. 9. Recall of the *Scan* strategy as a function of the fraction of retrieved documents, for $g(t) = 1$, $g(t) = 2$, and $g(t) = 4.4$.

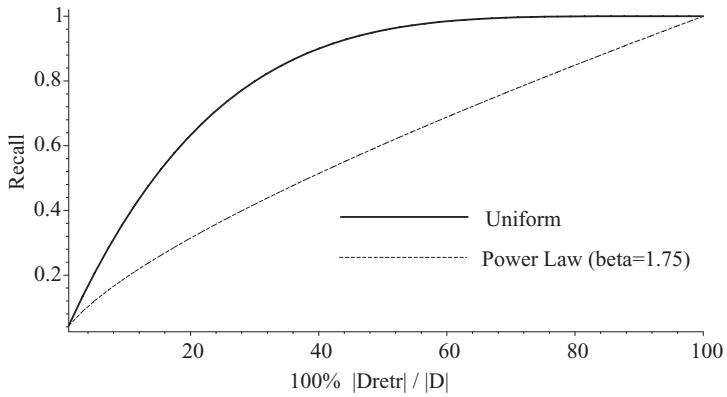

Fig. 10. Recall of the *Scan* strategy as a function of the fraction of retrieved documents, comparing the cases when $g(t)$ is constant for each token t and when $g(t)$ follows a power-law distribution (the mean value of $g(t)$ is the same in both cases, $E[g(t)] = 4.4$).

This is expected: while it is easy to discover the few very frequent tokens, it is hard to discover the majority of tokens, with low frequency. 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 (9) and (10) 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 token t: now each token appears, on average, $C_r \cdot g(t)$ times[8] 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:

$$E[|Tokens_{retr}|] = \sum_{t \in Tokens} 1 - \frac{(C_\sigma \cdot |D| - C_r \cdot g(t))!(C_\sigma \cdot |D| - S)!}{(C_\sigma \cdot |D| - C_r \cdot g(t) - S)!(C_\sigma \cdot |D|)!}. \quad (11)$$

As in the case of *Scan*, we use the probabilities $Pr\{g(t) = k\}$ instead of the individual $g(t)$ values:

$$E[|Tokens_{retr}|] = |Tokens| \cdot \sum_{k=1}^{\infty} Pr\{g(t) = k\} \cdot \left(1 - \frac{(C_\sigma \cdot |D| - C_r \cdot k)!\,(C_\sigma \cdot |D| - S)!}{(C_\sigma \cdot |D| - C_r \cdot k - S)!\,(C_\sigma \cdot |D|)!}\right). \quad (12)$$

Again, similar to *Scan*,

$$|\widehat{D_{retr}}| = \frac{|\widehat{D_{proc}}|}{C_\sigma} = \frac{\min\{S : E[|Tokens_{retr}|] \geq \tau |Tokens|\}}{C_\sigma}. \quad (13)$$

Equations (11) and (13) 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*

[8]We assume uniform recall across tokens, that is, 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.

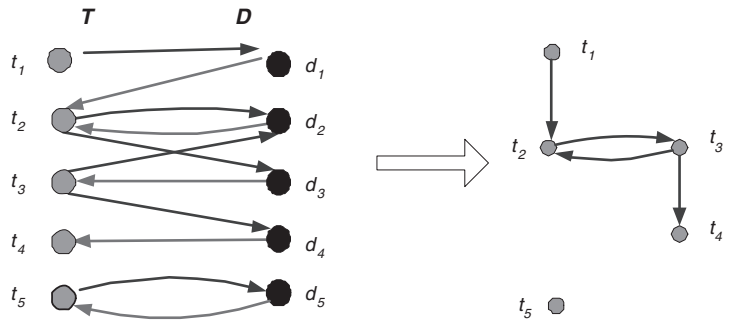

Fig. 11. Portion of the querying and reachability graphs of a database.

more formally as well as a graph-based representation of the querying process, originally introduced by Agichtein et al. [2003].

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*, as usual, 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 a document node d means that d is returned from D as a result to a query derived from the token t.

For example, in Figure 11, token t_1, after being suitably converted into a query, retrieves a document d_1 and, in turn, 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 at 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. [2001] 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[9] of a *randomly chosen* document and token, respectively:

$$Gd_0(x) = \sum_{k=0}^{\infty} pd_k \cdot x^k, \quad Gt_0(x) = \sum_{k=0}^{\infty} pt_k \cdot x^k, \tag{14}$$

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.

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. [2001], 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:

$$Gd_1(x) = x\frac{Gd_0'(x)}{Gd_0'(1)}, \quad Gt_1(x) = x\frac{Gt_0'(x)}{Gt_0'(1)}, \tag{15}$$

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 Newman et al. [2001] for the proof.)

For the rest of the analysis, we use the following useful properties of generating functions [Wilf 1990]:

—*Moments*: The ith factorial moment of the probability distribution generated by a function $G(x)$ is given by the ith 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 (14) and (15), 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

[9]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.

were discovered by following random edges on the graph; therefore, the degree distribution of these tokens is described by $Gt_1(x)$ (Equation (15)). 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[10]:

$$Gd_2(x) = [Gt_1(x)]^{|Q|}. \tag{16}$$

Now, we know that $|D_{retr}|$ in Equation (6) 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 (15)). Then, by the *Composition* property,[11] the distribution of the total number of *tokens* $|Tokens_{retr}|$ retrieved by the D_{retr} documents is given by the generating function[12]:

$$Gt_2(x) = Gd_2(Gd_1(x)) = [Gt_1(Gd_1(x))]^{|Q|}. \tag{17}$$

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

$$E[|D_{retr}|] = \left[\frac{d}{dx}[Gt_1(x)]^{|Q|}\right]_{x=1}, \tag{18}$$

$$E[|Tokens_{retr}|] = \left[\frac{d}{dx}[Gt_1(Gd_1(x))]^{|Q|}\right]_{x=1}. \tag{19}$$

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

$$|\widehat{Q_{sent}}| = \min\{Q : E[|Tokens_{retr}|] \geq \tau|Tokens|\}. \tag{20}$$

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 Agichtein et al. [2003] 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 11 shows the reachability graph derived from an underlying querying graph, illustrating how edges are added to the reachability graph. Since token t_2

[10]This is the number of *nondistinct* documents. To compute the number of distinct documents, we use the *sieve* method. For details, see Wilf [1990], page 110.
[11]We use the *Composition* property and not the *Power* property because $|D_{retr}|$ is a random variable.
[12]Again, this is the number of *nondistinct* tokens. To compute the number of distinct tokens, we use the *sieve* method. For details, see Wilf [1990], page 110.

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 11, 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 11.

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 [2002] 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. [2001] provide similar results for graphs with arbitrary degree distributions. Interestingly for our problem, the size distribution of the connected components can be estimated for many degree distributions using only a small number of parameters [Newman et al. 2001]. For example, for power-law graphs, which is the type of the reachability graphs for *Task 1* and *Task 2*, we only need an *estimate* of the average node out-degree [Chung and Lu 2002] to discover if there is a giant connected component and to compute the size distribution for the connected components. By estimating only a small number of parameters, we can thus characterize the performance limits of the *Iterative Set Expansion* strategy, which depends of the size distribution of the connected components of the reachability graph. In Section 6 we explain how we obtain such estimates.

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[13] 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:

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

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

$$E\left[\left|D_{retr}^{useful}\right|\right] = |D_{useful}| \cdot \left(1 - \prod_{i=1}^{|Q|}\left(1 - \frac{p(q_i)\cdot g(q_i)}{|D_{useful}|}\right)\right) \tag{21}$$

and the number of useless documents retrieved is:

$$E\left[\left|D_{retr}^{useless}\right|\right] = |D_{useless}| \cdot \left(1 - \prod_{i=1}^{|Q|}\left(1 - \frac{(1-p(q_i))\cdot g(q_i)}{|D_{useless}|}\right)\right). \tag{22}$$

Assuming that the "precision" of a query q is independent of the number of documents that q retrieves,[14] we get simpler expressions:

$$E\left[\left|D_{retr}^{useful}\right|\right] = |D_{useful}| \cdot \left(1 - \left(1 - \frac{E[p(q)]\cdot E[g(q)]}{|D_{useful}|}\right)^{|Q|}\right), \tag{23}$$

$$E\left[\left|D_{retr}^{useless}\right|\right] = |D_{useless}| \cdot \left(1 - \left(1 - \frac{(1-E[p(q)])\cdot E[g(q)]}{|D_{useless}|}\right)^{|Q|}\right), \tag{24}$$

where $E[p(q)]$ is the average precision of the queries and $E[g(q)]$ is the average number of retrieved documents per query. The expected number of retrieved documents is then:

$$E[|D_{retr}|] = E\left[\left|D_{retr}^{useful}\right|\right] + E\left[\left|D_{retr}^{useless}\right|\right]. \tag{25}$$

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 (23) 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

[13]The conditional independence assumption implies that the queries are only biased towards retrieving useful documents, and not towards any subset of useful documents.
[14]We observed this assumption to be true in practice.

the sampling is over the D_{useful} set. Therefore, we have an effective database size $|D_{useful}|$ and a sample size equal to $|D_{retr}^{useful}|$.[15] By modifying Equation (8) appropriately, we have:

$$E[|Tokens_{retr}|] = \sum_{t \in Tokens} 1 - \frac{(|D_{useful}| - g(t))!\left(|D_{useful}| - \left|D_{retr}^{useful}\right|\right)!}{\left(|D_{useful}| - g(t) - \left|D_{retr}^{useful}\right|\right)!|D_{useful}|!}. \quad (26)$$

Again, similarly to *Scan* and *Filtered Scan*, we use the probabilities $Pr\{g(t) = k\}$ instead of the individual $g(t)$ values:

$$E\left[|Tokens_{retr}|\right] = |Tokens| \cdot \sum_{k=1}^{\infty} Pr\{g(t) = k\} \cdot \left(1 - \frac{(|D_{useful}| - k)!\left(|D_{useful}| - \left|D_{retr}^{useful}\right|\right)!}{\left(|D_{useful}| - k - \left|D_{retr}^{useful}\right|\right)!|D_{useful}|!}\right). \quad (27)$$

A good approximation of the average value of $|Tokens_{retr}|$ can be derived by setting $|D_{retr}^{useful}|$ to be the mean value $E[|D_{retr}^{useful}|]$ (Equation (23)). Similarly to the analysis for *Iterative Set Expansion*, we have:

$$|\widehat{Q_{sent}}| = \min\{Q : E[|Tokens_{retr}|] \geq \tau |Tokens|\}. \quad (28)$$

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.

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 present two different optimization schemes. In Section 6.1, we present a "global" optimizer, which tries to pick the best execution strategy for reaching the target recall. Then, in Section 6.2 we present a "local" optimizer, which partitions the execution in multiple stages, and selects the best execution strategy for each stage. As we will show in our experimental evaluation in Section 8, our optimization approaches lead to efficient executions of the text-centric tasks.

6.1 Global Optimization Approach

The goal of our global optimizer is to select an execution plan that will reach the target recall in minimum amount of time. The optimizer starts by choosing one of the execution plans described in Section 4, using the cost model that we presented in Section 5.

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

Our cost model relies on a number of parameters, which are generally unknown before executing a task. Some of these parameters, such as classifier selectivity and recall (Section 5.3), can be estimated efficiently before the execution of the task. For example, 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 [Chaudhuri et al. 1998].

Other parameters of our cost model, namely the token and document distributions, are challenging to estimate. Rather than attempting to estimate these distributions without prior information, we rely on the fact that for many text-centric 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,[16] 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—once again—Chaudhuri et al. [1998], 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 steps 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. The basic challenge in this scenario is to guarantee that the parameter estimates that we obtain during execution are accurate. Below, we discuss how to perform the parameter estimation for each of the execution strategies of Section 4.

6.1.1 *Scan.* Our analysis in Section 5.2 relies on the characteristics of the token and document degree distributions. After retrieving and processing a few documents, we can estimate the distribution parameters based on the frequency of the initially extracted tokens and documents. Specifically, we can use a maximum likelihood fit to estimate the parameters of the document degree distribution. For example, the document degrees for *Task 1* tend to follow a power-law distribution, with a probability mass function $Pr\{g(d) = x\} = x^{-\beta}/\zeta(\beta)$, where $\zeta(\beta)$ is the Riemann zeta function $\zeta(\beta) = \sum_{n=1}^{+\infty} n^{-\beta}$ that serves as a normalizing factor. Our goal is to estimate the most likely value of β, for a given sample of document degrees $g(d_1), \ldots, g(d_s)$. Using a maximum likelihood estimation (MLE) approach, we identify the value of β that maximizes the likelihood function:

$$l(\beta | g(d_1), \ldots, g(d_s)) = \prod_{i=1}^{s} \frac{g(d_i)^{-\beta}}{\zeta(\beta)}.$$

[16]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 nonparametric 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 nonparametric framework is a topic for interesting future research. (See also Section 10.)

Taking the logarithm, we have the log-likelihood function:

$$\begin{aligned} L(\beta|g(d_1),\ldots,g(d_s)) &= \log l(\beta|g(d_1),\ldots,g(d_s)) \\ &= \sum_{i=1}^{s}(-\beta\log g(d_i)-\log\zeta(\beta)) \\ &= -s\cdot\log\zeta(\beta)-\beta\sum_{i=1}^{s}\log g(d_i). \end{aligned} \tag{29}$$

To find the maximum of the log-likelihood function, we identify the value of β that makes the first derivative of L be equal to zero:

$$\begin{aligned} \frac{d}{d\beta}L(\beta|g(d_1),\ldots,g(d_s)) &= 0, \\ -s\cdot\frac{\zeta'(\beta)}{\zeta(\beta)}-\sum_{i=1}^{s}\log g(d_i) &= 0, \\ \frac{\zeta'(\beta)}{\zeta(\beta)} &= -\frac{1}{s}\sum_{i=1}^{s}\log g(d_i), \end{aligned} \tag{30}$$

where $\zeta'(\beta)$ is the first derivative of the Riemann zeta function. Then, we can estimate the value of β using numeric approximation. Similar approaches can be used for other distribution families.

The estimation of the token degree distribution is typically more challenging than the estimation of the document degree distribution. While we can observe the degree $g(d)$ of each document d retrieved in a document sample, we cannot directly determine the actual degree $g(t)$ of each token t extracted from sample documents. In general, the degree $g(t)$ of a token t in a database is larger than the degree of t in a document sample extracted from the database. Hence, before using the maximum likelihood approach described above, we should estimate, for each extracted token t, the token degree $g(t)$ *in the database*.

We denote the sample degree of a token t as $s(t)$, defined over a given document sample. Using, again, a maximum likelihood approach, we find the most likely token frequency $g(t)$ that maximizes the probability of observing the token frequency $s(t)$ in the sample:

$$Pr\{g(t)|s(t)\} = \frac{Pr\{s(t)|g(t)\}\cdot Pr\{g(t)\}}{Pr\{s(t)\}}. \tag{31}$$

Since $Pr\{s(t)\}$ is constant across all possible values of $g(t)$, we can ignore this factor for this maximization problem. From Section 5.2, we know that the probability of retrieving $s(t)$ times a token t when it appears $g(t)$ times in the database follows a hypergeometric distribution, and then:

$$Pr\{s(t)|g(t)\} = \binom{g(t)}{s(t)}\cdot\binom{|D|-g(t)}{S-s(t)}\Big/\binom{|D|}{S}.$$

To estimate $Pr\{g(t)\}$, we rely on our knowledge of the distribution family of the token degrees. For example, the token degrees for *Task 1* follow a power-law distribution, with $Pr\{g(t)\} = g(t)^{-\beta}/\zeta(\beta)$. Then, for *Task 1*, we find the value

of $g(t)$ that maximizes the following:

$$Pr\{s(t)|g(t)\} \cdot Pr\{g(t)\} = \left(\binom{g(t)}{s(t)} \cdot \binom{|D| - g(t)}{S - s(t)} \Big/ \binom{|D|}{S}\right) \cdot \frac{g(t)^{-\beta}}{\zeta(\beta)}. \quad (32)$$

For this, we take the logarithm of the expression above and use the Stirling approximation to eliminate the factorials. We then find the value of $g(t)$ for which the derivative of the logarithm of the expression above with respect to $g(t)$ is equal to zero. Given the database size $|D|$, the sample size S, and the sample degree $s(t)$ of the token, we can estimate efficiently the maximum likelihood estimate of $g(t)$, for different values of the parameter(s) of the token degree distribution. Then, using these estimates of the database token degrees, we can proceed as in the document distribution case and estimate the token distribution parameters.

The final step in the token distribution estimation is the estimation of the value of $|Tokens|$, which we need to evaluate Equation (9). Unfortunately, the *Tokens* set is, of course, unknown. But during execution, we know the number of tokens that we extract from the documents that we retrieve, and this actual number of extracted tokens should match the $E[|Tokens_{retr}|]$ prediction of Equation (9) for the corresponding values of the sample size S. Furthermore, we know the values of $|D|$, S, and the probabilities $Pr\{g(t) = k\}$. Therefore, the only unknown value in Equation (9) is the value of $|Tokens|$. We can then estimate $|Tokens|$ as the value of $|Tokens|$ that solves Equation (9) for the given sample size S. Since the value of $|Tokens|$ also determines whether the execution strategy reached the target recall τ, we also compute the confidence intervals for the estimate of $|Tokens|$, using the variance of $E[|Tokens_{retr}|]$; to avoid false early terminations, we terminate the execution only when we have 95% confidence that $|Tokens_{retr}|/|Tokens| \geq \tau$.

6.1.2 *Filtered Scan.* The analysis for *Filtered Scan* is analogous to the analysis of *Scan*. Assuming that the only classifier bias is towards *useful* documents (see Section 5.3), we use the document degree distribution in the retrieved sample to estimate the database degree distribution. To estimate the token distribution, the only difference with the analysis for *Scan* is that the probability of retrieving a token $s(t)$ times when it appears $g(t)$ times in the database is now:

$$Pr\{s(t)|g(t)\} = \binom{C_r \cdot g(t)}{s(t)} \cdot \binom{C_\sigma \cdot |D| - C_r \cdot g(t)}{S - s(t)} \Big/ \binom{C_\sigma \cdot |D|}{S}, \quad (33)$$

where C_r is the classifier's recall and C_σ is the classifier's selectivity (see Section 5.3).

6.1.3 *Iterative Set Expansion.* The crucial observation in this case is that, during querying, we actually sample from the distributions generated by the $Gt_1(x)$ and $Gd_1(x)$ functions, rather than from the distributions generated by $Gt_0(x)$ and $Gd_0(x)$ (see Section 5.4). Still, we can use our estimation procedure that we applied for *Scan* to return the parameters for the distributions generated by $Gt_1(x)$ and $Gd_1(x)$, based on the sample document and token degrees observed during querying. However, these estimates are not the actual

parameters of the token and document degree distributions, which are generated by the $Gt_0(x)$ and $Gd_0(x)$ functions, respectively, not by $Gt_1(x)$ and $Gd_1(x)$. Hence, our goal is to estimate the parameters for the distributions generated by the $Gt_0(x)$ and $Gd_0(x)$ functions, given the parameter estimates for the distributions generated by the $Gt_1(x)$ and $Gd_1(x)$ functions.

For this, we can use Equations (14) and (15), together with the distributions generated by $Gt_1(x)$ and $Gd_1(x)$, to estimate the $Gt_0(x)$ and $Gd_0(x)$ distributions. Intuitively, $Gt_1(x)$ and $Gd_1(x)$ overestimate $Pr\{g(t) = k\}$ and $Pr\{g(d) = k\}$ by a factor of k, since tokens and documents with degree k are k times more likely to be discovered during querying than during random sampling. Therefore,

$$Pr\{g(t) = k\} = K_t \cdot \frac{\widehat{Pr_{\mathcal{ISE}}}\{g(t) = k\}}{k},$$
$$Pr\{g(d) = k\} = K_d \cdot \frac{\widehat{Pr_{\mathcal{ISE}}}\{g(d) = k\}}{k},$$

where $\widehat{Pr_{\mathcal{ISE}}}\{g(t) = k\}$ and $\widehat{Pr_{\mathcal{ISE}}}\{g(d) = k\}$ are the probability estimates that we get for the distributions generated by $Gt_1(x)$ and $Gd_1(x)$, and K_t and K_d are normalizing constants that ensure that the sum across all probabilities is 1.

6.1.4 *Automatic Query Generation.* For the document degree distribution, we can proceed analogously as for *Scan*. The crucial difference is that *Automatic Query Generation* underestimates $Pr\{g(d) = 0\}$, the probability that a document d is useless (the document retrieval process is biased towards retrieving useful documents), while it overestimates $Pr\{g(d) = k\}$, for $k \geq 1$. The *correct* estimate for $Pr\{g(d) = 0\}$ is:

$$Pr\{g(d) = 0\} = \frac{|D_{useless}|}{|D|} = \frac{|D_{useless}|}{|D_{useful}| + |D_{useless}|}. \quad (34)$$

To estimate the correct values of $|D_{useful}|$ and $|D_{useless}|$, we use Equations (21) and (22). For each submitted query q_i, we know its precision $p(q_i)$ and its degree $g(q_i)$. We also know the number of useful documents retrieved $|D_{retr}^{useful}|$ and the number of useless documents retrieved $|D_{retr}^{useless}|$. Hence, the only unknown variable in Equation (21) is $|D_{useful}|$, while the only unknown variable in Equation (22) is $|D_{useless}|$. It is difficult to solve these equations analytically for $|D_{useful}|$ and $|D_{useless}|$. However, Equations (21) and (22) are monotonic with respect to $|D_{useful}|$ and $|D_{useless}|$, respectively, so it is easy to estimate numerically the values of $|D_{useful}|$ and $|D_{useless}|$ that solve the equations. Then, we can estimate $Pr\{g(d) = 0\}$ using Equation (34). After correcting the estimate for $Pr\{g(d) = 0\}$, we proportionally adjust the estimates for the remaining values $Pr\{g(d) = k\}$, for $k \geq 1$, to ensure that $\sum_{i=0}^{+\infty} Pr\{g(d) = i\} = 1$.

To estimate the parameters of the token distribution, we assume that, given sufficiently many queries, *Automatic Query Generation* will have perfect recall. In this case, we assume that *Automatic Query Generation* performs random sampling over the D_{useful} documents, rather than over the complete database.

We then set:

$$Pr\{s(t)|g(t)\} = \binom{g(t)}{s(t)}\binom{|D_{useful}| - g(t)}{S - s(t)} \Big/ \binom{|D_{useful}|}{S}, \quad (35)$$

where $S = |D_{retr}^{useful}|$. Then, we proceed with the estimation analogously to *Scan*.

6.1.5 *Choosing an Execution Strategy.* Using the estimation techniques from Sections 6.1.1 through 6.1.4, we can now describe our overall global optimization approach. Initially, our optimizer is informed of the general token and document degree distribution (e.g., the optimizer knows that the token and document degrees follow a power-law distribution for *Task 1*). As discussed, the actual parameters of these distributions are unknown, so the optimizer assumes some rough constant values for these parameters (e.g., $\beta = 2$ for power-law distributions)—which will be later refined—to decide which of the execution strategies from Section 4 is most promising.

Our optimizer's initial choice of execution strategy for a task may of course be far from optimal, since this choice is made without accurate parameter estimates for the token and document degree distributions. Therefore, as documents are retrieved and tokens are extracted using this initial execution strategy, the optimizer updates the distribution parameters using the techniques of Sections 6.1.1 through 6.1.4, checking the robustness of the new estimates using cross-validation. We consider an estimate robust if the standard deviation of the estimated values is less than 10% of the corresponding mean.

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. In practice, we refine the statistics and reoptimize only after the chosen strategy has processed N documents.[17] (In our experiments, we set $N = 100$.) Figure 12 summarizes this algorithm.

6.2 Local Optimization Approach

The global optimization approach (Section 6.1) attempts to pick an execution plan to reach a target recall τ for a given task. The optimizer only revisits its decisions as a result of changes in the token and document statistics on which it relies, as we discussed. In fact, if the optimizer were provided with perfect statistics, it would pick a single plan (out of *Scan*, *Filtered Scan*, *Iterative Set Expansion*, and *Automatic Query Generation*) from the very beginning and continue with this plan until reaching the target recall.

Interestingly, often the best execution plans for a text-centric task might use different execution strategies at different stages of the token extraction process. For example, consider *Task 1* with a target recall $\tau = 0.6$. For a given

[17] An interesting direction for future research is to use confidence bounds for the statistics estimates, which dictate how often to reoptimize. Intuitively, the estimates become more accurate as we process more documents. Hence, the need to reconsider the optimization choice decreases as the execution progresses.

Input: database D, recall threshold τ, alternate strategies $\mathcal{S}_1, \ldots, \mathcal{S}_n$
Output: tokens $Tokens_{retr}$, documents D_{retr}
$statistics = \emptyset$, $D_{retr} = \emptyset$, $Tokens_{retr} = \emptyset$, $recall = 0$
while $recall < \tau$ *and* $|D_{retr}| < |D|$ **do**
  /* Locate best possible strategy */
  foreach $\mathcal{S}_i \in \{\mathcal{S}_1, \ldots, \mathcal{S}_n\}$ **do**
    Use available *statistics* to estimate $Time(\mathcal{S}_i, D)$, the time for $\mathcal{S}_i$ to reach target recall τ
  end
  $strategy = \arg\min_{\mathcal{S}_i}\{Time(\mathcal{S}_i, D)\}$, where $\mathcal{S}_i \in \{\mathcal{S}_1, \ldots, \mathcal{S}_n\}$
  /* Execute strategy */
  Execute *strategy* over N unprocessed documents and update D_{retr}, $Tokens_{retr}$, and *recall* accordingly
  Refine *statistics* using D_{retr} and $Tokens_{retr}$
end
return $Tokens_{retr}$, D_{retr}

Fig. 12. The "global" optimization approach, which chooses an execution strategy that is able to reach a target recall τ.

text database, the *Iterative Set Expansion* strategy (Section 4.3) might stall and not reach the target recall $\tau = 0.6$, as discussed in Section 5.4. So our global optimizer might not pick this strategy when following the algorithm in Figure 12. However, *Iterative Set Expansion* might be the most efficient strategy for retrieving, say, 50% of the tokens in the database. So a good execution plan in this case might then start by running *Iterative Set Expansion* to reach a recall value of 0.5, and then switch to another strategy, say *Filtered Scan*, to finally achieve the target recall, namely, $\tau = 0.6$. We now introduce a *local* optimization approach that explicitly considers such combination executions that might include a variety of execution strategies.

Rather than choosing the best strategy—according to the available statistics—for reaching a target recall τ, our local optimization approach partitions the execution into "recall stages" and successively identifies the best strategy for each stage. So initially, the local optimization approach chooses the best execution strategy for extracting the first k tokens, for some predefined value of k, then identifies the best execution strategy for extracting the next k tokens, and so on, until the target recall τ is reached. Hence, the local optimization approach can be regarded as invoking the global optimization approach repeatedly, each time to find the best strategy for extracting the next k tokens (see Figure 13). As a result, the local optimization approach can generate flexible combination executions, with different execution choices for different recall stages.

At each optimization point for a task over a database, the local optimization approach chooses the execution strategy for extracting the next batch of k tokens. The new tokens will be extracted from the unseen documents in the database, so the optimizer adjusts the statistics on which it relies accordingly, to ignore the documents that have already been processed in the task execution. Typically, the most frequent tokens are extracted early in the execution; the document and token degree distributions in the unseen portion of the database are thus generally different from the corresponding distributions in the

Input: database D, recall threshold τ, alternate strategies $\mathcal{S}_1, \ldots, \mathcal{S}_n$, optimization interval k
Output: tokens $Tokens_{retr}$
$statistics = \emptyset$, $D_{retr} = \emptyset$, $Tokens_{retr} = \emptyset$, $recall = 0$
while $recall < \tau$ *and* $|D_{retr}| < |D|$ **do**
  /* Optimize for the next-k tokens */
  $\{Tokens'_{retr}, D'_{retr}\} = \text{GlobalOptimizer}(D - D_{retr}, \frac{k}{|\widehat{Tokens}| - |Tokens_{retr}|}, \mathcal{S}_1, \ldots, \mathcal{S}_n)$
  $Tokens_{retr} = Tokens_{retr} \cup Tokens'_{retr}$
  $D_{retr} = D_{retr} \cup D'_{retr}$
  Update *recall*
  Refine *statistics* for $D - D_{retr}$, using D_{retr} and $Tokens_{retr}$
end
return $Tokens_{retr}$

Fig. 13. The "local" optimization approach, which chooses a potentially different execution strategy for each batch of k tokens.

complete database. To account for these differences, at each optimization point the local optimization approach follows the estimation procedures of Sections 6.1.1 through 6.1.4 to characterize the distributions over the *complete* database; then, the optimizer uses the distribution statistics for the complete database—as well as the statistics for the retrieved documents and tokens—to estimate the distribution statistics over the unseen portion of the database: we can easily compute the degree distribution for the unseen tokens and documents by subtracting the distribution for the retrieved documents from the distribution for the complete database.

Next, we report the results of our experimental evaluation of our optimization approaches, to highlight their strengths and weaknesses for choosing execution strategies that reach the target recall τ efficiently.

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

7.1.1 *Document Processor.* For this task, we use the Snowball information extraction system [Agichtein and Gravano 2000] 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 article. 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.

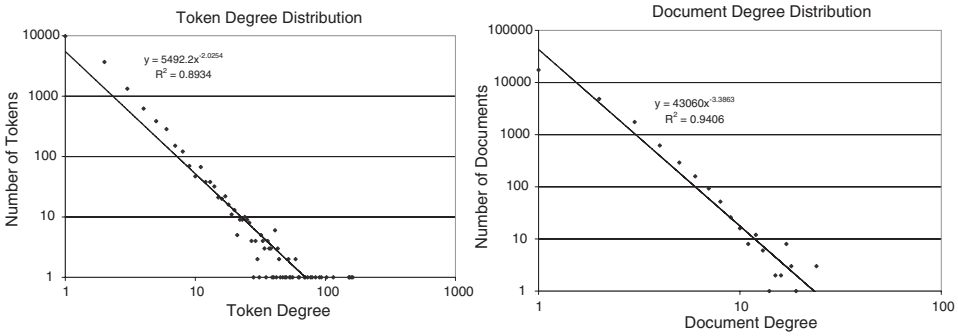

Fig. 14. Token and document degree distribution for *Task 1*'s *DiseaseOutbreaks*.

7.1.2 *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*. Figure 14 shows the token and document 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*.

7.1.3 *Execution Plan Instantiation.* For *Filtered Scan* we use a rule-based classifier, created using RIPPER [Cohen 1996]. We train RIPPER using a set of 500 useful documents and 1500 not useful documents from the NYT95 data set. We also use 2000 documents from the NYT95 data set as a training set to create the queries required by *Automatic Query Generation*. 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*]). Finally, for the query-based strategies, we use $maxD = 100$ as the upper limit for the maximum number of returned documents for a query.

7.2 Content Summary Construction

7.2.1 *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.

7.2.2 *Data Set.* We use the *20 Newsgroups* data set from the UCI KDD Archive [Blake and Merz 1998]. 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*. Figure 15 shows the token and document degree distribution (Section 5) for this task. The document degree follows a lognormal distribution [Mitzenmacher 2004] and the token degree follows, as expected [Zipf 1949], a power-law distribution.

7.2.3 *Execution Plan Instantiation.* For this task, *Filtered Scan* is not directly applicable, since all documents are "useful." For *Iterative Set Expansion*,

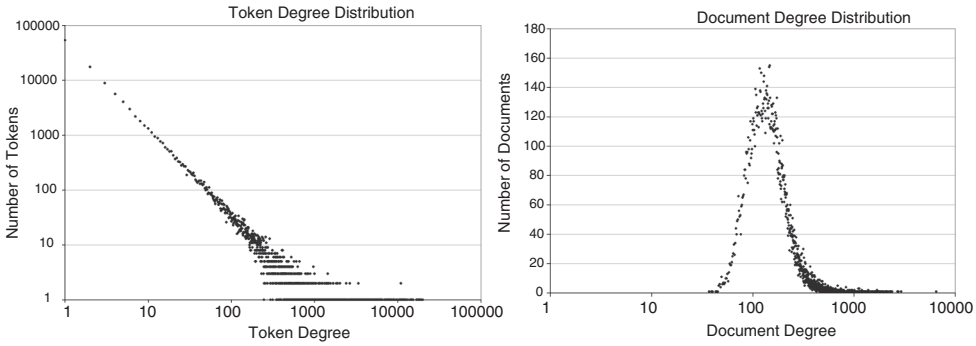

Fig. 15. Token and document degree distribution for *Task 2*.

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. [1999]. For *Automatic Query Generation*, 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 [Ipeirotis and Gravano 2002]. Finally, for the query-based strategies, we use $maxD = 100$ as the upper limit for the maximum number of returned documents for a query.

7.3 Focused Resource Discovery

7.3.1 *Document Processor.* For this task, the document processor is a multinomial Naive Bayes classifier, which detects the topic of a given Web page [Chakrabarti et al. 1999]. The topic of choice for our experiments is "Botany."

7.3.2 *Data Set.* We retrieved 8000 Web pages listed in Open Directory[18] under the category "*Top* $\rightarrow$ *Science* $\rightarrow$ *Biology* $\rightarrow$ *Botany.*" We selected 1000 out of the 8000 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

[18]`http://www.dmoz.org`.

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.

7.3.3 *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 used the "hard" version of the focused crawler described by Chakrabarti et al. [1999]. 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, and we use $maxD = 100$ as the upper limit for the maximum number of returned documents for a query.

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 strategies, which rely on approximate estimates of these parameters, as described in Section 6.

8.1 Accuracy of Cost Model with Correct Information

The goal of the first set of experiments was 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 degrees, 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 16, 17, 18, and 19 show the actual execution time of the respective strategies for the tasks described in Section 7. Then, to predict the execution time of each strategy, we used our equations from Section 5. The lines *SC_pred*, *FS_pred*, *ISE_pred*, *AQG_pred* in Figures 16, 17, 18, and 19 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 the characteristics and the limitations of each execution plan. For example, *Automatic Query Generation* is the fastest execution plan for *Task 1a* (Figure 16) when the target recall τ is under 0.15. However, due to the limited number of queries generated during the training phase, *Automatic Query*

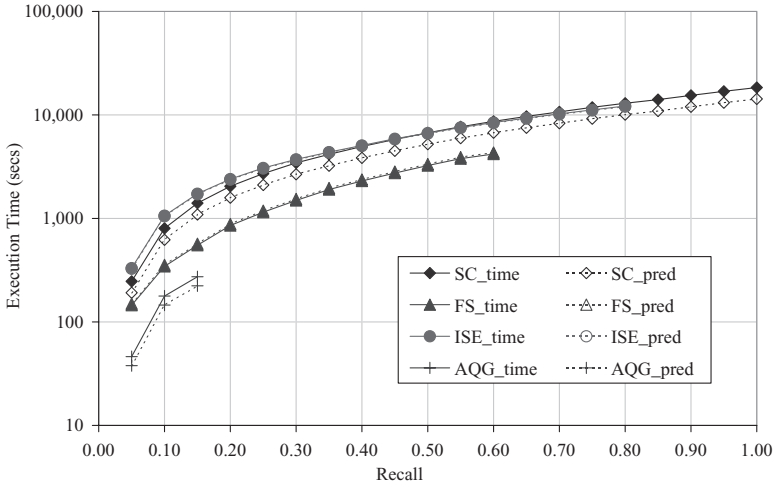

Fig. 16. Actual versus estimated execution times for *Task 1a*, as a function of the target recall τ.

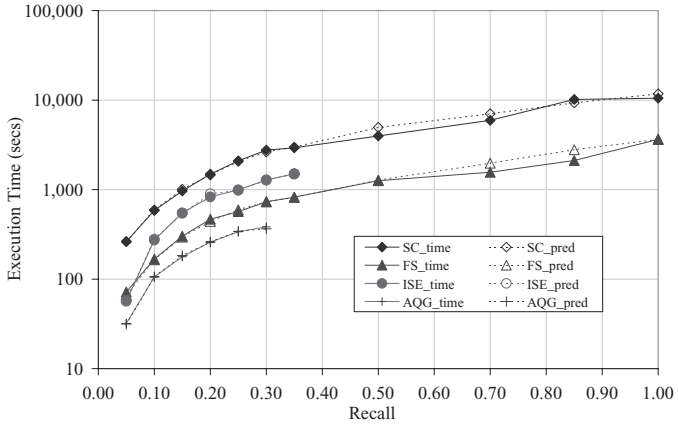

Fig. 17. Actual versus estimated execution times for *Task 1b*, as a function of the target recall τ.

Generation does not reach higher recall values in our scenario and implementation. (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 [Agichtein et al. 2003]. The results are similar for *Task 2* and *Task 3*: our analysis correctly predicts the execution time and the recall limitations of each strategy.

8.2 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 our optimization strategies lead to the choice of the fastest plan for each value of target recall τ.

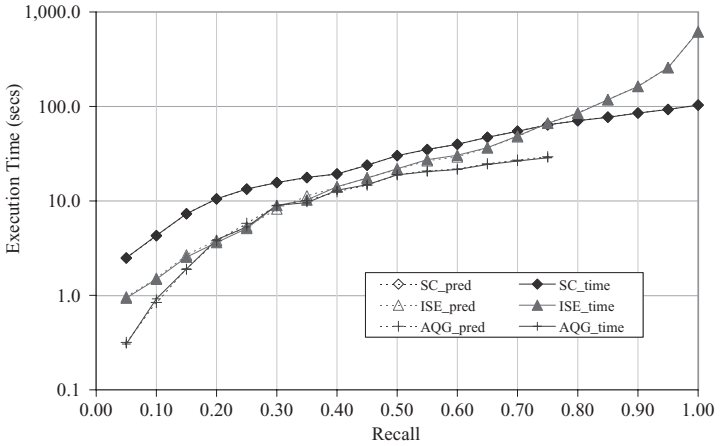

Fig. 18. Actual versus estimated execution times for *Task 2*, as a function of the target recall τ.

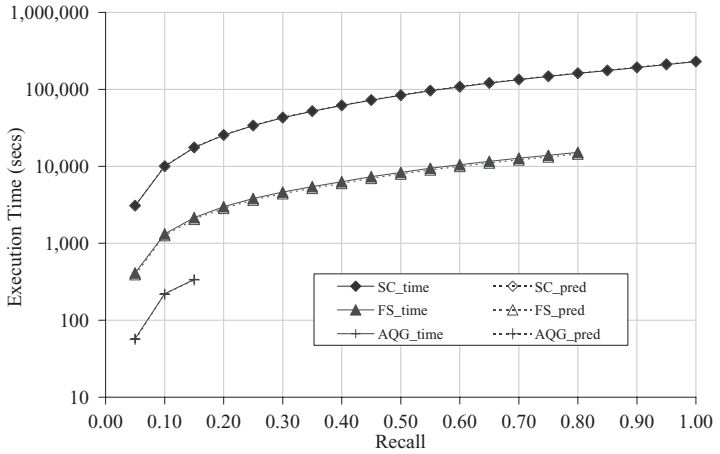

Fig. 19. Actual versus estimated execution times for *Task 3*, as a function of the target recall τ.

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). As we retrieve documents and extract tokens during the actual execution, the available statistics are refined and progressively lead to better estimates of the document and token degree distributions for the complete database. The "global" optimization approach reconsiders its choice of execution plan every N documents (see Figure 12). For our experiments, we use $N = 100$, which allows the statistics to change sufficiently between reoptimizations, but—at the same time—without allowing a suboptimal algorithm to run for too long. The "local" optimization approach defines "combination" executions by picking the best

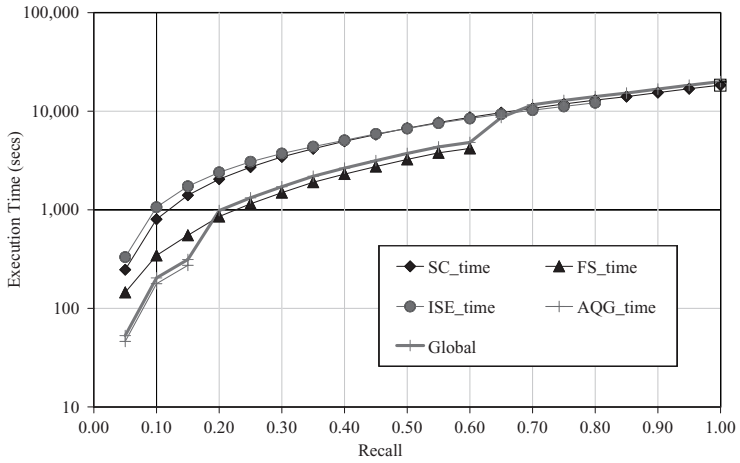

Fig. 20. Actual execution times for the four basic execution strategies, as well as for the "global" optimization approach, for *Task 1a* and as a function of the target recall τ.

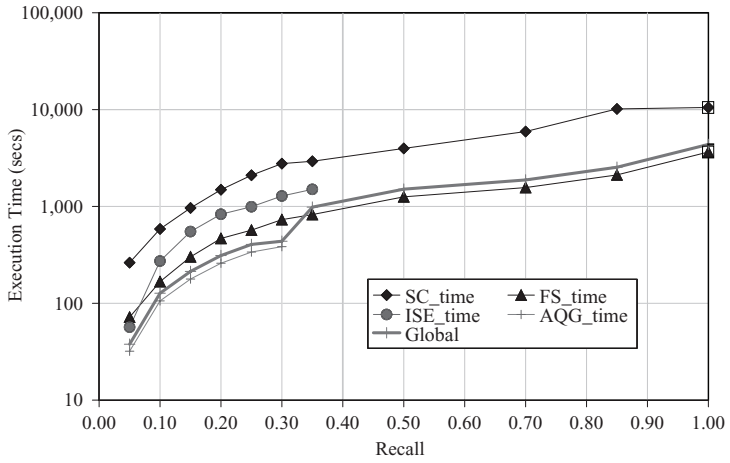

Fig. 21. Actual execution times for the four basic execution strategies, as well as for the "global" optimization approach, for *Task 1b* and as a function of the target recall τ.

strategy for selecting k tokens at a time (see Figure 13). For our experiments, we set $k = 0.05 \cdot |Tokens|$.

The "global" line in Figures 20, 21, 22, and 23 shows the actual execution time, for different recall thresholds, using our global optimization approach. Typically, our global 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 21): 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

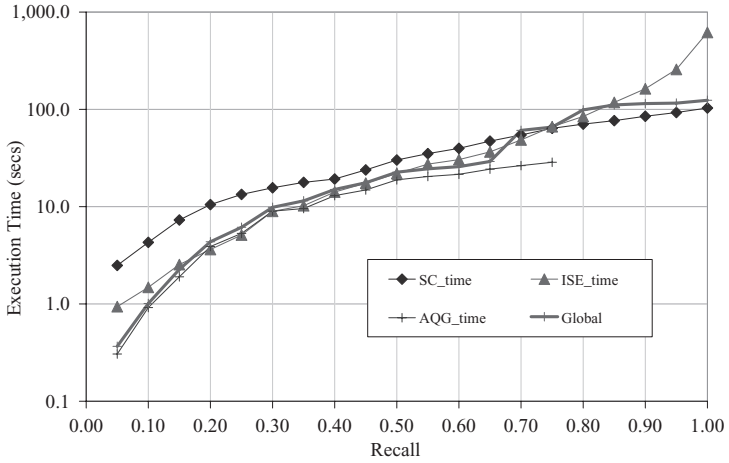

Fig. 22. Actual execution times for the three basic execution strategies, as well as for the "global" optimization approach, for *Task 2* and as a function of the target recall τ.

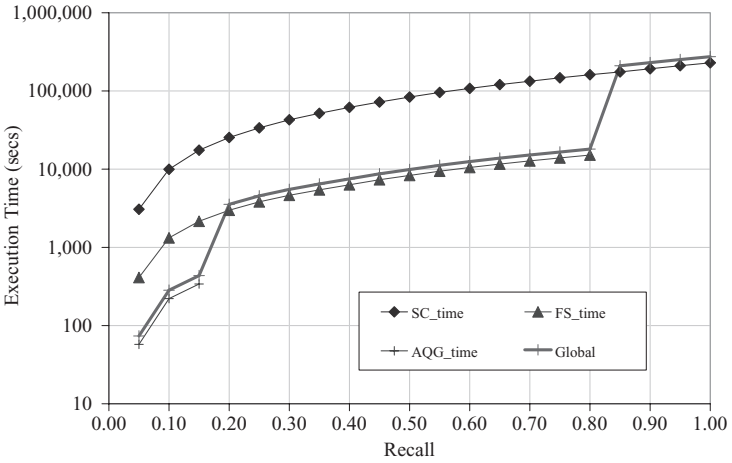

Fig. 23. Actual execution times for the three basic execution strategies, as well as for the "global" optimization approach, for *Task 3* and as a function of the target recall τ.

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.

We should note here that our optimizer's actual time estimates are often far from the actual execution times, especially at the beginning of the execution when parameter estimates are rough and usually inaccurate. Fortunately, these time estimates are only used to pick the best available strategy; therefore even coarse estimates are sufficient. We observed high time estimation errors frequently in our experiments but, due to the large differences in execution time between the strategies, our optimizer still managed to pick good execution

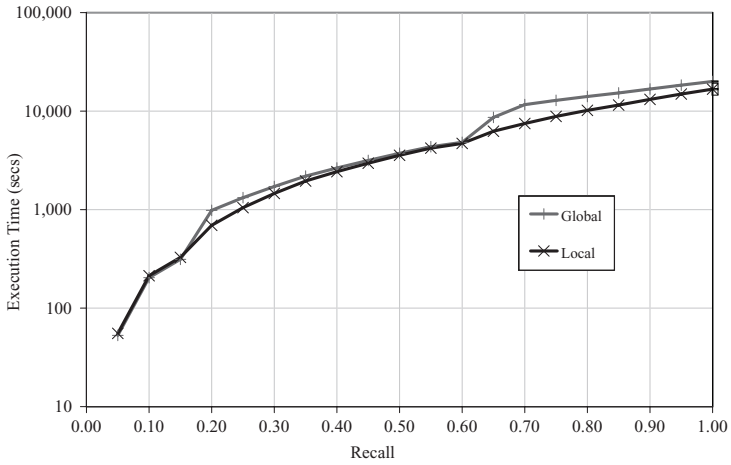

Fig. 24. Actual execution times for the "global" and "local" optimization approaches, for *Task 1a* and as a function of the target recall τ.

plans. As the execution progresses, the estimates become increasingly accurate and the optimizer not only identifies the best execution plans but also provides accurate time estimates as well.

As another interesting observation derived from our experiments, our prediction algorithm sometimes overestimates the achievable recall of a strategy (e.g., for *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 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 shows that a multistrategy execution can often perform better than an optimization strategy that attempts to pick a single execution plan, which is precisely the rationale behind our "local" optimization approach.

The "local" line in Figures 24, 25, 26, and 27 shows the actual execution time, for different recall thresholds, using our "local" optimization approach. Not surprisingly, the "local" optimizer behaves similarly to the "global" optimizer for low recall targets, where both optimization approaches proceed similarly. However, the "local" optimizer becomes noticeably preferable for higher target recall values that are beyond the reach of the fastest execution strategies: the "global" optimizer, by design, ignores an execution plan if this plan cannot reach the target recall. In contrast, the "local" optimizer can choose a fast execution strategy for extracting the initial batches of tokens, even if such strategy could not reach the overall target recall; then the "local" optimizer can pick a slower strategy to continue from the point where the fastest plan has stopped. Interestingly, the advantage of the "local" optimizer diminishes over time, and its execution times slowly converge towards the execution times of the "global" optimizer: the "local" optimizer targets the most promising parts of the database early on, through fast early executions, and the associated speedups in the execution diminish

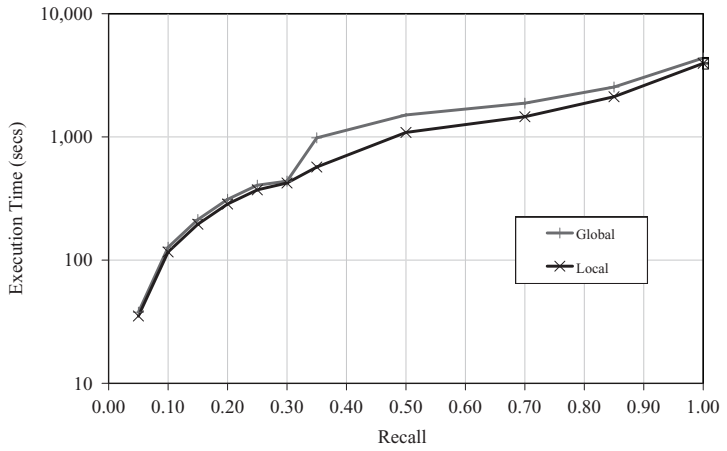

Fig. 25. Actual execution times for the "global" and "local" optimization approaches, for *Task 1b* and as a function of the target recall τ.

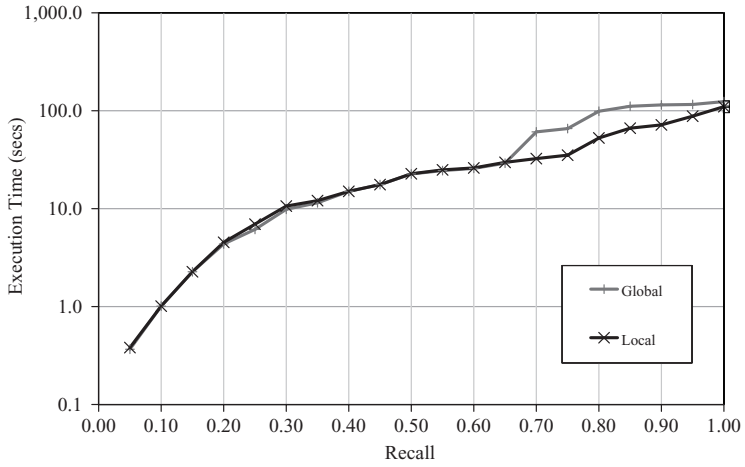

Fig. 26. Actual execution times for the "global" and "local" optimization approaches, for *Task 2* and as a function of the target recall τ.

as the distribution of tokens over the unseen documents becomes sparser and sparser.

8.3 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 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

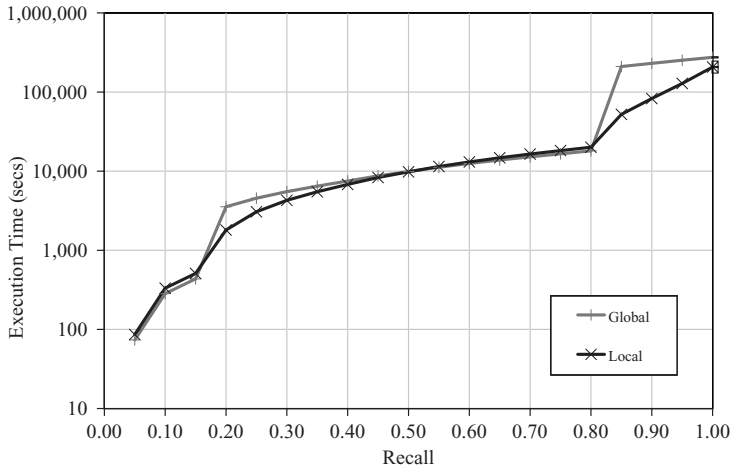

Fig. 27. Actual execution times for the "global" and "local" optimization approaches, for *Task 3* and as a function of the target recall τ.

automatically select the (combination of) best execution strategies to achieve this recall.

9. RELATED WORK

In this article, 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., Grishman [1997]; Yangarber and Grishman [1998]). Some systems use the *Filtered Scan* strategy of Section 4.2, where only the documents that match specific URL patterns (e.g., Brin [1998]) or regular expressions (e.g., Grishman et al. [2002]) are processed further. Agichtein and Gravano [2003] 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. (Different approaches to scaling up information extraction are surveyed by Agichtein [2005].) More recently, Etzioni et al. [2004] 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 [2005] 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 by Callan et al. [1999] can be cast as an instance of *Iterative Set Expansion*, as discussed in Section 4.3. Another strategy for the same task [Ipeirotis and Gravano 2002] 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 [Gravano et al. 2002], 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* [Chakrabarti et al. 1999, 2002; Diligenti et al. 2000; Menczer et al. 2004], where a classifier determines which links to follow for subsequent computationally expensive steps of retrieval and processing. Other strategies, which we model as variants of *Automatic Query Generation*, may also be effective for some scenarios [Cohen and Singer 1996].

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) [Oard 1997], which can be executed following either *Filtered Scan*, or, if appropriate, *Automatic Query Generation*. Another task is the construction of comparative Web shopping agents [Doorenbos et al. 1997]. 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*). For the task of training named entity recognition systems, Jones [2005] showed that named-entity co-occurrence graphs (e.g., involving person and location names) follow a power-law degree distribution, which suggests that the execution of this task might also be modeled in our framework. As another example, Web question answering systems [Banko et al. 2002] usually translate a natural language question into a set of Web search queries to retrieve documents for a subsequent answer extraction step over a subset of the retrieved documents. This process can thus be viewed as a combination of *Automatic Query Generation* and *Filtered Scan*. Recently, Ntoulas et al. [2005] 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 effective database sampling techniques (e.g., Chaudhuri et al. [1998]; Ling and Sun [1995]) 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 also related to previous research on optimizing query plans with user-defined predicates [Chaudhuri and Shim 1999], 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 [Chaudhuri et al. 2003]).

Our optimization approach is conceptually related to adaptive query execution techniques developed for relational data. In particular, Ives et al. [1999] describe the Tukwila system for distributed processing of joins over autonomous data sources, with no information about table cardinalities or value distributions and with unpredictable network delays and data arrival rates. Hence any initially chosen execution plan is expected to be adjusted during query execution, as the availability of sources changes or better relevant

statistics are obtained. Our optimization approach also revisits the choice of execution strategies for a text-centric task, as documents are retrieved and tokens extracted and, consequently, the statistics on document and token distributions are refined. Our focus in this article is on processing text-centric tasks over a single text "database," and not on gracefully recovering from unpredictable delays when executing a particular operator in a join pipeline. Our optimization approach is also conceptually related to the *eddies* work, where a query execution plan is continuously reevaluated after each output *tuple* [Avnur and Hellerstein 2000]. The *eddies* work thus focuses on effective join processing, allowing flexible reordering of the query operators.

Our optimization approach is also related to the reoptimization methods presented by Kabra and DeWitt [1998]: the statistics are updated at key points during query execution to reallocate memory resources for active operators and to potentially adjust the plan for the *rest* of the execution. The general reoptimization approach of Kabra and DeWitt [1998] for relational data was extended by Markl et al. [2004], where the cardinality estimation errors detected during query execution may trigger a reoptimization step for the execution plan. Our general optimization approach behaves similarly, albeit for text-centric tasks, which require different parameter estimation techniques.

This article substantially extends our previous work in Agichtein et al. [2003] and Ipeirotis et al. [2006]. Our earlier article [Agichtein et al. 2003] presented preliminary results on modeling and estimating the achievable recall of *Iterative Set Expansion*, for *Task 1* (information extraction) and *Task 2* (database content summary construction). Later, in Ipeirotis et al. [2006], we developed and evaluated rigorous cost models for *Iterative Set Expansion*, as well as for three additional general execution strategies, namely *Scan*, *Filtered Scan*, and *Automatic Query Generation*. In Ipeirotis et al. [2006], we also presented a principled, cost-based "global" optimization approach for selecting the most efficient execution strategy automatically. The current article substantially extends the analysis and experimental evaluation by Ipeirotis et al. [2006]. In this article, we present a detailed description of our methodology for estimating the parameter values required by our cost model (Sections 6.1.1 through 6.1.4), whereas in Ipeirotis et al. [2006], due to space restrictions, we only gave a high-level overview of our techniques. Another substantial new contribution with respect to Ipeirotis et al. [2006] is that now our optimizers do not rely on knowledge of the $|Tokens|$ statistics, but instead estimate this parameter "on-the-fly" as well, during execution of the task. Furthermore, in this article, we present a new, "local" optimizer that potentially builds "multistrategy" executions by picking the best strategy for each batch of k tokens (Section 6.2). In contrast, the "global" optimization approach in Ipeirotis et al. [2006] only attempts to identify a single execution plan that is capable of reaching the full target recall. We implemented the new local optimization approach and compared it experimentally against the global approach of Ipeirotis et al. [2006]; the results of the comparison are presented in Figures 24, 25, 26, and 27, in Section 8. The results show the superiority of the "local" optimizer over the "global" optimizer.

An alternative approach to processing SQL queries over unstructured text was presented by Cafarella et al. [2007], which requires a single scan over the

collection for all possible binary relations. Finally, Jain et al. [2007] have recently presented a query optimization approach for simple SQL queries over (structured data extracted from) text databases. This work heavily relies on information extraction systems and is thus closely related to our *Task 1* scenario. Jain et al. [2007] consider multiple document retrieval strategies to process a SQL query, including *Scan*, *Automatic Query Generation*, and other query-based strategies. Unlike our setting, however, Jain et al. [2007] focus on extraction scenarios that typically involve multiple information extraction systems, whose output might then need to be integrated and joined to answer a given SQL query. The SQL query optimization approach by Jain et al. [2007] accounts for errors originating in the information extraction process, and characterizes alternate query executions—which might differ in their choice of extraction systems—based on their precision, as well as on their execution time and recall. An interesting research direction is to incorporate the time and recall estimation models presented in this article into the model of Jain et al. [2007].

10. DISCUSSION: ASSUMPTIONS, LIMITATIONS, AND FUTURE WORK

We now further discuss some assumptions behind our work, as well as outline opportunities for future work.

Our work assumes that the document processors are perfect, meaning that they return only correct tokens and identify all the tokens that appear in a document. Unfortunately, for many tasks (e.g., *Tasks 1* and *3*) the document processors are inherently noisy. Relaxing the assumption that document processors are perfect and, correspondingly, predicting the precision of the output produced by different strategies are natural next steps for improving our models.

Another interesting direction for future research is to examine how to minimize the task-specific knowledge that is needed for our optimization techniques. Even though the analysis presented in Section 5 is nonparametric (i.e., it does not assume any particular distribution family for the token and document degrees), our optimizer of Section 6 uses a parametric setting to reduce the number of unknown values that need to be estimated. An interesting direction for future work is to use techniques for efficient histogram construction from the database literature to overcome this restriction.

Finally, our modeling of the crawl- and query-based strategies assumes that only a simple inverted index is available over the text database, built on keywords and only accessible via a query interface (i.e., we cannot access the contents of the index directly). In contrast, execution strategies for some text-centric tasks might sometimes benefit from other types of indexes (e.g., as is the case in the work by Cafarella and Etzioni [2005] for *Task 1*). Analyzing such strategies is another interesting direction for future work.

11. CONCLUSION

In this article, 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 optimizers for text-centric tasks, and showed that our optimizers help build efficient execution plans to achieve a target recall, resulting in executions that can be orders of magnitude faster than alternate choices.

REFERENCES

AGICHTEIN, E. 2005. Scaling information extraction to large document collections. *IEEE Data Eng. Bull. 28*, 4 (Dec.), 3–10.

AGICHTEIN, E. AND GRAVANO, L. 2000. Snowball: Extracting relations from large plain-text collections. In *Proceedings of the Fifth ACM Conference on Digital Libraries (DL'00)*.

AGICHTEIN, E. AND GRAVANO, L. 2003. Querying text databases for efficient information extraction. In *Proceedings of the 19th IEEE International Conference on Data Engineering (ICDE'03)*.

AGICHTEIN, E., IPEIROTIS, P. G., AND GRAVANO, L. 2003. Modeling query-based access to text databases. In *Proceedings of the Sixth International Workshop on the Web and Databases (WebDB'03)*. 87–92.

AVNUR, R. AND HELLERSTEIN, J. M. 2000. Eddies: Continuously adaptive query processing. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data (SIGMOD'00)*. 261–272.

BAAYEN, R. H. 2006. *Word Frequency Distributions*. Springer, Berlin, Germany.

BANKO, M., BRILL, E., DUMAIS, S., AND LIN, J. 2002. AskMSR: Question answering using the World Wide Web. In *Proceedings of 2002 AAAI Spring Symposium on Mining Answers from Texts and Knowledge Bases*.

BERGMAN, M. K. 2001. The deep Web: Surfacing hidden value. *J. Electron. Publish. 7*, 1 (Aug.).

BLAKE, C. L. AND MERZ, C. J. 1998. UCI repository of machine learning databases. `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

BRIN, S. 1998. Extracting patterns and relations from the World Wide Web. In *Proceedings of the First International Workshop on the Web and Databases (WebDB'98)*. 172–183.

CAFARELLA, M. J. AND ETZIONI, O. 2005. A search engine for natural language applications. In *Proceedings of the 14th International World Wide Web Conference (WWW'05)*. 442–452.

CAFARELLA, M. J., RE, C., SUCIU, D., AND ETZIONI, O. 2007. Structured querying of Web text data: A technical challenge. In *Proceedings of the Third Biennial Conference on Innovative Data Systems Research (CIDR'07)*. 225–234.

CALLAN, J. P. AND CONNELL, M. 2001. Query-based sampling of text databases. *ACM Trans. Inform. Syst. 19*, 2, 97–130.

CALLAN, J. P., CONNELL, M., AND DU, A. 1999. Automatic discovery of language models for text databases. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data (SIGMOD'99)*. 479–490.

CALLAN, J. P., LU, Z., AND CROFT, W. B. 1995. Searching distributed collections with inference networks. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'95)*. 21–28.

CHAKRABARTI, S., PUNERA, K., AND SUBRAMANYAM, M. 2002. Accelerated focused crawling through online relevance feedback. In *Proceedings of the 11th International World Wide Web Conference (WWW11)*. 148–159.

CHAKRABARTI, S., VAN DEN BERG, M., AND DOM, B. 1999. Focused crawling: A new approach to topic-specific Web resource discovery. *Comput. Netw. 31*, 11–16 (May), 1623–1640.

CHAUDHURI, S., GANJAM, K., GANTI, V., AND MOTWANI, R. 2003. Robust and efficient fuzzy match for online data cleaning. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD'03)*. 313–324.

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

CHAUDHURI, S. AND SHIM, K. 1999. Optimization of queries with user-defined predicates. *ACM Trans. Database Syst. 24*, 2, 177–228.

CHUNG, F. AND LU, L. 2002. Connected components in random graphs with given degree sequences. *Ann. Combin. 6*, 125–145.

COHEN, W. W. 1996. Learning trees and rules with set-valued features. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI-96), Eighth Conference on Innovative Applications of Artificial Intelligence (IAAI-96)*. 709–716.

COHEN, W. W. AND SINGER, Y. 1996. Learning to query the Web. In *Proceedings of the AAAI Workshop on Internet-Based Information Systems*. 16–25.

CUNNINGHAM, H. 2006. Information extraction, automatic. In *Encyclopedia of Language and Linguistics*, 2nd ed. Elsevier Science, Amsterdam, The Netherlands.

DILIGENTI, M., COETZEE, F., LAWRENCE, S., GILES, C. L., AND GORI, M. 2000. Focused crawling using context graphs. In *Proceedings of the 26th International Conference on Very Large Databases (VLDB'00)*. 527–534.

DOORENBOS, R. B., ETZIONI, O., AND WELD, D. S. 1997. A scalable comparison-shopping agent for the World-Wide Web. In *AGENTS '97: Proceedings of the First International Conference on Autonomous Agents*. 39–48.

DUDA, R. O., HART, P. E., AND STORK, D. G. 2000. *Pattern Classification*, 2nd ed. Wiley, New York, NY.

ETZIONI, O., CAFARELLA, M. J., DOWNEY, D., KOK, S., POPESCU, A.-M., SHAKED, T., SODERLAND, S., WELD, D. S., AND YATES, A. 2004. Web-scale information extraction in KnowItAll (preliminary results). In *Proceedings of the 13th International World Wide Web Conference (WWW'04)*. 100–110.

GRAVANO, L., GARCÍA-MOLINA, H., AND TOMASIC, A. 1999. *GlOSS*: Text-source discovery over the Internet. *ACM Trans. Database Syst. 24*, 2 (June), 229–264.

GRAVANO, L., IPEIROTIS, P. G., AND SAHAMI, M. 2002. Query- vs. crawling-based classification of searchable Web databases. *IEEE Data Eng. Bull. 25*, 1 (Mar.), 43–50.

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

GRISHMAN, R., HUTTUNEN, S., AND YANGARBER, R. 2002. Information extraction for enhanced access to disease outbreak reports. *J. Biomed. Inform. 35*, 4 (Aug.), 236–246.

IPEIROTIS, P. G., AGICHTEIN, E., JAIN, P., AND GRAVANO, L. 2006. 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'06)*. 265–276.

IPEIROTIS, P. G. AND GRAVANO, L. 2002. Distributed search over the hidden Web: Hierarchical database sampling and selection. In *Proceedings of the 28th International Conference on Very Large Databases (VLDB'02)*. 394–405.

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

JAIN, A., DOAN, A., AND GRAVANO, L. 2007. SQL queries over unstructured text databases (poster paper). In *Proceedings of the 22nd IEEE International Conference on Data Engineering (ICDE'07)*.

JONES, R. 2005. Learning to extract entities from labeled and unlabeled text. Ph.D. dissertation. Carnegie Mellon University, School of Computer Science, Pittsburgh, PA.

KABRA, N. AND DEWITT, D. J. 1998. Efficient mid-query re-optimization of sub-optimal query execution plans. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data (SIGMOD'98)*. 106–117.

LING, Y. AND SUN, W. 1995. An evaluation of sampling-based size estimation methods for selections in database systems. In *Proceedings of the 11th IEEE International Conference on Data Engineering (ICDE'95)*. 532–539.

MARKL, V., RAMAN, V., SIMMEN, D., LOHMAN, G., PIRAHESH, H., AND CILIMDZIC, M. 2004. Robust query processing through progressive optimization. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data (SIGMOD'04)*. 659–670.

MCCALLUM, A. 2005. Information extraction: Distilling structured data from unstructured text. *ACM Queue 3*, 9, 48–57.

MENCZER, F., PANT, G., AND SRINIVASAN, P. 2004. Topical Web crawlers: Evaluating adaptive algorithms. *ACM Trans. Internet Tech. 4*, 4 (Nov.), 378–419.

MITZENMACHER, M. 2004. Dynamic models for file sizes and double pareto distributions. *Internet Math. 1*, 3, 305–334.

NEWMAN, M. E. J., STROGATZ, S. H., AND WATTS, D. J. 2001. Random graphs with arbitrary degree distributions and their applications. *Phys. Rev. E (Statistical, Nonlinear, and Soft Matter Physics) 64*, 2 (Aug.), 026118 (1–17).

NTOULAS, A., ZERFOS, P., AND CHO, J. 2005. Downloading textual hidden Web content by keyword queries. In *Proceedings of the Fifth ACM+IEEE Joint Conference on Digital Libraries (JCDL'05)*.

OARD, D. W. 1997. The state of the art in text filtering. *User Model. User-Adapt. Interact. 7*, 3, 141–178.

ROSS, S. M. 2002. *Introduction to Probability Models*, 8th ed. Academic Press, New York, NY.

SEBASTIANI, F. 2002. Machine learning in automated text categorization. *ACM Comput. Surv. 34*, 1 (Mar.), 1–47.

VAPNIK, V. N. 1998. *Statistical Learning Theory*. Wiley-Interscience, New York, NY.

WILF, H. S. 1990. *Generatingfunctionology*. Academic Press Professional, Inc., New York, NY.

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

ZIPF, G. K. 1949. *Human Behavior and the Principle of Least Effort*. Addison-Wesley, Reading, MA.

Received October 2006; accepted January 2007