Information Processing and Management 53 (2017) 309–331

Contents lists available at ScienceDirect

Information Processing and Management

journal homepage: www.elsevier.com/locate/infoproman

# Sampling strategies for information extraction over the deep web

Pablo Barrio*, Luis Gravano

*Columbia University, Computer Science Department, 500 West 120th Street, Room 405, MC0401, New York, NY 10027, USA*

ARTICLE INFO

*Article history:*
Received 25 December 2015
Revised 21 November 2016
Accepted 23 November 2016
Available online 6 December 2016

*Keywords:*
Information extraction
Sampling
Deep web
Text mining
Scalability

ABSTRACT

Information extraction systems discover structured information in natural language text. Having information in structured form enables much richer querying and data mining than possible over the natural language text. However, information extraction is a computationally expensive task, and hence improving the efficiency of the extraction process over large text collections is of critical interest. In this paper, we focus on an especially valuable family of text collections, namely, the so-called deep-web text collections, whose contents are not crawlable and are only available via querying. Important steps for efficient information extraction over deep-web text collections (e.g., selecting the collections on which to focus the extraction effort, based on their contents; or learning which documents within these collections—and in which order—to process, based on their words and phrases) require having a representative document sample from each collection. These document samples have to be collected by querying the deep-web text collections, an expensive process that renders impractical the existing sampling approaches developed for other data scenarios. In this paper, we systematically study the space of query-based document sampling techniques for information extraction over the deep web. Specifically, we consider (i) alternative query execution schedules, which vary on how they account for the query effectiveness, and (ii) alternative document retrieval and processing schedules, which vary on how they distribute the extraction effort over documents. We report the results of the first large-scale experimental evaluation of sampling techniques for information extraction over the deep web. Our results show the merits and limitations of the alternative query execution and document retrieval and processing strategies, and provide a roadmap for addressing this critically important building block for efficient, scalable information extraction.

## 1. Introduction

*Information extraction systems* are complex software tools that discover structured information in natural language text. For example, an information extraction system trained to extract *Occurs* − *in*(*Natural Disaster, Location*) tuples would extract the tuple ⟨tornado, Adairsville⟩ from the text "the tornado caused significant damage in Adairsville." Having information in structured form enables much richer querying and data mining than possible over the natural language text. Unfortunately, information extraction is a time-consuming task. Since text collections routinely contain millions of documents or more,

* Corresponding author.
*E-mail addresses:* pjbarrio@cs.columbia.edu (P. Barrio), gravano@cs.columbia.edu (L. Gravano).

http://dx.doi.org/10.1016/j.ipm.2016.11.006
0306-4573/© 2016 Elsevier Ltd. All rights reserved.

improving the efficiency and scalability of the information extraction process over these large text collections is critical. In this paper, we focus on an especially valuable family of text collections, namely, the so-called *deep-web text collections*, whose contents are not crawlable and are only available via querying (Bergman, 2001; Gupta & Bhatia, 2014; Raghavan & Garcia-Molina, 2001; Sherman & Price, 2003). Deep-web text collections many times exhibit a full-text search interface. (We rely on this interface to access the contents of the collection, as we discuss in Section 4.) Moreover, deep-web text collections cover a wide range of topics and are hence relevant to a broad spectrum of information extraction tasks. Efficiently processing the contents of these collections is thus of significant interest.

Important steps for efficient information extraction over deep-web text collections require having, for each collection, a representative document sample of documents that lead to the extraction of tuples for a relation of interest. We refer to the documents that lead to the extraction of tuples for a relation of interest as the *useful documents* for the information extraction task.[1] The document samples can be valuable, for instance, to decide on which collections to focus the extraction effort, based on their contents (Barrio, Gravano, & Develder, 2015a). For example, such document samples can reveal that the Federal Emergency Management Agency (FEMA) collection,[2] an up-to-date resource for natural disasters and other hazards in the United States, is a better collection for the extraction of the $Occurs - in$ relation than the PubMed collection,[3] a database for life sciences and biomedical research. Similarly, a document sample from a collection can be valuable to help select and rank the collection documents for the extraction task: for efficiency, we should attempt to process only useful documents, so techniques such as QXtract (Agichtein & Gravano, 2003), FactCrawl (Boden, Löser, Nagel, & Pieper, 2012), PRDualRank (Fang & Chang, 2011), and BAgg-IE and RSVM-IE (Barrio, Simões, Galhardas, & Gravano, 2015b) use these samples to learn words and phrases that separate useful documents for the information extraction task from the rest. The samples on which these techniques rely must be collected in a collection-specific way, because the focus and language of each collection generally differs from those of other collections.

Given an information extraction task, producing high-quality, representative document samples from a deep-web text collection is a challenging process, for two main reasons. *(1) Sampling efficiency:* the document sampling process has to be efficient and lightweight because, as discussed above, it is often used to make the overall information extraction execution over deep-web text collections efficient and scalable. This efficiency requirement is complicated by the fact that document samples can only be collected, by definition, by querying the (remote) deep-web text collections, which is expensive. Furthermore, as we will see, analyzing the documents as we retrieve them, to decide the composition of the samples, is also an expensive proposition because it often involves running the extraction system at hand on the documents. *(2) Sampling quality:* the document sampling process has to return documents that represent the relevant extraction-related document characteristics in each deep-web text collection. This quality requirement is complicated by the fact that the useful documents for the information extraction task are often a small minority of the collection documents. For example, under 2% of the 1.03 million documents in TREC 1–5 collections[4] are useful for $Occurs - in$ when processed with a state-of-the-art information extraction system. Furthermore, even within a relatively small number of documents, the sampling process should capture the large variations in language and general content in the documents.

Earlier efforts to address the efficiency and scalability of the extraction process have incorporated sampling in a relatively ad-hoc manner. Notably, QXtract (Agichtein & Gravano, 2003), FactCrawl (Boden et al., 2012), PRDualRank (Fang & Chang, 2011), and BAgg-IE and RSVM-IE (Barrio et al., 2015b) rely on document sampling to develop document retrieval or ranking strategies for an information extraction task at hand. Despite the important role of sampling in these techniques, the sampling approaches that they use are far from ideal, as we will see. Specifically, these techniques adopt flavors of sampling that rely on high-precision queries to target certain documents efficiently, but fail to capture the large variety of extraction-relevant document characteristics discussed above. Consequently, they miss important groups of documents during sampling, which other sampling strategies can effectively obtain, as we will show experimentally.

Query-based document sampling has also been studied beyond information extraction, for other text-centric tasks. As notable examples, Bar-Yossef and Gurevich (2008), Zhang, Zhang, and Das (2011), Wang, Liang, and Lu (2014a), and Wang, Liang, and Lu (2014b) developed document sampling techniques for the generation of unbiased descriptors of the collections. Unfortunately, these approaches are ineffective for our information extraction scenario, because they focus on obtaining random document samples. As we discussed above, our scenario requires that the document samples represent the often small minority of documents that lead to extraction output for a given information extraction task. To sufficiently characterize the documents in such small portions of the collections through random sampling, the above techniques would require issuing an exorbitant number of queries to the deep-web text collections.

[1] We do not consider the correctness of extracted tuples in our work. Instead, we trust the output of the information extraction system and focus on efficiently and effectively identifying useful documents for our extraction task of interest. For correctness, we could use the confidence score that the information extraction system often assigns to each extracted tuple. This approach has been adopted in Agichtein and Cucerzan (2005); Jain and Srivastava (2009) for the (related) task of identifying text collections with high-quality, or correct, tuples. Alternatively, to deem tuples as correct, we could adopt the statistical approach proposed in Jain, Doan, and Gravano (2008); Jain and Ipeirotis (2009); Jain, Ipeirotis, Doan, and Gravano (2009); Simões, Galhardas, and Gravano (2013) for the (related) task of building efficiency- and quality-aware execution plans to extract tuples from large text collections.

[2] http://www.fema.gov/.

[3] http://www.ncbi.nlm.nih.gov/pubmed.

[4] http://trec.nist.gov/data.html.

In this paper, we systematically study the space of query-based document sampling techniques for information extraction over the deep web. Specifically, we consider (i) alternative query execution schedules, which vary on how they account for the query effectiveness; and (ii) alternative document retrieval and processing schedules, which vary on how they distribute the extraction effort over documents. We conduct a large-scale and fine-grained experimental evaluation over real deep-web text collections, and for a large variety of information extraction tasks, to assess the merits of the alternative query execution and document retrieval and processing strategies. We also explore several different query generation techniques, for robustness. Our conclusions are twofold. Regarding query execution, schedules that focus on queries with a high fraction—and number—of useful documents, namely, the *effective* queries, improve sampling efficiency. In contrast, schedules that prioritize less-effective queries improve sampling quality, because in this case many (potentially diverse) queries need to be issued to retrieve a desired number of useful documents, hence leading to high-quality document samples. Regarding document retrieval and processing, schedules that process the documents for each query exhaustively at once improve sampling efficiency when the sampling technique focuses on effective queries. In contrast, schedules that process documents incrementally and in rounds improve sampling quality, because a larger variety of documents—from a larger number of queries—is processed. As we will see, fundamentally different sampling techniques (i.e., with distinct implications in sampling efficiency and quality) are possible.

In short, the main contributions of this paper are:

- A thorough discussion of the sample generation problem for information extraction over deep-web text collections (Section 2).
- A systematic study of query-based document sampling techniques for information extraction over deep-web text collections that considers (i) alternative query execution schedules and (ii) alternative document retrieval and processing schedules (Section 3).
- The first large-scale and fine-grained evaluation of query-based document sampling techniques for information extraction over the deep web. We perform our experiments over real deep-web text collections and for a large variety of extraction tasks. We show the implications in sampling efficiency and quality of different query execution schedules, as well as of different document retrieval and processing schedules (Sections 4 and 5).

We now review necessary background and define our problem of focus in this paper (Section 2).

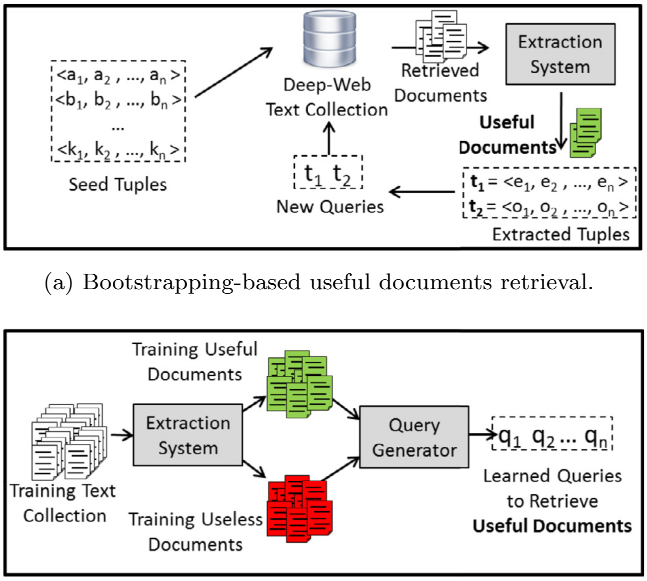
## 2. Background and problem definition

*Information extraction systems* extract structured information from natural language text. Because of all the operations that are typically involved (e.g., dependency parsing or named entity tagging), information extraction is a computationally expensive process. Therefore, processing all documents of a large, or rapidly evolving, text collection can become prohibitively time consuming.

In this paper, we focus on an especially valuable family of text collections, namely, the so-called *deep-web text collections*, whose contents are not crawlable and are only available via querying (Bergman, 2001; Gupta & Bhatia, 2014; Raghavan & Garcia-Molina, 2001; Sherman & Price, 2003). Examples of deep-web text collections include the FEMA and PubMed collections discussed in the Introduction. In addition to FEMA and PubMed, the deep web hosts a large number of high-quality collections across many domains (Zillman, 2008). Consequently, a wide range of information extraction tasks can benefit from gathering and exploiting the valuable information buried in such deep-web text collections.

To run an information extraction system over a deep-web text collection, we could attempt first to download all the collection documents, for which we could use proposed approaches for such "crawling" (e.g., Barbosa & Freire, 2010; Ntoulas, Zerfos, & Cho, 2005; Raghavan & Garcia-Molina, 2001; Vieira, Barbosa, Freire, & Silva, 2008; Wang, Li, Pi, & Lu, 2015). We could then run the information extraction system over the local copy of the collection documents. Unfortunately, such crawling approaches are expensive—they require large numbers of queries—and have far-from-perfect recall—they fail to retrieve many documents, given the query-only nature of the deep-web text collections (Tirado, Serban, Guo, & Yoneki, 2016). Furthermore, such expensive crawling approaches are unnecessary: the documents that lead to the extraction of tuples for a given information extraction task, namely, the *useful documents* for the extraction task, are often a small fraction of the collection, because relations are generally topic-specific, in that they are associated mainly with documents about certain topics. For these reasons, in this paper we move away from generic crawling approaches, and focus instead on more efficient and effective targeted query-based approaches.

Important steps for efficient information extraction over deep-web text collections require having a representative document sample from each collection. For example, these samples help identify the collections and documents within them on which to focus the extraction effort. To see how, consider the $Occurs - in(Natural\ Disaster, Location)$ relation discussed in the Introduction. We could start by analyzing the content and number of $Occurs - in$ tuples that samples from the FEMA and PubMed collections above include, to conclude that FEMA is more valuable than PubMed for this relation. In turn, we could run techniques like QXtract (Agichtein & Gravano, 2003), FactCrawl (Boden et al., 2012), PRDualRank (Fang & Chang, 2011), or BAgg-IE and RSVM-IE (Barrio et al., 2015b) over the document sample collected from FEMA to learn words and phrases, such as "Richter" or "hypocenter", that are discriminative of the useful documents for the $Occurs - in$ relation. We could then use these words and phrases as text queries to retrieve and prioritize the documents from the collection that the extraction system will ultimately process (Agichtein & Gravano, 2003).



(a) Bootstrapping-based useful documents retrieval.

(b) Learning-based useful documents retrieval.

**Fig. 1.** Two main families of existing query generation techniques for useful document retrieval.

To perform the above steps in the extraction process effectively, the document samples on which these steps rely must accurately reflect the extraction-related characteristics of the useful documents in the collections. Unfortunately, generic query-based sampling techniques (e.g., Callan & Connell (2001)) become impractical for this task, because useful documents are often a small minority in the collection, as discussed. Therefore, the problem of focus in this paper is that of efficiently collecting high-quality document samples for information extraction from deep-web text collections, as follows:

**Problem definition.** *Consider a deep-web text collection C and an information extraction system E trained to extract tuples for a relation from text. To enable efficient and effective information extraction over collection C, we need a sample of documents from C that represents the population of useful documents in C with respect to E. Specifically, the goal is to obtain a sample of useful documents that satisfies certain quality metrics (e.g., diversity in the tuples extracted with E from the sampled documents) while satisfying certain efficiency-related requirements (e.g., minimize the number of documents processed with E and the number of queries issued to C as part of the sampling process).*

**Existing techniques:** Existing query-based techniques for retrieving useful documents from a collection fall into two families. Techniques in the first family adopt a bootstrapping approach: Starting with a small number of "seed" tuples for the relation of interest, these techniques iteratively retrieve (potentially useful) documents by issuing as queries the seed tuples and, later, the new tuples that the extraction system discovers from documents as they are retrieved (Fig. 1(a)). Earlier efforts to address the efficiency and scalability of the extraction process (e.g., QXtract (Agichtein & Gravano, 2003), FactCrawl (Boden et al., 2012), and PRDualRank (Fang & Chang, 2011)) have adopted this family of techniques in their sample generation step, because queries tend to be high-precision. Unfortunately, as we will show experimentally, these techniques compromise recall and often miss important relevant groups of useful documents, which is undesirable during the sampling step.

Techniques in the second family adopt a statistical learning approach that aims to alleviate the recall limitation above: these techniques use a training sample of useful and useless documents labeled "for free," without human intervention, meaning that the documents are processed with the information extraction system at hand and labeled as useful if they produce tuples for the extraction task or useless otherwise. These techniques then learn keywords and phrases that are discriminative of the useful documents (Fig. 1(b)). Importantly, the learned keywords and phrases often include a score that roughly corresponds with their expected precision and recall for useful documents. These scores can be systematically exploited when issuing these learned keywords and phrases as text queries to retrieve potentially useful documents. For instance, QXtract (Agichtein & Gravano, 2003) issues the queries in descending score order, to first process queries that are likely to retrieve useful documents with high recall and precision. QXtract processes the documents retrieved by each query exhaustively at once before processing those retrieved by the following query. Unfortunately, these techniques mainly tackle the efficiency of the extraction process, one of the crucial aspects in our sampling problem. As we will see, to also address the sampling quality we need to choose carefully both the query execution as well as the document retrieval and processing strategies.

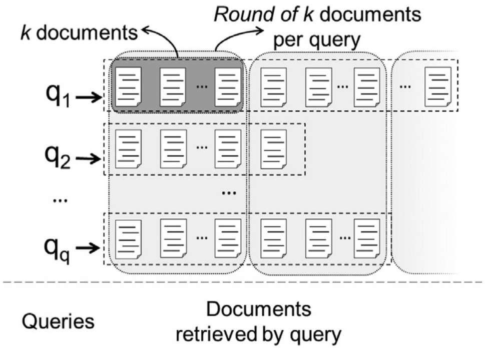


**Fig. 2.** Query-document space.

In the next section, we discuss different query execution and document retrieval and processing strategies, along with their implications in sampling efficiency and quality. We in turn introduce several different sampling techniques, which we evaluate experimentally in later sections.

## 3. Document sampling strategies

We now systematically study query-based document sampling techniques for information extraction over a deep-web text collection. We focus on learning-based methods, which rely on a learned set of text queries to retrieve potentially useful documents for an information extraction task of interest, as discussed in Section 2. (Section 4 describes the learning-based methods with which we experiment; these methods are orthogonal to the document sampling strategies that we study.) Unlike in the existing literature, though, we consider tackling both sampling quality and efficiency. We start by outlining—and analyzing the efficiency and quality of—different alternatives for processing the (learned) set of queries and their retrieved documents, namely, the *query–document space* of the queries (see Fig. 2). We then discuss how we can exploit the information that we gather from each query along the sampling process (e.g., the number of useful and useless documents that the query returns) to improve different aspects of the process. In turn, we introduce the sampling techniques that we study in this paper, which we evaluate experimentally in Sections 4 and 5.

**Exploring the query-document space:** We now consider different alternatives to exploring the query–document space of a set of queries for our sampling problem. We first consider alternative query execution schedules, which vary on how they account for the query effectiveness. Specifically, for a pool of documents retrieved by a query, we define the *effectiveness* of the query as the fraction of useful documents within this document pool. More formally, the effectiveness of a query is based on the so-called precision@$K$ in information retrieval, where relevance is defined in our case as usefulness and $K$ is the number of documents to process. Then, and in an orthogonal dimension, we consider alternative document retrieval and processing schedules, which differ on how they distribute the extraction effort over documents. We discuss these alternatives in detail next.

*Query execution:* The order in which we process queries during sampling, namely, the *query execution order*, is crucial to the efficiency and quality of the sampling process. For efficiency, on one hand, we need to prioritize effective queries (i.e., the queries that retrieve useful documents with high precision and recall), so that we mainly process—hence sample—useful documents. This is motivated by the fact that the sampling cost is a function of the number of issued queries—necessary to retrieve documents for the sample—and the number of documents retrieved and processed—necessary to decide the composition of the sample. The approach in Agichtein and Gravano (2003), for example, approximates this query order based on the learned query scores: This approach uses the learned score of a query as a surrogate of its effectiveness and arranges the queries in descending score order. Fig. 3 shows an example of such query order for $Occurs - in$: the (top) query [earthquake] is more effective than query [richter], because it retrieves more useful documents for the same number of processed documents.

Processing queries in decreasing effectiveness order leads to efficient executions that identify a sample of useful documents quickly. Unfortunately, if the query execution process is only guided by efficiency, the overall sampling quality might suffer. To see why, consider once again the example in Fig. 3. Specifically, if the query execution process were to focus, say, on queries [earthquake] and [richter], which are highly effective for $Occurs - in$, we would likely produce a document sample whose useful documents are predominantly about earthquakes and not about other natural disasters that should be included in the sample as well.

We thus argue that for quality we should sometimes prioritize less-effective queries, so that a larger—hence potentially more diverse—set of queries needs to be processed to obtain a desired number of useful documents. In our example in Fig. 3, for instance, such a query execution order would process query [aftermath] before processing other more effective queries (e.g., queries [vortex] or [earthquake]) and, more importantly, it would be more likely to cover documents about earthquakes, tornadoes, as well as other natural disasters, because a larger number of queries would be processed. This

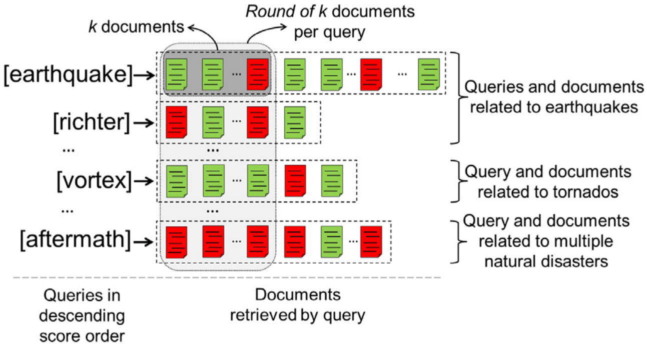

k documents
Round of k documents per query
[earthquake]→
[richter]→
...
[vortex]→
...
[aftermath]→
Queries and documents related to earthquakes
Query and documents related to tornados
Query and documents related to multiple natural disasters
Queries in descending score order
Documents retrieved by query

**Fig. 3.** Query-document space of a set queries for the *Occurs* – *in* relation. Useful and useless documents are illustrated in green and red, respectively. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

quality-driven approach, however, is problematic for two reasons. First, arranging the queries in such query order, or an approximation thereof, is nontrivial, unlike with the efficiency-driven query execution order above. Second, following this query execution order might compromise sampling efficiency dramatically, because many useless documents would need to be processed to retrieve a desired number of useful documents. Next, we discuss how different document retrieval and processing strategies can help address these limitations.

*Document retrieval and processing:* In addition to query execution, the strategy we adopt to retrieve and process the documents during sampling, namely, the *document retrieval and processing strategy*, is also crucial to the efficiency and quality of the sampling process. A possible choice is, of course, to process the documents returned by each query exhaustively at once, as suggested in Agichtein and Gravano (2003). Importantly, such an exhaustive strategy would promote the efficiency and quality considerations of the adopted query execution approach: If, for instance, the query execution is guided by efficiency (i.e., effective queries are prioritized), as in Agichtein and Gravano (2003), exhaustively processing the documents returned by each query will yield efficient sampling executions, because the number of queries to issue and documents to process to collect a desired number of useful documents will be relatively small. Analogously, if the query execution is guided by quality (i.e., it prioritizes less-effective queries), processing all documents returned by each query would produce high-quality sampling executions, because a larger, potentially more diverse set of queries will be processed. Unfortunately, by promoting the considerations of the adopted query execution, this exhaustive document processing strategy would also preserve their discussed quality and efficiency limitations.

An alternative document retrieval and processing strategy, and one that would alleviate the limitations of the exhaustive strategies above, would be to process the documents returned by each query iteratively and in rounds. Specifically, for a given query execution order, this strategy would iterate over the queries in order, processing only a certain number of documents per round. In Fig. 3, we identify the documents in the first round of an iterative strategy that processes $k$ documents from each query per round (see lightly shaded area in Fig. 3). As a result of this iterative process, documents will potentially be sampled from larger sets of queries—hence addressing sampling quality—and the extraction effort will be evenly distributed among queries during each round—hence addressing sampling efficiency. To better illustrate this, consider again the lightly shaded documents from our *Occurs* – *in* example in Fig. 3: These documents form a rather diverse sample—with documents about earthquakes, tornadoes, and other disasters—and only a fraction of the (many) useless documents retrieved by less-effective queries (e.g., query [aftermath]) need to be processed during the first round.

Despite the advantages of the iterative strategy above, specifying a number of documents per round that suitably balances efficiency and quality is a difficult proposition: Large values for such number would exhibit similar limitations to those of the exhaustive approach discussed above, while small values would affect sampling efficiency drastically due to the high querying cost that would be incurred. We experimentally evaluate the efficiency and quality implications of the choice of the number of documents to process per round in Sections 4 and 5. An additional problem of using small values is that we would be unable to precisely measure the real effectiveness of queries, a crucial measurement, as we discuss next.

**Exploiting observed information:** So far, we have discussed the query–document space exploration as a static, once-and-for-all process. However, there is valuable information (e.g., the number of useful and useless documents that a query returns) that we can gather gradually, as the sampling process progresses, and that we can use to improve this process. We now discuss how we can exploit this information (i) to revise the query execution order, for sampling efficiency and quality; and (ii) to filter underperforming queries, for sampling efficiency:

*Revising query execution order:* The learned score of a query is often used as a surrogate of its effectiveness, as argued earlier in this section, so we can expect the query order given by these scores to be correlated with that of the real effectiveness of the queries. However, for a given collection, these two query orders may differ considerably (e.g., due to the contents of the collection or the indexing and retrieval techniques thereof), and hence the query execution order may have to be revised. For instance, in our *Occurs* – *in* example in Fig. 3, prioritizing query [vortex] would yield more efficient sampling executions than processing the documents in descending score order.

**Table 1**
Sampling techniques and the alternatives they consider for each relevant aspect. For query execution, we consider prioritizing effective queries (>) or less-effective queries (<). For document retrieval and processing, we consider processing documents exhaustively at once (→) or iteratively and in rounds (↻). We finally consider techniques that perform query order revision or query filtering (+) and techniques that do not (−).

| Name | Query execution | Document retrieval and processing | Query order revision | Query filtering |
|---|---|---|---|---|
| QXtract | > | → | − | − |
| Cyclic | > | ↻ | − | − |
| Opportunistic | > | ↻ | + | − |
| Balanced | < | ↻ | + | − |
| F-QXtract | > | → | − | + |
| F-Cyclic | > | ↻ | − | + |
| F-Opportunistic | > | ↻ | + | + |
| F-Balanced | < | ↻ | + | + |

Fortunately, exhaustively processing the documents returned by a query to effectively measure its effectiveness is unnecessary: We can in fact gauge the real effectiveness of a query by only processing a relatively small subset of its returned documents, because the fraction of useful documents is expected to remain largely stable across its retrieved documents (Ipeirotis, Agichtein, Jain, & Gravano, 2007). This idea of “probing” queries to estimate their effectiveness is used in a preprocessing step in Boden et al. (2012) for the related problem of ranking documents to improve the efficiency of the extraction process. For instance, in our example in Fig. 3, we could process the first *k* documents returned by each query, to conclude that queries [vortex] and [aftermath] are, respectively, the most and least effective queries, and revise the query execution order in light of the observed information.

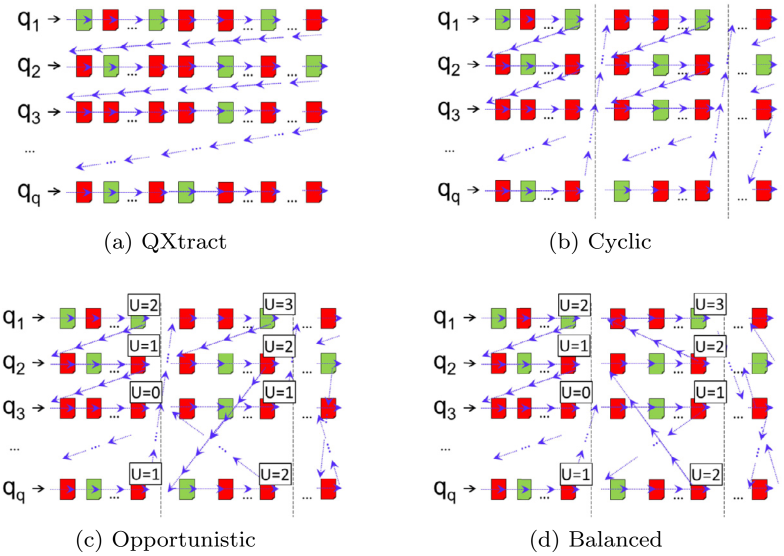
*Filtering underperforming queries:* By definition, there are two operations during sampling that hurt sampling efficiency, namely, issuing queries to the collection at hand that retrieve none—or a low fraction of—useful documents and retrieving and running the information extraction system of choice over a useless document. We argue that, for efficiency—and at the expense of a modest lost in quality—, we can exploit the gauged effectiveness of queries to avoid such (undesirable) cases and, in effect, focus only on cost-effective queries. For instance, in our example for *Occurs − in* in Fig. 3, if we filtered query [aftermath], we would avoid a considerable extraction effort—over multiple useless documents—at the expense of losing one useful document.

Based on the discussion above, we consider applying two filtering options. Our first alternative avoids issuing a query altogether if the observed effectiveness of previously issued queries is below a certain threshold. This filtering scheme is possible when we initially issue queries in descending score order, because the performance of the queries is expected to drop as a function of their order. In Fig. 3, for our *Occurs − in* example, we may filter query [aftermath] if previous queries exhibited poor effectiveness. Our second alternative filters already issued queries whose real, observed effectiveness drops below a certain threshold, to avoid processing useless documents. For instance, if we decide to filter queries that do not retrieve useful documents within the first *k* documents, the documents beyond *k* returned by query [aftermath] in Fig. 3 for *Occurs − in* would not be processed. Of course, deciding the settings for these filtering conditions is challenging, and we consider several options in Sections 4 and 5 together with their impact on sampling efficiency and quality.

**Sampling techniques:** So far, we have discussed the components involved in query–document space exploration as well as explained how we can exploit observed information to adaptively revise the query execution order and to focus the sampling effort. We now define the (arguably) most interesting query-based document sampling techniques for information extraction over the deep web, which we summarize in Table 1. As we will see in our experimental evaluation, we focus on techniques that collect high-quality document samples while keeping the sampling cost at reasonable levels. For the completeness of our evaluation, however, we include other sampling techniques, which we describe in the next sections, as needed. Importantly, some of the techniques in Table 1 (e.g., QXtract (Agichtein & Gravano, 2003)) have been introduced in the literature whereas others have not. We list them all here, to later assess their merits and limitations in Sections 4 and 5.

*QXtract: QXtract* (Agichtein & Gravano, 2003) explores the query–document space by issuing queries in descending learned score order and processing the documents retrieved by each query exhaustively at once (Fig. 4(a)). QXtract produces relatively efficient sampling executions; however, it may compromise sampling quality, as discussed earlier in this section. *Cyclic: Cyclic* explores the query–document space by issuing queries in descending learned score order and processing the documents retrieved by each query iteratively and in rounds (Fig. 4(b)). Cyclic addresses the sampling quality deficiencies of QXtract above, because it requires issuing a larger—hence potentially more diverse—set of queries to retrieve a desired number of useful documents. For instance, to collect three useful documents in Fig. 4(b), Cyclic processes the documents returned by two queries, namely, $q_1$ and $q_2$, whereas QXtract processes the documents returned by one query, namely, $q_1$.

*Opportunistic: Opportunistic* explores the query–document space by prioritizing—and issuing—effective queries and processing the documents retrieved by each query iteratively and in rounds (Fig. 4(c)). Opportunistic initially prioritizes queries according to the learned score; then, between rounds, and as it gathers relevant information for each query, Opportunistic revises the query execution order using the real, observed effectiveness of queries. The sampling quality of Opportunistic may suffer, though, because some groups of documents may still be underrepresented. To see why, consider Fig. 4(c): If the



(a) QXtract (b) Cyclic

(c) Opportunistic (d) Balanced

**Fig. 4.** Examples of query–document space exploration strategies. Useful and useless documents are illustrated in green and red, respectively. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

sampling process stops after collecting five useful documents (i.e., during the second round of documents retrieved by $q_1$), $q_1$ will contribute three useful documents to the sample whereas $q_2$, $q_3$, and $q_q$ will contribute at most one useful document each.

*Balanced: Balanced* explores the query–document space by prioritizing—and issuing—less-effective queries and processing the documents retrieved by each query iteratively and in rounds (Fig. 4(d)). Because finding an initial query order for the queries is problematic, as discussed, Balanced initially issues queries in descending score order; then, between rounds, and as it gathers relevant information for each query, Balanced revises the query execution order using the real, observed effectiveness of queries. By prioritizing less-effective queries, Balanced alleviates the quality limitation of Opportunistic above. Specifically, if in Fig. 4(d) we stop after collecting five useful documents (i.e., during the second round of documents retrieved by $q_3$, which will be now prioritized), each query will contribute a similar number of useful documents to the sample.

The techniques described thus far do not include the filtering step described earlier in this section. We define variants of these techniques that incorporate query filtering, which we refer to as *F-QXtract, F-Cyclic, F-Opportunistic*, and *F-Balanced*, respectively (see Table 1). These filtered techniques run as their unfiltered counterparts, and decide the queries to filter based on the filtering options described earlier in this section. Next, we describe the settings for our in-depth experimental evaluation of sampling techniques for information extraction.

## 4. Experimental settings

We describe the details of our experimental evaluation of the query-based document sample generation techniques for information extraction.

**Deep-web text collections:** We collected a representative set of 335 real Web text collections with a text search interface across different topics by following an approach similar in spirit to that of Gravano, Ipeirotis, and Sahami (2003) over the Open Directory Project directory[5]: We first selected the 8 categories with the highest number of entries, namely, Business, Society, Arts, Science, Computers, Recreation, Shopping, and Sports. From each category, we then selected the 5 most popular subcategories along with their corresponding 5 most popular subsubcategories, for a total of 200 subsubcategories. We then randomly chose 7 unique Web collections with a text search interface from each subsubcategory. (For each subsubcategory with fewer than 7 entries, we selected all its collections.) Finally, we randomly selected 335 collections from this set of collections, which we split into a tuning set (48 collections, or 15% of the collection set) and a test set (287 collections, or 85% of the collection set). We report our results over the test set.

**Training collection:**To learn the queries for our sampling strategies, and following Section 2, we need a text collection that includes useful documents for the extraction tasks of interest (see next). As discussed in Section 2, we label each

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

**Table 2**
Relations for our experiments along with fraction of useful documents in TREC 1–5 collections. In this table, Travel Destination and Winner are of type Location and Person, respectively.

| Relation | Useful documents (%) | |
|---|---|---|
| | SSK (%) | BONG (%) |
| Person–Career | 56.20 | 55.95 |
| Natural Disaster–Location | 2.03 | 2.74 |
| Man Made Disaster–Location | 0.80 | 0.87 |
| Person–Travel Destination | 1.08 | 4.67 |
| Person–Charge | 1.55 | 1.84 |
| Election–Winner | 0.24 | 0.84 |

document as useful if it produces tuples for the extraction task at hand and as useless otherwise. For this purpose, we combined all documents in the TREC 1–5 collections[6] to form a collection of 1,038,957 unique documents.

**Entity and relation extraction systems:** To include a variety of extraction approaches, we considered different relation extraction systems for each relation (see next), as well as different entity extraction systems for their corresponding entities. For relation extraction systems, we selected the two best performing combinations via 5-fold cross validation over a set of manually annotated documents. Likewise, for entity extraction systems, we selected the best performing combination for each entity type, and used it across all extraction tasks. However, for diversity, whenever we had ties in performance, we selected the (arguably) less common contender. We provide details next:

- *Relation extraction:*To extract our relations, we trained relation extraction systems using REEL[7] (Barrio, Simões, Galhardas, & Gravano, 2014). The two best performing systems, and the ones that we use in our experiments, are Subsequence Kernel (Bunescu & Mooney, 2005) (SSK) and Bag of *n*-grams (Giuliano, Lavelli, & Romano, 2006) (BONG).
- *Entity extraction:*To extract *person* and *location* entities, we used the StanfordNLP named entity tagger[8]; for other entities, we trained our own entity extractors using etxt2db.[9] Our final models are Maximum Entropy Markov Models (McCallum, Freitag, & Pereira, 2000) for *natural disasters* and Conditional Random Fields (McCallum & Li, 2003) for the remaining entities.

**Relations:** Table 2 shows the broad range of relations from different domains that we extract for our experiments. We also include the fraction of useful documents for each relation in our training collection for the different extraction systems above. Our relations include *sparse* relations, for which a relatively small fraction of documents (i.e., less than 2% of the documents) are useful, as well as *dense* relations.

**Bootstrapping-based sampling techniques:** In addition to the techniques discussed in Section 3, we evaluate the bootstrapping-based approach proposed in Agichtein and Gravano (2003)—and described in Section 2—that derives queries from all attributes in extracted tuples. We also experiment with queries derived from each attribute individually, as done in Fang and Chang (2011), to assess their individual impact in sampling quality and efficiency. The techniques that we explore are defined as follows:

- *Tuples* (Agichtein & Gravano, 2003) uses all tuple attributes in the query. For example, for the $Occurs - in$ relation, Tuples produces the Boolean conjunctive query [adairsville AND tornado] from the tuple ⟨adairsville, tornado⟩.
- *P-Tuples* (Fang & Chang, 2011) uses the most "specific" (see below) tuple attribute of the relation in the query with the goal of producing high-precision queries. To determine the most specific attribute in a relation, we analyze the schema of all relations supported by OpenCalais,[10] an online service for information extraction, and use the least common relation attribute. In our $Occurs - in$ relation, for instance, P-Tuples uses the *natural disaster* attribute, because this is the attribute that appears in the fewest OpenCalais relations, specifically in just one relation out of 83. P-Tuples thus produces the query [tornado] from the tuple ⟨adairsville, tornado⟩.
- *R-Tuples* (Fang & Chang, 2011) uses the most "general" (see below) tuple attribute of the relation in the query with the goal of producing high-recall queries. To determine the most general attribute in a relation, we analyze the schema of all relations supported by OpenCalais and use the most common relation attribute. In our $Occurs - in$ relation, for instance, R-Tuples uses the *location* attribute, because this is the attribute that appears in the most OpenCalais relations, specifically in 16 relations out of 83. R-Tuples thus produces the query [adairsville] from the tuple ⟨adairsville, tornado⟩.

As discussed in Section 2, given a collection, bootstrapping-based techniques start with a small seed of tuples for the relation of interest likely to be mentioned in the collection. We rely on a fully automatic approach to obtain such tuples:

[6] http://trec.nist.gov/data.html.

[7] http://reel.cs.columbia.edu/.

[8] http://nlp.stanford.edu/software/CRF-NER.shtml.

[9] http://web.ist.utl.pt/rui.lageira/.

[10] http://www.opencalais.com/.

We collected 20,000 unique documents from each collection using the crawling technique by Barbosa et al. (Barbosa & Freire, 2010). The technique in Barbosa and Freire (2010) generates initial queries using words in the main page of the text collection, and subsequently generates more queries using frequent keywords from the documents retrieved using the initial queries. We then run our information extraction systems over the crawled documents, to obtain tuples for each collection–information extraction system pair. We do not consider the cost of obtaining these tuples in the overall sampling cost reported in Section 5, because such initial cost is relatively low and by ignoring it we can focus more precisely on quantifying the (much more substantial) actual cost of sampling. For collections that did not produce tuples following this strategy, we generated seed tuples from the training collection.

**Learning-based query generation techniques:** We now describe different query generation techniques that learn queries from a training document sample, as discussed in Section 2. Query generation techniques rely on two building blocks, namely, the *candidate set of keywords* and the *query generation algorithm*. The candidate set of keywords specifies the words (e.g., all words except for stopwords) in the training documents that the query generation algorithm can use to construct queries. The query generation algorithm automatically learns as text queries discriminative words and phrases that separate useful from useless documents. As described in Section 2, these query generation techniques assign a score to each word or phrase, which is generally a function of its precision and recall for useful documents. In detail, our candidate set of keywords and query generation techniques are as follows:

*Candidate set of keywords:* We study two candidate set of keywords. For our first set, we removed: (i) English stopwords from MyISAM search indexes in MySQL,[11] as they are not effective as queries and (ii) rare words (i.e., words that appeared in less than 0.003% of the training documents) and frequent words (i.e., words that appeared in more than 90% of the training documents). For our second set, we also removed words in tuple attributes (e.g., "tornado"), as originally suggested in Agichtein and Gravano (2003). We refer to the first candidate set of keywords as *explicit*, since attribute values can be used to construct queries; accordingly, we refer to the second candidate set of keywords as *implicit*.

*Query generation algorithm:* We explored several techniques from two fundamentally different approaches: (i) *keyword selection*, which produce single-keyword queries from words that effectively separate useful from useless documents; and (ii) *keyword combination*, which produce phrase queries (e.g., ["tornado swept"]) or Boolean queries (e.g., [tornado AND vortex]) from word combinations that are discriminative of the useful documents. Specifically, we evaluated three keyword selection techniques (SVM, IG, and $\chi^2$) and two keyword combination techniques (Ripper and SP), which effectively cover existing query generation algorithms in the literature. We provide a brief description of these techniques, and explain how they score words and phrases:

- *SVM* (Mladenic, Brank, Grobelnik, & Milic-Frayling, 2004) trains a linear support vector machine classifier (Joachims, 1998) using the candidate set of keywords as Boolean features, and scores them with their corresponding learned weights.
- *IG* (Mladenic et al., 2004) scores each keyword in the candidate set with its *information gain* value (Kullback & Leibler, 1951). The information gain of a keyword $W$ is defined as $IG(C) = H(C) - H(C|W)$, where $C = \{\text{useful}, \text{useless}\}$, and $H(C)$ and $H(C|W)$ are the entropy and the conditional entropy, respectively. We ignore keywords that are more frequent in the useless documents than in the useful documents.
- *Chi-squared* ($\chi^2$) performs the Pearson's $\chi^2$ test (Pearson, 1900) over the candidate set of keywords and scores them with their corresponding $\chi^2$ value. Because the test runs over a 2 × 2 contingency table for each keyword—with usefulness of documents (i.e., useful or not) and occurrence of a keyword (i.e., it occurs in the document or not)—and because the observations in the table are rather small, we correct the test by applying Yates's correction (Yates, 1934) to the observations. The corrected $\chi^2$ value is obtained from $\chi^2(K) = \sum_{i\in\{0,1\}} \sum_{C\in\{+,-\}} \frac{((\mathcal{O}^K_{i,C} - \mathbb{E}^K_{i,C})-0.5)^2}{\mathbb{E}^K_{i,C}}$, where $k$ is the keyword, $\mathcal{O}^K_{i,C}$ and $\mathbb{E}^K_{i,C}$ are the observed and the expected value for $K$ of the contingency table, respectively, and $i$ and $C$ index the occurrence of the term and the usefulness of a document, respectively. Yates's correction alleviates the upward bias of Pearson's $\chi^2$ test in 2 × 2 contingency tables with low observations.
- *Ripper* (Agichtein & Gravano, 2003) uses the Ripper algorithm (Cohen, 1995) to generate classification rules consisting of combinations of words that define useful documents. The algorithm in Agichtein and Gravano (2003) then transforms the rules into Boolean conjunctive queries. For example, the rule <"vortex" AND "wind" ⇒ useful> is transformed into the Boolean conjunctive query [vortex AND wind]. A query is scored with its expected precision, defined as the ratio of useful documents to the total of documents in the training set that match its original rule.
- *Significant phrases* (SP) (Boden, Löser, Nagel, & Pieper, 2011, 2012) learns the most frequently collocated pairs of words (Dunning, 1993) from the useful documents and reports them as phrase queries. For example, for the *Occurs* $-$ *in* relation, SP produces queries like ["richter scale"] and ["snow storm"]. SP scores each phrase with the Pearson's $\chi^2$ value computed over its keywords, which indicates how independent its keywords are from one another. To guarantee that the queries (i.e., collocated pairs of words) are real phrases in the document sample, we generate all phrases and remove those that do not comply with the candidate set of keywords at hand: (i) for explicit, we remove phrases with only stopwords, rare, or frequent words; (ii) for implicit, we also remove phrases that include words in the attribute values.

[11] http://dev.mysql.com/doc/refman/5.7/en/fulltext-stopwords.html.

**Table 3**
Parameter setting for filtering conditions. The parameters correspond to: number of queries ($N$), round precision threshold ($\tau_r$), number of documents ($M$), and query precision threshold ($\tau_q$).

| Technique | Filtering condition | | | |
|---|---|---|---|---|
| | (1) | | (2) | |
| | $N$ | $\tau_r$ | $M$ | $\tau_q$ |
| F-QXtract | 75 | 0.15 | 150 | 0.05 |
| F-Cyclic | 75 | 0.15 | 150 | 0.05 |
| F-Opportunistic | 75 | 0.15 | 150 | 0.05 |
| F-Balanced | 75 | 0.15 | 50 | 0.05 |

We used Weka 3.6[12] with default settings to implement SVM (SequentialMinimalOptimization), IG (InfoGainAttributeEval), $\chi^2$ (ChiSquaredAttributeEval) with Yates correction, and Ripper (JRip). To implement SP, we used the significant phrases implementation of LingPipe[13] with default settings, as suggested in Boden et al. (2011).

**Sampling techniques:** We evaluate the techniques described in Section 3 and the bootstrapping-based techniques described above. For QXtract, we retrieve and process 1000 documents per query, while we consider different numbers of retrieved documents for Cyclic, Balanced, and Opportunistic. Also, for reference, we compare a sampling technique that prioritizes less-effective queries from the ground up (i.e., without previously assessing the real effectiveness of queries). Specifically, this technique, which we refer to as *Reverse*, proceeds as QXtract (see Section 3), although processing top-$Q$ queries in ascending score order. We use different values of $Q$ in our experiments.

**Filtering conditions:** We rely on two filtering conditions, which correspond to the filtering alternatives described in Section 3. The first filtering condition stops processing queries based on the performance of the latest $N$ queries that were issued. Specifically, we stop processing queries when, out of these $N$ queries, the fraction of queries that retrieve at least one useful document is below a certain threshold $\tau_r$ (see (1) in Table 3). The main impact of this filtering condition occurs during the first query round because, as discussed, queries are initially issued according to their effectiveness. The second filtering condition stops processing queries based on their actual performance, as follows: We stop processing a query $q$ if its effectiveness computed over the last $M$ retrieved documents (i.e., the precision@$M$ of the query) is below a certain threshold $\tau_q$ (see (2) in Table 3). We evaluated different values for the parameters in these conditions: We varied $N \in [10, 100]$, $\tau_r \in [0.02, 0.25]$, $M \in [5, 50]$, $\tau_q \in [0.05, 0.25]$. Finally, we kept for each strategy the settings that collected on average the largest and highest-quality samples for the same sampling cost. We summarize these settings in Table 3.

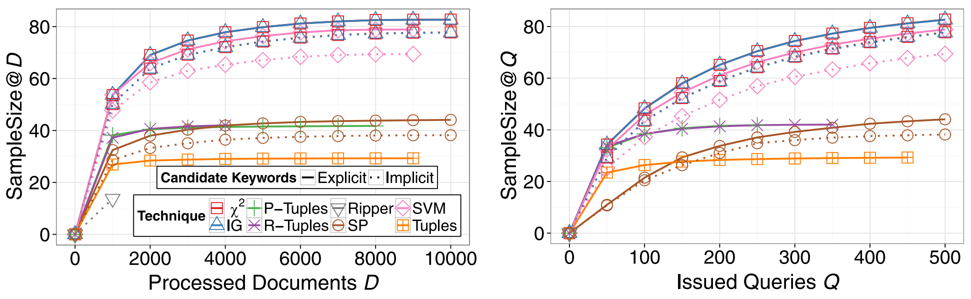
**Sampling execution and termination:** Given an information extraction system and a collection, the output document sample includes all useful documents for the extraction task that are found during the execution of the sampling process. We let each sampling execution issue up to 500 unique queries and process up to 10,000 unique documents with the information extraction system at hand, to keep the sampling cost to reasonable levels. We also terminate the sampling process after collecting 400 useful documents. According to the results over our tuning collections, conclusions are analogous for larger sample sizes.

**Performance metrics:** We use the following metrics:

- SampleSize@$Q$ and SampleSize@$D$ measure the size of the document sample (i.e., number of useful documents in the sample) as a function of the number of issued queries $Q$ and of the number of processed documents $D$, respectively. We do not report $S$ or $D$ as a percentage of the total number of documents in the deep-web text collection being sampled (e.g., 50% of the documents), since we are unaware of the real size of the collection.
- UniqueTuples@$S$, UniqueTuples@$Q$, and UniqueTuples@$D$ measure the quality of the sampling process in terms of the number of unique tuples and attributes as a function of sample size $S$, issued queries $Q$, and processed documents $D$, respectively. Specifically, we compute the number of unique tuples using case-insensitive string matching over each attribute.
- IssuedQueries@$S$ and ProcessedDocuments@$S$ measure the number of queries issued and documents processed, respectively, to collect a sample of size $S$. Given a technique and a sample size $S$, we only report IssuedQueries@$S$ and ProcessedDocuments@$S$ if the technique collects at least one sample of size $S$. Because not all sampling executions manage to collect document samples of all sizes, we complement these measures with the fraction of collections that the technique collects samples of size $S$ from, which we define next. Some sampling techniques may not reach all sample sizes $S$ for three main reasons: (i) collections may include (very) few useful documents for some relations; (ii) techniques that rely on filtering conditions may terminate the sampling process early; and (iii) only a limited number of issued queries and processed documents may be allowed, for efficiency.

[12] http://www.cs.waikato.ac.nz/ml/weka/.

[13] http://alias-i.com/lingpipe/.



(a) SampleSize@$D$ (b) SampleSize@$Q$

**Fig. 5.** Sample size for different useful document retrieval strategies, processing 50 documents per query and for the Person–Career relation. (P-Tuples and R-Tuples refer to the Career and Person attributes, respectively.)

- Coverage@$S$ measures the fraction of deep-web text collections from which the sampling process manages to collect samples of size $S$. We do not report Coverage@$S$ as a fraction of the ideal coverage, since we are unaware of the real contents of the collections. We evaluate Coverage@$S$ as a complementary measure to those defined above.

We run all sampling processes five times, to account for randomness, as follows: For bootstrapping-based techniques, each run uses a different initial set of seed of 20 tuples. For learning-based techniques, we built 5 disjoint training document samples from our training collection, each with 5000 randomly picked useful documents—or the maximum number of useful documents available for each training sample—and the same number of useless documents, so that the training samples are balanced. For example, for the Election–Winner relation and using the SSK extraction system, each training document sample included 499 useful documents, because there were 2494 useful documents in the training collection. (Other seed tuples and training sample sizes yielded similar results during tuning.) Given a collection, we finally report the average over all executions using the same sampling configuration.

## 5. Experimental results

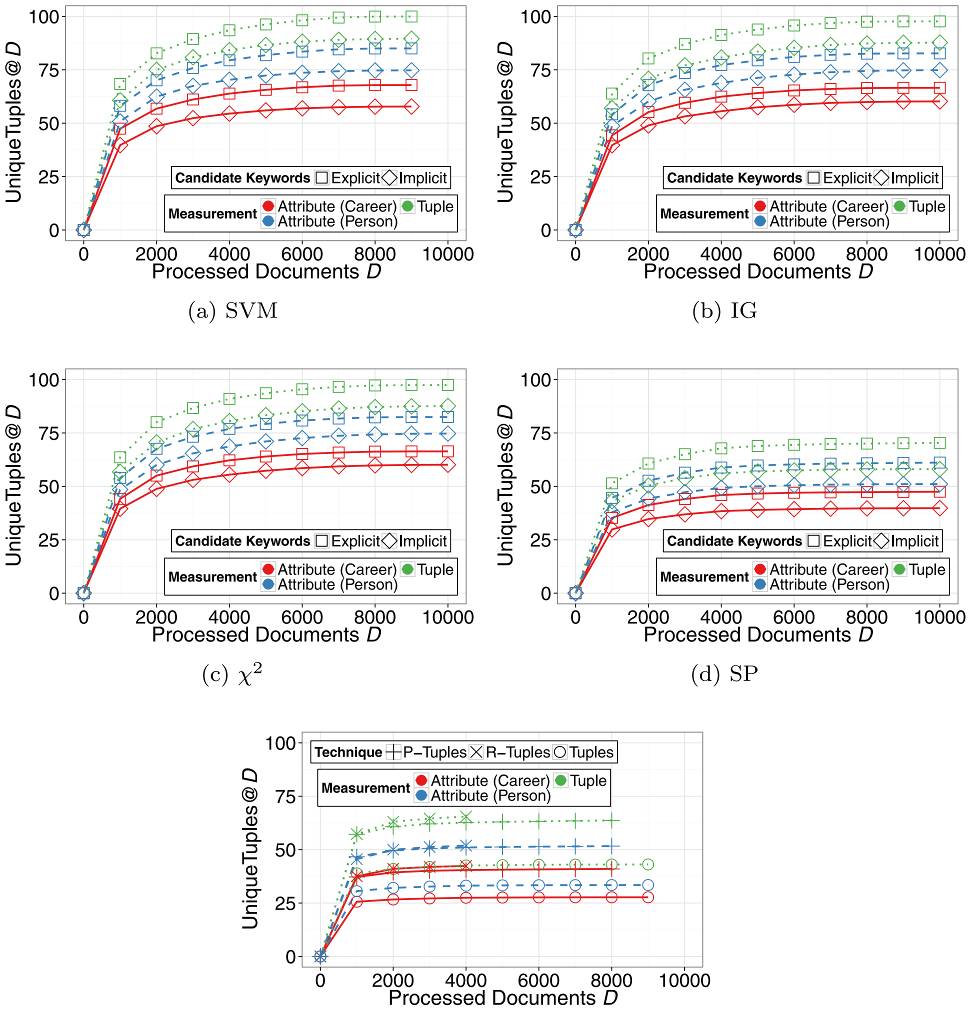
We now report our experimental results: We start by evaluating different families of useful document retrieval techniques (Section 5.1). We then evaluate different query execution schedules (Section 5.2) and document retrieval and processing strategies (Section 5.3). Finally, we evaluate the impact of revising the query execution order (Section 5.4) and of filtering underperforming queries (Section 5.5).

### *5.1.* Impact of useful document retrieval

We evaluate the two document retrieval strategies in Section 4, from the bootstrapping- and learning-based families discussed in Section 2.

**Efficiency analysis:** We first evaluate efficiency by considering sample size: Fig. 5 shows SampleSize@$D$ (Fig. 5(a)) and SampleSize@$Q$ (Fig. 5(b)) for different document retrieval strategies and processing the top-50 documents per query, for the Person–Career relation. (Other relations as well as number of documents per query yielded analogous conclusions.) As shown, learning-based techniques that employ keyword selection, namely, SVM, IG, and $\chi^2$, consistently outperform other techniques after processing 1000 documents and issuing 100 queries. The differences were statistically significant ($t$-test, $p < 0.001$) for all comparisons between SVM, IG, and $\chi^2$ and the rest. These techniques sample on average 100% more documents than other techniques for the same document processing and querying costs: These methods select popular—yet discriminative—keywords that are evenly distributed across useful and useless documents (Forman, 2003), thus improving the recall of the sampling process. For small numbers of processed documents and issued queries, bootstrapping-based techniques are comparable to keyword selection-based techniques, because the top queries from learning-based methods tend to overlap with those from bootstrapping-based techniques. This finding corroborates that of previous studies for the related problem of efficiently running an extraction process over a large text collection (e.g., Agichtein & Gravano, 2003), which state that bootstrapping-based techniques are rather high-precision. The results in Fig. 5(b) for issued queries, in particular, also provide empirical evidence of the analysis in Lu and Li (2010) and Ipeirotis et al. (2007): the number of retrieved—and sampled—useful documents decreases as queries are issued, due to overlap in the successive query results and decreasing effectiveness of the queries.

The choice of candidate set of keywords also affects sampling efficiency, as shown in Fig. 5: The explicit candidate set of keywords, which includes values of tuple attributes in the learned queries (see Section 4), targets useful documents more effectively than its implicit counterpart. The differences were statistically significant for all comparisons ($t$-test, $p < 0.2$) and more prominently for SVM ($t$-test, $p < 0.001$). (We observed analogous conclusions for all relations, with the exception of Natural Disaster–Location, which we study in detail later.) This result differs from that in Agichtein and Gravano (2003),



(a) SVM

(b) IG

(c) $\chi^2$

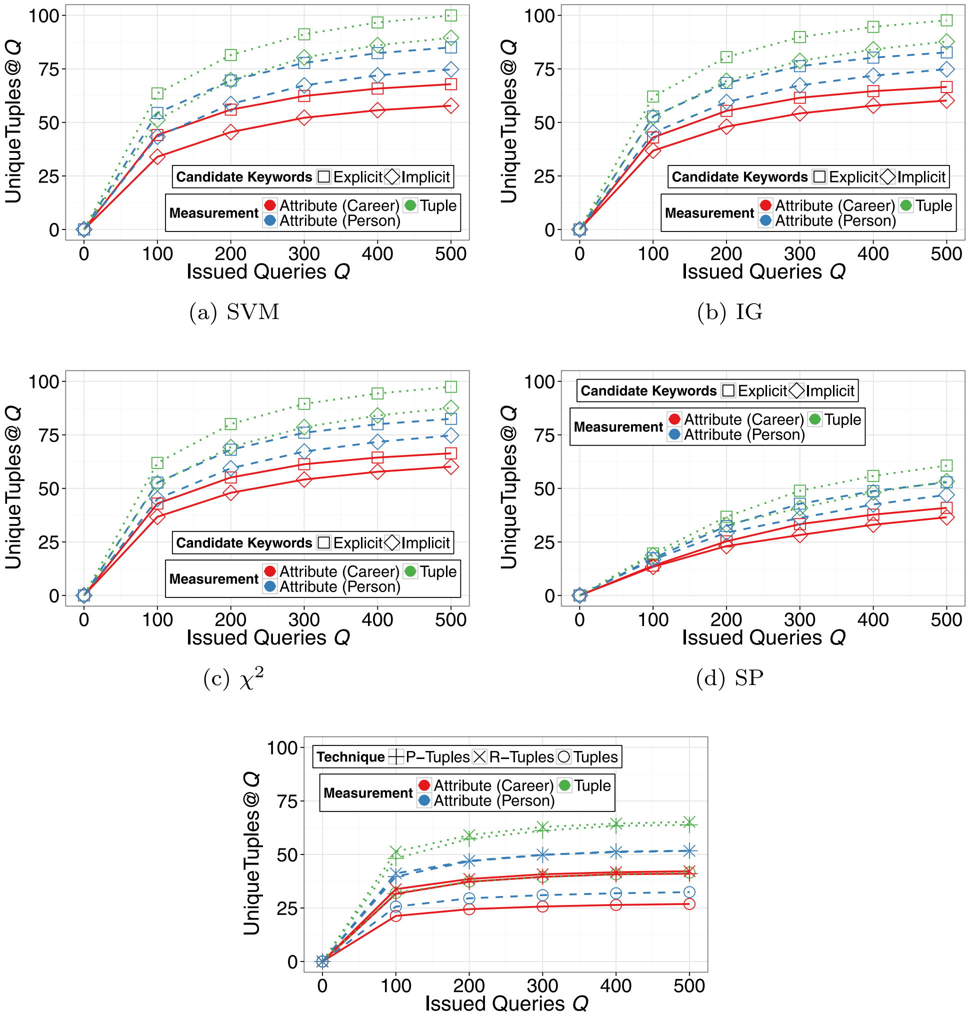
(d) SP

(e) All bootstrapping-based variants

**Fig. 6.** UniqueTuples@D for different useful document retrieval strategies, processing 50 documents per query and for the Person–Career relation. (P-Tuples and R-Tuples refer to the Career and Person attributes, respectively.)

where the implicit set of keywords (almost) always performed the best. We observe the largest performance gap for SVM, which gave considerably high weights to rare—yet discriminative—keywords in the training documents using the implicit candidate set of keywords. These keywords were in turn also rare in our test text collections. This finding corresponds with that of previous studies (e.g., see Chen, Lam, Tsang, & Wong, 2010) that conclude that SVMs are many times unable to generalize to other datasets.

**Quality analysis:** To evaluate the quality of the samples produced with different document retrieval strategies, we measured the number of unique tuples. Fig. 7 shows UniqueTuples@*D* (Fig. 6) and UniqueTuples@Q (Fig. 7), processing top-50 documents per query for the Person–Career relation. (We do not include Ripper since the number of processed documents and issued queries is small, as shown in Fig. 5. Other relations and values for the number of documents per query yielded similar conclusions.) Our first observation is that the most efficient techniques also exhibit the highest quality: For the same document processing and querying cost, these (efficient) techniques collect a larger number of tuples, which in effect include a higher number of unique tuples. The differences were statistically significant ($t$-test, $p < 0.001$) for all comparisons between SVM, IG, and $\chi^2$ and the rest after processing 1500 documents and issuing 50 queries. For bootstrapping-based techniques, in particular, the quality positively correlates with the domain of attributes (e.g., names of people, careers) used as queries. In the Person–Career relation, for instance, there are more people names than careers; as a result, we observe the highest quality for R–Tuples, which derives queries from the Person attribute. Unfortunately, the quality of bootstrapping-based techniques is low compared with that of $\chi^2$ and other learning-based techniques. As shown, these techniques reach their highest quality values early in the sampling process, which exhibits their quality limitations. This corroborates the find-



(a) SVM

(b) IG

(c) $\chi^2$

(d) SP

(e) All bootstrapping-based variants

**Fig. 7.** UniqueTuples@Q for different useful document retrieval strategies, processing 50 documents per query and for the Person–Career relation. (P-Tuples and R-Tuples refer to the Career and Person attributes, respectively.)

ing in Agichtein, Ipeirotis, and Gravano (2003), which states that bootstrapping-based approaches often only reach limited groups of documents—hence limited sampling quality—in the collections.

**Coverage analysis:** We finally evaluate Coverage@*S* of the document retrieval strategies: Fig. 8 shows Coverage@*S* for the learning- and bootstrapping-based variants of interest. As shown, learning-based techniques using the explicit set of keywords exhibit the highest coverage across different sample sizes. Specifically, learning-based techniques manage to collect useful documents from 30% more collections than other techniques on average. For bootstrapping-based techniques, P-Tuples collects small samples (75 documents or fewer for Person–Career relation) from 10% and 20% more collections than R-Tuples and Tuples, respectively. For larger samples (100 documents or more, for the Person–Career relation), in contrast, R-Tuples manages to effectively collect samples from 25% and 40% more collections than P-Tuples and Tuples, respectively.

**Conclusion:** Based on the evaluation above, we conclude that learning-based techniques with keyword selection strategies perform the best for document sampling: They (i) collect useful documents efficiently (e.g., in terms of processed documents and issued queries); (ii) sample representative, high-quality documents for all attributes in the extraction task at hand; and (iii) manage to collect useful documents from more deep-web text collections than those of other techniques.

### 5.2. Impact of query execution order

In Section 3, we argued that different query execution orders have distinct implications in sampling efficiency and quality. We now evaluate the discussed query execution orders: We compare (i) QXtract (see Section 3), to assess the performance of prioritizing effective queries; and (ii) Reverse (see Section 4), to assess the performance of prioritizing less-effective queries.

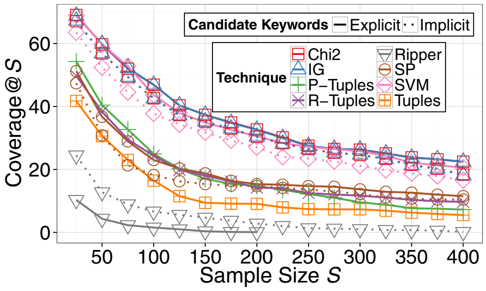
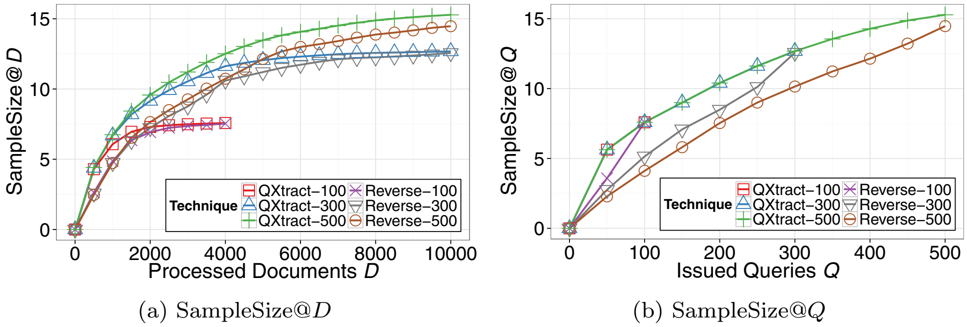


**Fig. 8.** Coverage@*S* for different useful document retrieval strategies for different sample sizes, processing 50 documents per query and for the Person–Career relation. (P-Tuples and R-Tuples refer to the Career and Person attributes, respectively.)



(a) SampleSize@$D$

(b) SampleSize@$Q$

**Fig. 9.** Sample size for different query execution orders and number of learned queries, processing 100 documents per query and for the Man Made Disaster–Location relation.

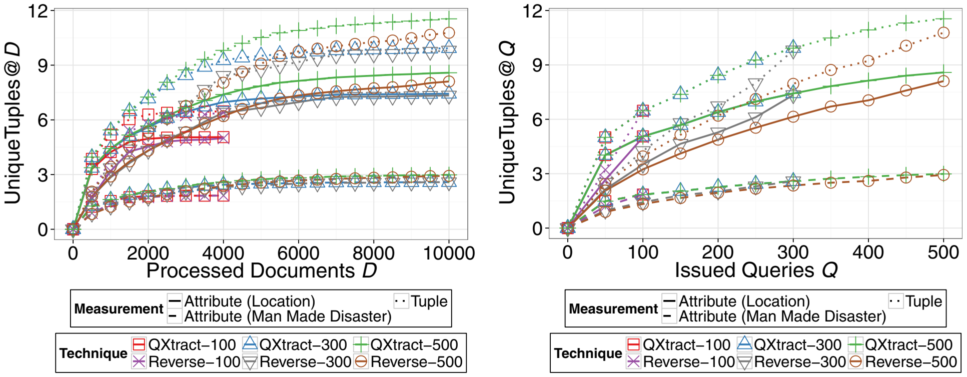
We report our evaluation using $\chi^2$ as query generation method and over the explicit candidate set of keywords, as it performed substantially better than other techniques and comparably to *IG* and *SVM*. We vary the number of (top) learned queries between 100 and 500.

**Efficiency analysis:** To assess the efficiency of different query execution orders, we evaluated QXtract and Reverse over all relations, and for different numbers of queries: Fig. 9 shows SampleSize@*D* (Fig. 9(a)) and SampleSize@Q (Fig. 9(b)) for different query execution orders and number of learned queries, for the Man Made Disaster–Location relation. (Other relations yielded analogous conclusions.) As shown, all versions of QXtract perform comparably to or better than the Reverse counterparts: For small values of the number of highly-effective queries (see QXtract-100 and Reverse-100 in Fig. 9), the query execution order has almost no impact on sampling efficiency, although the difference between QXtract-100 and Reverse-100 is statistically significant (*t*-test, $p < 0.001$) for the first 1500 processed documents and 75 issued queries. For large numbers of queries (see QXtract-500 and Reverse-500 in Fig. 9), the impact of the query execution order is considerable, with QXtract-500 collecting 100% more useful documents than Reverse-500. In this case, the difference between QXtract-500 and Reverse-500 is statistically significant (*t*-test, $p < 0.001$) for the first 2500 documents and 200 issued queries.

**Quality analysis:** Beyond efficiency, we also expect the query execution order to impact sampling quality. Fig. 10 shows UniqueTuples@*D* (Fig. 10(a)) and UniqueTuples@Q (Fig. 10(b)(b)), for different query execution orders and number of learned queries, and using the explicit candidate set of keywords over the Man Made Disaster–Location relation. (Other relations yielded analogous conclusions.) As shown, for the number of processed documents and issued queries, QXtract variants, which prioritize effective queries, collect a higher number of unique tuples and attributes. Similarly to our efficiency analysis above, differences were statistically significant (*t*-test, $p < 0.001$) for the same techniques and intervals. This happens because, as discussed above, effective queries lead to extracting more tuples—hence more unique tuples. However, we are also interested in the sampling quality of different query execution orders as a function of the sample size. This cannot be evaluated with UniqueTuples@Q and UniqueTuples@*D*, since we have different sample sizes across collections for the same values of Q and *D*.

To evaluate the intrinsic quality of different query execution orders, and to complement the quality analysis above, we evaluate sample quality across sample sizes. Fig. 11 shows UniqueTuples@*S* for different query execution orders, using the explicit candidate set of keywords, and over the Man Made Disaster–Location relation. As shown, for small sample sizes (100 sampled documents or fewer), Reverse variants exhibit sample quality at least as good as that of their QXtract counterparts. This also holds for sample sizes for which QXtract has collected more samples (see Sample Size=75 in Fig. 12).

**Coverage analysis:** We finally evaluate the coverage that different query execution orders exhibit. Fig. 12 shows Coverage@*S* for different query execution orders, using the Man Made Disaster–Location relation. (Other relations yielded analogous results.) Conclusions are manifold: Focusing on a small set of highly-effective queries drastically reduces the coverage



(a) UniqueTuples@*D* (b) UniqueTuples@*Q*

**Fig. 10.** Number of unique tuples for different query execution orders and number of learned queries, processing 100 documents per query and using the explicit candidate set of keywords and for the Man Made Disaster–Location relation.

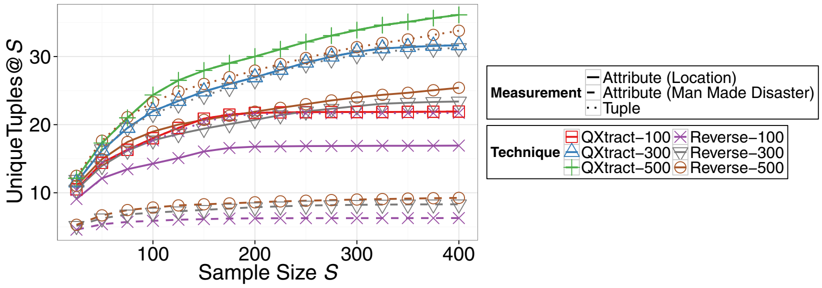


**Fig. 11.** UniqueTuples@*S* for different query execution orders and number of learned queries, processing 100 documents per query and using the explicit candidate set of keywords and for the Man Made Disaster–Location relation.

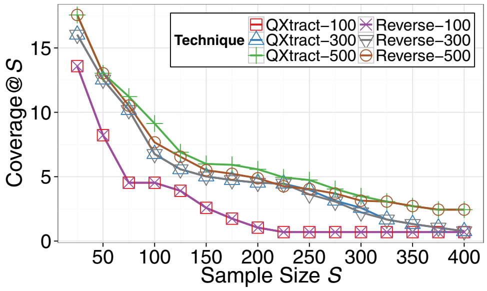


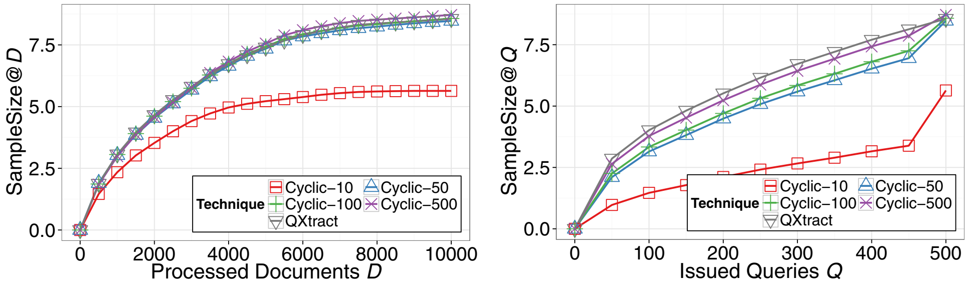
**Fig. 12.** Coverage@*S* for different query execution orders and number of learned queries for different sample sizes, processing 100 documents per query and for the Man Made Disaster–Location relation.

of the techniques for all sample sizes (see QXtract-100 and Reverse-100 in Fig. 12). More importantly, the query execution order does not affect the (poor) coverage in this case. Unlike what we expected, increasing the number of learned queries showed limited impact in coverage, while its querying overhead was considerable (see Fig. 9(b)).

**Conclusion:** We have empirically corroborated the efficiency and quality implications of different query execution orders: Prioritizing effective queries leads to more efficient sampling executions that, in turn, collect document samples from a larger number of collections than prioritizing less-effective queries. Prioritizing less-effective queries, in contrast, leads to high-quality document samples, but at a considerably high document processing and querying cost. Moreover, increasing the number of learned queries has limited impact.

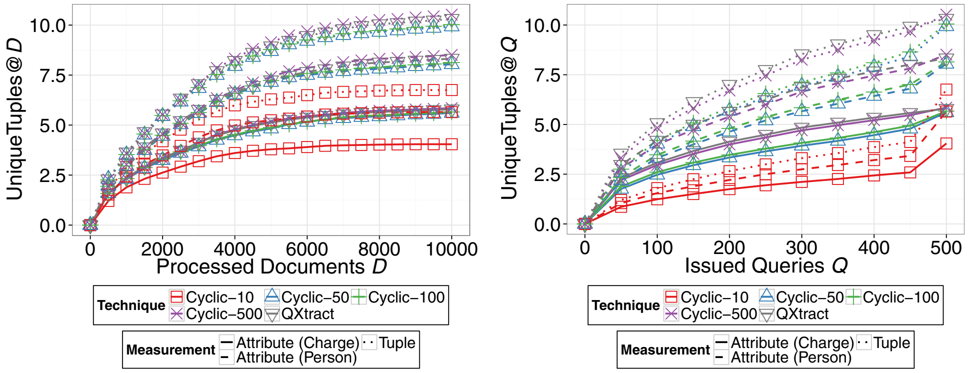
### *5.3.* Impact of document retrieval and processing

In addition to the query execution orders analyzed above, we also argued in Section 3 that different document retrieval and processing strategies also impact sampling efficiency and quality. We now compare: (i) QXtract, which retrieves and



(a) SampleSize@$D$ (b) SampleSize@$Q$

**Fig. 13.** Sample size for different document retrieval and processing strategies for the Person–Charge relation.



(a) UniqueTuples@$D$ (b) UniqueTuples@$Q$

**Fig. 14.** Number of unique tuples for different document retrieval and processing strategies, using the explicit candidate set of keywords and for the Person–Charge relation.

process documents exhaustively for each query; and (ii) Cyclic, which does so incrementally and in rounds. We report our evaluation using $\chi^2$ as our query generation method and over the explicit candidate set of keywords, as done in Section 5.2.

**Efficiency analysis:** We evaluate the efficiency of QXtract and Cyclic with different numbers of documents per round. Fig. 13 shows SampleSize@$D$ (Fig. 13(a)) and SampleSize@$Q$ (Fig. 13(b)) for different document retrieval and processing strategies, and using the Person–Charge relation. (Other relations yielded similar conclusions.) As shown, there is a positive correlation between the number of documents per round and the number of sampled useful documents: QXtract and Cyclic start with highly-effective queries, which are likely to retrieve useful documents with high precision and recall. This is better illustrated in Fig. 13(b), where QXtract consistently outperforms all variants of Cyclic. In terms of processed documents, though, the sampling process benefits from moving earlier to other queries (see Fig. 13(a)), because top queries may not be equally effective across collections. As a result, variants of Cyclic with rounds of 100 documents or more collect larger samples than QXtract, for the same number of processed documents. The differences were statistically significant ($t$-test, $p < 0.001$) along the sampling process between Cyclic-10 and all other techniques.

**Quality analysis:** Beyond efficiency, we also compared the quality of different document retrieval and processing strategies. Fig. 14 shows UniqueTuples@$D$ (Fig. 14(a)) and UniqueTuples@$Q$ (Fig. 14(b)) for different document retrieval and processing strategies, using the explicit candidate set of keywords and for the Person–Charge relation. (Other relations yielded analogous conclusions.) Surprisingly, low values of $k$ (e.g., $k = 10$) did not enhance sample quality: Even after processing 8000 documents with Cyclic-10, sampling quality was lower than that of other variants for only 4000 retrieved and processed documents. Similarly to what we observed above in our efficiency analysis, the differences were statistically significant ($t$-test, $p < 0.001$) along the sampling process between Cyclic-10 and all other techniques. Conversely, and similarly to what we observed for document retrieval strategies (Section 5.1), the number of sampled documents correlates with quality.

**Coverage analysis:** Fig. 15 shows Coverage@$S$ for different document retrieval and processing strategies, for the Person–Charge relation. (Other relations yielded similar conclusions.) As shown, the most efficient techniques, namely, QXtract and variants of Cyclic with 100 or more documents per round, also exhibit the best coverage. Processing fewer documents per round tended to deploy querying and document processing effort on less–effective queries and useless documents, thus compromising the overall sampling performance (see Cyclic-10 in Fig. 15).

**Conclusion:** Based on the evaluation above, techniques that focus on effective queries, namely, QXtract and variants of Cyclic with 100 or more documents per round, outperformed other configurations. In particular, although these techniques

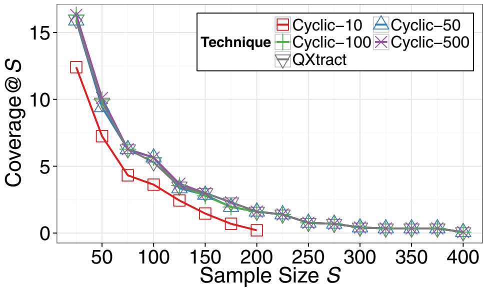


**Fig. 15.** Coverage@*S* for different document retrieval and processing strategies for the Person–Charge relation.

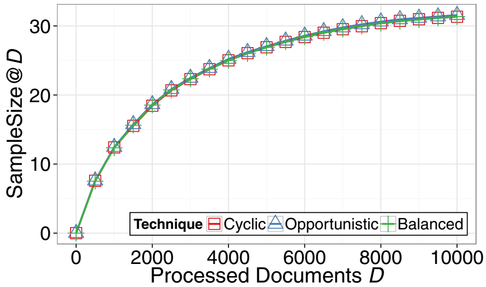


**Fig. 16.** SampleSize@*D* for different query execution schedules and processing 50 documents per round for the Natural Disaster–Location relation.

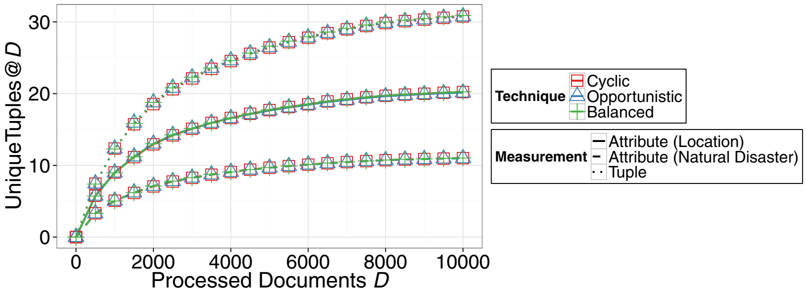


**Fig. 17.** UniqueTuples@*D* for different query execution schedules, processing 50 documents per round, using the implicit candidate set of keywords and for the Natural Disaster–Location relation.

perform comparably, QXtract is a better choice when querying cost dominates the sampling cost, while Cyclic prevails when document processing cost dominates sampling cost.

### *5.4.* Impact of revising query order

So far, our experimental evaluation is on the intrinsic performance of different query execution and document processing and retrieval strategies. However, as argued in Section 3, there is valuable information (e.g., the real, observed effectiveness of queries) that we can exploit along the sampling process. We now study the impact of using this information to revise the query execution order. We compare (i) Balanced and Opportunistic, which revise the order of the queries; and (ii) Cyclic, which maintains their original (learned) order along the sampling process. We report our evaluation using $\chi^2$ as our query generation method and over the implicit candidate set of keywords. Unlike in previous experiments, though, we only report the number of processed documents, as these techniques issue the same queries.

**Efficiency analysis:** We first evaluate the impact on sampling efficiency of revising the query order. Fig. 16 shows SampleSize@*D* for Cyclic, Opportunistic, and Balanced, processing 50 documents per round (i.e., $k = 50$) for the Natural Disaster–Location relation. (Other relations and values of $k$ yielded analogous conclusions.) From the techniques we evaluated, Opportunistic revises the query order to prioritize queries based on their real, observed effectiveness. As expected, Opportunistic exhibits the best sampling efficiency on average. However, none of the differences between techniques were statistically significant. Importantly, the improvement of Opportunistic over other techniques was more noticeable over collections with a large number of useful documents.

**Quality analysis:** We also evaluated the impact in sampling quality. Fig. 17 shows UniqueTuples@*D* for Cyclic, Opportunistic, and Balanced, processing 50 documents per round (i.e., $k = 50$), using the implicit candidate set of keywords, and

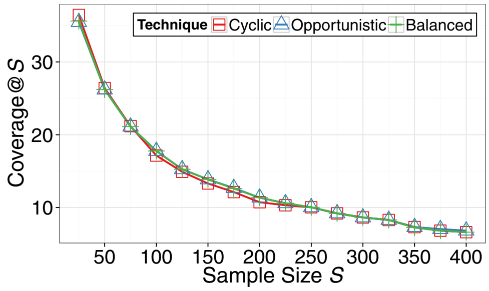


**Fig. 18.** Coverage@*S* for different query execution schedules, processing 50 documents per round and for the Natural Disaster–Location relation.

**Table 4**
ProcessedDocuments@*S* for filtered and unfiltered versions of QXtract and Cyclic (using $k = 50$), using the explicit candidate set of keywords and for the Election–Winner relation. Best techniques for each sample size are bolded.

| Technique | Sample size | | | | |
|---|---|---|---|---|---|
| | 25 | 50 | 100 | 200 | 400 |
| F-Cyclic | 1067.4 ± 261 | 1853.2 ± 53.4 | 3385.9 ± 517.6 | **5146.8** ± **1006.1** | - |
| Cyclic | 2374 ± 336.2 | 2804.5 ± 328.6 | 3517.9 ± 463.1 | 5457 ± 567.3 | **7126** ± **0** |
| F-QXtract | **975.1** ± **245.4** | **1761.5** ± **62.8** | **3266.4** ± **81.9** | 5193.5 ± 983.2 | - |
| QXtract | 1977.3 ± 134.4 | 2617.4 ± 466.4 | 3281.4 ± 838.7 | 5617.9 ± 776.3 | 7169.5 ± 0 |

**Table 5**
IssuedQueries@*S* for filtered and unfiltered versions of QXtract and Cyclic (using $k = 50$), using the explicit candidate set of keywords and for the Election–Winner relation. Best techniques for each sample size are bolded.

| Technique | Sample size | | | | |
|---|---|---|---|---|---|
| | 25 | 50 | 100 | 200 | 400 |
| F-Cyclic | **83.1** ± **12.9** | 128.9 ± 11.4 | 224.8 ± 24.8 | 306.5 ± 74.8 | - |
| Cyclic | 245.8 ± 19.2 | 316.4 ± 19.8 | 292.5 ± 48 | 374.7 ± 21.6 | 500 ± 0 |
| F-QXtract | 89.9 ± 23.9 | **125.7** ± **15.4** | 214.9 ± 11.3 | 305.6 ± 70.1 | - |
| QXtract | 119.9 ± 10.4 | 177 ± 22.3 | **201.1** ± **56.5** | **298.5** ± **28.2** | **485** ± **0** |

for the Natural Disaster–Location relation. (Other relations and values of *k* yielded similar conclusions.) As expected, Balanced exhibits the best sampling quality for all attributes, even when Opportunistic collected more useful documents (see efficiency analysis above). However, none of the differences between techniques were statistically significant. More importantly, and similarly to what we pointed out above, the impact on quality of Balanced is generally more noticeable over collections that include large numbers of useful documents. These collections tend to return many useful documents also for less-effective queries; therefore, these queries effectively enhance sampling quality when prioritized.

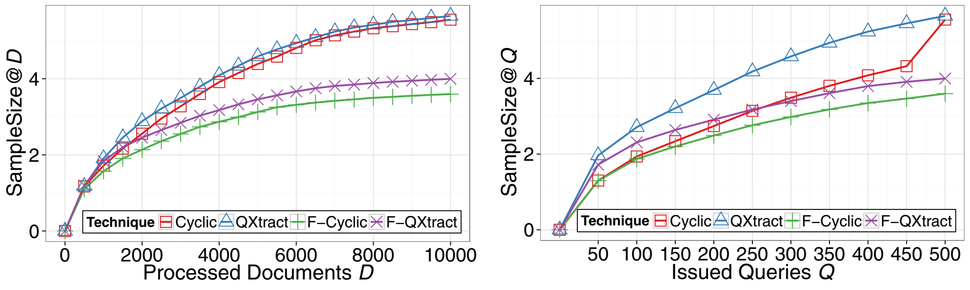
**Coverage analysis:** Finally, we evaluate the impact on coverage of revising query execution order. Fig. 18 shows Coverage@*S* for different sample sizes for Cyclic, Opportunistic, and Balanced, processing 50 documents per round (e.g., $k = 50$) and for the Natural Disaster–Location relation. These techniques exhibit similar coverage: Prioritizing less-effective queries based on their real, observed performance (e.g., in Balanced) does not impact the fraction of collections from which we can collect samples of different sizes.

**Conclusion:** Based on the evaluation above, we corroborated that we can further improve sampling efficiency and quality by accounting for the real, observed effectiveness of the queries. Although all techniques performed similarly, on average, Opportunistic and Balanced exhibited, respectively, the best sampling efficiency and quality, with noticeable effects on collections with large numbers of useful documents.

### *5.5.* Impact of filtering underperforming queries

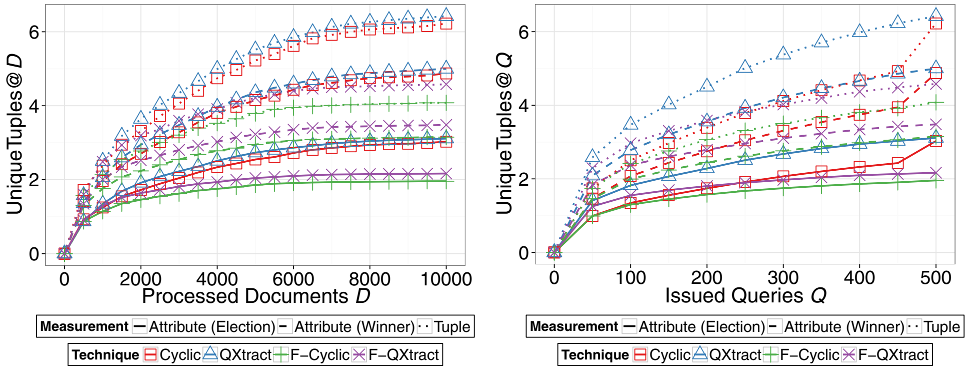
Our last experiment involves assessing the impact of filtering underperforming queries, which, as discussed in Section 3, can improve sampling efficiency. We compare (i) Cyclic and QXtract, which issue and process all queries; and (ii) F-Cyclic and F-QXtract, their filtered counterparts, which filter underperforming queries using the settings in Section 4. Conclusions were analogous for different techniques. We report our evaluation using $\chi^2$ as our query generation method and over the explicit candidate set of keywords.

**Efficiency analysis:** We first evaluate ProcessedDocuments@*S* and IssuedQueries@*S* for different sample sizes, which we show in Tables 4 and 5, respectively. (We later analyze the coverage of these techniques, which explains why, for instance,



(a) SampleSize@$D$ (b) SampleSize@$Q$

**Fig. 19.** Sample size for filtered and unfiltered versions of Cyclic (using $k = 50$) and QXtract for the Election–Winner relation.



(a) UniqueTuples@$D$ (b) UniqueTuples@$Q$

**Fig. 20.** Number of unique tuples for filtered and unfiltered versions of Cyclic (using $k = 50$) and QXtract, using the explicit candidate set of keywords and for the Election–Winner relation.

samples of 100 documents for Cyclic are on average less expensive than those of 50 documents.) As shown, filtered versions collect samples more efficiently than their unfiltered counterparts. For example, F-Cyclic processes 35% fewer documents and issues 55% fewer queries than Cyclic to collect samples of 50 useful documents. The differences between filtered and unfiltered versions of QXtract and Cyclic were statistically significant ($t$-test, $p < 0.001$) after processing 2500 documents and issuing 150 queries. The main benefit of the filtered versions is that they stop processing collections that include none—or insufficiently many—useful documents, which are a large portion of the collections. Overall, F-QXtract exhibits the best sampling efficiency across different sampling sizes; however, as we will see next, filtering underperforming queries has undesirable effects on all other relevant aspects of the sampling process.

In addition to the evaluation above, we study the impact of filtering underperforming queries on the sample size that we collect at different sampling costs. Fig. 19 shows SampleSize@$D$ (Fig. 19(a)) and SampleSize@$Q$ (Fig. 19(b)) for both the filtered and unfiltered versions of Cyclic, processing 50 document per round (i.e., $k = 50$), and QXtract, for the Election–Winner relation. (Other relations yielded similar conclusions.) As shown, filtered versions collect on average smaller sample sizes for the same cost, because they (mistakenly) stop processing queries that would retrieve useful documents otherwise: QXtract and Cyclic collect samples on average 100% larger than those of F-Cyclic and F-QXtract, respectively, for the same number of processed documents and issued queries.

**Quality analysis:** In Section 3, we argued that filtering certain queries has implications for sampling quality, because the sampling process only focuses on highly-effective queries. To evaluate their real impact, we compared Cyclic, processing 50 documents per query, and QXtract to their filtered counterparts: Fig. 20 shows UniqueTuples@$D$ (Fig. 20(a)) and UniqueTuples@$Q$ (Fig. 20(b)), for Cyclic, QXtract, F-Cyclic, and F-QXtract, using the explicit candidate set of keywords and for the Election–Winner relation. (Other relations yielded analogous conclusions.) As expected, filtering underperforming queries impacts sampling quality, because less-effective queries that potentially retrieve different groups of documents may not be processed. Also, and similarly to what we observed above, the techniques that collected more useful documents for the same document processing and query issuing cost also exhibit the best sample quality, for all tuple attributes. Finally, the differences between filtered and unfiltered versions of QXtract and Cyclic were statistically significant ($t$-test, $p < 0.001$) after processing 5000 documents and issuing 250 queries.

**Coverage analysis:** We finally evaluate how filtering underperforming queries impacts the coverage of the sampling techniques. Fig. 21 shows Coverage@$S$ for different sample sizes for Cyclic, QXtract, F-Cyclic, and F-Qxtract, and for the Election–

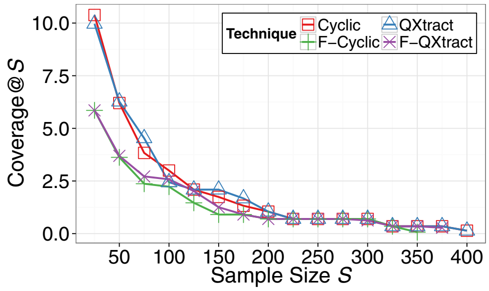


**Fig. 21.** Coverage@*S* for filtered and unfiltered versions of Cyclic (using $k = 50$) and QXtract for the Election–Winner relation.

Winner relation. (Other relations yielded similar conclusions.) We identify two regions in this figure worth analyzing. For small samples (e.g., 200 useful documents or fewer), QXtract and Cyclic consistently cover more collections than their filtered counterparts: Filtered technique rarely reach less-effective queries. For large samples (e.g., 200 documents or more), filtered and unfiltered techniques perform similarly: Filtering conditions do not affect the (typically) few collections that include large numbers of useful documents; instead, they effectively stop processing underperforming queries and focus on the rest.

**Conclusion:** Based on the evaluation above, we corroborate that filtering conditions help improve the efficiency of the sampling process, but affect other relevant aspects of the sampling process. We observed that the impact of the filtering step depends on the number of useful documents in the collections: Filtered techniques are as effective as their unfiltered counterparts over collections with large numbers of useful documents, while they tend to affect collections with only a small number of useful documents considerably.

## 6. Related work

We described relevant related work on document sampling over text collections for information extraction in Section 2, and we experimentally evaluated these techniques in Sections 4 and 5. Beyond these topics, another relevant area of related work is *focused crawling* (Chakrabarti, van den Berg, & Dom, 1999). In contrast to traditional exhaustive Web crawling (Olston & Najork, 2010), focused crawling aims to selectively discover Web pages on a specific set of topics. Focused crawling efforts (e.g., Diligenti, Coetzee, Lawrence, Giles, & Gori, 2000; Menczer, Pant, & Srinivasan, 2004; Pant & Srinivasan, 2006; Shchekotykhin, Jannach, & Friedrich, 2010) have often been devised for the crawlable Web, exploiting properties such as link structure that are nonexistent on the deep web. More recently, however, several focused crawling approaches (e.g., He, Xin, Ganti, Rajaraman, & Shah, 2013; Liakos, Ntoulas, Labrinidis, & Delis, 2016) have been proposed for the deep web. Similarly to our problem of interest, these strategies issue queries that are related to the topic at hand. The sampling strategies that we propose in this paper are complementary to those proposed for focused crawling over the deep web in two main ways. First, the document samples that our strategies produce can serve as input for the generation of topic-specific queries. Second, the different query execution and document retrieval and processing schedules that we discuss in Section 3, and that we evaluate in Section 5, can lead to fundamentally different (e.g., in terms of quality and efficiency) focused crawling executions. Importantly, our sampling strategies are crucial for other important building blocks of deep-web crawling, in general, namely, automatic filling of search forms (Kantorski, Moreira, & Heuser, 2015) and content monitoring (Khelghati, 2016), since they require high-quality and efficient document samples from the collection to select which queries to issue and to decide when to update the content summary of the collection, respectively.

Other more general approaches to document sampling over text collections (e.g., Bar-Yossef & Gurevich, 2008; Callan & Connell, 2001; Zhang et al., 2011, 2013) aim at collecting random samples from a text collection. Notably, the approaches in Bar-Yossef and Gurevich (2008); Callan and Connell (2001); Zhang et al. (2011) derive a large pool of queries from an external text collection (e.g., all *n*-grams in the external collection) that potentially reach all documents of interest in the collections. The approach in Zhang, Zhang, and Das (2013), on the other hand, generates these queries “on the fly,” as it retrieves documents from the collection in a random-walk fashion. To sample documents, these approaches pick queries (e.g., from the pool or from the document it retrieves), issue them to the collection at hand, and pick documents from the retrieved documents. Unfortunately, to effectively represent the (rather rare) useful documents in a collection, these approaches would require issuing an exorbitant number of queries. For a given (sub)population of interest (e.g., documents about sports), the approach in Zhang et al. (2011) proposes identifying queries that are positively correlated with this population (e.g., query “golf”) to, in turn, stratify the sampling process over correlated and uncorrelated queries. Unfortunately, this approach still requires issuing a large number of queries.

Stratified sampling (Särndal, Swensson, & Wretman, 1992) is often used to collect samples from subpopulations in the data. Specifically, stratified sampling separates these subpopulations into non-overlapping strata from which we can in turn sample independently. Existing approaches for efficiently running an information extraction system over a large text collection (e.g., QXtract (Agichtein & Gravano, 2003), FactCrawl (Boden et al., 2012), PRDualRank (Fang & Chang, 2011), and

BAgg-IE and RSVM-IE (Barrio et al., 2015b)) often require such stratification, to learn discriminative queries that retrieve useful documents: One stratum consists of useful documents, which we can collect using the techniques studied in this paper, while the other stratum consists of (rather frequent) useless documents, which we can obtain from a random sample (e.g., by using (Callan & Connell, 2001), as suggested in Agichtein and Gravano (2003)).

## 7. Conclusions

In this paper, we systematically studied the problem of sample generation for information extraction over the deep web. We considered (i) alternative query execution schedules, which vary on how they account for the query effectiveness, and (ii) alternative document retrieval and processing schedules, which vary on how they deploy the extraction effort over documents. Our large-scale evaluation, the first to the best of our knowledge, yielded several important conclusions: (i) schedules that focus on effective queries improve sampling efficiency, while schedules that prioritize less-effective queries favor quality; and (ii) processing the documents of highly-effective queries exhaustively consistently exhibits high sampling efficiency, but processing documents incrementally and in rounds can many times (e.g., with round sizes of 100 documents or more) exhibit better sampling efficiency and quality. We also evaluated several different useful document retrieval methods: Learned keyword queries performed substantially better than queries derived from tuples, which have been widely used in the existing literature. We also evaluated the implications of revising the order of the queries and of filtering underperforming queries. Revising query order during sampling helps improve sampling efficiency—when effective queries are prioritized in each round—and quality—when less-effective queries are prioritized instead. Also, filtering underperforming queries improves sampling efficiency considerably, although it compromises all other relevant aspects of the sampling process. Putting it all together, our study showed sampling configurations that can produce better-quality document samples for information extraction, and with executions that are several times more efficient than those possible with the sampling techniques adopted in the literature. In conclusion, our results provide a roadmap for addressing this critically important building block for efficient, scalable information extraction.

## Acknowledgments

This material is based upon work supported by the National Science Foundation under grant IIS-08-11038 and by Bloomberg L.P. This work was also supported by the Intelligence Advanced Research Projects Activity (IARPA) via Department of Interior National Business Center (DoI/NBC) contract number D11PC20153. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DoI/NBC, or the U.S. Government.

## References

Agichtein, E., & Cucerzan, S. (2005). Predicting accuracy of extracting information from unstructured text collections. In *Proceedings of the fourteenth ACM international conference on information and knowledge management (CIKM '05)* (pp. 413–420).

Agichtein, E., & Gravano, L. (2003). Querying text databases for efficient information extraction. In *Proceedings of the nineteenth international conference on data engineering (ICDE '03)* (pp. 113–124).

Agichtein, E., Ipeirotis, P., & Gravano, L. (2003). Modeling query-based access to text databases. In *Proceedings of the sixth international workshop on the web and databases (webDB '03)* (pp. 87–92).

Barbosa, L., & Freire, J. (2010). Siphoning hidden-web data through keyword-based interfaces. *Journal on Information and Data Management, 1*(1), 133–144.

Barrio, P., Gravano, L., & Develder, C. (2015). Ranking deep web text collections for scalable information extraction. In *Proceedings of the twenty-fourth ACM international conference on information and knowledge management (CIKM '15)* (pp. 153–162).

Barrio, P., Simões, G., Galhardas, H., & Gravano, L. (2014). REEL: A relation extraction learning framework. In *Proceedings of the 2014 ACM joint conference on digital libraries (JCDL '14)* (pp. 455–456).

Barrio, P., Simões, G., Galhardas, H., & Gravano, L. (2015). Learning to rank adaptively for scalable information extraction. In *Proceedings of the 2015 international conference on extending database technology (EDBT '15)* (pp. 241–252).

Bar-Yossef, Z., & Gurevich, M. (2008). Random sampling from a search engine's index. *Journal of the ACM, 55*(5), 1–74.

Bergman, M. K. (2001). The deep web: Surfacing, hidden value. *Journal of Electronic Publishing, 7*(1).

Boden, C., Löser, A., Nagel, C., & Pieper, S. (2011). Factcrawl: A fact retrieval framework for full-text indices. In *Proceedings of the fourteenth international workshop on the web and databases (webDB '11)*.

Boden, C., Löser, A., Nagel, C., & Pieper, S. (2012). Fact-aware document retrieval for information extraction. *Datenbank-Spektrum, 12*(2), 89–100.

Bunescu, R. C., & Mooney, R. J. (2005). Subsequence kernels for relation extraction. In *Proceedings of the nineteenth international conference on neural information processing systems (NIPS '05)* (pp. 171–178).

Callan, J. P., & Connell, M. (2001). Query-based sampling of text databases. *ACM Transactions on Information Systems, 19*(2), 97–130.

Chakrabarti, S., van den Berg, M., & Dom, B. (1999). Focused crawling: A new approach to topic-specific web resource discovery. *Computer Networks: The International Journal of Computer and Telecommunications Networking, 31*(11–16), 1623–1640.

Chen, B., Lam, W., Tsang, I. W., & Wong, T.-L. (2010). Location and scatter matching for dataset shift in text mining. In *Proceedings of the tenth IEEE international conference on data mining (ICDM '10)* (pp. 773–778).

Cohen, W. W. (1995). Fast effective rule induction. In *Proceedings of the twelfth international conference on machine learning (ICML '95)* (pp. 115–123).

Diligenti, M., Coetzee, F., Lawrence, S., Giles, C. L., & Gori, M. (2000). Focused crawling using context graphs. In *Proceedings of the 26th international conference on very large data bases (VLDB '00)* (pp. 527–534).

Dunning, T. (1993). Accurate methods for the statistics of surprise and coincidence. *Computational Linguistics, 19*(1), 61–74.

Fang, Y., & Chang, K. C.-C. (2011). Searching patterns for relation extraction over the web: Rediscovering the pattern-relation duality. In *Proceedings of the fourth ACM international conference on web search and data mining (WSDM '11)* (pp. 825–834).

Forman, G. (2003). An extensive empirical study of feature selection metrics for text classification. *The Journal of Machine Learning Research, 3*, 1289–1305.

Giuliano, C., Lavelli, A., & Romano, L. (2006). Exploiting shallow linguistic information for relation extraction from biomedical literature. In *Proceedings of the eleventh conference of the European chapter of the association for computational linguistics (EACL '06)* (pp. 3–7).

Gravano, L., Ipeirotis, P., & Sahami, M. (2003). QPRober: A system for automatic classification of hidden-web databases. *ACM Transactions on Information Systems, 21*(1), 1–41.

Gupta, S., & Bhatia, K. K. (2014). A comparative study of hidden web crawlers. *International Journal of Computer Trends and Technology, 12*(3), 111–118.

He, Y., Xin, D., Ganti, V., Rajaraman, S., & Shah, N. (2013). Crawling deep web entity pages. In *Proceedings of the sixth ACM international conference on web search and data mining (WSDM '13)* (pp. 355–364).

Ipeirotis, P., Agichtein, E., Jain, P., & Gravano, L. (2007). Towards a query optimizer for text-centric tasks. *ACM Transactions on Database Systems, 32*(4), 2–47.

Jain, A., Doan, A., & Gravano, L. (2008). Optimizing SQL queries over text databases. In *Proceedings of the 2008 IEEE 24th international conference on data engineering (ICDE '08)* (pp. 636–645).

Jain, A., & Ipeirotis, P. G. (2009). A quality-aware optimizer for information extraction. *ACM Transactions on Database Systems, 34*(1), 5:1–5:48.

Jain, A., Ipeirotis, P. G., Doan, A., & Gravano, L. (2009). Join optimization of information extraction output: quality matters!. In *Proceedings of the 2009 IEEE international conference on data engineering (ICDE '09)* (pp. 186–197).

Jain, A., & Srivastava, D. (2009). Exploring a few good tuples from text databases. In *Proceedings of the 2009 IEEE international conference on data engineering (ICDE '09)* (pp. 616–627).

Joachims, T. (1998). Making large-scale support vector machine learning practical. *Advances in kernel methods: Support vector machines*. Cambridge, MA: MIT Press.

Kantorski, G. Z., Moreira, V. P., & Heuser, C. A. (2015). Automatic filling of hidden web forms: A survey. *SIGMOD Record, 44*(1), 24–35.

Khelghati, M. (2016). Deep web content monitoring (Ph.D. thesis). University of Twente, Enschede, The Netherlands.

Kullback, S., & Leibler, R. A. (1951). On information and sufficiency. *The Annals of Mathematical Statistics, 22*(1), 79–86.

Liakos, P., Ntoulas, A., Labrinidis, A., & Delis, A. (2016). Focused crawling for the hidden web. *World Wide Web, 19*(4), 605–631.

Lu, J., & Li, D. (2010). Estimating deep web data source size by capture—recapture method. *Information Retrieval, 13*(1), 70–95.

McCallum, A., Freitag, D., & Pereira, F. C. N. (2000). Maximum entropy Markov models for information extraction and segmentation. In *Proceedings of the seventeenth international conference on machine learning (ICML '00)* (pp. 591–598).

McCallum, A., & Li, W. (2003). Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In *Proceedings of the ninth conference on computational natural language learning (coNLL '05)* (pp. 188–191).

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

Mladenic, D., Brank, J., Grobelnik, M., & Milic-Frayling, N. (2004). Feature selection using linear classifier weights: interaction with classification models. In *Proceedings of the twenty-seventh ACM international conference on research and development in information retrieval (SIGIR '04)* (pp. 234–241).

Ntoulas, A., Zerfos, P., & Cho, J. (2005). Downloading textual hidden web content through keyword queries. In *Proceedings of the 2005 ACM joint conference on digital libraries (JCDL '05)* (pp. 100–109).

Olston, C., & Najork, M. (2010). Web crawling. *Foundations and Trends in Information Retrieval, 4*(3), 175–246.

Pant, G., & Srinivasan, P. (2006). Link contexts in classifier-guided topical crawlers. *IEEE Transactions on Knowledge and Data Engineering, 18*(1), 107–122.

Pearson, K. (1900). On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that can be reasonably supposed to have arisen from random sampling. *Philosophical Magazine, 50*(302), 157–175.

Raghavan, S., & Garcia-Molina, H. (2001). Crawling the hidden web. In *Proceedings of the twenty-seventh international conference on very large databases (VLDB '01)* (pp. 129–138).

Särndal, C. E., Swensson, B., & Wretman, J. (1992). *Model assisted survey sampling*. Springer.

Shchekotykhin, K., Jannach, D., & Friedrich, G. (2010). Xcrawl: A high-recall crawling method for web mining. *Knowledge and Information Systems, 25*(2), 303–326. ISSN 0219–1377.

Sherman, C., & Price, G. (2003). The invisible web: Uncovering sources search engines can't see. *Library Trends, 52*(2), 282–298.

Simões, G., Galhardas, H., & Gravano, L. (2013). When speed has a price: Fast information extraction using approximate algorithms. *Proceedings of the VLDB Endowment, 6*(13), 1462–1473.

Tirado, J. M., Serban, O., Guo, Q., & Yoneki, E. (2016). Web data knowledge extraction. Technical report UCAM-CL-TR-881, University of Cambridge, Computer Laboratory. URL http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-881.pdf.

Vieira, K., Barbosa, L., Freire, J., & Silva, A. S. d. (2008). Siphon++: A hidden-web crawler for keyword-based interfaces. In *Proceedings of the seventeenth ACM international conference on information and knowledge management (CIKM '08)* (pp. 1361–1362).

Wang, Y., Li, Y., Pi, N., & Lu, J. (2015). Crawling ranked deep web data sources. In *Proceedings of the sixteenth international conference on web information systems engineering (WISE '15)* (pp. 384–398).

Wang, Y., Liang, J., & Lu, J. (2014). Estimating the size of hidden data sources by queries. In *Proceedings of the 2014 IEEE/ACM international conference on advances in social networks analysis and mining (ASONAM '14)* (pp. 712–719).

Wang, Y., Liang, J., & Lu, J. (2014). Discover hidden web properties by random walk on bipartite graph. *Information Retrieval, 17*(3), 203–228.

Yates, F. (1934). Contingency tables involving small numbers and the chi-square test. *Supplement to the Journal of the Royal Statistical Society, 1*(2), 217–235.

Zhang, M., Zhang, N., & Das, G. (2011). Mining a search engine's corpus: Efficient yet unbiased sampling and aggregate estimation. In *Proceedings of the 2011 ACM international conference on management of data (SIGMOD '11)* (pp. 793–804).

Zhang, M., Zhang, N., & Das, G. (2013). Mining a search engine's corpus without a query pool. In *Proceedings of the twenty-second ACM international conference on information and knowledge management (CIKM '13)* (pp. 29–38).

Zillman, M. P. (2008). Deep web research 2008. http://www.llrx.com/2007/11/deep-web-research-2008/ [Online; accessed 06 Nov 2016].