Prioritization of Domain-Specific Web Information Extraction

Jian Huang
Information Sciences and Technology
Pennsylvania State University
University Park, PA 16802, USA

Cong Yu
Yahoo! Research
New York, NY, USA

Abstract

It is often desirable to extract structured information from raw web pages for better information browsing, query answering, and pattern mining. Many such Information Extraction (IE) technologies are costly and applying them at the web-scale is impractical. In this paper, we propose a novel *prioritization approach* where candidate pages from the corpus are ordered according to their expected contribution to the extraction results and those with higher estimated potential are extracted earlier. Systems employing this approach can stop the extraction process at any time when the resource gets scarce (i.e., not all pages in the corpus can be processed), without worrying about wasting extraction effort on unimportant pages. More specifically, we define a novel notion to measure the value of extraction results and design various mechanisms for estimating a candidate page's contribution to this value. We further design and build the EXTRACTION PRIORITIZATION (EP) system with efficient scoring and scheduling algorithms, and experimentally demonstrate that EP significantly outperforms the naive approach and is more flexible than the classifier approach.

Introduction

A repository of structured information has many advantages over a collection of raw web pages. It allows users to issue queries with complex conditions (such as joins) and get direct answers (instead of pointers), as well as analyze the information for complex patterns. Although there are efforts in creating structured information from scratch through Wiki-style collaboration (Bollacker et al. 2008), the vast majority of information is still embedded within raw web pages. Hence, *information extraction* remains one of the most important and practical methods to obtain structured information on the Web.

However, extracting information from raw web pages is expensive: a page needs to be tokenized, tagged, matched against a non-trivial number of patterns and/or subject to inferencing through various machine learning models. Given the tremendous scale of the Web (tens of billions of pages nowadays), the full extraction approach, i.e., *performing extraction on every single page*, is prohibitively expensive and is limited to simple and light-weight extraction technologies only, such as Open-IE (Banko et al. 2007). As a result, many extraction systems are *domain-specific*, i.e., they build filters (essentially *classifiers*) to separate relevant web pages from irrelevant ones, and extract from the relevant ones only. This *classifier approach* works effectively for certain domains where relevant pages are concentrated and easy to determine (e.g., professional sports teams). However, it faces a difficult decision in many domains where good filters can not be easily designed (e.g., restaurant home pages): either making the filters restrictive at the expense of missing good results or making them loose and suffer a much higher computation cost for extraction. In this paper, we propose a novel *prioritization approach* that ranks each page instead of simply classifying them as relevant or not. Specifically, it identifies pages that are likely to yield the most value to the extraction results based on various factors, and prioritizes extraction on those pages within the resource constraints.

The main challenge of the prioritization approach is deciding which pages are more valuable than others and estimating those values. Consider the simple example

Web Pages:
P1: `joeshanghairestaurants.com/location.htm`
P2: `schmap.com/newyork/restaurants_french`
P3: `yelp.com/c/nyc/french`
P4: `yelp.com/nyc/fastfood`

Queries:
Q1: `joe's shanghai restaurant`
Q2: `joe's shanghai manhattan`
Q3: `french restaurants new york`
Q4: `chinese restaurants manhattan`

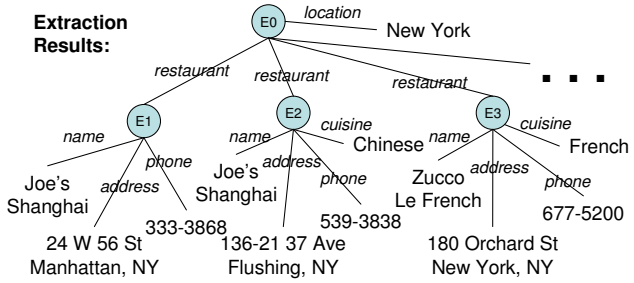

Figure 1: An example of a simple extraction scenario in the restaurant domain.

extraction scenario in Figure 1 in the restaurant domain. On the top are the web pages in the corpus and the set of queries posed by the users. On the bottom is the set of entities that can be extracted from the web pages: P1 contains E1 and E2 while E3 is contained within both P2 and P3. Intuitively, the value of a page with regard to the extraction task can be measured by several major factors. The first well-recognized factor is **relevance**: a page is only valuable if it contains information that is relevant to the domain of interest.

Relevance, however, fails to account for the impact of one particularly important external factor, *search demand*. For example, the restaurant "Joe's Shanghai" is more frequently requested by users than others (e.g., fast-food restaurants). Considering relevance only, a system that extracts many fast-food restaurants will be considered better than another system that only extracts the restaurant "Joe's Shanghai." However, to the users of this query workload, the latter is much more useful as the results it extracts can satisfy more of their needs. This observation leads us to the **importance** factor: a page is more valuable if it contains information that is needed by more users.

Furthermore, no extraction system can perfectly extract information from all the pages it processes. For example, it is possible that the web site `schmap.com` uses a lot of graphic content to convey its information, making it difficult to extract. So even if P2 contains important information (e.g., French restaurants), it is a waste of effort to extract from P2. We note this as the **extractibility** factor: a page is only valuable if the information it contains can be extracted by the underlying extraction system.

Finally, for web-scale extraction, a single piece of information is often extracted redundantly from multiple pages. For example, E3 can be extracted from both P2 and P3. Since redundant information is often wasteful (unless it helps to substantiate the correctness of the information already extracted), the last major factor, **novelty**, dictates that a page is more valuable if it contains more information that has not yet been extracted.

We formalize the notion of *page value* based on these main factors and describe how to estimate it effectively and efficiently. We design and implement a prototype system, called EP, capable of prioritizing pages for extraction based on the page value. We define the **extraction prioritization** problem as: given a corpus of candidates pages, determine the order by which those pages are to be extracted, at a reasonable additional overhead to the actual extraction, such that the value of the extraction results is maximized after each extraction iteration.

We make the following **main contributions**. First, we introduce a novel metric for the extraction results, *Consensus Graph Value*, which takes into account both relevance and importance of each extracted entity and relationship. Second, we formally define the notion of *Page Utility* for measuring a page's potential contribution to the extraction results and discuss how it can be estimated. Third, we design and build the EP system with efficient algorithms for scoring and scheduling candidate pages for extraction. Finally, we perform a comprehensive experimental evaluation of our approach using a real web data set.

The Value of Extraction Results

Consensus Graph is our logical data model for the extracted artifacts, i.e., entities and relationships, such as `restaurants` and `addresses`. Formally, we have:

Definition 1 (Consensus Graph) *A Consensus Graph (CG) is a 3-tuple,* $CG = (\mathsf{E}, \mathsf{A}, \mathsf{R})$*, where*

- E *is the set of typed entities, each entity* $\mathsf{e} \in \mathsf{E}$ *has a unique id within the CG and can be associated with multiple types.*
- A *is the set of atomic values of simple types like* String *or* Float.
- $\mathsf{R} = \mathsf{E} \times (\mathsf{E} \bigcup \mathsf{A}) \times \mathsf{String}$ *is the set of labeled typed relationships. A relationship with label* L *(of simple type* String*), which involves one entity* e_1 *and atomic value* a *or another entity* e_2*, can be written as* $\mathsf{e}_1.\mathsf{L} = \mathsf{a}$ *or* $\mathsf{L}(\mathsf{e}_1, \mathsf{e}_2)$ *respectively.*

For simplicity, the CG can be considered as a set of extracted semantic entities. When it's clear from the context, we simply write $CG = \{\mathsf{e}_i\}_{i=1}^{M}$ ($\mathsf{e}_i \in \mathsf{E}$ and $M = |\mathsf{E}|$). Each entity e_i can be involved in J semantic relationships, $\mathsf{e}_i = \{\mathsf{r}_{(i,1)}, ..., \mathsf{r}_{(i,J)}\}$ ($\mathsf{r}_{(i,j)} \in \mathsf{R}$), and each relationship can have K values, i.e., $\mathsf{r}_{(i,j)} = \{\mathsf{r}^1_{(i,j)}, ..., \mathsf{r}^K_{(i,j)}\}$, where each $\mathsf{r}^k_{(i,j)}$ is an atomic value or a pointer to another entity. We note here that our CG model is similar to the W3C RDF model (Manola and Miller 2004).

Consensus Graph Value

We design a comprehensive metric, *Consensus Graph Value*, that takes into account the search demand in measuring the value of extraction results. As shown in Figure 1, the CG value is higher if the extracted entities and relationships *satisfy* more user queries. At the core of this metric is the *coverage semantics*, i.e., what it means for an entity to satisfy a query. Given a structured information repository, users typically issue two major types of queries: *instance queries*, which look for specific entities or relationships, and *category queries*, which look for a large group of entities and relationships satisfying certain conditions. In Figure 1, Q1 and Q2 are instance queries, whereas Q3 and Q4 are category queries. For instance queries, there are typically few matching results, and all of the results are considered equally important. On the other hand, there are often many results satisfying category queries, and each additional result becomes less important as the user already has plenty of results to choose from. We define the value of Consensus Graph based on these intuitions next.

The value of an extraction result CG with respect to a query workload QW[1] is formally defined as the summation of the value of CG for each query q in QW:

$$\mathcal{V}^{QW}_{CG} = \sum_{q \in QW} \mathcal{V}(CG, q) \quad = \quad \sum_{q \in QW} \sum_{i=1}^{n_q} \alpha^{p_i} \mathcal{V}(\mathsf{e}_i, q) \quad (1)$$

[1] A query workload is the set of past queries taken within a time range. As topics shift over time, recency plays an important role in the predictiveness of a query workload for future queries.

where $\alpha > 1, p_i = \begin{cases} 0: & \text{q is an instance query} \\ 1-i: & \text{q is a category query} \end{cases}$

and $\exists j, k\ s.t.\ \mathsf{r}_{(i,j)}$ or $\mathsf{r}^k_{(i,j)}$ matches q

where n_q is the number of entities e_i matching the q: e_i matches q when some relation of e_i ($\mathsf{r}_{(i,j)}$) has a label or contains a value ($\mathsf{r}^k_{(i,j)}$) that matches some keywords in the query. Moreover, when q is a category query, we rank all entities that match q according to their values (defined next) and gradually reduce the value of each additional entity from top down. This models the fact that many low-ranked results for category queries are not that useful to the user. The parameter α determines how fast the value decays for each additional entity. Note that the query workload contains duplicate query terms (e.g. the same query terms may appear multiple times). Hence, the CG value implicitly captures the frequency of different query terms.

For each entity e_i in the CG with the given query q, its value is formally defined as:

$$\mathcal{V}(\mathsf{e}_i, q) = \sum_{j=1}^{J} \mathcal{DG}(\mathsf{r}_{(i,j)}, q) \quad (2)$$

$$\mathcal{DG}(\mathsf{r}_{(i,j)}, q) = \sum_{k=1}^{K} \beta^{1-k} c,\ \mathsf{r}^k_{(i,j)} \notin q \quad (3)$$

Intuitively, Equation 2 illustrates that the value of a matching entity e_i is the summation of the diminishing gains of all of its relationships. Equation 3 describes how the diminishing gain of each relationship is calculated. First of all, a relationship is only counted if its extracted value does not already appear in the query: returning the user some information s/he already knows is not useful. Also, each additional extracted value for the same relationship yields diminishing value (the rate of diminishment is determined by β). Finally, the constant c determines how much value we assign to each piece of extracted value. Because only the relative values are important, we can set c to 1.

Using diminishing gain in Equation 3 is desirable for two reasons. First, during the extraction process, each relationship can often have multiple extracted values (e.g., `cuisine`="Chinese, Thai, Japanese") and each additional information normally brings less value to the entity (and hence to the CG) than its predecessors. Second, even state-of-art information integration and reconciliation mechanisms are often imperfect, and as a result near-duplicate information often permeates the extracted results. Using diminishing gain for each relationship allows us to mitigate the impact of those redundant information on the CG value, and to redirect extraction efforts toward extracting information that has not been seen before.

Finally, we point out that the CG value we have defined is essentially a *generalized coverage measure*: it goes beyond incorporating the absolute number of extracted entities and relationships by considering the importance of individual pieces of information based on how often they are requested by the users.

Page Utility Estimation

CG value is defined in terms of the extraction results and thus is only computable *after the extraction*. To identify which pages to prioritize the extraction effort, we discuss how to estimate the potential value of a candidate page with *page utility* in this section.

Page Utility

The page utility $U(p)$ of a candidate page p is defined as:

$$\begin{aligned} U(p) &= E[\mathcal{V}_{(CG+ext(p))}] \cdot \Pr(p) \\ &\quad + \mathcal{V}_{CG} \cdot (1 - \Pr(p)) - \mathcal{V}_{CG} \\ &= \Pr(p) \cdot (E[\mathcal{V}_{(CG+ext(p))}] - \mathcal{V}_{CG}) \end{aligned} \quad (4)$$

where $\Pr(p)$ is the probability of p being extracted and $ext(p)$ is the entities and relationships that can be extracted from p. Intuitively, if page p is accepted for extraction *and* the extraction is successful, the value of the new consensus graph $CG + ext(p)$ (i.e., the current CG integrated with the entities and relationships from p) becomes $\mathcal{V}_{(CG+ext(p))}$, otherwise the CG value remains the same. Note that $\Pr(p)$ is Bernoulli-distributed, and the outcome of this event is revealed after the extraction. Also, in Eq.(4) the query workload is fixed in the prioritization and is omitted for notational convenience. Prior to the extraction, we aim to select the page for extraction with the highest expected page utility, which boils down to estimating the extraction probability $\Pr(p)$ and the new CG value $E[\mathcal{V}_{(CG+ext(p))}]$.

Estimating Extraction Probability Intuitively, the probability $\Pr(p)$ is affected by two factors, *relevance* and *extractibility*. If the page p is irrelevant to the domain, the system will reject it for extraction. The system may also fail to extract from p due to various other reasons (e.g., graphical contents on the pages). Hence the probability of p being extracted is a joint probability of p being both relevant to the domain and extractable by the underlying extraction system:

$$\Pr(p) = \Pr(p \text{ is relevant}) \cdot \Pr(p \text{ is extracted} | p \text{ is relevant})$$

The first term can be estimated by a lightweight page classifier, which only considers the metadata of the page (e.g., URL). The second term reflects the recall of the IE system and can be estimated based on prior knowledge. If we are agnostic about the performance of the IE system *a priori*, we can assume that the probability of successfully extracting a relevant page is largely related to the hosting domain of the page (as pages in the same domain share similar structural characteristics), and therefore estimate the probability in an online manner by examining the success rate on recently processed pages. Alternatively, as part of our future work, $\Pr(p)$ can be estimated in a more sophisticated way by regression based on features such as the DOM structure and lexical information of the page.

Estimating the New CG Value Let $E_p = \{\mathsf{e}_k\}$ be the set of entities extracted from page p. Formally, the expected CG value after integrating p can be computed as:

$$E[\mathcal{V}_{(CG+ext(p))}] = \mathcal{V}_{CG} + \sum_{\mathsf{e}_k \in ext(p)} E[\mathcal{V}[\mathsf{e}_k | CG]] \quad (5)$$

where $\mathcal{V}[\mathsf{e}_k|CG]$ denotes the increment in CG value when the entity e_k is integrated to the current CG. Here, we assume that entities extracted from the same page are deduplicated by the underlying extraction system and thus their additions to the CG are independent. We consider the probabilistic events of whether the entity e_k exists in the CG:

$$\begin{aligned} & E[\mathcal{V}_{(CG+ext(p))}] - \mathcal{V}_{CG} \\ = & \sum_{\mathsf{e}_k \in ext(p)} [\ \Pr(\mathsf{e}_k \in CG) \cdot (E[\mathcal{V}_{CG \cup \{\mathsf{e}_k\}}] - \mathcal{V}_{CG}) \\ & + (1 - \Pr(\mathsf{e}_k \in CG)) \cdot E[\mathcal{V}_{\{\mathsf{e}_k\}}]\ ] \end{aligned} \quad (6)$$

Intuitively, if e_k has not been extracted before, the increase in CG value is solely based on e_k regardless of the current CG. Alternatively, if e_k is already in the current CG, it should be consolidated with the existing entity and only *novel* information can contribute to the CG value increase.

For the former case (i.e., $\Pr(\mathsf{e}_k \in CG) = 0$), we have:

$$(6) = \sum_{\mathsf{e}_k \in ext(p)} E[\mathcal{V}_{\{\mathsf{e}_k\}}] = \sum_{\mathsf{e}_k \in ext(p)} \sum_{q \in QW} E[\mathcal{V}(\{\mathsf{e}_k\}, q)]$$

Therefore,

$$U(p) = \Pr(p) \cdot \sum_{\mathsf{e}_k \in ext(p)} E[\mathcal{V}(\{\mathsf{e}_k\}, QW)] \quad (7)$$

In words, $U(p)$ equals the product of its extraction probability and the sum of *importance* of the entities that can be extracted from p. As aforementioned, importance is measured according to the entity's prevalence in the query workload.

For the case where e_k may not be *novel*, we need to estimate $\Pr(\mathsf{e}_k \in CG)$ and $E[\mathcal{V}_{CG \cup \{\mathsf{e}_k\}}] - \mathcal{V}_{CG}$. The former characterizes the probabilistic event whether e_k appears in the current CG. This can be efficiently estimated by interpreting the similarity between two entities as the desired probability (e.g. the similarity of the name of a restaurant appearing on a candidate page and those already in the CG). To estimate the latter, by substituting p with e_k and e_k with $\mathsf{r}_{(k,j)}$, we can further drill down to the expected increase in the utility of an existing entity e_k by incorporating relationships in a similar form as in Equation (6). The details of these are omitted due to space limitation.

Page Scoring

Estimating a page's utility involves discovering the set of entities on the page and computing their expected values as in Equation (6). These tasks are accomplished by performing highly lightweight extraction on the page (see next section) to obtain an approximation of the set of extractable entities. To illustrate, consider $E[\mathcal{V}_{\{\mathsf{e}_k\}}]$ and only instance queries. Referring to Eq.(1), we have:

$$E[\mathcal{V}_{\{\mathsf{e}_k\}}] = E[\sum_{q \in QW} I[e_k \text{ matches } q] \cdot \mathcal{V}(e_k, q)] \quad (8)$$

Further assuming that $\mathcal{V}(e_k, q)$ is constant for all k:

$$E[\mathcal{V}_{\{\mathsf{e}_k\}}] \propto E[\sum_{q \in QW} I[e_k \text{ matches } q]] \quad (9)$$

which is the expected query coverage of the entity. Hence the system can use the query coverage (substituting the expectation) to compute the entity's utility.

Implementation

The extraction prioritization (EP) process is accomplished in *batches*. In each batch, a set of pages are sampled from the corpus and scored. The ones with the highest estimated utility are extracted immediately (i.e. prioritized) and the rest is either queued in the working set or discarded if their utility is too low. After the extraction completes, EP obtains the updated CG from the extraction system and incrementally updates the utilities of pages remaining in the working set, before moving on to the next batch. We note that, at one extreme, each page can be considered as an individual batch, in which case the system becomes the random sampling strategy. At the other extreme, the whole corpus can be considered as a single batch. This, however, means that before the first page can even be extracted, EP will have to analyze and score all the pages in the corpus — a delay that is often too costly. Here, we briefly describe the scheduling process and the page scoring operations.

Scheduling: Algorithm 1 illustrates the scheduling procedure. The core data structure is the working set $\mathcal{W}$, implemented as a priority queue. In each batch, a set of s pages is randomly sampled and removed from the corpus $\mathcal{P}$. For each sampled page, its expected utility is estimated as we will describe next. Those pages with expected utility greater than a threshold δ (line 7) are inserted into the working set and the top b pages are selected for extraction. Finally, the expected utilities of the top k pages in $\mathcal{W}$ are incrementally updated (line 13) according to the updated CG. We only update the estimated utilities of top remaining pages as the working set can become quite large and hence costly to re-estimate all pages.

Algorithm 1 EP scheduling procedure

Require: $\mathcal{P}$: a corpus of pages
1: $\mathcal{W} \leftarrow \phi$
2: **while** $\mathcal{P} \neq \phi$ || $\mathcal{W} \neq \phi$ **do**
3: $\mathcal{R} \leftarrow$ Random_Sample($\mathcal{P}, s$) // s: sample size
4: $\mathcal{P} \leftarrow \mathcal{P} - \mathcal{R}$
5: **for** $p \in \mathcal{R}$ **do**
6: $u \leftarrow$ Utility(p)
7: **if** $u > \delta$ **then**
8: $\mathcal{W}$.insertWithPriority(p, u)
9: **end if**
10: **end for**
11: $\mathcal{CG} \leftarrow$ IE.extract($\mathcal{W}$.removeTopN(b)) // b: batch size
12: **for** $p \in \mathcal{W}$.getTopN(k) **do**
13: update utility of p w.r.t. $\mathcal{CG}$ // $k \geq b$
14: **end for**
15: **end while**

Page Utility Estimation: Algorithm 2 shows the algorithmic steps of the *page scorer*, which accomplishes lightweight extraction and utility estimation. A given page p is first segmented into a set of sections C such as the page's hosting domain, URL, title, keywords, header, body, footer, etc. The sections are then cleaned up and tokenized (e.g. the domain name `www.timberlodgesteakhouse.com` becomes 'Timber Lodge Steakhouse'). Lightweight

extraction is then performed on the token sets T. Evidently, the speed of lightweight extractors is crucial. We use rule-based extractors and base noun phrase (BNP) chunkers for this purpose. Line 5-9 estimates each potential entities' value u. Finally, p's utility is computed by combining u and page extraction probability r using Equation (7).

Algorithm 2 Generic page scorer Utility (p)

Require: p: a page, QW: prioritization query workload
Ensure: u: estimated page utility
1: $S \leftarrow \mathsf{Segment}(p)$
2: $T \leftarrow \mathsf{Tokenize}(S)$
3: $E \leftarrow \mathsf{LightWeightExtract}(T)$
4: $v \leftarrow 0$
5: **for** $e_k \in E$ **do**
6: **for** $q \in QW$ **do**
7: $v \leftarrow v + \mathsf{Similarity}(e_k, q)$
8: **end for**
9: **end for**
10: Estimate page extraction probability r
11: Compute utility u as in Equation (7)

Experiments

We built the EP system in Java and conducted a comprehensive set of experiments in the *restaurant domain* using real world data. All experiments were conducted on a Linux machine with a Intel Core2 CPU and 4GB memory. We choose to focus our evaluation on the restaurant domain, for which logs of real user queries on restaurants are available to us from Yahoo! Local.

Corpus: We randomly sampled 100,000 web pages from the crawl of a major search engine (search.yahoo.com) and named this corpus **web**. A very small portion of those pages are expected to be related to restaurants. In practice, preprocessing techniques like focused crawling will likely lead to a corpus with a higher concentration of restaurant pages. As a result, we further randomly sampled 10,000 pages, whose URL domains matched those covered by a comprehensive set of restaurant domains in Yahoo! Local. We named this corpus **core**. To simulate the focused crawling effect, we combined the **web** corpus with the **core** corpus to generate the **focus** corpus with 110,000 pages.

Query Workload: We randomly sampled 150,000 queries from the log of the restaurant search engine [2] during the month of May 2009. We further randomly sampled 50,000 restaurant queries from the month of June 2009. The former is used as the **prioritization workload** and the latter as the **evaluation workload** in our experiments.

Restaurant Information Extraction: To extract restaurant information from a page, we leverage an extraction system (Bohannon et al. 2009) that includes a rule-based extractor for contact information extraction and an SVM-based extractor for cuisine type extraction.

[2]Yahoo! Local has a sophisticated way of determining which queries belong to the restaurant category with over 95% accuracy based on examining a 200-query random sample.

Page Scoring Strategies: We experiment with different page scoring strategies in prioritization: *metadata-only*, *content-only* and *combined*. The metadata-only scorer analyzes the page metadata that are available in the crawl (e.g. URL tokenized into words, title, page size, etc.). The content-only scorer analyzes easily accessible portions of the page content (e.g., header and footer) for scoring. The combined scorer simply uses an equal-weighted sum of both scores. We compare those strategies to *random*, which randomly selects pages at each iteration.

Effectiveness of Extraction Prioritization

We first examine the effectiveness of different EP strategies. The effectiveness is measured by the resulting CG value after extracting a certain number of pages. (In this section, all experiments are done with the batch size 100 for **focus** corpus and 20 for **core** corpus.)

Figure 2 shows that all three EP strategies far outperform *random* on the **focus** corpus. After extracting $10,000$ pages (9% of all pages) from the **focus** corpus, the EP strategies have achieved a CG Value of 5700, covering 85% of the total CG Value of 6770 (calculated from the **core** corpus as the number of restaurant pages in the **web** corpus is negligible). In contrast, the *random* strategy can only achieve a CG value of 1047 after extracting $10,000$ pages, wasting a lot of resources on the irrelevant pages. As expected, the CG value achieved by *random* grows almost linearly to the number of pages extracted. For EP strategies, there are roughly three stages. In the fast growth stage (the first 2000 pages extracted), the system identifies the most important pages. In the slow growth stage (roughly from 2000 to 7000 pages), the most important pages are exhausted and the system starts to extract less important but still valuable pages. Finally, in the random stage (the rest of the pages, not shown after $10,000$), useful pages have been exhausted and prioritization no longer has advantages over *random*.

We further examine the effectiveness of EP strategies on the restaurant-concentrated (and thus random-friendly) **core** corpus. As shown in Figure 3 (left), all EP strategies still significantly outperform *random*. This is especially evident for the fast growth stage (Figure 3 (right)). After high priority pages are extracted, EP strategies start to slow down until they reach the same CG value as *random* when all pages are extracted. We also compare the number of pages the EP strategies and the *random* strategy need to extract to reach the same CG value. As shown in Figure 4, to reach 60% of the total CG value, the EP strategies often only need

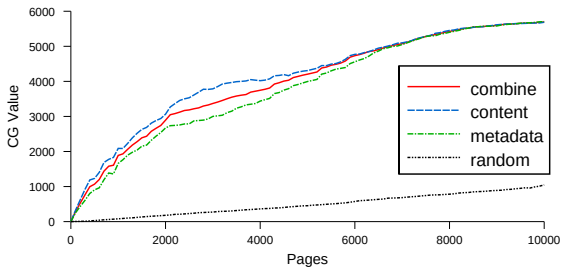

Figure 2: Comparison of Extraction Prioritization strategies on the corpus "focus".

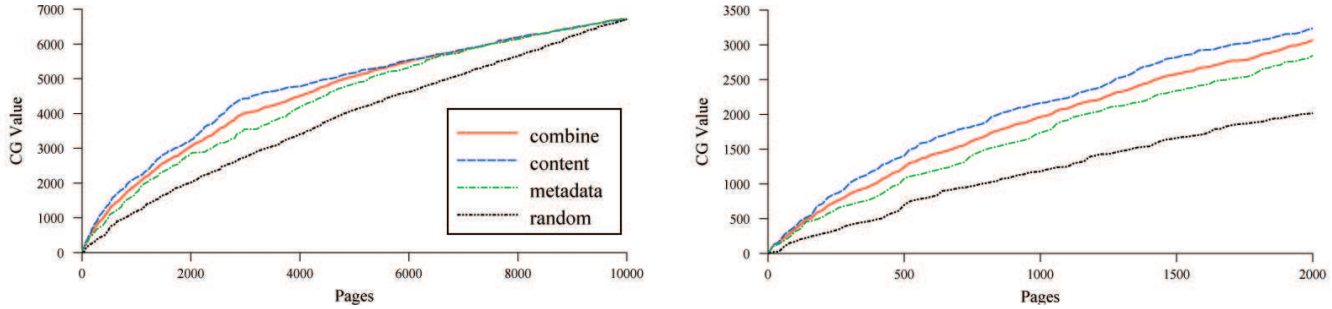

Figure 3: (Left) Comparison of prioritization strategies on the corpus "core" (Right) Results on first 2k pages.

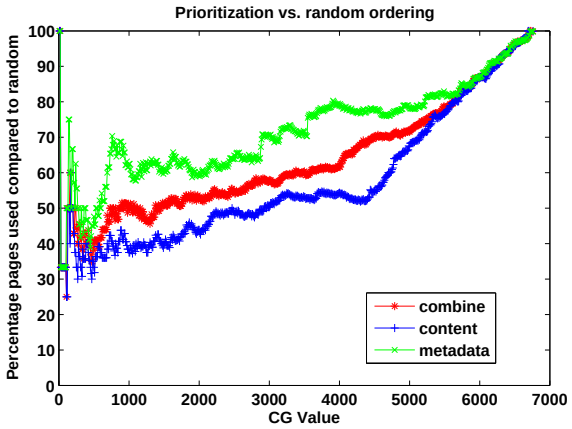

Figure 4: Percentage of pages used by different methods to reach the same CG value as *random* on the corpus "core".

to extract about half of the pages needed by the *random*, which is a significant improvement.

These experiments clearly demonstrate that prioritizing the extraction on pages with highest utilities achieves better extraction results faster. As the sheer size of the web and its evolution outpace state-of-the-art extraction methods, EP can have a significant impact in resource conservation. We perform some detailed analysis on EP further. All experiments use the combined strategy and the **core** corpus (if not otherwise specified).

Overhead of Prioritization: We weigh the benefit of prioritization against its overhead and show the results in Table 1. Even on the highly concentrated **core** corpus, EP strategies cost only 15% more running time, while achieving over 60% more value. The trade-off is even more impressive when prioritizing for the less concentrated **focus** corpus: over an order of magnitude more value in the extraction results with running time overhead of less than 35%.

Impact of Batch Size: Batch size determines the number of pages to extract at each iteration (b in Algorithm 1).

core	Time (secs)	CG Value
random	385	1178
combine	437 (+13.5%)	1964 (+66.7%)
focus	Time (secs)	CG Value
random	2336	80.64
combine	3122 (+33.65%)	1887 (+2258%)

Table 1: Overhead and benefit of prioritization on extracting 10% of the total pages ($1,000$ for **core** and $10k$ for **focus**).

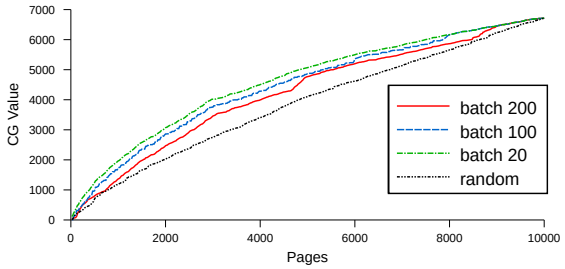

Figure 5: Impact of batch size on Extraction Prioritization on the corpus "core".

Figure 5 shows that the smaller the batch size, the more effective prioritization is, as a small batch size allows the extraction system to focus on the most valuable pages first.

Impact of Query Workload Size: Figure 6 shows that the prioritization strategy is relatively robust against the size of the query workload. A large query workload (150K queries) is slightly more effective than a smaller one (50K), and both significantly outperform the random strategy.

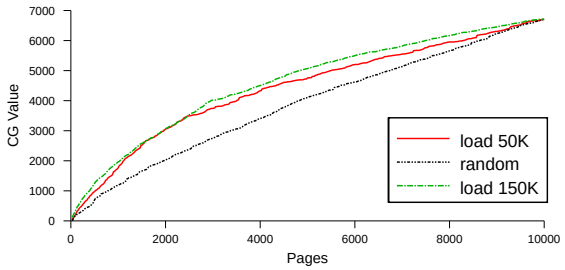

Figure 6: Impact of query workload size.

Comparison with Classifier Approach

The page utility estimation can be used by the so-called *classier approach* for filtering instead of prioritization. To compare these two approaches, we build two classifiers based on the combined scorer. The first one retains only pages with estimated utility greater than 0.8 for extraction, while the second one retains pages with utility higher than 0.2. We compare the prioritization strategy against both and the result is shown in Figure 7. Not surprisingly, the more selective classifier is slightly more effective than the prioritization approach in the initial stage, but it stops extracting pages after about $4,000$ pages, leaving a mere 55% coverage of the total CG value, compared with an

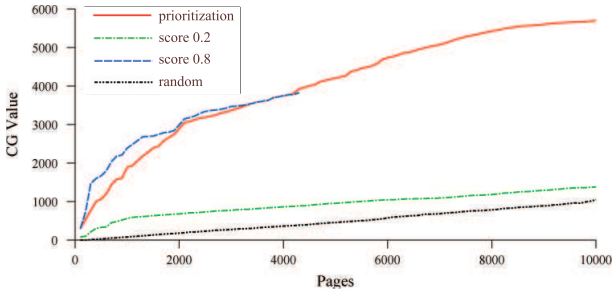

Figure 7: Comparing Extraction Prioritization with classifier approach on the corpus "focus".

85% coverage for the prioritization approach. Contrarily, the less selective classifier significantly under-performs the prioritization approach and more closely resembles the *random* strategy. In general, the classifier approach will struggle to find a good threshold to achieve the optimal balance between better overall coverage and better initial results. The prioritization approach addresses this problem by eliminating the need to define a single threshold (and make only binary decisions) on whether pages should be extracted or not — instead, pages compete with each other for the extraction slots and the more useful a page is, the earlier it will be selected for extraction.

Related and Future Works

Prioritization has spawned significant research interests recently in many research areas. In AI and data integration, (Jeffery, Franklin, and Halevy 2008) adopted a decision-theoretic approach to order candidate entity matches for user confirmation. The ordering is guided by the value of perfect information (VPI) (Russell and Norvig 1995), which is estimated from the query workload. The page utility in this paper also adopts the VPI technique to estimate the true value of the unknown. Besides the different goals (theirs for reconciliation and ours for extraction), our EP system has to handle the novelty factor — prioritization needs to be aware of the extraction results after each iteration. In search engine crawling, (Pandey and Olston 2008) proposed to prioritize the crawling of web pages according to the page impact, defined as the number of times the page appears in top search results. In information extraction, (Agichtein and Gravano 2003; Ipeirotis et al. 2006) introduced an iterative approach to identify useful pages for extraction. It starts with a small seed set of documents and gradually constructs keyword queries from the extracted pages, to fetch additional pages with similar content by a search engine. Unlike EP, it relies on the quality of the initial seed set and the external search engines to achieve good results, without considering the impact of user search demands.

There are several future directions to explore. First, social tagging (e.g. Digg) is an important form of explicit user endorsement and can be adopted as an alternative way of estimating page importance. Second, the value of the extracted entities and relationships are also influenced by their corroborative-ness: i.e., whether they can influence the scores of some already extracted information toward their true values. This is especially important since most extraction systems produce information that is uncertain and often conflicting. A formal model of the tradeoff between redundancy and novelty is an important future direction.

Conclusion

A vast amount of knowledge is hidden behind raw pages on the Web. Information extraction techniques transform them into structured information that facilitates human information consumption and data mining. The sheer size and the fast growth of the Web, however, are overwhelming the state-of-the-art IE techniques. We proposed an automatic technique, Extraction Prioritization, for obtaining the most valuable extraction results as early as possible in the extraction process. We formally defined a metric for measuring the quality of extraction results, which is suitable for the web retrieval context. We further developed statistical methods to efficiently estimate the page utilities without launching full-scale extractions. We implemented the EP system and validated its significant benefits in a large web data set over alternative strategies.

Acknowledgments

This work was done while Jian Huang was visiting Yahoo! Research, New York. The authors would like to thank Philip Bohannon, Raghu Ramakrishnan and the anonymous reviewers for their helpful comments and suggestions.

References

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

Banko, M.; Cafarella, M. J.; Soderland, S.; Broadhead, M.; and Etzioni, O. 2007. Open information extraction from the web. In *IJCAI*.

Bohannon, P.; Merugu, S.; Yu, C.; Agarwal, V.; DeRose, P.; Iyer, A.; Jain, A.; Kakade, V.; Muralidharan, M.; Ramakrishnan, R.; and Shen, W. 2009. Purple SOX extraction management system. *ACM SIGMOD Record* 37(4):21–27.

Bollacker, K.; Evans, C.; Paritosh, P.; Sturge, T.; and Taylor, J. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD*.

Ipeirotis, P. G.; Agichtein, E.; Jain, P.; and Gravano, L. 2006. To search or to crawl?: towards a query optimizer for text-centric tasks. In *SIGMOD*.

Jeffery, S. R.; Franklin, M. J.; and Halevy, A. Y. 2008. Pay-as-you-go user feedback for dataspace systems. In *SIGMOD*.

Manola, F., and Miller, E. 2004. RDF Primer W3C Recommendation.

Pandey, S., and Olston, C. 2008. Crawl ordering by search impact. In *WSDM*.

Russell, S., and Norvig, P. 1995. *Artificial Intelligence: A Modern Approach*. Prentice Hall.