A Structure-Driven Yield-Aware Web Form Crawler: Building a Database of Online Databases

Bin He, Chengkai Li, David Killian, Mitesh Patel, Yuping Tseng, Kevin Chen-Chuan Chang
Computer Science Department
University of Illinois at Urbana-Champaign
{binhe, cli, dkillian, mppatel2, ytseng1}@uiuc.edu, kcchang@cs.uiuc.edu

ABSTRACT

The Web has been rapidly "deepened" by massive databases online: Recent surveys show that while the surface Web has linked billions of static HTML pages, a far more significant amount of information is "hidden" in the deep Web, behind the *query forms* of searchable databases. With its myriad databases and hidden content, this deep Web is an important frontier for information search. In this paper, we develop a novel *Web Form Crawler* to collect the "doors" of Web databases, *i.e.*, query forms, to build a database for online databases in both efficient and comprehensive manners. Being *object-focused*, *topic-neutral* and *coverage-comprehensive*, such a crawler, while critical to searching and integrating online databases, has not been extensively studied. In particular, query forms, while many, when compared with the size of the Web, are sparsely scattered among pages, which brings new challenges for focused crawling: First, due to the topic-neutral nature of our crawling problem, we cannot rely on existing topic-focused crawling techniques. Second, existing focused crawling cannot achieve the comprehensiveness requirement because it is not able to be aware of the coverage of crawled content. As a new attempt, we propose a *structure-driven* crawling framework by observing *structure locality* of query forms– That is, query forms are often close to root pages of Web sites and accessible by following navigational links. Exploring this structure locality, we substantiate the structure-driven crawling framework into a *site-based* Web Form Crawler by first collecting the site entrances, as the Site Finder, and then searching for query forms within the scope of each site, as the Form Finder. Analytical justification and empirical evaluation of the Web Form Crawler both show that: 1) our crawler can maintain stable harvest and coverage throughout the crawling, and 2) compared to page-based crawling, our best harvest rate is about 10 to 400 times better, depending on the page traversal schemes used.

1. INTRODUCTION

We have witnessed the rapid growth of databases on the Web, or the so-called "deep Web." As Figure 1 conceptually illustrates, on this deep Web, myriad online databases provide dynamic query-based data access through their query interfaces, or *query forms*, instead of static URL links. A July 2000 survey [4] estimated 43,000-96,000 "search sites" and 7,500 terabytes of data– 500 times larger than the statically linked "surface Web." Our recent study [8] in April 2004 estimated 307,000 deep Web sources, reflecting a fast 3-7 times increase in 4 years (2000-2004) and resulting in a total of over 1.2 million query interfaces. With the virtually unlimited amount of information sources, the deep Web is clearly an important frontier for information search, integration and mining.

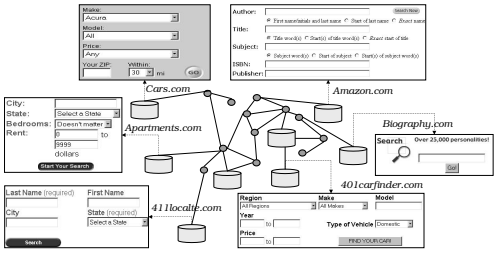


Figure 1: The deep Web: Databases on the Web.

However, while there are myriad databases online, users often have difficulties in first *finding* the right sources and then *querying* over them. Consider a user Amy, who is moving to a new town. To start with, different queries need different sources to answer: Where can she look for real estates (*e.g.*, *realtor.com*)? Study for a new car (*cars.com*)? Find a new job (*monster.com*)? Further, different sources support different query capabilities: After source hunting, Amy must learn to gruel the details of querying each source. Enabling search and integration is thus crucial for accessing the deep Web.

Toward effective access of online databases, as the "doors" to the deep Web, query interfaces are key to reach the data hidden behind. The discovery of query forms are thus essential for the subsequent search and integration tasks, much like today's "surface Web" search engines must start with crawling Web pages. While crucial, automatic collection of query forms in a large scale has remained largely unexplored– As a novel attempt, this paper thus develops a *Web Form Crawler*. Such crawling will realize our objective of building a *database of online databases*.

Such a Web Form Crawler has broad applications in searching and integrating data on the deep Web. First, *source search*: A Web database search engine can be directly built upon our crawling result, to facilitate users in finding online sources– *e.g.*, for directing Amy to the right sources in various domains. Second, *source integration*: To help users querying selected sources uniformly, many ongoing research efforts, *e.g.*, MetaQuerier [10] and WISE [19], have been investigating large scale deep Web integration. Our Web From Crawler will fill in the critical lacking of source discovery in the first place, to facilitate the subsequent integration tasks, *e.g.*, interface extraction [29, 20], schema matching [16, 19, 27, 18, 26, 17], and query translation [30]. Third, *data gathering*: Many integration tasks require access to data behind query interfaces by automatic querying, *e.g.*, [24, 23], for which query forms must again be discovered first.

Building the Web Form Crawler brings new challenges. In particular, while a large number, query interfaces as scattered on the entire Web are rather sparse: Our estimated 1,258,000 query interfaces (as just mentioned; in [8]) can appear anywhere in the

19.2 billion Web pages (as reported by the recent index of Yahoo.com [28], which can thus be viewed as the lower bound the Web size). As a baseline, the traditional *page-based* crawler (without a topic focus), which recursively follows links to traverse the entire Web, will thus expect to find only one interface in crawling 15262 pages.

To effectively build a database of online databases, our crawler has dual requirements: First, to be *efficient*, it must have a high *harvest* rate to collect Web forms without crawling many pages, which is defined as

$$harvest = \frac{\#\,of\ Forms\,Collected}{\#\,of\ Pages\,Crawled}. \quad (1)$$

Second, to be *comprehensive*, it must have a high *coverage* rate so as to cover a reasonable snapshot of the deep Web, which is defined as

$$coverage = \frac{\#\,of\ Forms\,Collected}{\#\,of\ Total\ Forms}. \quad (2)$$

To motivate, the traditional page-base crawler, as just mentioned, after crawling the entire Web (or $c\%$ pages), will in principle result in 100% (or proportionally $c\%$) coverage, but at the cost of a measly harvest of 6.6×10^{-5}.

For more effective crawling, instead of traversing arbitrary links, we must develop a *focused* crawling strategy tailored for finding query forms. Unlike general crawling (which collects all Web pages), a focused crawler targets a specific subset of pages on certain focus, say, "virtual reality." Such a focused crawler, with its specific target, can often find crawling paths of certain patterns that lead to the desired pages, and thus achieve higher harvest. Such focused crawling is critical for building focused search applications such as MetaQuerier and many vertical search engines, where it is unnecessary to crawl the entire Web. In particular, as our goal is to build a focused Web Form Crawler, we must address two new challenges, which cannot be solved by traditional focused crawling techniques:

First, our crawler is *object-focused* but topic-neutral– the opposite of traditional focused crawlers. That is, unlike existing settings of *topic-focused* crawlers [7, 14, 21, 1, 6, 25], which look for Web *pages* of certain *topics*, our crawler targets a certain type of *objects*, namely query forms, which can be of any subject topics (*e.g.*, Amy's example: real-estate, cars, jobs). With their topic-focus, existing focused crawlers are mainly *content-driven*, by exploiting *content locality* across links: A page of certain topics can often be reached through a path along which the *contents* of pages form some patterns. In the simplest form, such content locality means that a page on, say, "virtual reality" may be connected from pages of similar topics. At its core, a topic-focused crawler employs a classifier to distinguish the content orientations (*e.g.*, trained using keyword features) of pages to find a desirable path.

Such techniques, being topic-focus and thus content-driven, are unlikely to work for our object-focused but topic-neutral crawler, simply because pages containing objects (such as query forms) can be undistinguishable from other pages by topic. Moreover, in practice, we may encounter difficulty in finding positive and negative examples to train the classifier. For instance, it is hard to pick appropriate positive examples to cover all subject topics. Also, being content-driven, current classifiers in topic-focused crawling mainly explore textual information as features in the training stage. Therefore, even with a given set of examples, it is difficult for a topic-focused classifier to get topic-neutral features.

Second, more importantly, our crawler needs to balance both efficiency and comprehensiveness, with not only a high harvest but also a "reasonable" coverage. (With the ever expanding and changing Web, it is well accepted that 100% coverage is unrealistic.) We note that harvest and coverage are often *conflicting* metrics: While focusing on only promising pages will lead to a high harvest, its narrow focus of "not going beyond" may compromise the coverage. On the other hand, although combing through many pages will extend the coverage, the broad reach may lead to diminishing returns and thus compromise the harvest.

However, current topic-focused crawlers, by greedily pursuing promising paths, aim at high harvest with no explicit notion of coverage. That is, while a crawler may start with high harvest for what it *has* crawled, how long will such harvest sustain? Can it estimate the harvest for what it has *not* crawled, so as to bail out without wasting resources in diminishing returns that will not enhance coverage (but actually hurt harvest)? Without this sense for the "unexplored" territory, it cannot focus resources on achieving reasonable coverage while maintaining high harvest *throughout* crawling.

Overall, we aim at developing a crawling framework that, without assuming topic-focus, not only gives a high harvest for what it has crawled, but also estimates a low yield for what it decides not to crawl, and thus achieves a good overall coverage. Our insight hinges on that, for our object-focused crawling, there exists certain *structure locality* on the Web, which can guide a "scope" for our crawling to focus into and draw a boundary around. This concept of structure locality, in terms of how our target objects distributes in the scope, enables the balance of the dual goals of harvest and coverage.

Specifically, we observe that query forms indeed distribute with such structure locality: *First*, independent of topic domains, query forms often appear near the entrance point, *i.e.*, the root page, of a Web site. *Second*, around the entrance point of a site, query forms also distribute in certain ways– Within the site, they tend to appear shallowly and are often reachable through *navigational links* (*i.e.*, links in the navigational menus of the site). Thus, our topic-neutral crawler can focus on such structure locality: Viewing the Web as a graph of Web sites, it will crawl each site as a separate "scope." For each site, starting from its entrance and following navigational links, it will achieve a high harvest rate. Further, drilling deeper into the site, when the yield starts to diminish, it will bail out, while still maintaining satisfactory coverage.

We thus propose the new concept of *structure-driven* crawling for realizing our *object-focused* crawler (Section 3). In the structure-driven framework, a crawler conceptually partitions the Web into independent scopes (*e.g.*, Web sites in our case) and search for target objects in each scope, with certain intra-scope search strategy that matches the object distribution patterns. Our analysis shows that, iterating over sites, as each site gives good *local* harvest and coverage, structure-driven crawling will maintain predictable *global* harvest and coverage for the entire crawling process. To accurately estimate the local yields and further predict the global yields, we develop a *sampling-then-executing* methodology to guide the selection of the best intra-scope search strategy for structure-driven crawling. Finally, since our crawling assumes Web sites as independent "scopes," it is inherently parallelizable.

To realize the structure-driven crawling framework for collecting query forms, we develop the *Web Form Crawler* with two components: *Site Finder* for collecting Web sites as scopes, and *Form Finder* for searching each scope. Section 4 and Section 5 discuss our design of the Form Finder and Site Finder in details, respectively. We have implemented the crawler, in a naturally parallel architecture, and deployed it on a cluster of about 100 PC nodes. We report large scale experiments in Section 6, which validate that the framework indeed crawls Web forms effectively by maintaining steady harvest and growing coverage as it crawls. In summary, the contributions of this paper are:

- In terms of **problem**, we propose to build a crawler for collecting deep Web sources, to build a database of online databases, as a map of the deep Web. While critical for deep Web search and

Subject Topics	Crawling Target: *Page*	Crawling Target: *Object*
Any	General [5,12]	Object-Focused
Certain	Topic-Focused [7,14,21,1,6,25]	Topic&Object-Focused [15,2]

Figure 2: The taxonomy of Web crawlers.

integration, such a crawler has been lacking.

- In terms of **abstraction**, we recognize the crawling problem as a new object-focused crawling.
- In terms of **framework**, we develop the novel concept of structure-driven crawling and propose a sampling-then-executing methodology to guide the selection of the best intra-scope search strategy based on statistical modeling.
- For concrete **techniques**, we design effective strategies for finding Web sites (in Site Finder) and searching for query interfaces within each Web site (in Form Finder).

2. RELATED WORK

We review the recent work on Web crawling. To facilitate our discussion, we present a taxonomy of 4-quadrant of Web crawlers in Figure 2, which contains two dimensions. Along the dimension of *subject topics*, crawlers are either topic-neutral on *any* topic or specific to *certain* topics. Along the dimension of *crawling target*, while traditional crawlers collect HTML pages, new type of crawlers collect certain Web artifacts (which we call *objects*), such as Web sites, query forms, products (*e.g.*, digital cameras), or addresses. This taxonomy thus classifies Web crawlers into four categories. For example, the traditional link-following crawlers [5, 12] fall into the category of topic-neutral and page-targeting crawlers, and the category of topic-focused crawlers [7, 14, 21, 1, 6, 25] look for pages on given topics.

As mentioned in Section 1, the traditional link-following (*i.e.*, page-based) crawlers [5, 12] are very inefficient for finding specific type of objects (query forms in our case) because of its poor harvest. They are also inefficient in finding pages with specific topics, due to the same reason. Such inefficiency indeed motivated the development of the topic-focused crawlers [7, 14, 21, 1, 6, 25]. However, although topic-focused crawlers improve the crawling efficiency significantly for topic-focused tasks, they are not appropriate for crawling topic-neutral objects such as query forms. As Section 1 discussed, topic-focused crawlers employ classifiers to distinguish the textual features of Web pages in finding promising crawling paths to follow. However, this intuition does not apply in finding topic-neutral objects. Moreover, topic-focused crawlers are not able to maintain a balance between harvest and coverage, as mentioned in Section 1 as well.

The focus of this paper is crawling certain type of objects. A recent work [15] describes a crawler for collecting Web sites related to specific topics. Given its target objects, *i.e.*, Web sites, it is natural for [15] to crawl site by site. Thus it is functionally similar to our Site Finder component but with specific topic-focus. The work closest to ours is [2], in which a crawler is developed to find query forms on given topics. While objects targeted by [15] are Web sites, the target objects in [2] and our work are query forms. Note that both [15] and [2] belong to the category of topic-focused and object-focused crawlers. By exploiting content-driven techniques (*e.g.*, content classifiers), they are not applicable in finding topic-neutral query forms or Web sites.

In contrast, to the best of our knowledge, ours is the first attempt of building a topic-neutral object-focused Web crawler. Building upon the insight of structure-driven crawling, this new framework on the one hand eliminates the reliance on content focus, and on the other hand enables us to balance between high harvest (as virtually all the traditional crawlers focus on) and coverage.

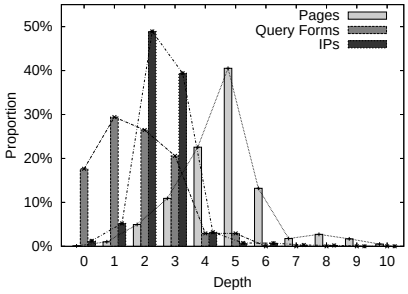


Figure 3: Proportion of pages, query forms, and IPs over depth.

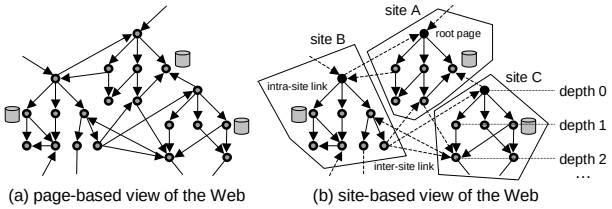


Figure 4: Page-based view vs. site-based view.

3. SYSTEM ARCHITECTURE

In this section, we present our architectural design of the Web Form Crawler in details. Specifically, we first observe the existence of concerted structure locality (Section 3.1), then motivate the structure-driven yield-aware crawling framework (Section 3.2), and finally discuss the development of the system architecture (Section 3.3).

3.1 Motivation: Structure Locality

Our object-focused crawling aims at comprehensively collecting query forms as the *target objects*. Being topic-neutral and coverage-aware, unlike traditional topic-focused crawling, our crawling cannot rely on content locality (as Section 1 mentioned)– We thus wonder, is there any new type of "locality," as distribution patterns of the target objects, that we can resort to?

We take a "divide-and-conquer" approach for solving the object-focused crawling. We first divide the Web into a set of non-overlapping *scopes*, where each scope contains a set of pages. With appropriate partitions, we hope that each scope will contain some type of locality, which can be explored to conquer the problem of object-focused crawling. Then our question becomes: Can we find a good way to divide the Web into scopes with the localities we need for the crawling task?

We notice that Web sites, as the intermediate concept between pages and the entire Web, seem to be natural partitions for scopes. We thus conduct a survey over Web sites and the answer is positive– Our result shows that Web sites are the appropriate scopes with a new locality feature for finding query forms. In particular, we studied the locations of query interfaces in their Web sites. For each query interface, we measured its *depth* as the minimum number of hops from the root page of the site to the interface page. We randomly sampled 1 million IP addresses, from which we identified 281 Web servers, crawled these servers up to depth 10, and identified a total of 34 databases with 129 query interfaces. Since a database can be accessed through multiple query forms in many sites, we manually check all the query interfaces to identify such "same-databases".

Our study shows an interesting phenomenon: Query forms tend

to locate "shallowly" in their sites and thus have *structure locality*. Figure 3 shows the distribution, in terms of proportion of total query forms, at progressively deeper levels from depth 0 to 10. The result clearly shows that most query forms can be found within depth 3 and none deeper than 5. To contrast in perspective, Figure 3 also shows the distribution of pages– which grows *exponentially* from 0 up to 5 and decreases after. While there are significantly more pages toward deeper in a site, most query forms are in the shallow levels. In particular, the top 3 levels (from the root page to depth 3) contain only 17% of total pages but 94% of forms.

This observation inspires us a site-based view of pages on the Web. To begin with, Figure 4(a) shows the typical page-based view of the Web, in which all pages and all links are equal. On top of the page graph, we now view the Web as a collection of Web *sites*, as Figure 4(b) shows. Each site is an HTTP server containing a subgraph (of the Web) for pages on the server, and is uniquely addressed by a distinct IP domain name and an HTTP port number [1], *e.g.*, `http://xyz.com:8080`. For brevity, we will simply use *site*, *IP*, or *domain name* interchangeably.

From Figure 4, we can see that although query forms seem to distribute sparsely and randomly on the traditional page view, the structure locality as we observed means that, within each site as a scope, the distribution of query forms is rather "predictable" (in a statistical sense)– they follow the pattern as Figure 3 shows, in which we expect to find query forms, if any, around the entrance of each scope.

To compare, in the traditional page-based view, we essentially consider each page itself as a scope. Under this view, since each scope is "atomic" with only one page, there is no intra-scope locality. We can only explore the inter-scope locality, *i.e.*, the linkage closeness among scopes (*i.e.*, pages) with the same topic or the so-called content locality.

On the contrary, in the site-based view, we view each site as a scope and employ structure locality as the "maps" to guide the crawling within scopes for finding target objects. Unlike the inter-scope content locality, structure locality, as a new type of locality, explores intra-scope information (*e.g.*, the depth of links, the navigational menu links) and has two excellent features: 1) topic-neutral: By exploring structure information, an intra-scope search strategy can equally handle any scope regardless of its domain. 2) coverage-aware: Equally treating any scope, an intra-scope search strategy is likely to achieve stable harvest and coverage within scopes and further make the overall yields predictable.

3.2 Methodology: Structure-Driven Crawling

The observation of the structure locality motivates us a new concept, *structure-driven* crawling, as a framework for building object-focused crawlers. This concept parallels and contrasts the implicit notion of *content-driven* crawling framework behind existing topic-focused crawlers, as Section 1 introduced. In a structure-driven framework, a crawler conceptually partitions the Web into independent scopes and searches for target objects in each scope, with certain intra-scope search strategy that matches the object distribution patterns. If such structure-locality patterns indeed exist, the in-scope search strategy can explore different ways to achieve predictable harvest and coverage for the crawling.

For each scope, we crawl from its entrance to search within the scope, guided by an *in-site* search strategy. By matching the structure locality of the scope, different strategies will result in different tradeoff of yield rates in this scope, or *local harvest* h and *local coverage* $c\%$. To confidently predict the "global" harvest H and coverage $C\%$ of the entire crawling process from our local yields, we develop a high-level methodology for structure-driven crawling.

First, *sampling phase*: Suppose we have a set of alternative intra-scope search strategies of crawling a certain type of objects. Our goal in the sampling phase is to select the best strategy by testing the strategies over a randomly sampled set of scopes [2]. To show that such a sampling phase can indeed help us predict the overall performance, we need to address two issues: 1) We need to show that, by choosing an appropriate sampling size, we can guarantee a confident estimation of the local yields of a strategy. 2) We can predict the global yields from the local yields. We will discuss these two issues respectively in this section.

Second, *executing phase*: We apply the selected strategy for crawling over the whole Web. With the accurate estimation of the local yields based on the sampled scopes, we thus can accurately predict the global yields in this phase. Our empirical study in Section 6 shows that the executing phase can indeed maintain steady harvest and coverage in practice.

To illustrate why local yields can be accurately estimated and they can further imply global yields in structure-driven crawling, let us for now consider a scope as a Web site and its entrance as the root page. Assume we use a simple strategy, *Exhaustive*(3), for crawling pages in a site from the root (*i.e.*, depth 0) up to depth 3.

Local Harvest h and Coverage $c\%$: As the same intra-scope crawling strategy may not generate the same local yields for different Web sites, we wonder whether we can observe stable local yields and further estimate them (in a statistical sense) with a randomly sampled set of sites. According to the Central Limit Theorem [3], when the sample size is large (usually more than 30), we can calculate confidence intervals of the mean values of the local harvest and local coverage using Equation 3. That is, the mean value μ of a random variable X has $1-\alpha$ probability to be in the range $[\overline{X}_n - \frac{z_{\alpha/2}s}{\sqrt{n}}, \overline{X}_n + \frac{z_{\alpha/2}s}{\sqrt{n}}]$, where n is the sampling size, X_i is the ith sample, $\overline{X}_n = \frac{\sum_{i=1}^{n} X_i}{n}$, and $s^2 = \frac{\sum_{i=1}^{n}(X_i - \overline{X}_n)^2}{n-1}$. For instance, with a trial of sampling 1000 sites, we have that the 95% confidence intervals of the local harvest mean and the local coverage mean of *Exhaustive*(3) are 0.119 ± 0.02 and 0.898 ± 0.016 respectively.

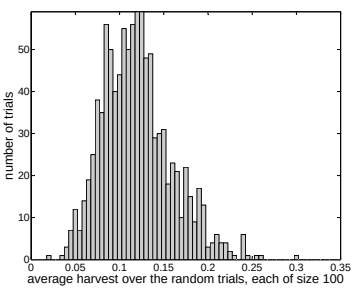
$$P(\overline{X}_n - \frac{z_{\alpha/2}s}{\sqrt{n}} < \mu < \overline{X}_n + \frac{z_{\alpha/2}s}{\sqrt{n}}) \quad = \quad 1-\alpha \qquad (3)$$

To visually illustrate the above estimation of confidence intervals, we conduct 1000 trials, with each trial sampling a different set of 100 sites. We still use *Exhaustive*(3) as the intra-scope strategy to crawl each site. We then compute the average harvest and coverage for each trial and draw the distribution of the average harvest and coverage among the 1000 trials, as Figure 5 shows. We can clearly observe that the mean values of both local harvest and coverage show normal distributions, with most values (about 95%) falling into the estimated confidence intervals.

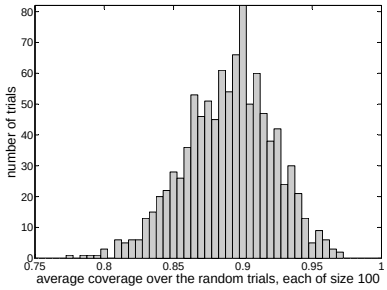
In practice, if we feel the confidence interval we get is not convincing enough to estimate the local yields, we can enlarge the sampling size. According to Equation 3, by doing so, we can obtain a smaller confidence interval for the same confidence $1-\alpha$ and thus a better estimation. From our experience, Web sites tend to share the structure locality shown in Figure 3. Therefore, we can often achieve accurate estimation of local harvest and coverage with a relatively small sampling size. Our experiment in Section 6 empirically verifies this argument for a set of different intra-scope crawling strategies, *e.g.*, the approaches we developed in Section 4.

[1] With IP aliasing and virtual hosting, there is generally a many-many mapping between IP and domain names. To be precise, a site should be recognized by (domain-name, IP, port-number).

[2] The criterion of judging the best is specific to the crawling task. For instance, a possible criterion can be choosing the strategy with the highest harvest among all strategies satisfying a given coverage.



(a) Normal distribution of the mean harvest.



(b) Normal distribution of the mean coverage.

Figure 5: Normal distribution of the mean yields.

Global Harvest H and Coverage $C\%$: How does the local performance imply the global yields in the entire crawling? We build a simple analytical model for this "prediction": We assume that query forms can only have duplicates in the same site (*e.g.*, Amazon.com has "product search" repeated in every page), and we consider forms from different sites as distinct (thus a form at *Amazon*.com cannot be found at *BN*.com). Assume the following characteristic parameters: 1) There are totally n Web sites. 2) In average, each site has m query forms. 3) The in-site searcher can achieve local harvest h and local coverage $c\%$ in average in a site.

We can now derive the global performance: During the entire crawling, the crawler will find $n \times m \times c\%$, among the total $n \times m$ forms. The number of pages crawled is $\frac{n \times m \times c\%}{h}$, since it crawls h times more pages than forms. Thus:

$$\text{Global harvest } H \quad = \quad \frac{n \times m \times c\%}{\frac{n \times m \times c\%}{h}} = h \qquad (4)$$

$$\text{Global coverage } C\% \quad = \quad \frac{n \times m \times c\%}{n \times m} = c\% \qquad (5)$$

Although it is possible to build more sophisticated modeling to more accurately capture the relationship between local yields and global yields, our empirical study in Section 6 shows that the simple modeling is pretty good in predicting the global yields.

Overall, we thus observe two desirable properties entailed by structure-driven crawling:

- **Steady local yields and predictable global yields:** In structure-driven crawling, we can accurately estimate the steady local harvest and coverage with an appropriate sampling size. Such steady local yields will help us to predict the global performance. Such features cannot be supported by traditional topic-focused crawling.
- **Yield-guided crawler design:** The analytical model, while simple, can guide the design of intra-scope search strategies (*e.g.*, selecting d for *Exhaustive*(d)). Guided by a desirable H and $C\%$, we can pick an intra-scope strategy that generates the corresponding local h and $c\%$, thus allowing a principled way of harvest-coverage tradeoff and resource allocation.

Although our analyses above assume Web sites as the scopes, we believe they are generally applicable to other scope definitions as long as some topic-neutral structure locality can be found within scopes.

3.3 Implementation: Architectural Design

To realize the structure-driven crawling framework for collecting query forms, we develop the *Web Form Crawler*. Our crawler conceptually takes a site-based view of the Web, as Figure 4(b) shows. In this view, each site is an independent scope, with an expected structure locality of our target objects. (With further analysis, more refined patterns can be constructed; Section 4.) We thus substantiate the concept of structure-driven crawling into a *site-based* framework: Crawl each site as a scope, with an *in-site* strategy that matches the locality, to achieve the objective of having predictable harvest and coverage.

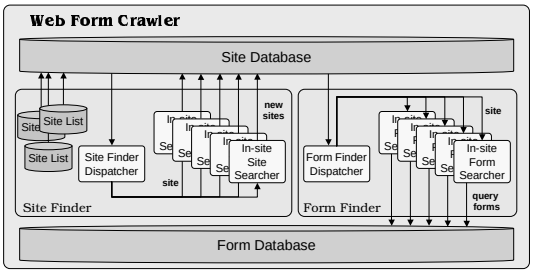


Figure 6: System architecture of the Web Form Crawler.

Figure 6 shows the architecture of the Web Form Crawler, with a pair of concurrent components: *Site Finder* for finding site entrances and *Form Finder* for searching each site for query forms. In this site-based framework, the finding of site IPs and the subsequent in-site search will run concurrently. In particular, the Site Finder collects new site IPs into a *Site Database*. From there, the Form Finder then continuously gets a site entrance, searches for query forms, and collects them into a *Form Database*.

Site Finding: To begin with, we need to find a set of sites in terms of entrance IP addresses. There are multiple ways to collect site IP entrances. First, *dynamic discovery*: We can build a crawler to specifically search for site entrances from Web pages. Second, *directory databases*: Some Web directories provide pre-compiled site lists. For instance, DMOZ has compiled a list containing 860,000 sites [22]. Third, *piggyback crawling*: We can add site-IP discovery as a "side effect" of other crawling activities. In particular, as our crawler searches for query forms in-site, it can also "piggyback" IP entrances found alone the way as byproducts. Our Site Finder currently supports two ways of collecting sites: from directory databases (*e.g.*, DMOZ) and from dynamic discovery.

Although our framework will employ multiple means of site finding, with the changing and expanding nature of the Web, we believe dynamic discovery remains essential for covering comprehensive all site-IPs. As a support, Section 6 will compare our dynamically discovered IPs with the DMOZ list, which reveals that such (manually) compiled list can be rather limited.

Such dynamic discovery of IPs turns out to be itself an object-focused crawling task (with "sites" as target objects) and can thus also be realized by structure-driven techniques. We observe the same phenomenon as query forms– That is, pages containing new site IPs are close to root pages. We randomly select 100 sites from the DMOZ site list and crawl each site up to depth 10. (We name this dataset as *Random100*, which will be used throughout this paper.) We measure the distribution at each depth, as Figure 3 shows. The result indicates that structure locality indeed exists: 95% IPs can be found within depth 3. (Section 5 will further refine the lo-

```
Algorithm: GENERALINSITESEARCHER:
Input: a site IP ip, maximal depth d
Output: a set of discovered objects from site ip
begin:
1  Q = ∅ /* Q: the queue of urls to be crawled */
2  B = ∅ /* B: blacklist: the set of urls already crawled */
3  I = ∅ /* I: the set of objects found in site ip */
4  Q.enqueue(ip)
5  while Q ≠ ∅
6    /* get a url to crawl and then add it into the blacklist*/
7    url = Q.dequeue()
8    page = retrieve the page of url
9    B = B ∪ {url}
10   /* add new objects in the crawled page page into I */
11   O = OBJECTEXTRACTION(page)
12   I = I ∪ O
13   /* select promising intra-links to crawl */
14   L = LINKSELECTION(page)
15   for each link u ∈ L and u ∉ B ∪ Q and DEPTH(u) ≤ d
16     Q.enqueue(u)
17 return I
end
```

Figure 7: Algorithm GENERALINSITESEARCHER.

cality.) Similar to finding query forms, we can resort to structure-driven crawling for site finding. Therefore, while our site-based crawling framework relies on the function of site finding, this function, recursively, can be realized in the same site-based framework.

The Site Finder thus shares the same design as the Form Finder: Within the Site Finder, we schedule site IPs (that are already in Site Database) to search; for each site, we devise an in-site searcher. Hence, our discussion next on site scheduling and in-site search are applicable for both the Form Finder and the Site Finder. (Their different in-site strategies will be explored in Sections 4 and 5).

Site Scheduling: After collecting sites as scopes, we must develop a scheduling strategy, to order these scopes for in-site search of query forms. There are various alternatives in scheduling: To begin with, *simple iteration* orders all sites arbitrarily, and crawls each till completion. This scheme requires minimal scheduling logic, but may not optimize for important sites, and may not interleave crawling traffic to a single site. To contrast, we can use *ranked interaction* to prioritize site rankings with estimated importance (*e.g.*, some "PageRank"). Similarly, we can adopt *round-robin iteration* to go in "rounds," each of which crawls progressively larger part into a site (*e.g.*, depth d in round d), and thus interleaves site traffic.

Our implementation currently uses simple iteration, for its simplicity. In particular, our experience shows that the concern of site traffic is not significant: Since we aim at searching each site minimally (by exploiting structure locality), the traffic is often rather minor. We emphasize that, for a given set of sites to crawl, different scheduling strategies will *not* affect the global yield (as Eq. 4 and 5 show), since in principle we will eventually crawl all the sites.

Specifically, to schedule, the dispatchers in Figure 6 send site IPs to the concurrent in-site searchers at parallel machines. Note that, since our structure localities suggest that target objects are connected and reachable from their site entrances, our structure-driven framework will search a scope *independently*, without requiring cross-site communication [11]– parallelization is thus immediate.

In-Site Search: We develop a generic in-site search logic. As Figure 7 outlines: URLs to be crawled are added into a queue Q. In each while-loop, the searcher gets a URL from Q to crawl and extract objects (either site entrances or query forms in our case) from the page by calling OBJECTEXTRACTION. It then selects links to crawl by executing LINKSELECTION and adds these links to Q. The parameter, maximal depth d, controls the depth of crawling. The process terminates when Q is empty.

The function OBJECTEXTRACTION extracts target objects from a page. While this extraction is necessary, it is not our focus in this paper, and we only briefly explain our implementation: Extracting site IPs is straightforward– We identify inter-site hyperlinks, extract IPs (*e.g.*, `foo.com:8080/abc.html` to `foo.com:8080`), and store them to the Site Database. However, extracting query forms is more involved: For each potential form, as marked by the HTML tag <FORM>, we first decide if it is indeed a query form, to avoid non-interesting forms (for our purpose), *e.g.*, site searches, logins, and polls. We implement a form-detection classifier similar to [13] for this decision. For each positive form, we then remove duplicates (by comparing to forms already found in the same site), extract its query structure (by the visual parser in our earlier work [29]), and store it into the Form Database.

Our remaining task is thus to design effective in-site search strategies, *i.e.*, to substantiate the LINKSELECTION function, as guided by the objective harvest and coverage. We study such strategies for the In-Site Form Searcher and the In-Site Site Searcher in Sections 4 and 5, respectively. The development of such strategies, albeit for different objects, essentially follow the same approach: First, to explore structure locality, we will start with making deeper *observations* to find more structure locality *patterns*. Second, guided by the patterns, we then formulate search *strategies* and provide our specific *implementations*. Such strategies often are configured with parameters (*e.g.*, maximal depth d) which will lead to different harvests and coverages. Finally, we use the *sampling-and-executing* methodology (Section 3.2) to select a good intra-scope strategy that leads to desirable performance.

4. IN-SITE FORM SEARCHER

In this section we develop the In-Site Form Searcher for efficiently finding query forms within a site. Section 3.1 discussed the simple strategy *Exhaustive(d)*, which can already outperform the base harvest rate with reasonable coverage. However, can we do better? In this section we observe further structure locality (in addition to the "shallow distribution" mentioned in Section 3.1) specific to finding query forms. In realizing it, as Section 3.3 outlined, we discuss our observations and discovered patterns for the structural locality of query forms (Section 4.1), then formulate strategies and concrete implementations (Section 4.2).

4.1 Observations and Patterns

Observations: We observe that, as a common feature, most Web sites provide navigational menus to guide users in browsing the sites. Such navigational menus are often presented in order to bring users to important pages, among which of particular interests to us are those containing query forms. To be more concrete, Figure 8(a) shows the navigational menu at *http://www.bn.com*. By following the link at the tag `"BOOKS"`, we go to another navigational menu (Figure 8(b)) that contains a simple query form and the link `"More Search Options"` to the advanced query form of the book department of *Barnes&Noble*. In fact, we can reach the query forms of all major departments of *Barnes&Noble* by similarly following the links on the tags in Figure 8(a). Figure 9 illustrates a variety of navigational menus from real-world Web sites.

To verify that the structure locality provides high coverage in finding query forms, we surveyed 100 query forms from the UIUC Web integration repository [9]. These forms are randomly selected from forms that are not on root pages (such root-page forms are always covered, as the In-Site Form Searcher starts with the root page). We find that 87 out of the 100 forms can be reached from the root pages by following navigational links[3].

[3]The remaining 13 forms can mostly be reached by other simple heuristics. For example, most links to "advanced" query forms that could not be directly reached by navigational links are around the

Figure 8: Navigational links in *BN.com*.

Figure 9: More navigational links.

Reachable Patterns: *First*, as navigational links serve the purpose of connecting from everywhere to the important information of a site, they naturally reach the query forms, which provide a crucial functionality of the site. *Second*, as mentioned in Section 3.1, query forms distribute "shallowly" in Web sites, *i.e.*, they are close to the entrances.

4.2 Strategy and Implementation

Strategy: There can be hundreds of intra-site links on a Web site. How do we effectively find the navigational links among them? Our method is based on the following two insights.

First, navigational links in a page are presented with distinguishing visual characteristics, such as alignment, position, size, color, and font. Such characteristics enable the prominent and intuitive visual presentation of navigational links so that users can easily identify them. This observation suggests to analyze the visual pattern of hyperlinks to detect the navigational links. Note that such *visual parsing* approach has also been applied in understanding Web query forms [29].

Second, navigational links are often part of a page "skeleton" of a site that repeatedly appears in many pages in the site. Such repeating occurrences present the navigational links as shortcuts available everywhere in the site, thus the important information is always reachable. This observation suggests to enforce a *link overlap* analysis across multiple pages.

Implementation: Based on the above insights, we specialize the LINKSELECTION function in Figure 7 to an in-site search strategy, Algorithm NAVMENU, for detecting navigational links. Given a page p containing a set of hyperlinks L, it returns a set of candidate navigational links $L_{nav} \subseteq L$. The details of NAVMENU are shown in Figure 11. At the high level, the algorithm takes two stages, corresponding to the aforementioned two insights, as illustrated in Figure 10.

The first stage, *link grouping and ranking*, explores the visual characteristics for selecting and ranking navigational links. The hyperlinks in L are formed into multiple disjoint *link groups* based on visual parsing. To begin with, a link group is a set of more than, say 3, consecutive links representing a menu, aligned horizontally or vertically. This captures the visual characteristic of most navigational menus in terms of alignment. Further we rank the link groups by other visual characteristics. For example, the top-3 link groups are annotated in Figure 10. More specifically, we compute the ranking score of a link group g as the weighted average of three factors. That is, $rank_g = w_s \times size_g + w_w \times wordcount_g + w_d \times dist_g$, where $size_g$ is the number of links in g , $wordcount_g$ is the percentage of anchortext words in all the words positioned in the range of g, and $dist_g$ is the distance between g and the bottom of the page. We note that this formula is simply to capture the important visual characteristics of a link group as a menu, although other heuristics can be developed as well.

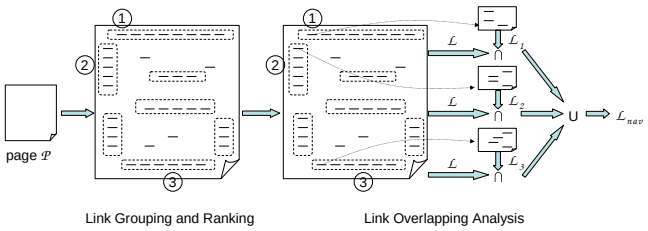


Figure 10: NAVMENU: navigational link detection.

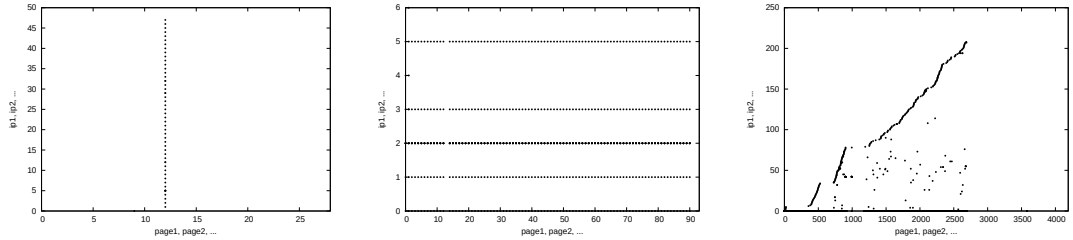
The second stage, *link overlap analysis*, identifies the links that repeatedly appear. This captures the observation that the same navigational links are likely to appear in the page itself and the pages referred to by the navigational links. To be more specific, given a navigational link $l \in L$ in p (*e.g.*, Figure 8(a)), the target page of l (*e.g.*, Figure 8(b)) likely contains l as well. From each of the top k (*e.g.*, $k = 3$ in Figure 10) ranked groups, x links (*e.g.*, the first link) are followed. The links in each resulting page (*e.g.*, L_1) are intersected with L, the links in p. The resulting intersections are unioned to form the set of candidate navigational links L_{nav}.

The visual information utilized in NAVMENU can be obtained from the Web page rendering engine of browsers such as *IE* and *Mozilla*. To ensure the high efficiency of the crawler, we use the open source browser *Lynx*, which renders a Web page in the text mode. For instance, the dumping output of *Lynx* corresponding to Figure 8(a) is shown in Figure 12. (The association of the anchortext and the target URL of each hyperlink is not shown.) The (anchortexts of) hyperlinks aligned horizontally in a graphical browser will appear in the same line of the textual output. Similarly the hyperlinks aligned vertically will appear as multiple lines. Note that although *Lynx* only provides approximate visual presentation of a page, compared to graphical rendering engines, and our algorithm even only exploits partial information (such as alignment, size, and distance) from the output of *Lynx*, the result is quite satisfactory according to our experiments in Section 6. Further improvements of NAVMENU can be made by exploring more visual characteristics related to navigational links.

Local Harvest and Coverage of Forms: The In-Site Form Searcher captures the reachable pattern of query forms with respect to navigational links, thus can achieve both high harvest and coverage. It only follows navigational links, therefore crawls much less pages than the baseline approach of exhaustively following links, and thus has higher harvest. Navigational links can lead to 87% of query forms (that are not on root pages), therefore the coverage of the form searcher can reach higher than 87% since many forms can be found on root pages.

Moreover, to capture the "shallow" distribution pattern of query forms, we combine NAVMENU with the simple strategy *Exhaustive(d)* in Section 3.2 to obtain NAVMENU(d), which follows navigational links within the maximal depth d. The appropriate d for NAVMENU(d) will be empirically determined by the experiments in Section 6. In practice, we use the sampling-and-executing methodology dis-

simple query forms that can. Here we focus on the navigational links only and do not consider other heuristics.



(a) Link-page template. **(b)** Skeleton template. **(c)** Directory template.

Figure 13: Three typical templates of the distribution of IPs within a site.

Algorithm: NAVMENU (p):
Input: a page, p
Output: a list of selected intra-site links, $links$
begin:
1 /* obtain the lines of texts and the URLs by *Lynx* */
2 ($< l_1, ..., l_n >, < u_1, ..., u_m >$) = lynx(p)
3 /* link grouping*/
4 $t = 0$ /* total number of link groups */
5 **for** each line l_i, $1 \leq i \leq n$ **do**
6 /* NAW: non-anchortext words */
7 **if** line l_i contains single URL u_j and no NAW **then**
8 **if** line l_{i-1} contains multiple URLs or NAW **then**
9 $t = t$+1
10 $g_t = g_t \cup \{u_j\}$
11 /* all the words in l_i are the anchor text of u_j */
12 $wc[g_t]$ = $wc[g_t]$ + number of words in l_i
13 **elif** line l_i contains multiple URLs $\{u_j, ..., u_{j+k}\}$ **then**
14 $t = t$+1
15 $g_t = \{u_j, ..., u_{j+k}\}$
16 /* including both anchortexts and NAW */
16 $wc[g_t]$ = number of words in l_i
17 /* a group should have at least 3 links */
18 remove those groups with $|g_i| \leq 2$
19 /* link group ranking */
20 **for** each group $g_i = \{u_j, ..., u_k\}$, $1 \leq i \leq t$ **do**
21 $total\_size$ = $total\_size$ +$|g_i|$
22 $total\_words$ = $total\_words$ + $wc[g_i]$
23 $total\_dist$ = $total\_dist$ + m - $(j + k)/2$
24 **for** each group $g_i = \{u_j, ..., u_k\}$, $1 \leq i \leq t$ **do**
25 $rank[g_i] = \frac{w_s \times |g_i|}{total\_size} + \frac{w_w \times wc[g_i]}{total\_words} + \frac{w_d \times (m-(j+k)/2)}{total\_dist}$
27 /* link overlap analysis */
28 $ls = \emptyset$
29 **for** each of the top k ranked groups g **do**
30 ls = $ls \cup$ {the first x links in g}
31 $P\_url = < u_1, ..., u_m >$
32 $links = \emptyset$
33 **for** each link $u \in ls$ **do**
34 $child = retrieve(u)$
35 C_url = URLs in $child$
36 $links$ = $links \cup (P\_url \cap C\_url)$
37 **return** $links$
end

Figure 11: Algorithm NAVMENU.

cussed in Section 3.2 to estimate the local yields, as we will empirically illustrate in our experiments.

5. IN-SITE SITE SEARCHER

In this section, we discuss specializing the Algorithm GENERALINSITESEARCHER for the task of finding site IPs within a site. In particular, we need to specialize the LINKSELECTION function for selecting links that are likely to contain new sites. Similar to the procedure taken in Section 4, we discuss our observations and discovered patterns for the structural locality of site entrances (Section 5.1), from which we develop the link selection strategy and implementation (Section 5.2).

5.1 Observations and Patterns

Figure 12: The output of Lynx corresponding to Figure 8(a).

Observations: We study the occurrences of external site IPs in a site by surveying the 100 sites in the *Random100* dataset. For each site, we draw a "matrix" of IP occurrences. The x-axis is all pages in the site in their breadth first traversal order. The y-axis is all IPs in the site ordered by their first discovery (because an IP can occur in many pages of a site). If an IP occurs in a page, we mark the corresponding position in the matrix with a dot.

From all the occurrence matrices, we observe that the distribution of IPs in a site has three typical shapes, which we call "templates," as Figure 13 illustrates. 1) *Link-page template*, in which there is one (or a few) "link" page that contains many external IPs, while other pages have none. 2) *Skeleton template*, in which some external IPs occur in almost every page, mainly because these IPs appear in the common "skeleton" that many pages share. 3) *Directory template*, in which new IPs keep growing and show a triangle shape of distribution– That is, new IPs can occur in not only shallow pages but also deeper ones and thus form a triangle. A site with this template is often a directory site, *e.g.*, Yahoo directory and DMOZ directory, where each page contains IP references for a certain subject category. Finally, some sites may show a mix of these typical templates in their distributions. Among the 100 sites, 11 sites have no new IPs, 44 sites follow only link-page template, 8 only skeleton template and 8 only directory template. There are totally 29 sites that have mixed templates.

Reachable Patterns: From our survey of templates, we summarize two patterns to reach pages containing IPs: First, *target-page pattern*: Some pages are important by themselves. In particular, it is crucial to find the path to the "link" pages. Second, *continuous pattern*: Some sites contain external site IPs in a "continuous" distribution across depths. That is, searching more pages in a site will either continuously find different IPs (*i.e.*, the directory template) or the same ones (*i.e.*, the skeleton template).

5.2 Strategy and Implementation

Strategy: We design our link selection strategy by leveraging both reachable patterns. *First*, for sites of the target-page pattern, we observe that link pages are often either close to root page (*i.e.*, within depth 1) or contain some keywords (*e.g.*, "`links`", "`resources`"). We thus design our crawling strategy as: Crawling pages up to depth 1 and then for deeper pages, we build a classifier using anchortext keywords as distinguishing features to reach special link pages. *Second*, for sites of the continuous pattern, it is clear that we want

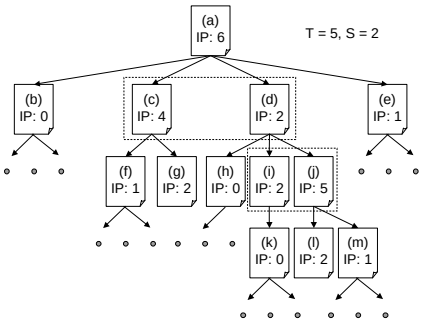


Figure 14: Adaptive crawling for finding IPs.

to leverage the continuity to use the past to "predict" the future and stop early if no more new IPs are found. We thus develop an adaptive crawling strategy, which dynamically decides whether to further crawl or not by its current crawling yields.

Specifically, the continuous distribution of IPs indicates that if we observe enough IPs from a group of recent pages, we are likely to see new IPs in their children. To realize this idea, we need to define what "recent" pages are and how many IPs are "enough." Given a page, we define recent pages as a sliding window of a group of S adjacent intra-site links in the page. For each window, we crawl its pages and compute the number of new IPs found in pages of the window. If the number of new IPs is no lower than a threshold T, we will crawl children pages for every page in the window.

Example 1: To illustrate the adaptive crawling with an example, consider a Web site as Figure 14 shows. For simplicity, we alphabetically mark each page, *i.e., a, b, ...*, instead of using URLs. Also, for each page, we give it the number of new IPs it yields. Suppose we set the window size S as 2 and the IP threshold T as 5.

To begin with, we crawl the root page *a* and find 6 new IPs[4]. Since 6 is more than the threshold 5, we will continue to crawl its children pages in depth 1, *i.e., b, c, d* and *e*.

Then, we use sliding windows to decide whether to crawl pages in depth 2 or not. As our window size is 2, the first sliding window contains pages *b* and *c*. Since their total new IPs is 4, which is less than 5, we will not crawl their children pages. We then move forward the sliding window. The second sliding window contains pages *c* and *d*, and has 6 new IPs. We thus crawl children pages of both pages *c* and *d*, and get pages *f, ..., j*. The last sliding window in this depth contains pages *d* and *e*, which has only 3 new IPs. We thus will not crawl page *e*.

Next, we repeat the above process for each depth of pages until no pages can be selected to crawl. In this example, we will further crawl children pages of pages *i* and *j*, since their total new IPs is 7. Since there are no other windows that can pass the threshold, the crawling will stop after crawling page *m*. ■

Finally, because the above strategies are only likely but not certain, we may still miss some pages containing new IPs. We thus introduce a random crawling behavior for pages that are originally not selected. Specifically, when a page is not selected by any of our strategy, we still give it a chance to be crawled with a small probability p, *e.g.*, 0.05. This random behavior complements our deterministic crawling strategies in a statistical sense.

Implementation: Putting together all the strategies we have discussed so far, we develop the overall Algorithm ADAPTIVE(T, S, p) to realize the LINKSELECTION function for finding Web sites, as

[4]Note that this example is just for illustration. As we will show in Algorithm ADAPTIVE, in practice, to avoid missing the entire site due to a low harvest rate at a single root page, we crawl all pages up to depth 1 regardless of the harvest rate of the root page.

```
Algorithm: ADAPTIVE (T, S, p):
Input: a page pg, IP threshold T, window size S, probability p
Output: a list of selected intra-site links
begin:
1  L = ∅  /* L: the set of selected intra-site links */
2  W = all intra-site links in pg
3  /* deal with target-page pattern */
4  if DEPTH(pg) ≤ 1 then L = L∪ {u| for all u ∈ W}
5  else L = L∪ {u| for all u ∈ W and LINKKEYWORDS(u) = true}
6  /* deal with continues pattern with adaptive crawling*/
7  /* get a set of pages with respect to the window size S */
8  pw = GETPAGEWINDOW(pg, S)
9  E = the set of new IPs found from pages in pw
10 /* check whether there are enough IPs in the window of pages */
11 if |E| ≥ T then L = L∪ {u| for all intra-site links u in pw}
12 /* add random crawling behavior */
13 elif we hit a probability p then L = L∪ {u| for all u ∈ W}
14 return L
end
```

Figure 15: Algorithm ADAPTIVE.

Figure 15 shows. In particular, lines 3-5 realize the crawling strategy for the target-page pattern. Function DEPTH is to return the depth of a page and function LINKKEYWORDS is to check whether the URL contains some keywords about link pages. Lines 6-11 realize the adaptive crawling strategy for the continuous pattern. Function GETPAGEWINDOW returns a window of pages with the given page pg as the last page in the window. Lines 12-13 realize the random crawling behavior.

Local harvest and coverage: Our algorithm, with its parameters, allows us to control the local harvest and coverage. We have three parameters to set: the IP threshold T and the window size S in adaptive crawling, and the random probability p. As our experiments in Section 6 show, the combination of these three parameters affect both the local harvest and coverage. It is thus possible to choose the parameters that are likely to have a good balance of local harvest and coverage with respect to user's requirements.

6. EXPERIMENTS

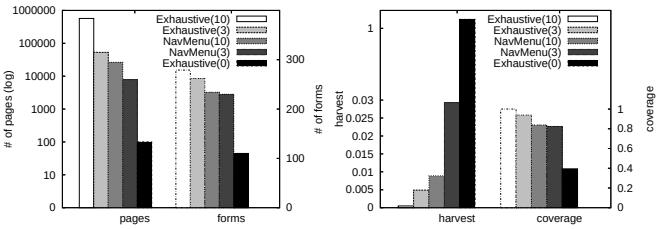
To evaluate the Web Form Crawler, we extensively test each of the core components as well as the entire system, for their (local and global) harvest and coverage. The experimental results verify that 1) by our sampling-then-executing strategy over a small sample of Web sites, we can compare various in-site search strategies and select the appropriate one; and 2) compared to page-based crawling, our best harvest rate is about 10 to 400 times higher, depending on the page traversal schemes used.

To begin with, we have implemented the Web From Crawler, as Figure 6 shows, with our control logic built upon several modified open-source softwares. In particular, we build our implementation of the in-site searchers (*i.e.*, the In-Site Site Searcher and the In-Site Form Searcher) based on wget (http://www.gnu.org/-software/wget/wget.html). For the In-Site Form Searcher, we revise the text-based browser *Lynx* (http://lynx.browser.org) to extract visual information of Web pages. We implement the dispatchers in C and use PostgreSQL (http://www.postgresql.org) to support the Site Database and Form Database.

We deployed the crawler, with its parallel architecture, on the HAL PC cluster at the University of Illinois at Urbana-Champaign (http://hal.cs.uiuc.edu). The HAL cluster consists of 100 dual processor machines each with two 500MHz Pentium III Xeon processors, 1 GB of memory and a 9GB SCSI drive. The database servers have Xeon 2.80 GHz dual CPU with 2GB memory.

We extensively test the Web Form Crawler in its core components as well as the entire system:

1) Form Finder: We evaluate the local and global performance of the Form Finder in terms of its harvest and coverage.



(a) Crawling result. **(b)** Performance.

Figure 16: Form Finder: Local study.

2) Site Finder: We briefly evaluate the performance of the Site Finder in terms of its harvest and coverage.

3) Overall: We evaluate the Web Form Crawler with a large scale crawling and report the crawling result. We also compare our performance with the one using traditional page-based crawlers.

1a. Local performance of the Form Finder: In this study, we measure the local harvest and coverage of the In-Site Form Searcher under various settings. We randomly choose 100 deep Web sites from the TEL-8 dataset of the UIUC Web Integration Repository [9] as our test set. For each site, we run the In-Site Form Searcher in three cases: Maximal depth d as 0, 3, and 10, denoted as *Navmenu*(0), *Navmenu*(3), and *Navmenu*(10), respectively. For each case, we measure its local harvest and coverage. As the baseline, we also run the simple strategy of crawling all pages with depth 0, 3 and 10, (*i.e.*, *Exhaustive*(0), *Exhaustive*(3) and *Exhaustive*(10), as Section 3.2 introduced). Figure 16(a) shows, for each case, the number of pages crawled and the number of forms found. Figure 16(b) shows, for each case, the local harvest rate and coverage. As *Exhaustive*(0) is effectively the same as *Navmenu*(0), we only list the result of *Exhaustive*(0).

The result in Figure 16 is consistent with our analysis in Section 3.2: With a deeper depth, the harvest is lower, while the coverage is higher. Meanwhile, the further structure locality developed in Section 4, *i.e.*, navigational menus, developed in Section 4 indeed results in better performance. By following navigational menus, we can significantly speed up the harvest while in the meantime maintain a high coverage. In particular, *Navmenu*(3) is the best setting, with a good balance between harvest and coverage.

We note that as the query form classifier (developed in Section 3.3) may have false positives, *i.e.*, some non-query forms may be classified as query forms, to have an accurate evaluation of the local harvest and coverage in this small scale study (in contrast to the large scale global study later), we perform a manual inspection to verify query forms and duplicates. The result in Figure 16 is the one after manual inspection.

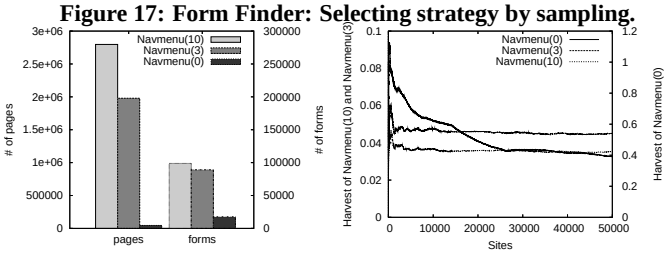
Our manual inspection shows that the more pages we crawl, the more false positives we have in the collected forms. For instance, in depth 0, only 9% forms are false positives, while in depth 10, 60% to 70% forms are. Also, exhaustive crawling, as it crawls more pages, has more false positives than navigational menu crawling. Further, as the 100 deep Web sites we choose in this local study are all "big" sites, the harvest is likely to be underestimated, because for smaller sites, we may not need to crawl many pages to find forms. Therefore, in practice, the global harvest in large scale crawling, where no manual inspection is taken, may increase over 5 times, as our following experiments will show. We believe a more accurate classifier should be and can be developed, but such a topic is beyond the scope of this paper.

1b. Sampling to select the strategy of the Form Finder: To select a good in-site search strategy for the Form Finder, we follow the methodology of sampling-then-executing that is mentioned in Section 3.2. In the sampling stage, we apply various strategies over a small sample of 1000 Web sites. For each strategy, we compute its local harvest and coverage over each individual Web site. According to the Central Limit Theorem, following the method in Section 3, we obtain the 95% confidence interval (CI) for the mean harvest ($\overline{h}$) and mean coverage ($\overline{c}$), respectively, of the underlying population over the whole Web. The results for the five crawling strategies of the Form Finder are shown in Figure 17.

method	*mean harvest*	95% *CI*
Exhaustive(10)	0.114	0.020
Exhaustive(3)	0.119	0.020
Navmenu(10)	0.192	0.028
Navmenu(3)	0.214	0.029
Exhaustive(0)	0.537	0.049

method	*mean coverage*	95% *CI*
Exhaustive(10)	1.0	0.0
Exhaustive(3)	0.898	0.016
Navmenu(10)	0.630	0.026
Navmenu(3)	0.598	0.026
Exhaustive(0)	0.287	0.025

Figure 17: Form Finder: Selecting strategy by sampling.

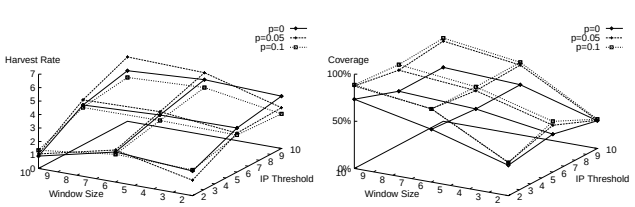


(a) Crawling result. **(b)** Trend of global harvest.

Figure 18: Form Finder: Global study.

In the sampling procedure, and the following large scale global study and the evaluation of the entire system, we explore automatic query form detection mainly based on a rule-based classifier similar to [13]. However, this automatic detection can result in false positives in both form detection and duplicate removal. The global harvest rate thus will be higher than the one in local study due to the existence of false positives. For instance, one particular error is that the classifier often makes a mistake on considering product configuration forms as query forms, which we specifically removed in the manual inspection. A product configuration form is an HTML form used for configuring features of a specific product, *e.g.*, selecting options of a specific car. For E-commerce sites, it is quite often that each product may have a unique configuration form and thus there are a large number of such forms. Therefore, misclassifying this type of forms will result in a significant increase of harvest rate. However, as this issue affects every strategies, we still obtain accurate comparison of the strategies. For example, although the mean harvest obtained from the sampling procedure may not be the same as the real mean harvest across the underlying population, the mean harvests of different strategies still indicate their performance ranking when compared with each other.

1c. Global study of the Form Finder: We then evaluate the global performance of the Form Finder with a large scale crawling. We execute the Form Finder in three cases: Using *Navmenu*(0), *Navmenu*(3), and *Navmenu*(10) as the In-Site Form Searcher respectively. We crawl the same set of 50,000 sites for all cases and compare their performance. Figure 18(a) shows, for each case, the number of pages crawled and the number of forms found. Figure 18(b) shows, for each case, the trend of global harvest rate. The result shows that, after crawling a few sites, the harvest rate of the Form Finder is quickly stabilized. The harvest of *Navmenu*(0) takes longer to get stable than the other two because it only crawls



(a) Harvest. **(b)** Coverage.

Figure 19: Site Finder: Local performance.

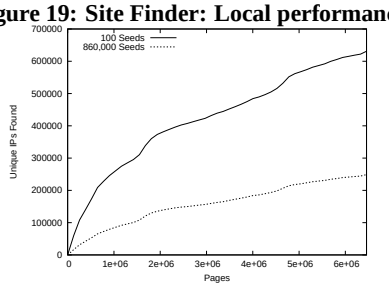


Figure 20: Site Finder: Global performance.

one page from every site. This result is consistent with our analysis in Section 3.2– That is, the structure-driven crawler can indeed maintain stable harvest.

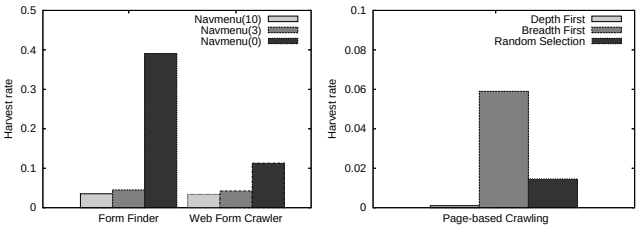
2. Performance of the Site Finder: While we have given detailed analyses of the Form Finder above, we briefly summarize the results of the Site Finder, as the main goal of our Web Form Crawler is to collect query forms.

We first evaluate the performance of the Algorithm ADAPTIVE in the In-Site Site Searcher. As Section 5 discussed, by tuning the three parameters in ADAPTIVE, *i.e.*, the IP threshold T, the window size S and the random probability p, we should be able to control the local performance of the In-Site Site Searcher for finding site entrances. To verify this argument, we evaluate the In-Site Site Searcher over the *Random100* dataset with a set of combinations for $T \in \{2, 5, 10\}$, $S \in \{2, 5, 10\}$ and $p \in \{0, 0.05, 0.1\}$. For each combination, we evaluate its local harvest and coverage. In all the executions, we set the maximal crawling depth d (in Algorithm GENERALINSITESEARCHER) as 10, which is deep enough to cover almost all pages. Figure 19 shows the performance.

From Figure 19, we observe interesting trade offs between local harvest and local coverage. In general, when the parameters allow more pages to be crawled (*i.e.*, T is smaller, S is larger, and p is higher), the coverage will be higher, while the harvest will be lower. Overall, any of three parameters can affect the trade off between harvest and coverage. It is thus possible to choose an appropriate parameter setting according to user's desired crawling goal. We apply the sampling method to choose the parameter setting, *i.e.*, the specific in-site search strategy, similar to the procedure in Form Finder. For example, the medium values for all the three parameters, *i.e.*, $T = 5$, $S = 5$ and $p = 0.05$, can achieve good harvest as well as reasonable coverage.

With the chosen strategy, we evaluate the performance of the Site Finder over the HAL cluster by crawling a large set of sites. We test the Site Finder in two cases: Starting from the small *Random100* dataset of 100 IPs and from the large DMOZ list of 860,000 IPs. The top curve in Figure 20 shows the global harvest of starting from *Random100* and the bottom one from the DMOZ list. Comparing the two curves, we can see that the more IPs we have in the Site Database, the lower global harvest the Site Finder achieves. However, even with the large starting set of 860,000 IPs, the harvest rate is still reasonably good.

The bottom curve in Figure 20 also indicates that the Site Finder can find many new sites that are not indexed by the DMOZ site list.



(a) Harvest of the Web Form Crawler. **(b)** Harvest of page-based crawling.

Figure 21: Evaluation of the entire system.

To measure the percentage of IPs we can find beyond the 860,000 IPs from DMOZ, we execute the Site Finder for a long time and collect 2,067,068 IPs, among which 703,691 overlap with the DMOZ list. That is, only 34% of IPs are indexed by DMOZ and 66% are not. Therefore, it is crucial to develop the In-Site Site Searcher for finding sites besides directly using pre-compiled site lists.

3. Evaluation of the entire system: We next evaluate the overall performance of the Web Form Crawler, including both Site Finder and Form Finder. In particular, the harvest rate of the Web Form Crawler becomes $\frac{\#FormsCollected}{\#PagesCrawledInBothFinders}$. Recall that, in the Site Finder, we have two ways to collect site entrances: Importing from site lists and crawling with In-Site Site Searchers. For the first situation, the Site Finder does not need to crawl pages and thus the harvest rate will be the same as the one in the global study of the Form Finder. For the second situation, as Site Finder also crawls pages, the harvest rate will be lower.

First, overall performance: We can use the result in the global study of the Site Finder to measure the harvest rate in the second situation. Checking our crawling result according to Figure 20, we know that starting with 100 sites as seeds, we crawled 110,814 pages to find 50,000 sites. By counting these pages, we can compute the harvest of the Web Form Crawler. Figure 21(a) compares the harvest of Web Form Crawler with the corresponding one of Form Finder. In all cases, after counting the pages crawled by the Site Finder, we can still achieve a good harvest rate, although the harvest is more significantly affected for the In-Site Form Searcher with smaller maximal depth.

Second, comparison to page-based crawling: We compare the harvest of site-based crawling and page-based crawling. As we argued in Section 1, page-based crawling, without focusing on the structure locality, may result in low harvest. We run the traditional page-based crawler to find query forms by simply following links. We test three common link following strategies in page-based crawling: Breadth first, depth first and random selection. For each strategy, we crawl about 50,000 to 150,000 pages and evaluate its harvest.

Figure 21(b) shows the result, from which we can see that, site-based crawling achieves better harvest than page-based crawling in finding query forms. For instance, using the depth first strategy, page-based crawling can only find 1 query form in every 1000 pages. Our highest harvest is about 400 times better than this depth first case. Breadth first strategy can achieve better harvest because by following external links and only crawling about 50,000 pages, the page-based crawler is very likely to crawl in the shallow part of many distinct sites and thus behaves similar to a site-based crawler. Even so, our highest harvest is about 10 times better than this breadth first case. Note that while the breadth first strategy "accidentally" explores the structure locality when crawling a relatively small portion of pages, our goal of structure-driven crawling is to formalize and explicitly explore such locality and balance harvest and coverage.

Since the harvest we list here is the one without manual inspection of query forms, we then wonder if the comparison is still valid. Our answer is yes. Recall that our manual inspection shows that the more pages we crawl, the more percent of false positives we have in the collected forms. Compared to site-based crawling, a page-based crawler will be more likely to touch the large number of pages in deeper depth and thus have more false positives. Therefore, our comparison here in fact disfavors the site-based crawling, although the result of site-based crawling is still better.

We notice that the initial harvest of page-based crawling in Figure 21(b) is higher than our average estimation of 6.6×10^{-5}. There are three reasons: First, since many query forms are duplicated in a large number of pages in their own sites, *e.g.*, the keyword book search in *BN.com* may appear in many pages, the initial chance to see a query form in page-based crawling is thus higher. When we crawl more and more pages, the harvest of page-based crawling will slow down and become worse and worse, since many query forms are already seen. Second, the false positive problem in the query form classifier also makes the harvest significantly higher than its real value. Third, our survey of the scale of query forms was done in April 2004. With the rapid growth of the deep Web, we believe there are more query forms available on the Web now.

7. CONCLUSION

This paper aims at building a crawler for collecting query forms on the Web. Although critical to information search and integration over the deep Web, such a problem has not been extensively studied. As a new attempt, we abstract this problem as object-focused, topic-neutral crawling and propose a structure-driven crawling framework for such a crawling task by observing the existence of structure locality of query forms. We develop the Web Form Crawler to realize the framework. The experimental results show that our crawler can maintain stable yields in the entire crawling process and thus we can pursue a yield-guided crawler design. Such features are not supported by existing focused crawlers. Compared to page-based crawling, our best harvest rate is about 10 to 400 times better, depending on the page traversal schemes used.

8. REFERENCES

[1] R. Baeza-Yates and C. Castillo. Balancing volume, quality and freshness in web crawling. In *Hybrid Intelligent Systems*, 2002.

[2] L. Barbosa and J. Freire. Searching for hidden-web databases. In *WebDB Workshop*, pages 1–6, 2005.

[3] P. Bickel and K. Doksum. *Mathematical Statistics: Basic Ideas and Selected Topics*. Prentice Hall, 2001.

[4] BrightPlanet.com. The deep web: Surfacing hidden value. Accessible at `http://brightplanet.com`, July 2000.

[5] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[6] S. Chakrabarti, K. Punera, and M. Subramanyam. Accelerated focused crawling through online relevance feedback. In *WWW*, pages 148–159, 2002.

[7] S. Chakrabarti, M. van der Berg, and B. Dom. Focused crawling: a new approach to topic-specific web resource discovery. In *WWW Conference*, 1999.

[8] K. C.-C. Chang, B. He, C. Li, M. Patel, and Z. Zhang. Structured databases on the web: Observations and implications. *SIGMOD Record*, 33(3):61–70, 2004.

[9] K. C.-C. Chang, B. He, C. Li, and Z. Zhang. The UIUC web integration repository. Computer Science Department, University of Illinois at Urbana-Champaign. http://metaquerier.cs.uiuc.edu/repository, 2003.

[10] K. C.-C. Chang, B. He, and Z. Zhang. Toward large scale integration: Building a metaquerier over databases on the web. In *CIDR Conference*, 2005.

[11] J. Cho and H. Garcia-Molina. Parallel crawlers. In *WWW Conference*, 2002.

[12] J. Cho, H. García-Molina, and L. Page. Efficient crawling through URL ordering. *Computer Networks and ISDN Systems*, 30(1–7):161–172, 1998.

[13] J. Cope, N. Craswell, and D. Hawking. Automated discovery of search interfaces on the web. In *Proc. of the 14th Australasian Database Conference*, pages 181–189, 2003.

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

[15] M. Ester, H.-P. Kriegel, and M. Schubert. Accurate and efficient crawling for relevant websites. In *VLDB*, pages 396–407, 2004.

[16] B. He and K. C.-C. Chang. Statistical schema matching across web query interfaces. In *SIGMOD Conference*, 2003.

[17] B. He and K. C.-C. Chang. Making holistic schema matching robust: An ensemble approach. In *KDD Conference*, 2005.

[18] B. He, K. C.-C. Chang, and J. Han. Discovering complex matchings across web query interfaces: A correlation mining approach. In *KDD Conference*, 2004.

[19] H. He, W. Meng, C. Yu, and Z. Wu. Wise-integrator: An automatic integrator of web search interfaces for e-commerce. In *VLDB Conference*, 2003.

[20] H. He, W. Meng, C. Yu, and Z. Wu. Automatic extraction of web search interfaces for interface schema integration. In *WWW Conference, poster paper*, 2004.

[21] S. Mukherjea. Wtms: A system for collecting and analyzing topic-specific web information. In *WWW Conference*, 2000.

[22] OpenDirectoryProject. DMOZ site list. http://rdf.dmoz.org/rdf/content.rdf.u8.gz.

[23] M. S. Panagiotis G. Ipeirotis, Luis Gravano. Probe, count, and classify: Categorizing hidden web databases. In *SIGMOD Conference*, 2001.

[24] S. Raghavan and H. Garcia-Molina. Crawling the hidden web. In *VLDB Conference*, 2001.

[25] S. Sizov, M. Theobald, S. Siersdorfer, G. Weikum, J. Graupmann, M. Biwer, and P. Zimmer. The BINGO! system for information portal generation and expert web search. In *CIDR*, 2003.

[26] J. Wang, J.-R. Wen, F. Lochovsky, and W.-Y. Ma. Instance-based schema matching for web databases by domain-specific query probing. In *VLDB Conference*, 2004.

[27] W. Wu, C. T. Yu, A. Doan, and W. Meng. An interactive clustering-based approach to integrating source query interfaces on the deep web. In *SIGMOD Conference*, 2004.

[28] YahooSearchBlog. http://www.ysearchblog.com/archives/000172.html.

[29] Z. Zhang, B. He, and K. C.-C. Chang. Understanding web query interfaces: Best-effort parsing with hidden syntax. In *SIGMOD Conference*, 2004.

[30] Z. Zhang, B. He, and K. C.-C. Chang. Light-weight domain-based form assistant: Querying web databases on the fly. In *VLDB Conference*, 2005.