Query Selection Techniques for Efficient Crawling of Structured Web Sources

Ping Wu[1]
University of California Santa Barbara
pingwu@cs.ucsb.edu

Ji-Rong Wen
Microsoft Research Asia
jrwen@microsoft.com

Huan Liu
Arizona State University
hliu@asu.edu

Wei-Ying Ma
Microsoft Research Asia
wyma@microsoft.com

Abstract

The high quality, structured data from Web structured sources is invaluable for many applications. Hidden Web databases are not directly crawlable by Web search engines and are only accessible through Web query forms or via Web service interfaces. Recent research efforts have been focusing on understanding these Web query forms. A critical but still largely unresolved question is: how to efficiently acquire the structured information inside Web databases through iteratively issuing meaningful queries? In this paper we focus on the central issue of enabling efficient Web database crawling through query selection, i.e. how to select good queries to rapidly harvest data records from Web databases. We model each structured Web database as a distinct attribute-value graph. Under this theoretical framework, the database crawling problem is transformed into a graph traversal one that follows "relational" links. We show that finding an optimal query selection plan is equivalent to finding a Minimum Weighted Dominating Set of the corresponding database graph, a well-known NP-Complete problem. We propose a suite of query selection techniques aiming at optimizing the query harvest rate. Extensive experimental evaluations over real Web sources and simulations over controlled database servers validate the effectiveness of our techniques and provide insights for future efforts in this direction

1. Introduction

Structured Web databases are becoming an increasingly popular habitat for relational data. Since data residing in Web databases is guarded by search interfaces and inaccessible to conventional Web crawlers, this portion of the Web is commonly referred to as "deep Web" or "hidden Web" [7]. To enable a "one-stop" access for such a wealthy amount of high quality data, there are two major kinds of design choices among others. One is the data warehouse-like approach [1,2], where the data is gathered from a large number of Web data sources and can be searched and mined in a centralized manner. The other is the "MetaQuerier" approach [10] that abstracts different "dialects" of Web databases by providing a mediated schema to the users. In addition to many differences between these two approaches, they have different goals in offline data acquisition. For the data-warehouse-based approach, integrating data from structured Web sources and collecting them to a centralized repository is a critical first step, since user queries are answered solely based on the data stored in the centralized warehouse. In contrast, MetaQuerier imposes fewer requirements on data acquisition because several query probes will normally suffice for query mapping and schema matching [24].

While it is still unclear whether the MetaQuerier approach or the crawling-based warehousing approach is more suitable to access the deep Web, data acquisition from structured Web sources is a pressing research issue in its own right. Many exciting applications can be developed using the high quality, structured data crawled from the hidden Web sources. For example, one can learn domain knowledge by feeding the data to existing learning techniques [20,23]. One can create Internet services such as "comparison shopping" by integrating data from different, potentially competing product providers. Hidden Web data can also be used to assist structuring other less structured documents. In fact, some major Web search engines are beginning to provide product search services based on the structured data gathered from a number of Web sources [1,2].

In principle, there are two ways to acquire the data from Web sources. The most efficient and effective way is to let the data providers export their backend databases by licensing their data so that their product information can be directly indexed by the product search services. Unfortunately, in an autonomous, uncooperative, and competitive environment like the public Web, this approach is difficult to scale up with the huge number of deep Web sites and potentially requires significant amount of time and extensive human efforts. The alternative approach, which is the focus of our paper, relies on a hidden-Web crawler to actively query the target product database and iteratively "uncover" the database content. Queries are issued either through the Web query forms or through the published Web services' interfaces (e.g., Amazon Web Service). Unlike the traditional crawling methods, query-based database

[1] Work was done when the author was visiting Microsoft Research.

crawling can be characterized by a "query-harvest-decompose" loop that incrementally uncovers a target Web database. It starts with some seed queries prepared in the form of attribute value pairs (e.g., "Actors, 'Hanks, Tom'", "Brand, 'IBM'", etc.) Then, it automatically queries the target data source by feeding the attribute values into appropriate query boxes or Web service procedures. It harvests the data records from the returned pages, which may be in the form of HTML Web pages or as XML documents. It populates the extracted records to its local database and decomposes these records into attribute values, which are stored as candidates for future query formulation. This process is repeated until all the possible queries are issued or some stopping criterion is met.

Previous efforts have primarily focused on the problem of automating the above process. To this end, technical challenges mainly lie in the "smart form filling" [22] and structured data extraction (e.g., [5]). However, a critical but largely unresolved problem is: how to select good queries to issue so that satisfactory database coverage can be achieved within affordable communication costs with the server? Intuitively, while the ultimate database coverage (or the coverage convergence) is predetermined by the seed values and the target query interfaces, the communication costs used to reach this coverage are greatly dependent on the query selection method used. In practice, it is often infeasible for a Web database crawler to query the target database exhaustively. Hence, an efficient query selection method is necessary that can achieve "good" database coverage with reasonable communication costs. We show that a sound solution to this problem is technically non-trivial. We also demonstrate that to achieve the same database coverage, a good query selection method can incur considerably less overhead than a naive one.

As one of the initial efforts to address this problem, the contribution of the paper is summarized as follows:

First, we identify this novel problem of query selection for efficient Web database crawling. Contrary to the traditional surface Web crawling that focuses on URLs' ordering, we demonstrate that a central issue of efficient deep Web crawling lies in query selection.

Second, we provide a theoretical framework that formally models query-based deep Web crawling as graph traversal. We show that the goal of query selection algorithms is to find a *Weighted Minimum Dominating Set* in the corresponding attribute-value graph.

Third, we study the database graphs of real structured Web databases and observe that the degree distribution in such graphs closely resembles the power-law distribution. Motivated by this case study, we propose a greedy link-based query selection method that rapidly traverses the database graph by following some "hub" nodes. In addition, we developed two optimization techniques to further optimize the greedy crawler. The effectiveness of these methods is demonstrated with experiments on 4 real world databases with 1 million records in total.

Fourth, we present another family of query selection methods that leverages on domain knowledge to overcome the limitations associated with the greedy methods. Its effectiveness is evaluated by a real crawling experiment on the Amazon DVD databases with domain statistics tables built from the Internet Movie Database (IMDB).

The rest of this paper is organized as follows: Section 2 presents our theoretical framework for query-based database crawling. Section 3 describes a greedy query selection method and its variants. A domain knowledge-based query selection algorithm is proposed in Section 4. We discuss our experimental findings in Section 5. Section 6 reviews some related work. Section 7 concludes this paper.

2. Modeling Query-based Database Crawling

2.1. Attribute-Value-based Graph Model (AVG)

We view a structured Web database as a single relational table *DB* with n data records $\{t_1,t_2,...,t_n\}$ over a set of m attributes $AS=\{attr_1,attr_2,...,attr_m\}$. The distinct attribute value set *DAV* consists of all distinct attribute values occurred in *DB*.

DEFINITION 2.1.: An *attribute-value graph* (AVG), *G(V,E),* for *DB* is an undirected graph that can be constructed as follows: for each distinct attribute value $av_i \in DAV$ there exists a unique vertex $v_i \in V$. An undirected edge $(v_i,v_j) \in E$ i.f.f. av_i and av_j coexist in one relational instance $t_k \in DB$. Each edge in AVG stands for a *relational link* between av_i and av_j. □

According to the definition, attribute values from each relational instance form a *clique*. If two data records share the same attribute value, the corresponding vertex "bridges" the two cliques. Depending on the database structure, an AVG is not necessarily fully connected. Although we only discuss the relational data model, AVG can also model more flexible data models (e.g., XML) graphically with straightforward extension.

By characterizing structured Web databases using AVG, a query-based crawling process is transformed into a graph traversal activity in which the database crawler starts with a set of seed vertices and at each step a previously seen vertex v is selected to visit (query formulation) and all the neighbors of v will be discovered (result extraction) and stored (decomposition) for future visit.

Example 2.1.: Figure 1 *shows a relational database (represented as a single table on the left) and its corresponding distinct attribute value graph. Consider a database crawler starts with the attribute value a_2 as its seed value. In the first round, it will query the database with a_2. From the returned results, it will see c_1,b_2,c_2,b_3.*

In the second round, if c_2 is selected and sent as query, a new record will be discovered with two new attribute values: a_3,b_4. In the final round, if c_1 is selected, the one remaining record will be obtained. □

a_1	b_1	c_1
a_2	b_2	c_1
a_2	b_2	c_2
a_2	$b3$	c_2
a_3	b_4	c_2

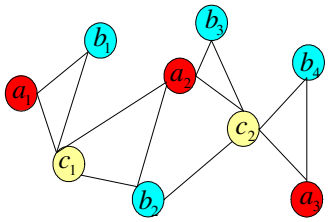

Figure 1. An example of attribute value graph

In summary, the significance of AVG is that it conceptually transforms the query-based Web database crawling into a graph traversal problem, which is parallel to the traditional problem of crawling the surface Web following hyperlinks.

2.2. Query Model

Structured Web databases can be queried via query forms or through Web service interfaces. We uniformly refer to both access methods as "query interfaces". Through query interfaces, data consumers (e.g. end users) are able to express their information needs by imposing selection conditions on certain attributes of interest.

DEFINITION 2.2. The database crawler views a Web database as a single relational table *DB* with a set of queriable attributes $A_q = \{attr_{q1}, attr_{q2},\ldots,attr_{qn}\}$ (interface schema) and a set of result attributes $A_r = \{attr_{r1}, attr_{r2},\ldots,attr_{rm}\}$ (result schema) [24]. Each $attr_{qi} \in A_q$ represents the queriable attribute through the query interface, while the result attribute $attr_{rj} \in A_r$ corresponds to the attributes displayed in the result pages. Each query operation can be modelled using SQL syntax as: "SELECT $attr_{r1}$, $attr_{r2},\ldots,attr_{rm}$ FROM *DB* WHERE $attr_{q1} = val_{q1}$, $attr_{q2}$,=val_{q2} ,…,$attr_{qn} = val_{qn}$", where val_{qi} is the corresponding attribute value filled into the query form.□

In this paper, we focus on the simplest selection queries with only one equality predicate. The reason for this choice is twofold: (1) The single attribute-value query is sufficient for query-based crawling in many domains. For example, many on-line product providers have already been supporting *keyword-based* queries to search their transactional data repositories. (2) Many product domains with more structured query interfaces also allow for single attribute-value queries (e.g., users can query Amazon.com with book title only). Below we conduct a real world case study to illustrate the applicability of this simplified query model in practice.

Case study: The applicability of the simplified query model in practice

Domain	K.W.	S.Q.M.
Book	82%	100%
Job	98%	96%
Movie	63%	100%
Car	14%	58%
Music	65%	100%

Domain	K.W.	S.Q.M.
DVD	78%	96%
Electronic	96%	96%
Computer	100%	100%
Games	91%	96%
Appliance	100%	100%
Jewellery	96%	100%

Table 1. Case study from 480 Web sources using UIUC Repository (the left Table) and Bizrate.com (the right table)

In this case study, we are interested in the percentage of structured Web sources that allow single attribute query. In addition, we are also interested to see how many Web product databases allow "keyword-based" search (structured sources with unstructured interfaces), which also naturally falls into the category of "single-attribute-queriable" interface. We manually examine two sets of real Web sources consisting of 480 sites from 11 different domains. The first dataset contains 5 domains selected from the UIUC Web Repository [3]. The second data set is from the Bizrate.com, a major online store that integrates data from hundreds of e-commerce sites in more than ten different domains. We investigate 6 major product domains in Bizrate.com, and for each domain, the top 25 stores (according to the customer ratings ranked by Bizrate.com) are selected for this case study.

Table 1 shows that most product databases can be modelled by simplified query model (S.Q.M), but the acceptance percentage varies dramatically across different domains. For example, in many product domains such as DVD and Books, more than 95% sources can be queried by specifying only one attribute value. However, there exist some domains like airplane fare vendors and hotel reservation sites (which are not shown here) where most query forms are highly structured and restrictive in the sense that only multi-attribute queries are accepted. Another interesting finding is that contrary to the common belief that structured Web databases are usually guarded by structured query interfaces with "rich" query semantics, most e-commerce Web sites also support keyword based search (K.W.) over their transactional product databases. This trend of "*fading schema*" opens exciting opportunities for query-based database crawling. In the presence of keyword query interfaces, the crawler can then easily "throw" attribute values into target query box and safely rely on the end sites' query processing mechanism to decide which "column" that value should actually match. In summary, our case study confirms that the simplified query model can model most e-commerce product sources. Thus, we argue that the research on query selection techniques can and should proceed without delay for more sophisticated query capability

models. We leave the issues relating to crawling multi-attribute Web sources as our future work.

2.3. Cost Model

We define the cost of crawling a Web database as the total number of communication rounds between the crawler and the Web server. In this paper, we are only interested in the transmission overhead that is typically a dominant component in Web database crawling. It is important to distinguish the total number of communication rounds with the total number of queries issued. This is because each result page can typically hold a fixed number k of matched records and thus every initiated connection retrieves *at most* k data records. For example, if there are 95 car records in the used car database matching the attribute value "Brand, Toyota" and each result page displays the next 10 records, the total cost to retrieve the entire answer set will be $\lceil 95/10 \rceil = 10$ communication rounds.

DEFINITION 2.3. The crawling cost $cost(q_i,DB)$ of querying the database DB with query q_i is defined as: $cost(q_i,DB) = \lceil \text{num}(q_i,DB)/k \rceil$, where $\text{num}(q_i,DB)$ stands for the number of all the records in DB matched by q_i and k corresponds to the maximum number of records displayed in each result page from the target Web site. □

2.5. Problem Formulation

We now describe the context in which a query selection algorithm works. In general, a Web database crawler consists of three components: *Query Selector*, *Database Prober* and a *Result Extractor*. The *Query Selector* implements three internal data structures: $L_{to\text{-}query}$, $L_{queried}$, and a statistics table. $L_{queried}$ stores the attribute values queried previously. $L_{to\text{-}query}$ contains the attribute values discovered so far but still yet to be probed. The statistics table maintains all the information needed for the *Query Selector* to make the selection decision. Once a query is chosen, the *Database Prober* module queries the target database DB and the *Result Extractor* extracts data records from the result pages and feeds them into DB_{local}. This process is equivalent to "expand" a local database graph G_{local}, a sub-graph of the complete graph G.

We now present a theoretical framework to formalize the problem of optimal query selection. In the traversal of a database graph, one needs to determine which vertex to visit next, *i.e.* how the crawler chooses the attribute value for the next query formulation. Intuitively, there exists a subset of vertices V' (queries) that retrieves the entire *reachable* database content in DB, i.f.f. all other vertices in corresponding database graph are adjacent to at least one vertex in V'. The goal of the *optimal* query selection strategy is hence equivalent to finding the V' with *minimum* cardinality. Furthermore, according to our cost model, the query cost varies across different graph vertices. Thus we further generalize the crawler's goal as to find a "dominating" vertex set V' with minimum total weight. Formally, we give the definition of the optimal query selection algorithm as follows:

DEFINITION 2.4. (Optimal Query Selection Method) Given a database graph $G(V,E)$, seed vertex set $V_{seed} \subseteq V$ and a weight function $W: V \rightarrow (0,1)$ where $G(V,E)$ and W are defined according to the graph model and cost model respectively, an optimal query-based database crawler finds a subset $V_{optimal}$ of V s.t. 1) $\forall v_i \in V\text{-}V_{optimal}$, $\exists v_j \in V_{optimal}$, s.t. $(v_i,v_j) \in E$; and 2) there does not exist another such subset V' of V with less total weight than $V_{optimal}$. □

By the above definition, the query selection problem is strikingly similar to the *Weighted Minimum Dominating Set* problem (NP-Complete). Slightly different from the traditional problem setting, the database crawler is facing a *more* challenging problem as it lacks the "big picture" of the whole graph and can thus only make a decision of query selection based on its partial knowledge about the target database. Starting with a set of seed attribute values and little information about the global graph, a database crawler essentially expands the local database graph and more knowledge about the global picture is accumulated during this process. We borrow the concept of *harvest rate* from focused Web crawling [9] and define a new metric called *query harvest rate* to capture the "productivity" of each candidate query.

DEFINITION 2.5. (Query Harvest Rate) Given a target Web database DB, and a local database DB_{local} containing the data records already crawled from DB, the harvest rate of q_i is defined as:

$$HR(q_i) = [num(q_i,DB) - num(q_i,DB_{local})]/cost(q_i,DB) \quad (2.1)$$

where $num(q_i,DB)$ and $num(q_i,DB_{local})$ corresponds to the number data records matched by q_i in DB and DB_{local}, respectively; $cost(q_i,DB)$ stands for the cost of obtaining all the result pages. □

The goal of a local optimal solution is to select the attribute value with maximum harvest rate as the next query. It is essentially a greedy approach. In our discussion so far, we by default consider the acquisition of the entire database. However, in reality, a more reasonable goal is to harvest a satisfying portion of records as quickly as possible, especially for big Web databases, *i.e.* to achieve a certain database coverage with the minimum cost. Now the key issue is to accurately estimate the harvest rate for a given candidate query.

3. Relational Link-based Query Selection Methods

3.1. Naïve Query Selection Methods

We start with three straightforward approaches: *breath-first selection*, *depth-first selection* and *random selection*.

For the breath-first selection, $L_{to\text{-}query}$ is organized as a queue. Newly discovered attribute values are added to the end of a queue. Each time the first query in the queue is selected to query *DB*. For the depth-first query selection, $L_{to\text{-}query}$ is implemented as a stack. Each time the top element is popped out as the next query. Finally, the random query selector picks a random element from $L_{to\text{-}query}$ to query *DB*.

The naïve methods described above do not utilize any database information that has already been available to the crawler. The random approach simply assigns a uniform harvest rate to all candidate queries, the breath-first approach assigns higher harvest rates to earlier found attribute values, and the depth-first approach assigns higher harvest rates to newly found attribute values.

3.2. Relational Link-based Query Selection Method

As the crawler incrementally "uncovers" the target database *DB*, the local data repository DB_{local} contains more and more records obtained so far. Hence its corresponding graph G_{local} constitutes a sub-graph of the complete database graph *G*. Intuitively, G_{local} reveals some properties of *G*, *i.e.* a node with greater degree in G_{local} is also likely to be more popular in *G*. This assumption lies critically on the link distribution in the attribute value graph. Recently there has been a resurgence of interest in link analysis over database graphs for authority-based ranking [14]. However, little is known about the actual degree distribution in database graphs. Do there exist some common distributions that are followed by a majority of structured Web sources? While this problem has been studied extensively in the context of surface Web[8], little work exists on this problem for the deep Web. A similar problem was studied in [4] for several text databases.

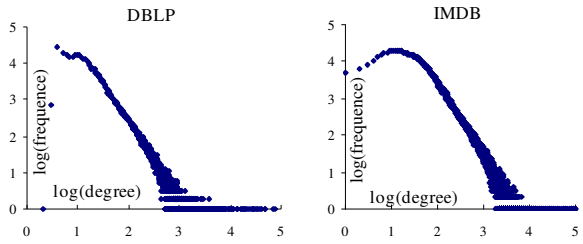

Figure 2. Case study: Relational Link Degree Distribution

We study corresponding database graphs of three popular structured Web sources: ACM Digital Library (similar to DBLP and omitted here due to space limitation), DBLP, and Internet Movie Database (IMDB). Figure 2 shows that in both the movie domain and the research publication domain, the degree distribution of the attribute value graph is very close to power-law distribution, *i.e.* a few attribute values are extremely popular, while "the massive many" are sparsely connected. This implies that in a database graph, there exists a small portion of "hub" attribute values that link to a significant portion of database content. Therefore these nodes can help the crawler rapidly "uncover" a considerable portion of the target source. Thus, we propose a greedy relational-link based database crawler. At each step it selects from $L_{to\text{-}query}$ the next attribute value with greatest link number in G_{local} for query formulation. In other words, the greedy link-based algorithm estimates $HR(q_i)$ as proportional to $degree(q_i, G_{local})$.

3.3. Exploiting Attribute Value Dependency

There is one fundamental limitation inherited in the above approach: the basic query selection criterion employed is the query's popularity, rather than a query's potential to discover *new* database content. The greedy linked-based strategy always favours popular nodes and does not take into consideration the dependency between the queries to issue and the queries already issued. However, in real world databases, the dependency between attribute values is very common. For example, many authors often publish paper together and thus after one of the authors are queried, the other author name may not be a good choice even if it is the most popular "node" in the G_{local}. Our experience indicates that in the beginning of the crawling process, the strategy of following dense link works well. However, after the database coverage reaches a certain percentage (e.g. 85%), the negative effects of query dependency catches up and finally outweighs the query popularity due to a large number of *duplicate* contents. We call this the "*low marginal benefit*" phenomena in Web database crawling.

In order to effectively "squeeze" out the marginal content, it is important that we alleviate the negative effects caused by the query dependency. To this end, we propose a new "Min-Max Mutual Information-based Query selection Method"(MMMI) that employs the concept of *mutual information* to capture the correlation between two queries (attribute values): q_i and q_j by counting the number of records in DB_{local} where they occur together. Specifically, each query q_i in $L_{to\text{-}query}$ will be assigned a score $s(q_i)$ that is proportional to its correlation with the queries in $L_{queried}$ and a sorting process will be invoked to sort $L_{to\text{-}query}$ in ascending order. The dependency between q_i and *all* the past queries in $L_{queried}$ is defined as follows:

DEFINITION 3.1. Given the part of target database DB_{local} that has already been discovered, the dependency $s(q_i, L_{queried}[1\ldots m])$ between q_i and $L_{queried}[1\ldots m]$ is:

$$s(q_i, L_{queried}[1\ldots m]) = MAX \ln\left(\frac{P(q_i, q_j \mid D_{local})}{P(q_i \mid D_{local})P(q_j \mid D_{local})}\right),$$

for each $q_j \in L_{queried}[1\ldots m]$ and $HR(q_i) \propto 1/s(q_i, DB_{local})$. □

Thus each query in $L_{to\text{-}query}$ is rated by its maximum dependence among all the issued queries in $L_{queried}$. By

doing so, we penalize those candidate queries that demonstrate strong dependency on previously issued queries. While it is open to discussion whether *max()* is the best function to capture the correlation between q_i and all the previous queries (e.g. the linear weighted function can be a good alternative), the reason for choosing *max()* is only to avoid "bad" decisions (similar to the "common wisdom" in query optimization) that brings back a lot of duplicate records and we believe that *max()* is sufficient for this goal.

In this paper, MMMI is used together with the greedy link-based approach to harvest the marginal database content. Specifically, the crawler starts with the strategy of favouring popular attribute values and MMMI is later invoked when the crawling process becomes saturated. The reason for this combination is two fold. First, the greedy link-based approach usually performs remarkably well when crawling the first 85% portion of the target database. Second, when the DB_{local} is small in the beginning, the query dependency calculated based on the limited information available is unlikely to be accurate and useful. Currently we apply a set of heuristics to determine the saturation point. Automatic saturation detection is left for future work.

The main challenge of incorporating the attribute dependency is to control its computing cost. It is expensive to update the attribute dependencies every time when new records are discovered. Our current solution is to adopt a batch-mode operation, *i.e*. the dependency information is recomputed when a batch of queries has been issued.

3.4. Heuristic-based Query Abortion

There are several simple heuristics that can be applied to improve the crawling performance. First, most Web sources report the number of total query results in the first return page. Therefore, a crawler is able to accurately calculate the exact number of new records in the following pages and thus can abort a query if the harvest rate is below some threshold. Second, when such information is not available, one can still apply other heuristics to abort queries that retrieves significant number of duplicate records in the first several pages. Our experiments show that these heuristics can greatly improve crawling performance. Due to the space limitation, we do not discuss these methods further in the paper and refer to a forth-coming journal version of this work.

4. Leveraging Domain Knowledge in Query Selection

So far, we have developed the greedy link-based query selection algorithm and optimization techniques to handle the "low marginal benefit" problem. While our experimental results in Section 6 show that link-based techniques can considerably outperform best naïve query selection methods, they have two fundamental limitations.

Limitation 1. Query selection decision is made solely based on the statistical information available in $DB_{local.}$ Without a bigger picture, any estimation about query harvest rate is doomed to be inaccurate. We call it "*near-sighted estimation problem*".

Limitation 2. Techniques introduced so far select queries only among the previously results returned from the target database, *i.e.* only those attribute values in DB_{local} are eligible for future query formulation. This *limited query candidate pool problem* will lead to another two issues. First, since the current query is selected from the previous query results, it is hard to avoid the negative impact of the attribute value dependency completely. Second, in some real world sources, the underlying database graphs may not be fully connected. As a result, from a small number of seed attribute values, the convergence coverage may constitute only a small fraction of the target database ("data islands").

4.1. Incorporating Domain Knowledge to Query Selection Framework

Structured Web sources are often domain specific. Databases from the same domain are similar both in terms of their attribute values and their attribute value distributions. For instance, in product databases, the set of brand names is usually stable across different sellers and the frequency of each brand name is also likely to be similar within different databases. Furthermore, a common situation in practice is that before crawling a target product database, the crawler may have already acquired access to the structured content from some databases in the same domain. This situation is especially true when some online sellers are willing to share their product catalogue with major search engines. Consequently it is unwise to ignore this fact and treat domain-specific databases as though they were independent of each other. Leveraging on some sample databases from a particular domain, the database crawler not only acquires the categorical attribute values for *query generation* (addressing Limitation 2), but also obtains global statistical information to estimate the query harvest rate more accurately (addressing Limitation 1).

Now the remaining question is how we can effectively integrate the domain knowledge into our existing query selection framework. For example, if we already have some DBLP data at hand, how can the database crawler utilize this piece of *prior* knowledge when crawling the ACM Digital Library? Similarly, if we have already obtained the data from the Internet Movie Database, how can this knowledge enhance the crawling performance of the Amazon DVD product database? Before presenting our solution to these questions, we first discuss how the domain statistics are represented.

DEFINITION 4.1. The domain statistics table *DT* of domain *DM* consists of a collection of entries in the form of

$<q_i, P(q_i, DM)>$, where q_i stands for a candidate query and $P(q_i, DM)$ is the *domain probability* that q_i occurs in DM. □

With the domain statistics table being incorporated, the database crawler now faces a larger candidate pool to select query from, which can be classified into two groups Q_{DB} and Q_{DT}. Q_{DB} consists of the queries whose corresponding attribute values have been discovered in the target database DB from the previous results; Q_{DT} corresponds to the queries in the domain table DT but not yet seen from DB. In the following two subsections, we derive two estimators for harvest rates of these two types of queries.

4.2. Harvest Rate Estimation for $q_i \in Q_{DB}$

Using the domain statistics to estimate the harvest rate for $q_i \in Q_{DB}$, we rewrite $HR(q_i)$ from Definition 2.5 as follows:

$$\mathrm{HR}(q_i) = k * \left[1 - \frac{num(q_i, DB_{local})}{num(q_i, DB)}\right] \quad (4.1)$$

where $num(q_i, DB_{local})$ is the number of matched records of query q_i in DB_{local}, $num(q_i, DB)$ is the total number of matched results in the target database DB, which is unknown before q_i is executed and k corresponds to the number of records contained in each result page. Below we show how to estimate the only unknown factor $num(q_i, DB)$. Our assumption is based on the observation discussed: the probability that query q_i occurs in the database DB equals to its *domain probability* $P(q_i, DM)$. This simplified assumption is similar to the statistical language model employed in the Information Retrieval (IR) literature, *i.e.* we basically assume that databases from the same domain are generated by a similar "hidden" model. According to this assumption, all queries in $L_{queried}[1\ldots m]$ (queries that have already been issued) have the same occurrences probability belonging to DB as its domain probability. Omitting the derivation, we estimate the value of $num(q_i, DB)$ using the following estimator:

$$num(q_i, DB) = \frac{|DB_{local}| * P(q_i, DM)}{P(L_{queried}[1\ldots m], DM)} \quad (4.2)$$

So far we assume a comprehensive domain table and do not take into consideration the situation where candidate query q_i does not appear in the domain statistics table DT. In order to handle the situations where DT misses q_i, we add a "*smoothing*" factor in the calculation of $P(q_i, DM)$:

$$P(q_i, DM) = \frac{num(q_i, \Delta DM) + num(q_i, DM)}{|\Delta DM| + |DM|} \quad (4.3)$$

In the above equation, $\triangle DM$ is the set of newly discovered records from DB containing at least one attribute value that is not in DM; $num(q_i, \triangle DM)$ and $num(q_i, DM)$ stand for the number of records matched by q_i in $\triangle DM$ and DM, respectively.

4.3. Harvest Rate Estimation for $q_i \in Q_{DT}$

We now show how we estimate the harvest rate of a query q_i in the domain table DT while its corresponding attribute value has not been discovered from the target database DB. Since we have not seen q_i in the previous query results, all the records matched by q_i in DB, if any, must be new, *i.e.* $HR(q_i)$=1. However, q_i may not exist in DB in the first place. In this case, simply no record will be returned, *i.e.* $HR(q_i)$=0. Therefore, the expected value of $HR(q_i)$ of $q_i \in Q_{DT}$ can be written as the weighted sum of the above two possible conditions and we get $HR(q_i) = P(q_i \in DB | q_i \in DM)$. With a simplified assumption that $P(q_i \in DB)$ equals to $P(q_i \in DM)$, we obtain the following: $HR(q_i) = P(q_i \in DB | q_i \in DM) = P(q_i \in DM | q_i \in DB)$, where $P(q_i \in DM | q_i \in DB)$ can be easily evaluated as DM's "hit rate" of the previously returned results in the crawling history.

4.4. Optimization Issues

We briefly discuss two optimization choices that are employed to accelerate the query selection process. First, we adopt the notion of *lazy harvest rate evaluation* in calculating the harvest rate for $q_i \in Q_{DB}$. The basic idea is to postpone as much as possible the real calculation of $HR(q_i)$. We internally organize a separate queue for all the queries that belong to Q_{DB}. All the queries in Q_{DB} are compared via an intermediate harvest rate value, since $HR(q_i) \propto P(q_i, DM)/num(q_i, DB)$. Consequently, each time we only need to calculate the exact value of only one query from Q_{DB}. Second, $P(L_{queried}[1\ldots m], DM)$ changes after the latest issued query being added to $L_{queried}$. Recalculating $P(L_{queried}[1\ldots m], DM)$ from scratch at each query selection step can be very time consuming. We incrementally maintain this value by rewriting $P(L_{queried}[1\ldots m], DM)$ as follows:

$$P(L_{queried}[1..m], DM) = \frac{|S(L_{queried}[1...m-1], DM) \cup S(L_{queried}[m], DM)|}{|DM|}$$

where $S(L_{queried}[1\ldots m\text{-}1], DM)$ stands for the set of record IDs in the sample databases DM matched by any query from $L_{queried}[1\ldots m\text{-}1]$. And $S(L_{queried}[m], DM)$ represents the set of record IDs that are matched by the latest issued query $L_{queried}[m]$. We keep $S(L_{queried}[1\ldots m\text{-}1], DM)$ as a sorted list, the union between the two sets can be efficiently implemented as merging of two sorted lists with duplicate elimination. The resulting list is a new sorted list $S(L_{queried}[1\ldots m], DM)$ to be used by the same procedure in the next round.

5. Experimental Evaluation

Our experiment consists of two groups: controlled simulation on local database servers and real crawling of the Amazon DVD product database.

For the controlled crawling simulation, the crawler and the database server are located on two separate machines interconnected with high speed LAN. The database is hosted by a Dual 3.06G HZ Windows Server with 3.87 GB of RAM running Microsoft SQL Server 2000 RDBMS. We simulate the crawling process by implementing server programs that mimic Web server behaviour on top of the database server. Real data from 4 structured Web databases is used in this controlled simulation. Specifically, the eBay Auction database consists of 20,000 auction items crawled in 2001 by Wisconsin database group. The ACM Digital Library database contains 150,000 research article records. We crawled the library and populate our local database with the structured data extracted from the page crawled. Both DBLP and the Internet Movie Database (IMDB) are freely available on the Web. DBLP contains 500,000 paper records and IMDB has information about 400,000 movies records. For each database, we join all the information into one single universal table. For those attributes that may contain variable number of attribute values among different records (e.g. consider the "Authors" attribute in research paper databases), we concatenate and combine all the attribute values into one single column and make the column full-text searchable. The query interfaces implemented in our experiments and the corresponding number of distinct attribute values is shown in Table 2.

Given the above query interfaces, all the four controlled databases are "well connected": 99% of all the records are connected in the sense that starting from any of these records, a database crawler can obtain 99% database records within a finite number of queries. Each query selection algorithm is evaluated four times with different seed values (starting points) to avoid the possible noise due to individual seed, and the average result is reported.

	Queriable Attributes	Attribute Value #
eBay	Categories, Seller, Location, Price	22950
ACM	Title, Conference, Journal, Author, Subject keywords	370,416
DBLP	Title, Conference, Journal, Author, Volume	860,293
IMDB	Actor, Actress, Director, Editor, Producer, Costumer, Composer, Photographer, Language, Company, Release Location	1,225,895

Table 2. Database Query Interface Schemas

We also conduct experiments on some real Web sources. The Amazon DVD product database is used to evaluate our domain knowledge-based query selection algorithm. Amazon.com provides developers with direct access to its product data repository via a set of well-defined Web service interfaces. The returned query results are in the format of XML documents, which eliminates the possible accuracy problems of extracting structured records from Web pages. Since we do not have the size information, "overlap analysis" is used [18] to obtain an estimation of approximate size. Specifically, we conducted 6 independent crawls starting from 6 randomly selected seed values. Each crawl terminates after 5000 interactions with the server. Then we calculate the overlap size of every two result sets and based on which, we obtain in total $C_6^2 = 15$ size estimation about the Amazon DVD database. Finally, statistical hypothesis testing is applied (*t-testing* in our case) on these 15 size estimations. Our results show that with 90% confidence, the Amazon DVD product database contains less than 37,000 data records.

5.1 Performance of Link-based Query Selection Algorithm

In Figure 3, we compare the crawling performance of the greedy link-based query selection method with three naïve ones in terms of communication overhead incurred to achieve the same database record coverage. In particular, x-axis represents the database coverage and y-axis plots the number of result pages requested by the database crawler. All the experiments here are performed over controlled database servers. In these experiments, the maximum target database coverage is 90%, which is a fairly ambitious goal in practice. By default, each result page contains at most ten data records from the backend database.

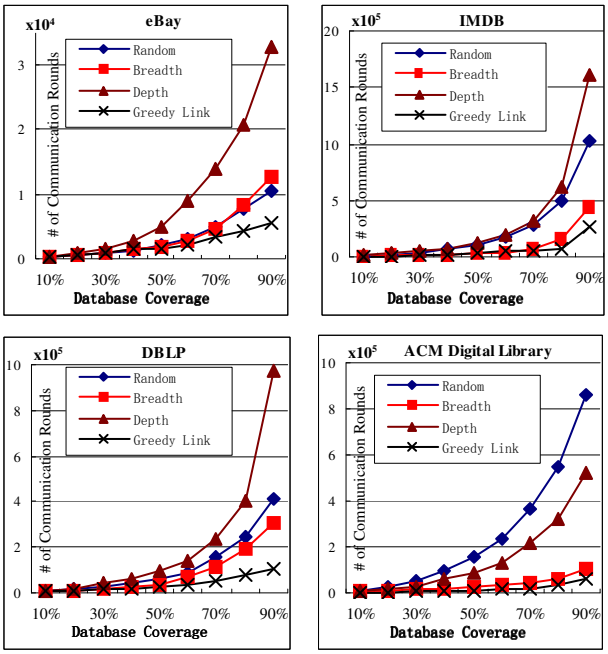

Figure 3. Performance comparison between the greedy and naïve methods on four controlled database servers.

As expected, the greedy link-based query selection method (GL) consistently achieves the best coverage with the least communication cost. This confirms that the presence of hub nodes in the database graph indeed enables GL to quickly retrieve the "dense portion" of a target

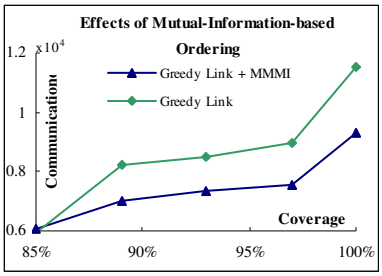

Figure 4. Effects of Attribute Value Dependency

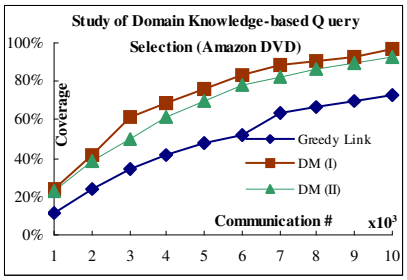

Figure 5. Comparison Domain Statistics v.s. Greedy Link

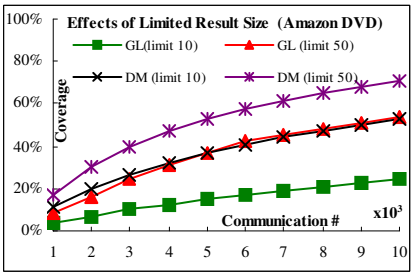

Figure 6. Effects of Limited Result Size

structured Web database. Moreover, on every dataset, the cost of all methods increases dramatically when the coverage exceeds 80%. This shows that after the coverage reaches a certain level, most returned records overlap with the previous results. This is what we called the "*low marginal benefit*" phenomena in query-based database crawling.

5.2 Effectiveness of Incorporating Attribute Value Dependency

Figure 4 illustrates the effects of using attribute value dependency in obtaining the "marginal database content" on the eBay auction dataset. The greedy crawler will invoke mutual information-based ordering and switch to a new policy when the database coverage reaches 85%. As is shown in Figure 4, MMMI achieves the same coverage with about 1200 communication rounds less than the original greedy crawler. This shows that by assigning higher priority to the queries with less correlation with the previously issued queries, MMMI speeds up the process of obtaining the "marginal database content".

5.3 Effectiveness of leveraging domain statistical information

In Figure 5, we conduct a head-to-head comparison between the domain knowledge-based approach and the greedy link-based algorithm (GL). We use the Amazon DVD product database as the target source. The IMDB dataset is used for the domain table construction (since IMDB and Amazon DVD are from the same Movie domain). In addition, we also investigate the effect of the sample database size on DM's performance. We construct two domain tables (DT) from two distinct subsets of IMDB. DM(I) constructs the DT from a larger subset containing all the movie records released after year 1960 (270,000 records) in IMDB. In contrast, DM(II) only contains movies after year 1980 (190,000 records). In Figure 5, x-axis depicts the total number of page requests sent by the crawler and y-axis represents the estimated database coverage. All methods are allowed to retrieve 10,000 result pages from the Amazon Web Service and we take snapshots of the database coverage between every 1,000 requests. As is shown in Figure 5, both DM-based crawlers outperform GL. In particular, after ten thousand communications with the Amazon Web service, DM(I) acquires around 95% of the estimated database records, while the final coverage of the greedy approach is less than 70%. Furthermore, with 5,500 communication rounds, DM(I) has already retrieved approximately 80% DVD database records, which is pretty good in practice.

By comparing DM (I) with DM (II), we observe that a smaller domain data set results in a slight performance degradation. With a more comprehensive domain table, DM(I) benefits from more accurate query probability estimation as well as a larger candidate query pool. Note that in this experiment, Amazon sets a limit of 3200 on its result size for any query, *i.e.* for each incoming query, Amazon Web service returns at most 3200 matched records. This value is quite "generous" compared to most other Web sources.

5.4 Crawling performance with limited result set size

Our experiments on controlled database servers assume that all the matched result records can be retrieved. However, in reality, most Web databases set an upper bound on the number of results that can be accessed to save the server resource. For example, Yahoo! Automobile Finder may claim that it has 5000 cars records matching with a certain query and the user may not be able to access the records beyond the first 20 pages.

In Figure 6, we are interested in the crawling performance with much tighter limits on the result size. We still use the Amazon DVD database as target source and the movie records after 1960 (270,000 records) from IMDB is used to construct the domain table. For both the DM crawler and GL crawler, we run two crawls with the limit size set to 10 and 50. Figure 6 shows that with tighter limits on the result size, both methods experience degradation in productivity. Compared to their performance in Figure 6 with a limit of 3200 (the original limit imposed by Amazon), the performance of both methods drops by about 50% and 20% under limit=10 and 50 respectively. Intuitively, the result limit reduces the connectivity of the target database, and thus delays the discovery of the "hub" nodes.

6. Related Work

Raghavan and Molina first introduced the problem of "hidden Web crawling". A prototype deep Web crawler, *HiWe*, was presented to automatically extract and analyse the interface elements and submits queries through these query interfaces. In [6], Barbosa et. al. experimentally investigate the construction of keyword queries to obtain documents from large Web text collections. We also notice that another independent effort by Ntoulas et. al. [21] proposes similar keyword query selection techniques for downloading the textual content from Web repositories.

Another important related area is Web information extraction. Lots of proposals have been made. For example, [5] studies the problem of fully automatic data extraction by exploring the repeated patterns from multiple template-generated result pages. [19] utilizes the information on the "detailed" record pages pointed by the current result page to identify and extract data records. Note that Web data extraction is orthogonal to the query selection problem investigated in this paper. Breakthroughs in that frontier will certainly increase the potential impact of this work.

There has been an active research interest in understanding the semantics of the query interfaces of the structured Web databases [16],[24],[25],[15]. For example, [16] introduces WISE-integrator that employs comprehensive meta-data, such as element labels and default value of the elements to automatically identify matching attributes. [24] proposes an instance-based schema matching scheme that uses domain specific query probes to discover the attribute mappings. [15] uses statistical models to find the hidden domain-specific schema by analysing co-appearance of attribute names.

7. Conclusion

The high quality data from structured Web sources is in high demand from many applications. Parallel to URL ordering in surface Web crawling [12], the central issue of efficient data acquisition from the deep Web is query selection. In this paper, we modeled the query-based database crawling, conducted two case studies to investigate real Web sources' properties, proposed two suites of query selection methods and experimented our methods extensively on both controlled local servers and a real Web structured source with more than 1 million data records in total. The results motivate the need for sophisticated query selection mechanism, validate the effectiveness of our approaches and provide insights for future efforts. The domain-knowledge based database crawler shows the greatest promise among all methods. We believe that a practical solution for real world applications is to combine the domain-knowledge-based query selection with a set of fine-tuned heuristics, which is a part of our future work. Our future work also includes the implementation and deployment of a real world product database crawler.

References

[1] MSN Shopping. http://shopping.msn.com/.

[2] Google Shopping. http://froogle.google.com/.

[3] UIUC Repository. http://metaquerier.cs.uiuc.edu/repository 2003.

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

[5] A. Arasu, and H. Garcia-Molina. Extracting structured data from Web pages. In *SIGMOD*, 2003.

[6] L.Barbosa and J.Freire. Siphoning hidden-Web data through keyword-based interfaces. In *SBBD*, 2004.

[7] M.K.Bergman. The deep Web: Surfacing hidden value, http://www.press.umich.edu/jep/07-01/bergman.html.

[8] A.Broder,R.Kumar,F.Maghoul,P.Raghavan,S.Rajagopalan, R.Stata, A. Tomkins, J.Wiener. Graph structure in the Web. In *WWW* 1999.

[9] S.Chakrabarti, M. Berg and B. Dom. Focused Crawling: A New Approach to Topic-Specific Web Resource Discovery. In *WWW*,1998.

[10] K. C.-C. Chang, B. He, and Z. Zhang. Toward Large Scale Integration: Building a MetaQuerier over Databases on the Web. In *CIDR* 2005.

[11] W.B. Croft and J.Laffery, Language Modelling and Information Retrieval. *Kluwer Academic Publisher*, 2003.

[12] J. Cho, H. Garcia-Molina, and L. Page. Efficient crawling through URL ordering. In *WWW*, 1998.

[13] L. Gravano, P. Ipeirotis, and M. Sahami. Query- vs. Crawling-based Classification of Searchable Web Databases. In *IEEE Data Engineering Bulletin*, 2002.

[14] F. Greets, H. Mannila, and E. Terzi. Relational link-based ranking. In *VLDB*, 2004.

[15] B. He, and K. C. Chang. Statistical Schema Matching across Web Query Interfaces. In *SIGMOD*, 2003.

[16] H. He, W. Meng, C, Yu and Z. Wu. WISE-Integrator: an automatic integrator of Web search interfaces for E-commerce. In *VLDB*, 2003.

[17] P. Ipeirotis, L. Gravano, and M. Sahami. Probe, Count, and Classify: Categorizing Hidden-Web Databases. In *SIGMOD*, 2001.

[18] S. Lawrence and C. L. Giles, Searching the World Wide Web. In *Science*.1998.

[19] K. Lerman, L.Getoor, S. Minton and C. Knoblock. Using the structure of Web sites for automatic segmentation of tables. In *SIGMOD*, 2004.

[20] A. Maedche and S.Stabb: Ontology learning for the semantic Web. In *IEEE Intelligent System*, 2001.

[21] A. Ntoulas, P.Zerfos and J. Cho. Downloading textual hidden Web content through keyword queries. In *JCDL*, 2005.

[22] S. Raghavan, and H .Garcia-Molina. Crawling the hidden Web. In *VLDB*, 2001

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

[24] J. Wang, J. Wen, F. H. Lochovsky, and Wei-ying Ma. Instance-based Schema Matching for Web Databases by Domain-specific Query Probing. In *VLDB*, 2004.

[25] Z. Zhang, B. He, and K. C. Chang. Understanding Web Query Interfaces: Best-Effort Parsing with Hidden Syntax. In *SIGMOD*, 2004.