Automatic Filling of Hidden Web Forms: A Survey

Gustavo Zanini Kantorski Viviane Pereira Moreira Carlos Alberto Heuser
{gzkantorski,viviane,heuser}@inf.ufrgs.br
Institute of Informatics – UFRGS – Porto Alegre – Brazil

ABSTRACT

A significant part of the information available on the Web is stored in online databases which compose what is known as *Hidden Web* or *Deep Web*. In order to access information from the Hidden Web, one must fill an HTML form that is submitted as a query to the underlying database. In recent years, many works have focused on how to automate the process of form filling by creating methods for choosing values to fill the fields in the forms. This is a challenging task since forms may contain fields for which there are no predefined values to choose from. This article presents a survey of methods for Web Form Filling, analyzing the existing solutions with respect to the type of forms that they handle and the filling strategy adopted. We provide a comparative analysis of 15 key works in this area and discuss directions for future research.

1. INTRODUCTION

The Hidden Web [11] or Deep Web [4] is the part of the Web that is not accessible through traditional crawling (*i.e.,* direct link navigation) [19]. The contents of the Hidden Web can only be accessed by filling out Web forms, such as the ones in Figure 1, which are then submitted as queries to the online database behind the form. The Hidden Web covers several topic domains, such as government, education, entertainment, business, health, news, and sports. There are thousands of online databases for each of those domains – most of them containing structured information [7]. Exposing the contents of an online database can be achieved by designing wrappers, *i.e.,* programs that extract data from a specific Web site. However, since wrappers are specific for each site, this solution is not feasible when dealing with thousands of hidden Web sites. Thus, a more scalable approach is to use a Hidden Web Crawler to automatically identify and retrieve data from online databases.

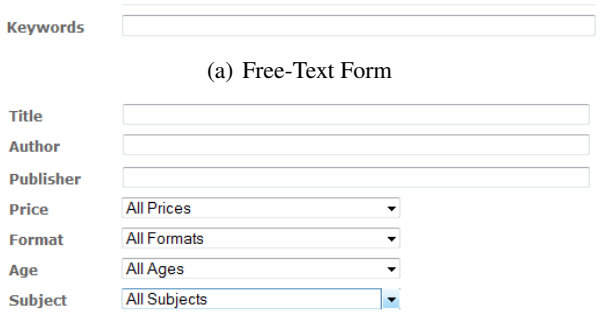
Hidden Web Crawling has many applications. The discovered content can be indexed by generic search engines, such as Google, Bing, and Yahoo!. Furthermore, it can be used to create vertical search engines, which focus on a specific segment such as real estate, online auction, books, airline tickets, etc.

Hidden Web crawling can be divided into three key phases: (i) discovery of the entry points to the Hidden Web, *i.e.,* Web forms that allow searching on-line databases [3, 21–23]; (ii) identification, filling, and submission of forms [1,2,9,14,15,20,22,25–27,29–31]; and (iii) data extraction from the results of submissions [5, 6, 8, 17, 32, 33].

This survey is focused on the second phase. The identification of the fields is reasonably straightforward and can be achieved by parsing the HTML code of the page containing the form looking for specific tags such as `input` and `select`. The challenge is *Web Form Filling* (WFF), *i.e.,* how to automatically fill the fields using suitable values in order to retrieve meaningful data. This is a critical step since filling a form with unsuitable values will result in blank or error pages representing a waste of resources. The goal is not to find all possible values, but to select a subset of values so as to minimize the number of submissions and maximize the coverage, i.e., retrieve more distinct data behind the form. The fact that forms are designed to be handled by human users and this leads to a diversity of designs poses further difficulties to automatic processing.

Figure 2 shows a fragment of the HTML code for the Web form shown in Figure 1(b). An HTML form is defined within a <*form*> and a <*/form*> tag (see Figure 2, lines 1 and 23). Form fields can be text boxes, selection lists, checkboxes, radio buttons, or submit buttons. Selection lists, radio buttons, and checkboxes show a list of options to the user. On the other hand, a text box field does not contain options, so the values need to be discovered somehow. More concisely, fields can be grouped into two types: *fields with a finite domain* such as selection lists (see Figure 2, lines 8 to 21); and *fields with an infinite domain*, such as text fields, in which a user can type any value (see Figure 2, lines 2 to 7), which brings further challenges to automatic filling.

Authors classify forms under different names with respect to the kinds of fields that they have. Forms with a



(a) Free-Text Form

(b) Complex Form

Figure 1: (a) Free-Text and (b) Complex Web Forms

single text field are called *simple* [24,25] or *free-text* [28]. Forms with several fields are called *multi-field* [25], *advanced* [24], or complex [28]. In this survey, we adopt the terminology by Tjin-Kam-Jet [28] which refers to the former as *free-text* and as *complex* to the latter.

Although the Hidden Web also has unstructured content (*e.g.*, text, images, videos, etc.), structured contents are the most prevalent (over 75%, according to Chang *et al.* [7]). These are usually relational databases containing attribute-value pairs (*e.g.*, a movie Web site that returns movie information such as director, actors, title, etc.). The vast majority of the existing work on WFF is focused on uncovering structured data, so this represents the primary focus of this survey.

Contributions. This article contains a survey of the key works on WFF. We analyzed the existing literature on Hidden Web Crawling and report on 15 works we believe to be the most influential in WFF. These works are discussed with respect to some key features such as text field seed generation, value generation, prior knowledge, human intervention, and submission method. While distinct works employ different filling strategies, two aspects are common to all: (i) filling method and (ii) form type. In this article, we consider two categories for filling method: *heuristics* and *machine learning*; and two categories for form type: *free text Web forms* and *complex Web forms*. We present a comparative study under two perspectives: a *holistic view*, in which each method is treated in its fullness; and a *Cartesian view*, in which each method is studied as a collection of parts. To the best of our knowledge, this is the first survey to address WFF in the Hidden Web.

2. DEFINITIONS AND PROBLEM OVERVIEW

Web Form Filling (WFF) is the process of selecting values for filling the fields in a Web form. Generating values for fields with finite domains is fairly easy as the possible values are found in the form itself. For fields with infinite domains, the values have to be predicted. This process may be divided into two sub-problems: (i) selecting an appropriate set of initial values (*seeds*); and (ii) selecting an appropriate set of input values.

```
1<form method="get" action="/results.html">
2     <span>Title</span>
3     <input type="text" name="title" />
4     <span>Author </span>
5     <input type="text" name="authorName" />
6     <span>Publisher</span>
7     <input type="text" name="publisher" />
8     <label for="price">Price</label>
9       <select name="prc" id="prc">
10        <option value="all">All Prices</option>
11        <option value="1">under $10</option>
12        <option value="2">$10-$25</option>
13        ..........
14      </select>
15    <label for="select">Format</label>
16      <select name="fmt" id="fmt">
17        <option value="all">All Formats</option>
18        <option value="0">Hardcover</option>
19        <option value="1">Paperback</option>
20        ..........
21      </select>
22      ................
23</form>
```

Figure 2: Fragment of HTML Code from Figure 1(b)

Existing solutions for WFF can be classified with respect to their reliance on prior knowledge. Approaches which rely on prior knowledge need to build the knowledge base beforehand and generate values according to the domain (*i.e.,* topic) of the form. On the other hand, methods that do not rely on prior knowledge typically generate new candidate values by analyzing the results from previous submissions.

The problem in which this survey is focused may be formulated as "given an HTML form, identify the fields and find suitable values to fill these fields so that the form retrieves meaningful data". A form F is represented by two sets. The first set describes the fields and their values. The second set represents the results of form submissions as a single table. More formally, the fields and values of a form F are represented as a set of *(field, domain)* pairs:
$F = \{(f_1, dom(f_1)), (f_2, dom(f_2)), ..., (f_n, dom(f_n))\}$
where the f_i's are the form fields and the $dom(f_i)$ are the domains.

A form field represents input objects, such as selection lists, checkboxes, text boxes. The domain of a field is the set of values it can take. For example, if f_j is a selection list ($\langle select \rangle$ tag in HTML), then $dom(f_j)$ are the values in the list $\langle option \rangle$ tag in the HTML.

A form field is usually associated with a *label* which has a descriptive text that helps understanding the field.

Thus, $label(f_i)$ refers to the label associated with the i^{th} form field. For example, the label *Title* is associated with the first field in the form shown in Figure 1(b).

The second set in form F is represented as a single table R with m records $r_1, r_2, ..., r_m$ over a set of k attributes $A = a_1, a_2, ..., a_k$, in which each attribute has $dom(f_k)$. Then, we have:
$F = \{(f_1, dom(f_1)), (f_2, dom(f_2)), .., (f_n, dom(f_n))\}$
$R = \{r_1, r_2, ..., r_m\}$ and $A = \{a_1, a_2, ..., a_k\}$

An HTML form can be submitted by two methods, *get* or *post*. In the *get* method, field values are included as part of the URL in the HTTP request. For example, suppose a user entered the value 'Hidden Web' in the field 'Title' in the form illustrated in Figure 1(b). The value entered for the title field is appended to URL generating '*../results.html?title=Hidden+Web*'. In the *post* method, field values are sent inside the HTTP request and the URL contains only the action that is about to be performed.

3. APPROACHES FOR WEB FORM FILLING

We surveyed 15 works that address the problem of filling fields in Web forms. This survey organizes the solutions found in the literature according to two aspects: *filling method* and *type of interface*. There are two possible filling methods: *heuristic-based* and *machine learning-based*. Heuristic-based approaches are usually guided by statistical information (such as term frequencies, number of rows retrieved, etc.) and apply thresholds as stopping criteria. Machine learning approaches use this statistical information to create a model which decides which values to use to fill the forms. Similarly, there are two types of interface: *free-text* Web form where users type a list of keywords in a single search text box and *complex* Web forms which contain several fields. We analyze each existing approach according to its filling method and the type of interface that it supports. It is important to notice that this analysis does not separate the surveyed approaches into disjoint groups, as it is possible for an approach to use both filling methods and/or types of interfaces.

Existing works are grouped into four categories: (i) heuristics applied to free text forms; (ii) heuristics applied to complex Web forms; (iii) machine learning applied to complex Web forms; and (iv) overlapping combinations. We classify under *overlapping combinations* approaches which tackle more than one type of interface and/or filling methods. We found no works which apply machine learning applied to free text forms. Thus, there is no subsection here dealing with this category. Also, we found only one approach that works with complex Web forms and machine learning. In the next subsections, we analyze each of these categories.

3.1 Heuristics Applied to Free Text Forms

Since free text forms have only one keyword text field, the formalism presented in Section 2 can be reduced. As a result, a Web form can be simply represented as $F = \{(f_1, dom(f_1)\}$, where f_1 is a keyword field and $dom(f_1)$ is the set of values for form field f_1. In general, heuristic-based methods rely on statistical information about the submissions.

Barbosa and Freire [2] select a set of keywords with high frequency to build queries with high coverage for form field f_1. The discovery of values for the domain $dom(f_1)$ is based on the data coming from the database itself instead of a random word generation. The approach is composed of two steps. The first step is the selection of initial keywords from the page that contains the form. The selected keywords are used to fill the form.

The algorithm proceeds to find additional keywords by iteratively submitting keywords obtained in the results of previous submissions. The goal is to select high-frequency keywords and use them to construct a query that has high coverage The stopping condition is determined by two parameters: *maxterms* or *maxSubmissions* probe queries submitted. The best choice for these parameters depends on the database. The rationale is that values found in the database are more likely to result in higher coverage than randomly selected values.

The main advantage of this method is that Web forms with keyword fields do not need detailed knowledge of the data structure or schema. Experiments on different domains and form sizes show that using *stopwords* increases coverage. This happens because *stopwords* have high frequencies (*i.e.,* appear in many documents).

Soulemane *et al.* [27] present a method which selects values for form field f_1 based on term frequency. The rationale is that frequency determines the importance of a value in a set of documents (or database rows).

The focus of Soulemane's work is on providing an automatic indexing mechanism for dynamic Web contents. The method comprises form detection, selection of search keywords, dynamic content extraction, and detection of duplicate URLs. Form detection consists in identifying Web forms with a single general input text field. Selection of search keywords tries to generate an optimized search result. Dynamic content extraction is the extraction of the data from the result pages. Detection of duplicate URLs deals with cases in which two distinct values may generate the same URL twice.

As in [2], the initial keyword values are selected from the page containing the form. After obtaining the first results, values for the domain $dom(f_1)$ are chosen from the successfully retrieved pages. A threshold *max* submissions per form prevents the crawler from falling into

an infinite loop. But if the method fails to obtain a convenient keyword from a given page, a value is chosen from the repository containing results from previous submissions. As a last resort, an external dictionary can be queried to provide values.

Term frequency measures how often a value is found in a collection. Suppose a value V_i occurs n_p times within a Web page P which contains N_p values. Term frequency is given by $F_{tf} = \frac{n_p}{N_p}$.

Ntoulas *et al.* [25] describe an adaptive algorithm based on results from previous submissions which adapts its query selection policy automatically based on such results. They assume that the crawler downloads pages from a Web site that has a set of pages S. Each potential query q_i which may be issued can be treated as a subset of S. Each subset is associated with a weight that represents the cost of issuing the query. Thus, the goal is to find which subsets cover the maximum number of Web pages with the minimum total weight (cost).

The heuristics employed by this method include the cost and the amount of new data returned for a query that has not been retrieved by previous queries. The maximum total weight takes a number of factors: the cost of submitting the query to the form, the cost of retrieving the result index page, and the cost of downloading the actual pages. The authors assume that submitting a query incurs a fixed cost of c_q. The cost c_d to download a matching item is also fixed, while the cost of downloading the result index page is proportional to the number of retrieved results. Then the overall cost of query q_i is as described in Eq. 1:

$$cost(q_i) = c_q + c_r P(q_i) + c_d P_{new}(q_i) \quad (1)$$

where $P_{new}(q_i)$ is the fraction of the new documents from q_i that have not been retrieved from previous queries.

Based on the cost and the amount of data retrieved, the authors use the efficiency metric to quantify the desirability of the query q_i (Eq. 2):

$$Efficiency(q_i) = P_{new}(q_i)/cost(q_i) \quad (2)$$

where $P_{new}(q_i)$ is the number of new documents returned for q_i and $cost(q_i)$ is the cost of issuing the query q_i.

Efficiency measures how many new documents are retrieved per unit cost and it can be used as an indicator of how well the resources are spent when submitting q_i. Thus, the crawler can estimate the efficiency of every candidate q_i and choose the one with the highest value. For estimating efficiency, the method has a *query statistics table*. This table stores the counts of how many times a value q_i appears within the documents downloaded from $q_1, ..., q_{i-1}$. The set of q_i's determines the domain $dom(f_1)$.

One issue here is the choice of the keyword to be used as the first query. The selection is not done by the adaptive algorithm as it has to be manually set because the query statistics table has not been populated yet. Thus, the selection is generally arbitrary. So, for the purpose of fully automating the whole process, the authors describe that some additional investigation is necessary.

Some Web Hidden Web sites limit the number of results returned for a query. Thus, if a query has a large number of matching results, only a fraction will be returned (e.g., the first 1000). This is problematic since the probability that a query q_i appears in the pages from $q_1, q_2, ..., q_{i-1}$ considers the entire database. To solve this issue, the approach assumes that the results returned are a random sample of the complete set of results which match the query and adjust the estimates to calculate $P(q_{i+1} | \, q_1 \vee ... \vee q_i)$.

Wu *et al.* [31] present a form filling method based on feedback of the previously submitted values. The form is treated as a single table, referred to as *DB*, with a set of queriable attributes $AS = \{attr_{s1}, attr_{s2}, ..., attr_{sm}\}$ and a set of result attributes $AR = \{attr_{r1}, attr_{r2}, ..., attr_{rn}\}$. Table *AS* is equivalent to set F and table *AR* is equivalent to sets R and A.

The set of distinct attribute values (DAV) consists of all distinct attribute values in *DB*. An attribute-value graph (AVG), *G(V,E)* for *DB* is a non-directional graph that is built as follows: for each distinct value a_{vi} in *DAV* there is only one vertex $v_i \in V$. A non-directional edge $(v_i, v_j) \in E$, if, and only if, av_i and av_j coexist in a relational instance $t_k \in DB$. Each edge in AVG represents a relational link between av_i and av_j. These values compose the domain $dom(f_1)$. The determination of values that are used for form filling is reached by the use of a seed value and, from the results, values related to the one used in the query are extracted. The authors rely on a heuristic cost to evaluate the query. The cost of a query q_i in the *DB* database is defined as in Eq. 3:

$$cost(q_i, DB) = \frac{num(q_i, DB)}{k} \quad (3)$$

where *num(q_i,DB)* represents the total number of records in *DB* matching q_i, and k corresponds to the maximum number of records in each result page.

For selecting the next query, the authors define a new metric called *query harvest rate* to capture the productivity of each candidate query. 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 in Eq. 4:

$$HR(q_i) = \frac{[num(q_i, DB) - num(q_i, DB_{local})]}{cost(q_i, DB)} \quad (4)$$

where $num(q_i, DB)$ and $num(q_i, DB_{local})$ is the number data records matched by q_i in *DB* and DB_{local}, respectively; $cost(q_i)$ stands for the cost of obtaining all the

result pages. The goal is to select the attribute value with the highest harvest rate as the next query.

Furthermore, the authors also integrate domain knowledge to query selection. The method uses a domain statistics table *DT* of a domain *DM*. Table *DT* 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 table *DT*, two groups of queries appear: Q_{DB} and Q_{DT}. Q_{DB} consists of queries whose corresponding attribute values have been discovered in the target database *DB* from the previous results, and Q_{DT} corresponds to the queries in the domain table *DT*, but not yet seen by *DB*.

Wang *et al.* [30] gather a set of documents as a sample that represents the original database. From the sample, they choose a set of values representing the domain $dom(f_1)$ that cover most of the items in the sample with a low cost (*i.e.,* retrieve the most results with the fewest submissions). These values are used to extract data from the original database.

The solution considers two heuristics: the *hit rate*, which denotes the set of data retrieved from the database, and the *overlapping rate*, which refers to the amount of duplicated data retrieved. More formally, the hit rate of a set of queries Q in a database, denoted by $HR(Q, DB)$, is defined as the ratio between the number of unique data items collected by sending the queries in Q to *DB* and the size of the database *DB*. The overlapping rate of Q in *DB*, denoted by $OR(Q, DB)$, is defined as the ratio between the total number of collected links and the number of unique links retrieved by sending queries in Q to *DB*.

The algorithm runs with the database *DB* as input, the sample size s, and the query pool size p. The sample should have an appropriate size to produce a satisfactory query list in the query pool. In order to select the queries to issue, the method creates a query pool using the terms found in a random sample from the database.

The values selected from the query pool are those that have document frequency (*df*) ranging between 2% and 20% of the sample size. Values that occur in fewer than 2% of the sample are most probably rare values, while values that appear in more than 20% of the documents are too common to consider. Then, the relative query pool size of a set of queries Q on database *DB*, denoted by $poolSize(Q, DB)$, is defined as in Eq. 5:

$$poolSize(Q, DB) = \sum_{q \in Q} df(q, DB)/|DB| \qquad (5)$$

where $df(q, DB)$ is the document frequency of q in *DB*, *i.e.,* the number of items in *DB* matching query q.

Once the query pool is populated, values from this pool are selected and sent to *TotalDB*. The selection criteria are to have Hit Rate equal to 1 and the minimum Overlapping Rate available in the sample. These values are the domain $dom(f_1)$ of form field f_1.

In practice, the total number of documents in a real deep Web database is unknown; hence the calculation of *HR* becomes impossible.

3.2 Heuristics applied to Complex Web Forms

Lage *et al.* [18] present a method to fill fields using a set of heuristics and a sample data repository for automatically finding forms, filling them out, and collecting pages containing useful data. The method starts crawling from the main page of the site looking for forms in a blind search (*i.e.,* the search is not guided by any heuristics). A set of heuristics is used to discard non-query forms. Next, it extracts the labels from the remaining forms and, using a sample data repository, it tries to learn how to fill them out. Finally, it submits all filled forms in order to identify data-rich pages. The process ends when these pages are not found.

A key component is the *sample data repository*. It is used to identify evidences in the traversed pages that such pages belong to a specific application domain. The repository is a set of attribute-value pairs of the form $< label(f_i), dom(f_i) >$ that describe objects from the application domain. The repositories are generated by extracting data from Web sources of specific domains.

The task of form filling consists in finding a mapping between form fields and repository attributes. Heuristics extract the labels which are above input fields. If the labels do not match the attributes in the repository, the form is disregarded.

Mapping is straightforward for search forms which contain only fields with finite domains, since one can easily obtain the matching attributes, which are explicitly available in the *select* HTML tags. If matches are found, the agent knows how to fill the form and all necessary information to submit it is already available, so the process continues.

Raghavan *et al.* [26] propose a task-specific Hidden Web crawler called *Hidden Web Exposer* (HiWE). The approach was developed to automatically process, analyze and submit forms through an internal model of forms and form submissions.

The model treats a form F as a set of $(element, domain)$ pairs: $F = (E_1, D_1), (E_2, D_2), ..., (E_n, D_n)$ where the E_i is the element and the D_i is the domain. The elements E_n are the form fields f_i and domains D_n are the field domains $dom(f_i)$.

The values used to fill out forms are maintained in a special table called *Label Value Set* (LVS). LVS tables are associated to form fields. Each row in LVS table is a pair (L, V), where L is a label and $V = \{v_1, v_2, ..., v_n\}$ is

a value set assigned to label L. The V set has a M_v function that associates weights, between 0 and 1, to each set member. Each v_i is a possible value that is assigned to form field E if $label(E)$ matches with L. The estimated value $MV(v_i)$ represents the correctness of value v_i in relation to element E.

The main issue in this method is filling LVS table with the desired values for queries and, after this, the association of values to form fields. The LVS table also allows *label aliasing*, *i.e.,* two or more labels may share the same value set V.

The HiWE crawler supports four strategies for populating the LVS table: *(i) explicit initialization*, in which it can be supplied with labels and associated value sets at startup time, *(ii) built-in categories*, which it has built-in entries in the LVS table for some common categories, such as times, months, dates, days of week, etc., *(iii) wrapped data source*, in which entries for the LVS table are used for querying data sources through a well-defined interface, and *(iv) crawling experience*, which provides useful information which can be used when crawling new sites.

Liddle *et al.* [20] perform automatic form filling by assigning a default value to form fields. The authors have created a prototype tool that automatically retrieves the data behind a specific HTML form.

The strategy involves three steps: *(i)* issue the default query, *(ii)* retrieve a sample to determine whether the default query produces acceptable results, and *(iii)* analyze the retrieved information and submit new queries exhaustively until a limiting threshold is reached.

The authors heuristically select a reasonable minimum number of submissions to maximize the coverage. In order to do that, the size of the database behind the form is estimated, and then queries are issued until a certain percentage of completeness is reached.

Heuristics include the percentage of data retrieved, the number of queries issued, the number of bytes retrieved, the amount of time spent, and the number of consecutive empty queries. Each of these thresholds constitutes a sequential stopping criterion that can terminate the crawl before trying all possible queries (*i.e.,* all combinations of values for fields with finite domains).

The sampling batch needs to be large enough to cover the margins of the sample space. Let $f_1, f_2, ...f_n$ be the n be the fields with finite domains, and let $|f_i|$ represent the number of values for the i^{th} factor. $|f_i|$ stands for the field domain $dom(f_i)$. Then, the total number of possible combinations N for this form is $\prod |f_i|$, and the cardinality c of the largest field is $max(|f_1|, |f_2|, ..., |f_n|)$. Next, C is defined as the size of a sampling batch. C is calculated as $max(c, log_2 N)$.

This accounts for the cases in which there are many fields of small cardinality. If the C sample queries yield new data, the method proceeds by sampling additional batches of C queries at a time, until it reaches one of the user-specified thresholds or it exhausts all the possible combinations.

This method does not handle text fields, ignoring them whenever possible. If they are mandatory, and thus cannot be ignored, user's intervention is requested.

Alvarez *et al.* [1] propose an architecture called *DeepBot* for crawling the Hidden Web. The crawler works in three steps: (i) for every domain, the system tries to match its attributes with the fields of the form, using visual distance and text similarity metrics, (ii) by using the output of the previous step, the system determines whether the form is relevant with respect to the domain and, (iii) if the form is relevant, the crawler uses it to run the queries defined in the domain.

For each query, the system obtains a new URL to add to the list of URLs. The authors describe the domain definitions used to guide the data collection task. A domain definition is composed of a set of attributes $A = a_1, a_2, ..., a_n$, a set of queries $Q = q_1, q_2, ..., q_m$, and a relevance threshold μ.

The method uses an attribute set that represents form fields and a query set associated with the domain. A set of attributes has a name, a nickname list, and a specificity index s_i. The nickname list represents alternative labels that may identify the attribute in a query form. For instance, the attribute *author*, from a domain used for collecting data about books, could have nicknames such as *writer* or *writtenby*. The specificity index s_i is a number between 0 and 1 indicating how likely a query form containing such an attribute is actually relevant to the domain. For instance, in the book domain, the attribute ISBN would have a very high s_i, since the presence of this attribute in a form is a strong evidence that it deals with book search. On the other hand, *price* would have a low s_i value, since it could be related to any type of product.

The set of queries is a list of pairs *(attribute, value)* where *attribute* is an attribute a_i from the set of attributes A and *value* is a string. The query set is run on the discovered relevant forms. The labels of the fields are extracted and compared to attributes of the domain through textual similarity. The domain attribute values that match the fields are selected for filling the form. For that, heuristics based on visual distance measures between the form fields and the texts surrounding them are used.

Finally, the domain also includes a relevance threshold μ. The specificity indexes and threshold will be used to determine whether a given form is relevant to a domain. The authors do not present details of how a query list for each attribute is built.

3.3 Machine Learning applied to Complex Web Forms

The methods that employ machine learning techniques generally rely on manually labeled data to serve as training instances. As a consequence, user intervention is needed to validate the data and to make corrections on the labeling.

Toda *et al.* [29] describe a method, called *iForm*, for WFF based on value extraction from free text documents. The method is divided into two sub-problems: extracting values from the input text; and filling the form field using the extracted values. It automatically chooses segments from the input text and assigns them to the appropriate form fields. Free text documents are treated as sequences of tokens $t_1, t_2, ..., t_N$, representing individual words or punctuation. The extraction task consists in identifying segments from the documents, *i.e.,* a sequence of contiguous tokens, which are suitable for the fields in the form.

The method exploits features related to the content and the style of the values. These are combined in a Bayesian framework. The conditional probabilities (probability using content related features and probability using style related features) of a field associated with an extracted value of the text document are computed. The final conditional probability can be computed using a disjunctive operator *or* over the probabilities derived from each feature.

The probabilities are assigned to form fields f_i and the extracted values are the domains $dom(f_i)$. The approach relies on the knowledge obtained from the values of previous submissions for each field and on manual textual input (to correct errors). The authors do not report on experiments run on search forms.

3.4 Overlapping Combinations

Jian *et al.* [13, 14] present a method for Hidden Web crawling. The method combines heuristics and machine learning techniques for filling free-text Web forms. Therefore, we can reduce the model presented in Section 2 to a single form field f_1. The domain $dom(f_1)$ is the set of values with the highest harvest rates.

In this approach, the harvest rate for each query is encoded as a tuple representing its linguistic, statistic, and HTML features. The linguistic features are *part of speech* which represents the category of the word (noun, verb, adjective, etc.);the *length*, which represents the length of values in number of characters; and the language that the values fall into (useful for multilingual crawls). Statistical features include term frequency (TF), document frequency (DF), Term Frequency times Inverse Document Frequency (TF$\times$ IDF), and the Residual IDF (RIDF). Finally, the HTML features are the *TAG*, which consists of the HTML tags and attribute information, the *Location*, which represents the location information of node in the DOM tree derived from the HTML document, and the *Markedness* which determines how much the word stands out from the normal text in the HTML document (e.g. bold, underlined, italic, etc). The keywords with high harvest rates are used to train a machine learning model which will be applied to estimate the harvest rates for issued keywords which have not been submitted yet.

Jian *et al.* [13] present a framework based on Reinforcement Learning for Deep Web crawling. In the framework, a crawler is regarded as an agent and the Hidden Web database is the environment. The agent perceives its current state and selects an action (query) to submit to the environment (database) according to a long-term reward. The environment responds by giving the agent some reward, *i.e.,* new records, and changing it into the next state. The rewards of unexecuted actions are evaluated by their executed neighbors. Because of the learning policy, a crawler can avoid using unpromising queries, as long as some of them have been issued. Zheng [34] extends the work by [13, 14] by developing a Q-value approximation algorithm that allows the crawler to select a query by learning from the experience of previous queries. The Q-value is the metric that estimates the long-term rewards.

Dong and Li [9], similarly to Jian *et al.* [14], work with free-text forms applying machine learning and heuristics. Consequently, the model can be represented as a single form field f_1 and the domain $dom(f_1)$ is represented by the values according to the query harvest rate.

A sample if taken from the target database behind the form to get a sampling database. The same measures used in Wu *et al.* [31] are applied. Then, it automatically chooses several types of features (number of records retrieved, the length of the values) from the sampling database. Next, it learns a query harvest model from a multi-linear regression approach and employs the model to select queries to submit to the form.

The heuristics include a cost model, a coverage rate, and a query harvest rate. The cost of crawling a Web database as the total number of communication rounds between the crawler and the Web server. It is important to distinguish between the total number of communication rounds and 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.

The crawling cost $cost(q_i, DB)$ of querying the database DB with query q_i is defined as in Eq. 6:

$$cost(q_i, DB) = \frac{|R(q_i, DB)|}{k} \tag{6}$$

where $|R(q_i,DB)|$ stands for the number of all records in DB matched q_i and k corresponds to the maximum number of records displayed in each result page from the target Web site.

The coverage rate of a query q_i is defined as in Eq. 7:

$$CR(q_i,DB) = \frac{|R(q_i,DB)|}{|DB|} \quad (7)$$

where $|R(q_i,DB)|$ stands for the number of all the records in DB matched q_i and $|DB|$ corresponds to the number of total records in the Web database DB.

Finally, given a target Web database DB and a local database DB_{local} containing the data records already crawled from DB, the query harvest rate of q_i $HR(q_i,DB)$ is defined as in Eq. 8:

$$HR(q_i,DB) = k \times (1 - |R(q_i,DB_{local})|/|R(q_i,DB)|) \quad (8)$$

where $|R(q_i,DB_{local})|$ and $|R(q_i,DB)|$ correspond to the number of data records matched by q_i in DB_{local} and DB, respectively.

The training set is constructed by simulating the Hidden Web crawling using the sampling database. It uses features of the query result in each round to generate the features for candidate queries. Since the sample database is known, the method calculates the harvest rate of each candidate query. Finally, it selects the query with the highest harvest rate as the next query and continues the construction of the training set until all records in the sample database are crawled.

Madhavan *et al.* [22] employs just heuristics to fill free-text and complex Web forms. The goal of the method is to index the resulting HTML pages.

The authors present an algorithm to select input values for text search interfaces that accept keywords and an algorithm for identifying inputs that take only values of a specific type. HTML forms have n inputs and the method introduces the *query template* concept for Web forms. A query template fills a subset of the inputs, called *binding inputs*. The remaining are regarded as *free inputs* and discarded. The number of inputs that make up a template will be referred to as the *dimension* of the template. Multiple form submissions can be generated by assigning different values to the binding inputs. There are no details on how the values are assigned to a template. Each query template and its values are submitted and the results are evaluated to check how much information is retrieved.

A signature function is calculated for the results of submissions, which are compared against each other. A query template is considered *informative* if the generated result pages are sufficiently distinct. Otherwise, it is *uninformative* and thus discarded.

For infinite domain fields, the authors adopt an iterative probing approach to identify the candidate keywords for a field. At a high level, they assign an initial seed set of words as values for the text field and build a query template with the text field as a single binding input. The method exploits the values from a page by identifying the most relevant values to its contents. Thus, the technique uses TF×IDF to choose the values. For the initial values, the top $N_{initial}$ words on the form page are selected.

For the candidate keywords in iteration $i+1$, assume that W_i is the set of all Web pages generated and analyzed until iteration i. Let C_i be the set of words that occur in the top N_{probe} words on any page in W_i. From C_i, the words discarded are those that have so far occurred in too many pages in W_i (since they are likely to correspond to boilerplate HTML that is is found on all pages on the form site), or those that occur only in one page in W_i (since they may be too specific and thus not representative of the contents of the site). The field domains $dom(f_k)$'s of form fields f_k are the remaining values in C_i.

Kantorski *et al.* [15] present an automatic method that combines heuristics for filling both free-text and complex Web forms. The method explores two strategies. The first is how to select good values, or queries, to submit to a particular form in order to retrieve more data with fewer submissions. The second strategy is how to fill the fields efficiently, especially text fields.

The authors employ the concepts of *template* (similar to Madhavan *et al.*'s [22] notion of query template) and *template instance*. Templates are represented by form fields and their combinations. A template instance assigns a value to each field considered for the form submission. A template is *informative* if its template instances retrieve enough distinct data and *uninformative* templates are discarded. The idea is to use information from previous submissions to avoid wasteful submissions (*i.e.,* which do not add new information to the existing set). The informativeness evaluation is repeated for all generated templates and avoids unnecessary submissions in templates of higher order. An instance template considered non-informative will cause instance templates of higher order being discarded.

The choice of values for fields with infinite domains is based on a feedback loop, in which each element has an effect on the next one, until the last element produces feedback on the first element. The idea is use information from the form itself plus the data retrieved from previous submissions as input to future submissions. Heuristics include ranking functions, such as the collection frequency (CF), IDF, and the number of distinct records retrieved (nr) combined into two scores $r1 = cf_t \times idf_t$ and $r2 = nr \times idf_t$.

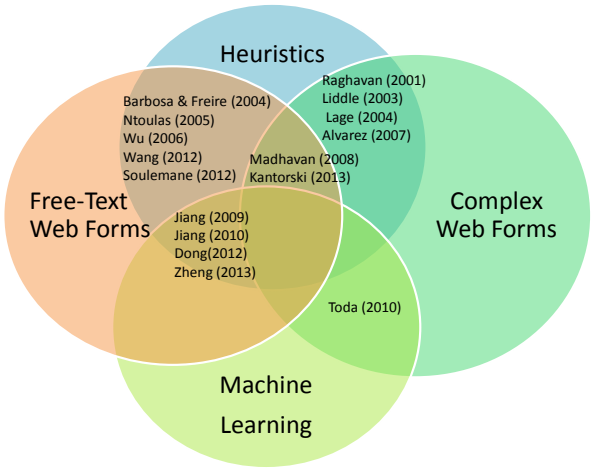


Figure 3: Holistic View

4. COMPARATIVE ANALYSIS

This section presents a comparative analysis of the surveyed methods according to two perspectives. In the *holistic analysis*, each method is studied in its entirety considering the aspects presented in Section 2, *i.e.,* the filling method and the type of interface. In the *Cartesian analysis*, the methods are studied as a collection of dissociated parts.

Holistic Analysis. The surveyed approaches apply heuristics or machine learning techniques or a combination of both. Figure 3 shows how the surveyed approaches are classified under the holistic view. Heuristics are used to simplify the process of WFF and yield good results for both types of interface. Machine learning techniques were added to improve the results reached using heuristics. The surveyed approaches report achieving similar scores using the evaluation metrics, regardless of whether they apply heuristics or machine learning.

Looking at Figure 3, we notice that the research is condensed around certain parts of the space as most of the existing WFF methods [1,2,15,18,20,22,25–27,30, 31] employ heuristics for selecting values for fields in free-text and complex Web forms. The reason for that is heuristics simplify the process of WFF and yet they yield good results.

Methods that rely solely on machine learning techniques (such as [29]) have the limitation of requiring manual labeling. To avoid human intervention, some methods [9, 14] combine heuristics and machine learning. Yet, there are methods that combine heuristics and machine learning to handle free-text Web forms only [9,13,14]. Finally, there are no approaches that combine heuristics and machine learning for handling both type of interfaces (free-text and complex Web forms). This probably happens because the existing methods reach good results for free-text forms. This is not true for complex Web forms, as approaches have yet to be proposed to choose good values for them.

While there are many methods that handle one type of interface, a gap is still open in the selection of proper values for both kinds of forms. Only two methods [15, 22] select values for both Web forms and both use only heuristics [15,22].

Figure 3 also shows the evolution of WFF in the past decade. In the early 2000s, progress was made in terms of free-text and complex forms, separately, using heuristics. In the late 2000s, solutions that work with both free text and complex types of interface, were developed. Since 2010, a considerable progress has been made in terms of machine learning techniques. The adoption of machine learning shows an increase in the level of sophistication of WFF.

Cartesian Analysis. Table 1 presents a summarized view of the methods showing a number of aspects: (i) how they generate seed (initial) values; (ii) how they generate the remaining values; (iii) how much they rely on prior knowledge; (iv) the type of submission method (get/post) of the forms handled by the approach; and (v) dependency on human intervention. Table 1 also contains information about the experiments reported on the surveyed works, such as number of forms and evaluation results. These values cannot be directly used to compare the approaches since experiments have been performed on different forms and evaluated under different metrics. Our goal is just to provide an indication of quality.

Regarding the generation of the initial values for text fields, several methods [1, 18, 25, 26, 30, 31] depend on a predefined list of values. This list is, generally, defined manually or previously built for each form domain. This is shown in the column entitled "prior knowledge". Other methods [2,15,22,27] extract the information from the HTML page where the form is located.

The advantage of getting the initial values from a previously assembled list is that these values tend to retrieve valid results. The disadvantage is that such lists need to be assembled for each form, which becomes prohibitive when dealing with a large number of forms. On the other hand, methods that do not rely on predefined lists have the advantage of being automatic. Their problem is to discover what are the good initial values for filling fields. Also, there is a higher submission cost associated with the task of discovering these values.

Distinct criteria are used by the approaches to select the remaining values. Liddle *et al.* [20] do not select values automatically for text fields as user intervention is needed. Madhavan *et al.* [22] use traditional information retrieval metrics such as TFxIDF while [2, 27] adopt only the term frequency. Kantorski *et al.* [15]

Table 1: Summary of the surveyed works

Method	Text Field Seed Generation	Value Generation	Prior Knowledge	Submission Method	Human Intervention	# Forms	Evaluation Metric
Barbosa and Freire[2]	page containing the form	iteratively submitting queries using values obtained in previous iterations based on term frequency	No	unknown	No	8	Coverage (79% - 8 forms)
Soulemane *et al.[27]*	page containing the form	iteratively submitting queries using values obtained in previous iterations based on term frequency	No	get	No	1	Number of values
Ntoulas *et al.[25]*	manually defined	iteratively submitting queries using values obtained in previous iterations based on term probability	No	get/post	Yes	4	Coverage (84%)
Wang *et al.[30]*	random sample of domain corpora	iteratively submitting queries using values obtained in previous iterations based on set of data retrieved	No	get/post	No	4	Sample Size (2,000)
Lage *et al.[18]*	sample data repository	sample data repository	Yes	unknown	No	27	Precision and recall (93%)
Raghavan *et al.[26]*	label value set table	label value set table	Yes	get/post	Yes	50	Submission Efficiency
Liddle *et al.[20]*	default values	not applicable	No	get	Yes	13	Coverage (80%)
Wu *et al.[31]*	randomly selected values from a pre-exisitng database	Attribute Value Graph Domain Statistics Table (DT)	Yes	get/post	No	5	Coverage (without DT 90%) (with DT 95%)
Alvarez *et al.[1]*	pre-defined domain attributes	domain attributes table	Yes	unknown	Yes	30	Precision and recall (>90%) except in one case
Toda *et al.[29]*	textual document given by user	probability of a field given a word n-gram	Yes	get/post	Yes	5	Precision, recall and F-Measure (73%)
Jian *et al.[13,14]*	page containing the form	query harvest rate using features of values	No	unknown	No	3	Coverage (>80%)
Dong and Li[9]	sample data repository	query harvest rate using features of values	No	get/post	No	3	Coverage (95%)
Kantorski *et al.[15]*	page containing the form	iteratively submitting queries using values obtained in previous iterations based on CFxIDF and distinct rows retrieved	No	get/post	No	11	Coverage and Efficiency (>82%)
Madhavan *et al.[22]*	page containing the form	iteratively submitting queries using values obtained in previous iterations based on TFxIDF	No	get	No	10	Coverage (> 55%)

combine CF together with IDF in the CF×IDF measures and the total number of distinct rows retrieved for choosing values. Wu *et al.* [31] adopt the HR. Wang *et al.* [30] match HR and OR. The value of HR is a limitation in Wang *et al.*'s work [30] because the total number of rows behind the form is needed and, for most real Hidden Web sources, this number is unknown. Finally, there are some methods [9,13,14] that use features about the submissions to select values.

Most approaches handle the *get* [20, 22, 27] submission method, while some can work for both *get* and *post* [9, 15, 25, 26, 29–31]. This information, however, is not evident in some of the surveyed works [1, 2, 14, 18]. We believe that approaches which do not clearly state the submission method use only the *get* method. This submission method has an advantage compared to the *post* method as field values are included as part of the URL in the HTTP request. As a result, the URLs identified can be directly indexed by the search engines.

The number of forms used in the experiments varies considerably (from 1 to 50). Experimenting with real Web forms is tricky as they frequently change, may have high response times or even be unavailable for some periods. These difficulties impact the scale of the tests. Approaches that use machine learning techniques report good evaluation results (averaged across all forms). However, they performed experiments with a small number of Web forms.

Considering the two methods used by the state-of-the-art in WFF (*i.e.,* heuristics and machine learning), we notice that heuristic-based methods have been favored over machine learning. However, many of the approaches rely on user intervention. Both heuristic [1, 18, 20, 25, 26] and machine learning [29] techniques require manual specification of initial values or annotation of the training data. Only one method [9] is completely automatic; however, it handles only free-text forms.

In terms of interface, methods for one type of form are more common than for both. There was also a logical transition from free-text to complex and then to both types. The only approach that handles complex Web forms and uses machine learning is the one by Toda *et al.* [29]. This approach, however, is designed for forms that add rows to a database and not search forms.

5. FUTURE DIRECTIONS

This section presents some insights into trends and future directions in WFF.

Initial seed generation. Most methods report evaluation results in terms of coverage [2, 9, 14, 15, 20, 22, 25, 31]. Precision, recall, and F-measure are also used. [1, 18, 29]. Others propose new evaluation measures [26, 27, 30]. We analyzed the the number of forms used in the experimental evaluation of approaches that apply the coverage metric [2, 9, 14, 15, 20, 22, 25, 31] and how each approach generates the initial values (automatically or using a predefined list). The average coverage for both approaches is similar ($\tilde{8}2\%$); however, experiments with automatic seed generation have been done on more forms. This suggests automatic solutions for generating initial values are more scalable and thus should gain more attention in future approaches.

Reducing human intervention. Human intervention is required by 5 out of the 15 surveyed methods [1, 20, 25, 26, 29]. The amount of intervention varies from labeling data to having to input the values manually. While human intervention may be feasible when dealing with a reduced number of forms, it poses a bottleneck on the approach. Removing human intervention while keeping good results is likely to be the goal of future approaches which aim at being scalable.

Handling complex forms. In complex Web forms, future research could concentrate on modeling the relationship among the multiple attributes in the form. In order to tackle that, understanding Hidden Web forms is necessary. Form understanding is the process of extracting semantic information from an interface [10, 12, 16]. The combination of filling methods and semantic services could be an alternative for improving the automatic filling of complex Web forms.

Using Machine Learning. Regarding machine learning approaches, future investigations could encompass the evaluation of several learning models to determine which is the best suited to address WFF. Furthermore, meta-learning approaches could also include the identification of the appropriate subset of learning algorithms is recommended for the task of choosing field values.

Machine Learning applied to complex forms. Figure 3 shows that there is no method that combines both types of interface with heuristics and machine learning. Thus, a possible future direction may be filling this gap by creating a technique that works for both types of Web forms and applies both types of filling method. Future approaches could adopt heuristic rules for extracting meta information (features) about the submissions and employ machine learning techniques to obtain values for the fields, without human intervention.

Allowing incremental updates. A limitation of all surveyed approaches is the crawling process is always repeated from the start. This is undesirable when dealing with large databases as it has the overhead of discovering already crawled data. Future approaches should aim for incremental updates so as to optimize the process.

Dealing with specific domains. Finally, most WFF approaches have been designed for general crawling. We found no methods that deal with WFF that is focused on a specific topic. Domain-specific features could be explored so as to yield improved results in WFF.

6. CONCLUSION

The Hidden Web represents an important portion of the Web which can only be reached by filling a form. Uncovering Hidden Web data is a challenging task. A scalable approach to gather such data depends on automatically filling forms. The goal is to choose suitable values so that meaningful data can be retrieved.

In this article, we present a survey dedicated to works that address the problem of Web form filling. Two types of analysis were performed over 15 key works in this area. The *holistic analysis* describing each method according to the filling method and the type form that they handle, and a *Cartesian analysis* which considers how the methods perform each of the subtasks involved in the process. This work concludes presenting future directions in this area.

Acknowledgments

The authors thank the anonymous reviewers for their helpful suggestions. This research was partially supported by CNPq/Brazil, project 478979/2012-6. G. Kantorski received a CAPES-Brazil scholarship.

7. REFERENCES

[1] M. Álvarez, J. Raposo, A. Pan, F. Cacheda, F. Bellas, and V. Carneiro. Crawling the content hidden behind web forms. *ICCSA*, pages 322–333, 2007.

[2] L. Barbosa and J. Freire. Siphoning hidden-web data through keyword-based interfaces. In *Brazilian Symposium on Databases*, pages 309–321, 2004.

[3] L. Barbosa and J. Freire. An adaptive crawler for locating hidden-web entry points. In *WWW*, pages 441–450, 2007.

[4] M. Bergman. The deep web: Surfacing hidden value. *Journal of Electronic Publishing*, 7(1):07–01, 2001.

[5] J. Caverlee, L. Liu, and D. Buttler. Probe, cluster, and discover: focused extraction of qa-pagelets from the deep web. In *Data Engineering*, pages 103 – 114, march-2 april 2004.

[6] C.-H. Chang, M. Kayed, R. Girgis, and K. F. Shaalan. A survey of web information extraction

systems. *TKDE*, 18(10):1411–1428, 2006.
[7] K. Chang, B. He, C. Li, M. Patel, and Z. Zhang. Structured databases on the web: Observations and implications. *SIGMOD Rec*, 33(3):61–70, 2004.
[8] V. Crescenzi, G. Mecca, P. Merialdo, et al. Roadrunner: Towards automatic data extraction from large web sites. In *VLDB*, volume 1, pages 109–118, 2001.
[9] Y. Dong and Q. Li. A deep web crawling approach based on query harvest model. *Journal of Computational Information Systems*, 8(3):973–981, 2012.
[10] E. C. Dragut, W. Meng, and C. T. Yu. *Deep Web Query Interface Understanding and Integration*. Morgan & Claypool Publishers, 2012.
[11] D. Florescu, A. Levy, and A. Mendelzon. Database techniques for the world-wide web: a survey. *SIGMOD Rec*, 27:59–74, September 1998.
[12] T. Furche, G. Gottlob, G. Grasso, X. Guo, G. Orsi, and C. Schallhart. Opal: automated form understanding for the deep web. In *WWW*, pages 829–838, 2012.
[13] L. Jiang, Z. Wu, Q. Feng, J. Liu, and Q. Zheng. Efficient deep web crawling using reinforcement learning. In *PAKDD*, pages 428–439, 2010.
[14] L. Jiang, Z. Wu, Q. Zheng, and J. Liu. Learning deep web crawling with diverse features. In *WI/IAT-Volume 01*, pages 572–575, 2009.
[15] G. Z. Kantorski, T. Moraes, V. Moreira, and C. Heuser. Choosing values for text fields in web forms. In *ADBIS*, pages 125–136, 2013.
[16] R. Khare, Y. An, and I. Song. Understanding deep web search interfaces: A survey. *SIGMOD Rec*, 39(1):33–40, 2010.
[17] A. H. Laender, B. A. Ribeiro-Neto, A. S. da Silva, and J. S. Teixeira. A brief survey of web data extraction tools. *SIGMOD Rec*, 31(2):84–93, 2002.
[18] J. P. Lage, A. S. da Silva, P. B. Golgher, and A. H. F. Laender. Automatic generation of agents for collecting hidden web pages for data extraction. *Data Knowl. Eng.*, 49(2):177–196, May 2004.
[19] S. Lawrence and C. L. Giles. Searching the world wide web. *Science*, 280(5360):98–100, 1998.
[20] S. Liddle, D. Embley, D. Scott, and S. Yau. Extracting data behind web forms. *Advanced Conceptual Modeling Techniques*, pages 402–413, 2003.
[21] Y. Lu, H. He, H. Zhao, W. Meng, and C. Yu. Annotating structured data of the deep web. In *ICDE*, pages 376–385. IEEE, 2007.
[22] J. Madhavan, D. Ko, Ł. Kot, V. Ganapathy, A. Rasmussen, and A. Halevy. Google's deep web crawl. *Proc. of the VLDB Endowment*, 1(2):1241–1252, 2008.
[23] M. C. Moraes, C. A. Heuser, V. P. Moreira, and D. Barbosa. Pre-query discovery of domain-specific query forms: A survey. *TKDE*, 25(8), 2013.
[24] U. Noor, Z. Rashid, and A. Rauf. A survey of automatic deep web classification techniques. *International Journal of Computer Applications*, 19(6):43–50, Apr. 2011.
[25] A. Ntoulas, P. Zerfos, and J. Cho. Downloading textual hidden web content through keyword queries. In *JCDL*, pages 100–109, 2005.
[26] S. Raghavan and H. Garcia-Molina. Crawling the hidden web. In *VLDB*, pages 129–138, 2001.
[27] M. Soulemane, M. Rafiuzzaman, and H. Mahmud. Article: Crawling the hidden web: An approach to dynamic web indexing. *International Journal of Computer Applications*, 55(1):7–15, Oct. 2012.
[28] K. Tjin-Kam-Jet, D. Trieschnigg, and D. Hiemstra. Free-text search over complex web forms. In *Multidisciplinary Information Retrieval*, pages 94–107. Springer, 2011.
[29] G. Toda, E. Cortez, A. da Silva, and E. de Moura. A probabilistic approach for automatically filling form-based web interfaces. *Proc. of the VLDB Endowment*, 4(3):151–160, 2010.
[30] Y. Wang, J. Lu, J. Liang, J. Chen, and J. Liu. Selecting queries from sample to crawl deep web data sources. *Web Intelligence and Agent Systems*, 10:75–88, January 2012.
[31] P. Wu, J. Wen, H. Liu, and W. Ma. Query selection techniques for efficient crawling of structured web sources. In *ICDE*, pages 47–47, 2006.
[32] Y. Zhai and B. Liu. Structured data extraction from the web based on partial tree alignment. *TKDE*, 18(12):1614–1628, 2006.
[33] H. Zhao, W. Meng, Z. Wu, V. Raghavan, and C. Yu. Fully automatic wrapper generation for search engines. In *WWW*, pages 66–75, 2005.
[34] Q. Zheng, Z. Wu, X. Cheng, L. Jiang, and J. Liu. Learning to crawl deep web. *Information Systems*, 38(6):801–819, 2013.