Choosing Values for Text Fields in Web Forms

Gustavo Zanini Kantorski, Tiago Guimaraes Moraes, Viviane Pereira Moreira and Carlos Alberto Heuser

Abstract Since the only way to gain access to Hidden Web data is through form submission, one of the challenges is how to fill Web forms automatically. In this paper, we propose algorithms which address this challenge. We describe an efficient method to select good values for text fields and a technique which minimizes the number of form submissions and simultaneously maximizes the number of rows retrieved from the underlying database. Experiments using real Web forms show the advantages of our proposed approaches.

Key words: Crawling, Deep Web, Filling Forms, Hidden Web

1 Introduction

The surface Web is the portion of the World Wide Web that can be reached by direct link navigation. However, a vast portion of the information on the Web is available in online databases and can only be reached through Web form filling and submission. This portion of the Web is known as *Hidden Web* [16] or *Deep Web* [4] and it is not indexed by conventional search engines. The Hidden Web is more diversely distributed, has a large number of structured databases, and suffers an inherent limitation of crawling [5].

Gustavo Z. Kantorski · Tiago Guimaraes Moraes · Viviane Pereira Moreira · Carlos Alberto Heuser
UFRGS, Porto Alegre, Brazil,
e-mail: gzkantorski@inf.ufrgs.br
Tiago G. Moraes
e-mail: tgmoraes@inf.ufrgs.br
Viviane P. Moreira
e-mail: viviane@inf.ufrgs.br
Carlos A. Heuser
e-mail: heuser@inf.ufrgs.br

In this context, one of the challenges is how to automatically fill forms in order to gain access to the data. This task is not trivial, since forms were designed to be used by human beings. Most architectures aim at retrieving as much information as possible from the online database behind the form. In order to do that, the simplest solution would be submitting the combination of all possible field values in a cartesian product. However, this solution is not feasible when the number of fields and possible values are large. For example, a Web form composed of five fields with thirty possible values each, has over 24 million filling combinations. Some combinations are wasteful, *i.e.,* they either return identical results (not adding new data) or fail to retrieve any data.

In this paper, we propose an automatic method for filling forms. Our 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, specially the text fields, which do not have a set of pre-determined values. The strategy to minimize the set of queries, *i.e.,* the number of form submissions, involves pruning the set of all possible queries. As each query is submitted, data extracted from the resulting page is used to identify wasteful queries and prune them. For filling form fields, this paper presents a solution called FTF (*Filling Text Fields*) which focuses on fields which do not have a set of pre-determined values, such as text boxes. Most of the existing work on form filling [1, 6, 8, 10, 11, 13] overlooks the problem of finding good values for text fields. Existing solutions for dealing with text fields usually rely on a list of values previously built by a specialist [10], on a sample of known values [3], or they entail the extraction and understanding of the fields [1, 7, 11] (which depends on the language and on the domain of the forms). Our proposed solution is automatic, requires no training, and relies on feedback from previous submissions. Domain often influences value selection for the fields. Although our method does not use domain knowledge explicitly, our experiments show that the values generated for the fields are domain-related. The results show that in most cases, our approach achieves superior coverage compared to a baseline method.

2 Definitions

Data available on the Hidden Web are accessible through Web forms (usually HTML forms). A form contains fields to be filled so that a form submission may retrieve data, accessing the online database hidden behind the form. Fields can be text boxes, selection lists, check boxes, radio buttons, and submit buttons. They can be classified in two groups: (i) fields with finite domains (such as selection lists) and fields with infinite domains (such as text boxes, in which a user can type any value). Once the values have been filled, the form can be submitted. A form can be submitted by two methods, *get* or *post*. Our strategies work with Web forms of both methods. In the context of this paper, it is important to distinguish between form domain and attribute domain. *Form domain* is the subject or topic to which the form belongs

(*e.g.,* books, airfares, hotel, jobs, etc.). For a study on the domains found on the Hidden Web, please refer to Chang *et al.* [15]. The *attribute domain* is the set of values for a field. For example, the domain of a selection list field is presented inside the *option* HTML tag.

Madhavan *et al.* [3] and Cafarella *et al.* [14] divide text boxes into two groups: *(i) generic text boxes* which represent text boxes in which the words are used to retrieve all documents in the backend database containing those words; and *(ii) typed text boxes* which serve as a selection predicate on a specific attribute in the where clause of a SQL query over the backend database. For a set of submissions, we define the concepts of *template* (similar to Madhavan *et al.*'s [3] notion of query template) and *instance template*. Templates are represented by form field combinations. For example, for a form with four fields *A, B, C*, and *D*, we have the following templates: *A, B, C, D, A&B, A&C, A&D, B&C, B&D, C&D, A&B&C, A&B&D, A&C&D, B&C&D and A&B&C&D*. If there are n fields in a form, there are $2^n - 1$ templates. The number of fields that make up a template will be referred to as the *dimension of the template*. For example, the dimension of the template *A&B&C&D* is four and the dimension of the template *A&B* is two. An instance template associates a value to each field considered in the form submission. For example, an instance template for the template *A&B&C* could be *A=a1&B=b1&C=c1* and it could be represented by the pair (A&B&C,a1&b1&c1). In practice, instance templates are represented by URLs. More formally we have:

Definition 1. (Template) Let $F=\{f_1, f_2, ..., f_n\}$ be a set of fields located in a Web form and let $T=\{t_1, t_2, ..., t_m\}$ be a set of all combinations of the elements in F. Each element in T is defined as a template.

Definition 2. (Instance Template) Let t be a template in T and let $V=\{v_1, v_2, ..., v_m\}$ be the domain of t. An instance template is an attribute-value pair (t,v) where v is an element from V.

3 Related Work

The literature has several solutions for automatic form filling. Raghavan *et al.* [10] manage Web form filling through tables called *Label Value Set* (LVS). LVS tables are associated to form fields. The main issue in this method is filling the LVS table with the desired values for queries and the association between values and fields.

In Liddle *et al.* [8], automatic form filling is carried out by assigning default values to form fields. Text fields are ignored and, if they are mandatory, user intervention is needed.

Barbosa *et al.* [6] present an approach for filling forms based on keywords. The discovery of words is based on the data coming from the database itself, instead of random word generation. On the other hand, they do not handle Web forms that do not contain keyword fields.

Wu *et al* [12] present a form filling technique based on a feedback process of the values filled in the form. A limitation is that, in the query, just one form field can be used at a time–combining form fields for several queries is not possible.

Jian *et al.* [1] present a method in which each keyword obtained from the result pages is encoded as a tuple representing its linguistic, statistic, and HTML features. These tuples (issued keywords) are used to train a model using machine learning algorithms which will be used to evaluate the harvest rates for un-issued keywords.

Toda *et al* [11] describe a form filling solution based on value extraction from a text document. This solution exploits features related to the content and to the style of values, which are combined through a Bayesian framework. The approach relies on the knowledge obtained from the values of previous submissions for each field.

The work by Madhavan *et al.* [3] describes a system for surfacing the content of the Hidden Web. They improved the keyword selection algorithm by ranking keywords with respect to their TF-IDF.

Existing methods for filling form fields [1, 3, 6, 8, 10, 11, 13] focus on selecting values for fields with finite domains and for fields with infinite domains classified as *keyword text boxes*. A gap is still open in the selection of good values for fields with infinite domains classified as *typed text boxes* (See Section 2). Proposed solutions usually employ a set of pre-defined values defined by a specialist [10], require the extraction and understanding of field labels [1, 7, 11], use the generic frequency distribution of each keyword [3, 6, 8, 13] or use a sampling of the known values [3]. Our main contribution is a totally automatic approach for finding values for typed text boxes. To the best of our knowledge there are no automatic solutions for filling this type of field.

4 Automatic Filling of Web Forms

Figure 1 shows the proposed architecture. Our main contributions are in the highlighted modules *Value Selection* and *Instance Template Generation and Submission*. The *Candidate template generation* module represents the HTML form processing to generate all possible templates. The *Value Selection* module finds the values to fill each field in the form. Here the main problem is how to choose values for fields with infinite domains. Details are discussed in Section 4.2.

The *instance template generation and submission* module attaches input values to each field of each template, creating its set of instance templates. More details are given in Section 4.1. The *Data extraction* module extracts the information from the pages resulting from form submissions. This extraction is needed to find where the data region is located in the result page. Information about ads and presentation from the result page are discarded. The extracted data is used to evaluate each template. This data is also used to populate a database, which is serves as basis for the generation of values for fields with infinite domains (*Value Selection* module). The intuition is that higher coverage may be achieved if instead of using randomly selected data, values resulting from previous submissions are used. The *informativeness evalua-*

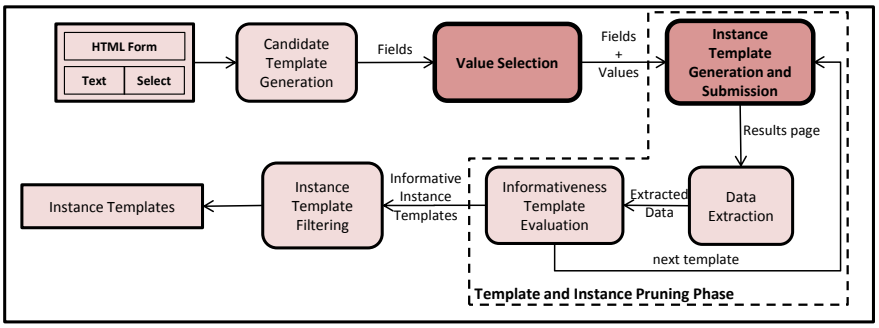

Fig. 1 Proposed Architecture

tion module checks if the template is informative after the submission of the selected instance templates. A template is *informative* if its instance templates retrieve sufficiently distinct data. If a template t is considered non-informative, higher order templates including t will be discarded. At the end of the process, after processing all templates, the *Filtering Instance Templates* module determines the minimum number of instance templates needed to retrieve all distinct rows extracted from the pages resulting from form submissions.

4.1 Instance Template Generation and Submission

The main idea here is that, for each new submission, information from previous submissions is considered in order to avoid wasteful submissions which will incur in blank pages, error pages, or pages with duplicate rows which do not add new information to the existing set. The Instance Template Pruning (ITP) method is used to prune the wasteful instances of a template. In order to identify such wasteful instances, we employ the concept of informativeness of an instance template. Each instance template submitted is evaluated as informative or non-informative. All non-informative instance templates are added to a *pruning list* and are discarded in higher-order templates. This process is shown in Figure 2. The pruning list is initially empty and starts being filled as non-informative instances are discovered.

For each template, a set of possible instance templates are generated, taking into account the pruning list. Instances are submitted one by one and, for each of them, the information from the response page is extracted. The extracted data feeds the informativeness evaluation module. This process is repeated for all generated templates and avoids unnecessary submissions in templates of higher order. Note that an instance template considered non-informative implies the pruning of other instance templates in templates of higher order. The method usually generates templates with order less than three because most templates of order greater than three do not return distinct data and thus are discarded [3].

Figure 3 shows an example of instance template pruning. Three fields (A, B, C) are considered, with three possible filling values each. The underlined values were

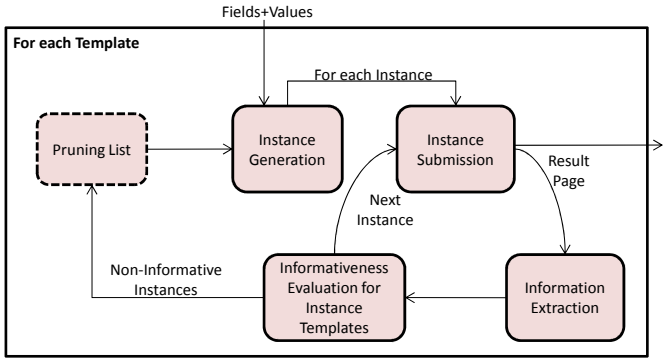

Fig. 2 Instance Template Generation and Submission module

submitted and considered non-informative. The strikethrough values were pruned and not even submitted. The other values were submitted and considered informative. The arrows show which values were considered in the pruning list for the creation of the instances for each template. For example, the instance of the template A, A = A1 was considered non-informative. Thus, other templates that take the filling of field A (templates AB, AC and ABC) into account had instances with value A1 removed from the group of possible instances for submission. This reduces the number of instances to be tested. In a naive method that considers the cartesian product of possibilities, 63 instance templates would be submitted. In the ITP method, that number is reduced to 33 since with the information that there are four non-informative instances, the system has learned that there are another 30 instances which would not return new data and thus do not need to be submitted.

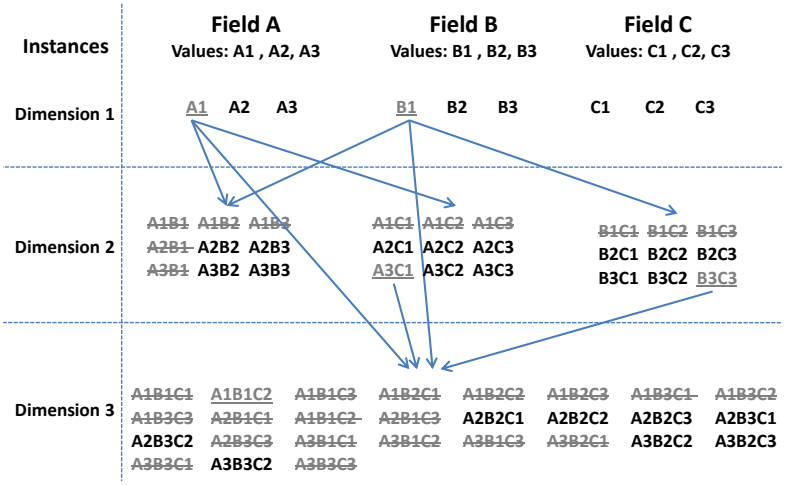

Fig. 3 An example for ITP

4.2 Value Selection

The goal of selecting good values for text boxes is to discover the values to be used in queries in order to retrieve as many rows hidden behind the form, at an acceptable cost. We present the *Filling Text Fields* (FTF) method which generates values for text boxes. FTF 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 to use information from the form itself plus the data retrieved from previous submissions as input to future submissions.

The generation of values for fields with finite domains is fairly easy as the values which may be selected are found inside the Web form code. However, the same does not happen for fields with infinite domains. It is not possible to know beforehand the quantity and the quality of the desired values. By quantity we mean the number of selected values which will lead to the desired coverage and by quality we mean the choice of values that retrieve more distinct data. These characteristics turn the task of finding good values for text boxes into a challenging problem. One way to find initial values could be to design a list of words associated with the text box. However, this is not feasible since there are Web forms in multiple domains that may have similar fields whose values present a wide variation.

In the FTF method, the selection of values for fields with infinite domains is divided into two steps. In the first step, a score r_1 is calculates with the aim of selecting meaningful tokens from previous submissions. The second step catches tokens from the first step to compose new queries and a new score r_2 is generated. Pages resulting from each submission are stored in a database. We calculate the score r_{1t} (Equations 1 and 2) for each token, and take the k-highest scoring tokens to use in future submissions. The intuition here is that, by associating cf, more records are retrieved. The *idf* component is used to remove tokens present in all results, for instance, header and title table. For each new template containing fields with infinite domains, the database is analyzed again and new k tokens are extracted. Thus, for distinct templates that have fields with infinite domains, distinct values are generated. Finally, we calculate the score r_{2t} (Equation 3) for each token. The ranking is based on number of rows retrieved by tokens (r_2). The difference between r_1 and r_2 is that, r_1 uses the values for discovering new values for text fields while r_2 uses the values for filling text fields. The FTF method works well for both keyword fields and for typed fields.

$$r_{1t} = cf_t \times idf_t \tag{1}$$

and

$$idf_t = log\frac{N}{df_t} \tag{2}$$

and

$$r_{2t} = nr \times idf_t \tag{3}$$

where cf_t is the number of times a token t appears in the database containing the rows obtained so far; N is the number of submissions; df_t is the number of result pages containing t; and nr is the number of records retrieved from the submission.

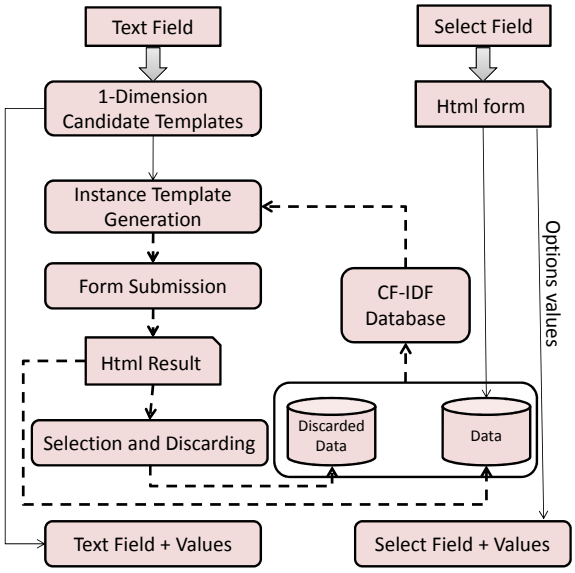

Fig. 4 Value Selection Module

For our purposes, we divide the Web forms in two types: those that contain only text boxes and those that contain, besides text boxes, finite domain fields, such as selection lists. The division is necessary, because depending on the type of form, the generation of initial values for the fields is different.

Figure 4 shows the proposed process for discovering good values for filling text boxes. The *Select Field* component represents the fields with finite domains. These fields are filled by the values extracted from the code of the form, in the *option* tag from HTML form. Queries are generated by values extracted from option tag and then submitted. Query results are stored in a database.

The order of submission of queries always starts by finite domain fields followed by infinite domain fields. The database containing query results will be used later to generate values that are used to fill infinite domain fields. The information inside the HTML page which contains the form is extracted only when the form has just infinite domain fields. The information is tokenized and ranked according to Equations 1 and 2. The n most frequent terms are selected to compose new queries.

The *Text Field* component represents the text box fields. Our proposal selects values only for dimension-1 templates. These values will be combined for higher dimension templates. For each template, n instances are generated. The submission of all instances is called *iteration*

The results for each instance are evaluated by checking the number of records retrieved (r_2). Queries that do not return records are discarded. Queries that return just a single row are also discarded because its likely that the single row contains just the heading and not data. In addition, the terms that return the fewest rows are

also discarded. The intuition is that, by discarding values that return few rows, we are making room for the selection of new terms that will return more rows in the next iteration.

The FTF method works well for both keyword fields and for typed fields. The main difference between the FTF method and other techniques [1, 6, 8, 10, 11, 13] is that it generates good values for typed fields in a fully automatic manner.

5 Experiments

This section presents experiments performed in order to evaluate the proposed strategies for input value selection and instance template pruning. All experiments were carried out using real Web forms. The proposed strategies and the architecture are domain independent. Forms from several domains (such as, Jobs, Books, Movies, and Food), and sizes were used in experiments. Table 1 shows details about the forms used in experiments. The number of Web forms used in our experiments is similar to what is used in related work [6, 8, 10, 11].

Three metrics were used in the evaluation. The coverage [6] C_f, of a form f is the number of distinct records extracted during the whole process. The execution efficiency [13], EE_f, for a form f is the ratio between coverage and the number of URLs submitted ($Total_{submitted}$) in the process. The indexing efficiency, IE_f, for a form f is the average number of records obtained for each distinct URL indexed ($Total_{indexed}$). For 1-dimension templates and finite domain fields, URL submissions are generated according to the options that exist for each field. For n-dimension templates, where $n > 1$, there are several strategies for selection. Kantorski *et al* [2] discuss four strategies to generate URLs for templates. In this paper, the *k-allValues* strategy was used. This strategy uses all values of fields at least once. The intuition behind this method is that the selection of all values can reach higher data coverage. All instance templates (URLs) are generated and only a subset is submitted according to the selected values by *k-allValues* strategy. For the data extraction process, the MDR method [9] is used.

Table 1 Form properties

ID	Web Form	#Fields text	#Fields select	Id	Web Form	#Fields text	#Fields select
Forms for FTF method				Forms for ITP/ITTP method			
1	http://www.beerintheevening.com/pubs/	3	1	1	http://www.foodandwine.com/search/	3	1
2	http://www.foodandwine.com/search/	3	1	2	http://www.global-standard.org/	1	3
3	http://www.mymusic.com/advancedsearch.asp	6	2	3	http://onlineraceresults.com/search/index.php	2	0
4	http://www.posteritati.com/advanced_search.php	1	1	4	http://www.phillyfunguide.com/	1	2
5	http://www.e4s.co.uk/	1	2	5	http://www.whoprofits.org/	2	1
6	http://www.whoprofits.org/	2	1	6	http://www.hcareers.com/seeker/search/	1	5
7	http://www.mldb.org/search-bf	4	0	7	http://www.rtbookreviews.com/rt-search/books	1	2
8	http://usajobs.opm.gov/	2	0	8	http://formovies.com/search/combined.html	3	0
9	http://www.movlic.com/k12/search.asp	2	0	9	http://www.careerbuilder.com/	2	1
10	http://jobs.careerbuilder.com/	2	0	10	http://www.policechiefmagazine.org/magazine/	1	2
11	http://onlineraceresults.com/search/my_results.php	2	0				

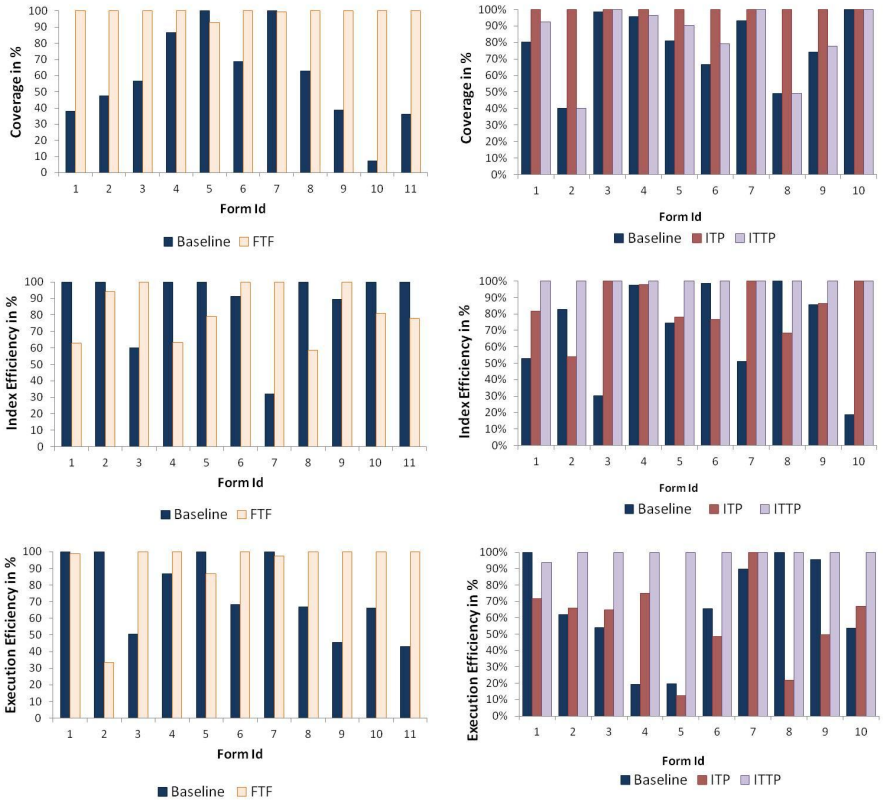

Fig. 5 Evaluation results for ITP, ITTP and FTF methods

Our evaluation is divided in two parts. In the first, we test the ITP method, and, in the second, we test the FTF method. The Web forms used in each part are different because the FTF method needs forms with text and select fields and forms with only text fields. In both cases, our baseline is an implementation based on the method proposed by Madhavan *et al.* [3] (see Section 2). Two versions of our pruning method were tested: ITP which selects informative instance templates (as described in Section 4) and ITTP which, as proposed by Madhavan et al. [3], employs an additional test on the informativeness of each template.

Figure 5 shows the coverage, and the execution efficiency normalized by the best method for each metric and form. In all forms tested, the coverage of the ITP method had the best performance because it uses all templates which means it makes further exploration of the submission possibilities. In some cases the ITTP method obtained the same results of ITP, and both were always better than the baseline. This occurs because ITP does not prune the templates (just the instances), submitting more instances compared to ITTP. On the execution efficiency aspect, ITTP was better than ITP in all forms. This happens because the ITTP method does not submit

non-informative templates, reducing the number of non-informative instance templates submitted. On the other hand, compared to the baseline, which also considers only informative templates, the ITTP method submits fewer instance templates with a greater probability of having better quality, since the baseline considers the all the instances of an informative template as informative.

For the FTF method, the number of URLs indexed was higher in all forms. This happens because the generated values are better, i.e., they retrieve more rows. Thus, the likelihood of a template being informative is higher for our method than for the baseline. Another finding is that, for forms that contain only text fields, the templates with dimension greater than one have a small probability of being informative. Our experiments showed that all 1-dimension templates are informative, increasing the chance of templates with higher dimension also being informative. The indexing efficiency presented by FTF was lower than the baseline. This happened because the baseline generated fewer URLs. For instance, form id 2 has 852 URLs for FTF and 135 URLs for the baseline. The templates from the baseline method, which contain the text fields, were considered non-informative. The FTF method reached a higher coverage compared to the baseline. The average coverage of the baseline was 47,8%, while the average coverage by FTF was 81,8%. This shows an advantage of our method. The *e4s* dataset (form id 5) showed a distinct behavior between baseline and FTF. This happened because the text field from this form is of keyword type and the remaining fields are selection lists. FTF was always better for forms that contain only text box fields.

6 Conclusion and Future Work

This paper described an approach for filling Web forms automatically. It includes two main methods for improving the search on the Hidden Web: *i.* ITP, a method to minimize the number of queries submitted to the form; and *ii.* FTF, an automatic method for selecting values for text fields. The ITP strategy evaluates each submission of an instance template. These instances are classified as informative (which retrieve new data) or non-informative (which do not return new distinct data). We can reduce the number of submissions by pruning the non-informative instances. The FTF method extracts values for text fields based on a feedback loop. The advantage is that FTF is totally automatic and can be used in any Web form that has text fields. Although the FTF method is domain-independent, the values selected for text box fields adapt to the domain. Our experiments demonstrate that our approaches are able to properly deal with text fields and query selection and have shown to be useful as a solution for filling Web forms automatically.

In future work, we will carry on with the research into two phases. In the first phase, we will use machine learning techniques to discover field values in templates with order higher than one. The second phase will entail the definition of a new method to determine values for text fields considering the number of distinct rows and the order of instance template submissions to assess the quality of the selected

values. Further investigations will also include handling fields generated dynamically and considering dependencies between values in distinct fields of a form.

Acknowledgements This research was partially supported by the National Counsel of Technological and Scientific Development, CNPq/Brazil, project number 480283/2010-9.

References

1. Jiang, L. and Wu, Z. and Zheng, Q. and Liu, J.: Learning Deep Web Crawling with Diverse Features. WI/IAT, (2009) 572–575
2. Kantorski, G. and Moraes, T. and Heuser, C.: Strategies for Automatically Filling Web Forms. Technical Report RP367-PPGC, Instituto de Informatica, UFRGS, Brazil (2012)
3. Madhavan, J. and Ko, D. and Kot, Ł. and Ganapathy, V. and Rasmussen, A. and Halevy, A.: Google's Deep Web Crawl. Proc. of the VLDB Endowment. VLDB Endowment,(2008), Vol. 1,Number 2,1241–1252.
4. Bergman, M.K.: The Deep Web: Surfacing hidden value. Journal of Electronic Publishing. (2001), Vol. 7, Number 1, 07–01.
5. He, B. and Patel, M. and Zhang, Z. and Chang, K.C.C.: Accessing the Deep Web. Communications of the ACM. ACM 50(5): 94–101. (2007).
6. Barbosa, L. and Freire, J.: Siphoning Hidden-Web Data through keyword-based interfaces. SBBD, 309–321, (2004).
7. Khare, R. and An, Y. and Song, I.Y.: Understanding deep web search interfaces: A survey. SIGMOD Rec., ACM, 39(1): 33–40. (2010).
8. Liddle, S. and Embley, D. and Scott, D. and Yau, S.: Extracting data behind web forms. Advanced Conceptual Modeling Techniques. Springer, 402–413, (2003).
9. Liu, B. and Grossman, R. and Zhai, Y.: Mining data records in Web pages. SIGKDD, ACM, 601–606, (2003).
10. Raghavan, S. and Garcia-Molina, H.: Crawling the Hidden Web. VLDB, 129–138, (2001).
11. Toda, G.A. and Cortez, E. and da Silva, A.S. and de Moura, E.: A probabilistic approach for automatically filling form-based web interfaces. Proc. of the VLDB Endowment, 4(3): 151–160. (2010).
12. Wu, P. and Wen, J.R. and Liu, H. and Ma, W.Y.: Query selection techniques for efficient crawling of structured web sources. ICDE, IEEE, 47–47. (2006)
13. Ntoulas, A. and Zerfos, P. and Cho, J.: Downloading textual hidden web content through keyword queries. JCDL, 100-109. (2005).
14. Cafarella, M.J. and Madhavan, J. and Halevy, A.: Web-scale extraction of structured data. SIGMOD Rec. ACM. 37(4): 55-61. (2009).
15. Chang, K.C.C. and He, B. and Li, C. and Patel, M. and Zhang, Z.: Structured databases on the web: Observations and implications. SIGMOD Rec. ACM. 33(3): 61–70. (2004).
16. Florescu, Daniela and Levy, Alon and Mendelzon, Alberto.: Database techniques for the World-Wide Web: a survey. SIGMOD Rec. ACM. 27(3): 59-74. September. (1998).