ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.19, No.2, February 2008, pp.246−256
DOI: 10.3724/SP.J.1001.2008.00246

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

使用分类器自动发现特定领域的深度网入口*

王 辉+, 刘艳威, 左万利

(吉林大学 计算机科学与技术学院,吉林 长春 130012)

Using Classifiers to Find Domain-Specific Online Databases Automatically

WANG Hui+, LIU Yan-Wei, ZUO Wan-Li

(College of Computer Science and Technology, Jilin University, Changchun 130012, China)

+ Corresponding author: Phn: +86-431-85166492, E-mail: whui05@yahoo.com.cn

Wang H, Liu YW, Zuo WL. Using classifiers to find domain-specific online databases automatically. *Journal of Software*, 2008,19(2):246−256. http://www.jos.org.cn/1000-9825/19/246.htm

Abstract: In hidden Web domain, general-purpose search engines (i.e., Google and Yahoo) have their shortcomings. They cover less than one-third of the data stored in document databases. Unlike the surface Web, if combined, they cover roughly the same data. Hidden Web is a highly important information source since the content provided by many hidden Web sites is often of very high quality. This paper proposes a three-step framework to automatically identify domain-specific hidden Web entries. With those obtained query interfaces, they can be integrated to obtain a unified interface which is given to users to query. Eight large-scale experiments demonstrate that the technique can find domain-specific hidden Web entries accurately and efficiently.
Key words: deep Web; hidden Web; surface Web; hidden Web entry; searchable form

摘 要: 在深度网研究领域,通用搜索引擎(比如 Google 和 Yahoo)具有许多不足之处:它们各自所能覆盖的数据量与整个深度网数据总量的比值小于 1/3;与表层网中的情况不同,几个搜索引擎相结合所能覆盖的数据量基本没有发生变化.许多深度网站点能够提供大量高质量的信息,并且,深度网正在逐渐成为一个最重要的信息资源.提出了一个三分类器的框架,用于自动识别特定领域的深度网入口.查询接口得到以后,可以将它们进行集成,然后将一个统一的接口提交给用户以方便他们查询信息.通过 8 组大规模的实验,验证了所提出的方法可以准确高效地发现特定领域的深度网入口.
关键词: 深度网;深度网;表层网;深度网入口;搜索表单
中图法分类号: TP393 **文献标识码**: A

1 Introduction

According to how its data is stored, the World Wide Web can be classified into two categories that are surface Web and deep Web (also called hidden Web). In the surface Web, data are stored in document files; while in the

* Supported by the National Natural Science Foundation of China under Grant No.60373099 (国家自然科学基金); the Science and Technology Development Program of Jilin Province of China under Grant No.20070533 (吉林省科技发展计划)

Received 2007-08-02; Accepted 2007-11-06

deep Web, data are stored in databases[1]. Unlike the surface Web, the deep Web refers to the collection of Web data that is accessible by interacting with a Web-based query interface, and not through the traversal of static hyperlinks. A July 2000 white paper[2] estimated that deep Web has 450 000 databases, 7 500 terabytes of information and 550 billion individual documents. In contrast, the surface Web contains 19 terabytes of information and 1 billion individual documents. In addition, according to many studies, the size of the hidden Web increases rapidly as more organizations release their valuable content online through an easily used Web interface[3]. The content provided by many hidden Web sites is often of very high quality and can be extremely valuable to many users. The site of the US Patent and Trademarks Office is an example, which makes existing patent documents available in order to help potential inventors examine the arts which had been invented already.

To retrieve data from online databases, three main problems should be considered. They are interface unification (also called interface integration), query translation and result merging. Before hidden Web database is queried, search system first characterizes the available search interfaces and then, given a query, it selects a subset of useful domain-specific search interfaces, queries them and presents results to the users.

In this paper, we consider an overlooked problem that precedes the three main problems. It is discovering deep Web entries. A search system must discover a set of search interfaces or be provided with such a set before it can proceed with the other three steps.

Much work has been done in each of these three areas. For each domain, the MetaQuerier[4] constructs a unified interface which is provided for users to query. Users send their queries via the unified interface. A mediator translates the queries to each specific online database and then returns the integrated query results to the users. Chang, *et al*.[5] use the parsing approach achieving above 85% accuracy for extracting query conditions across randomly selected deep Web sources and for query interface matching. Wu, *et al*.[6] develop a novel approximation algorithm LMax, which builds the unified interface via recursive applications of clustering aggregation. Moreover, they extend LMax to handle the irregularities that frequently occur in the interface schemas.

The interface Extractor[7] can achieve a deeper understanding of Web search interfaces in the sense that more semantic/meta information on search interfaces can be extracted. With such semantic/meta information, the enriched interface schema can be used in many applications, for instance, query translation, search result extraction and annotation. Chang, *et al*.[8] pursue a source-based and rule-driven framework to implement query translation across different deep Web sources. On the contrary, He, *et al*.[9] propose a generic type-based and search-driven query translation framework to reach the same goal.

A deep Web wrapper is a program that extracts contents from search results. Nakatoh, *et al*.[10] propose a new automatic generation algorithm which discovers a repetitive pattern from search results. Hedley, *et al*.[11] describe a Two-Phase Sampling (2PS) technique to detect templates and extract query-related information from the sampled documents of a database. Mundluru, *et al*.[12] give a highly effective and efficient solution for automatically mining result records from search engine response pages. Experimental results showed that their proposed system significantly outperforms MDR[13], a state-of-the-art record mining system.

Though much work has been done in those areas, little work has been done in interface discovery, especially since the three main problems depend on having a set of known hidden Web interfaces. The remaining paper is organized as follows. We review related work in Section 2. Section 3 concerns with page and form classifiers. Three-step framework of our hidden Web crawler is described in Section 4. Experimental results are elaborated in Section 5. Section 6 is concerning with conclusion and future work.

2 Related Work

In recent years, the hidden Web is becoming a hot research spot. It is estimated that there are several million hidden Web sites, which contain a large amount of information that is of high quality[14]. The difficulties in automatically filling out structured Web forms have been documented in the literature[15]. The work in Ref.[16] describes a semi-automatic crawler called HiWE, which is aided by domain knowledge to generate reasonable queries for hidden Web interfaces. Cope, *et al*.[17] use an automatic feature generation technique to depict candidate forms and a C4.5 decision tree to classify them. In their two testbeds—ANU collection and random Web collection, they get an accuracy of more than 85% and a precision of more than 87% respectively.

Bergholz, *et al*.[18] describe a crawler which starts from the Publicly Indexable Web (PIW) to find entry points into the hidden Web. This crawler is domain-specific and is initialized with pre-classified documents and relevant keywords. Luciano and Juliana[19] compose two classifiers in a hierarchical fashion to identify online databases among a heterogeneous set of Web forms automatically gathered by a focused crawler. In Ref.[20], they present a new adaptive focused crawling strategy for efficiently locating hidden Web entry points. Unfortunately, the *ACHE* framework they proposed can not handle very sparse domains efficiently. Besides, the *ACHE* framework is complex and its overhead is large. Our technique is different from theirs. Firstly, our modified best-first crawler just finds domain-specific hidden Web entries. Secondly, we use a three-step framework to guide our deep Web crawler in this paper.

3 Page and Form Classifiers

In order to find domain-specific hidden Web entries, we use three classifiers which work in a hierarchical fashion to guide our deep Web crawler. The three classifiers include form structure classifier, form text classifier and page text classifier.

3.1 Form structure classifier

A form is made up of two parts that are structural and textual parts. Consider the famous Perl CPAN Web page as an example, where we can find distributions, modules, documents and ID's. The HTML source code of its form contained in this Web page is listed below:

```
⟨form method="get" action="/search" name="f" class="searchbox"⟩
  ⟨input type="text" name="query" value="" size="35"⟩
  ⟨br⟩ in ⟨select name="mode"⟩
    ⟨option value="all"⟩ All ⟨/option⟩
    ⟨option value="module"⟩ Modules ⟨/option⟩
    ⟨option value="dist"⟩ Distributions ⟨/option⟩
    ⟨option value="author"⟩ Authors ⟨/option⟩
  ⟨/select⟩ & nbsp; ⟨input type="submit" value="CPAN Search"⟩
⟨/form⟩
```

When displayed in IE browser, the result is shown in Fig.1.

Fig.1 An illustrated form interface displayed in IE browser

From Fig.1, we can see that a form contains not only textual contents such as ‘in’, ‘CPAN Search’, but also structural contents such as select elements, submission buttons. In order to identify whether a form is a domain-specific searchable form or not, in this paper we use form structural and textual features to train form structure and form text classifiers respectively. Once obtaining a form structure classifier, we can get rid of these non-searchable forms, such as forms for login, discussion group interfaces, mailing list subscriptions, purchase forms and Web-based email forms. Luciano, *et al*.[21] and Cope, *et al*.[17] demonstrate that optimum result will be obtained by using a decision tree to classify searchable and non-searchable forms. Accordingly, we also use decision tree algorithm to train a form structure classifier.

3.2 Form text classifier

With the aid of decision tree classifier, we can identify whether a form is a searchable form or not. To further ascertain if a searchable form is a domain-specific one, we must make full use of form textual features. According to previous researches[19,20,22], libsvm learning algorithm[22] should be used in this case. To extract textual features from forms, two text extracting methods are tried in this paper. One is called FT (full text) technique and the other is named PT (partial text) method. The FT method simply uses all HTML codes of forms and splits them using non-alphanumeric strings. In contrast with FT method, the PT technique extracts those textual features, which can be seen by our human beings (when displayed in a browser) as well as the form action attribute to which all form data are sent. For example:

〈form action=“http://www.hotwire.com/car/search-options.jsp”
method=“get” name=“searchCar”〉
This is a form which is used for demo.
〈/form〉

In the above form, the action attribute value is:

http://www.hotwire.com/car/search-options.jsp

which is also extracted by the PT method.

In order to use these extracted textual features, some pre-processing steps are needed. First, all characters other than alphanumeric ones are replaced by a space character; second, uppercase characters, if any, are converted to their lower case equivalents; third, stop words, if any, are removed, using CPAN[23] Perl package Lingua::EN::StopWords; fourth, each word in the remaining texts is stemmed, using CPAN Perl package Lingua::Stem::En; finally, TFIDF[24] is used to transform each training example into its corresponding vector. The same procedure is also applied to page text classifier (see Section 3.3). Using those extracted textual features, we can train a SVM classifier which can be used to identify whether a searchable form is domain-specific or not.

3.3 Page text classifier

To decide automatically whether a Web page is relevant or not, we use a page text classifier. Given a Web page, we first obtain its corresponding plain texts. After that, some pre-processing steps (see Section 3.2) are needed in order to use these texts to train a SVM classifier.

With these three classifiers on hand, we can apply them to guiding a focused crawler to find deep Web entries:

- First, using the page text classifier to decide whether the Web pages corresponding to the given URLs are relevant or not;
- Second, if a Web page is relevant, we extract searchable forms from it with the aid of the form structure classifier;
- Third, if the relevant Web page contains searchable forms, we further use the form text classifier to

ascertain whether they are domain-specific or not.

The reason why we use classifiers in this hierarchical fashion is that the hierarchical composition of classifiers leads to modularity. In this case, a complex problem is decomposed into simpler sub-components and each is devoted to a subset of the hypothesis. This has several merits:

- First, the overall classification process is not only accurate but also robust;
- Second, we can apply to each part a learning method that is best suited for the feature set of the partition.

4 Three-Step Framework

Figure 2 shows the high-level architecture proposed in this paper.

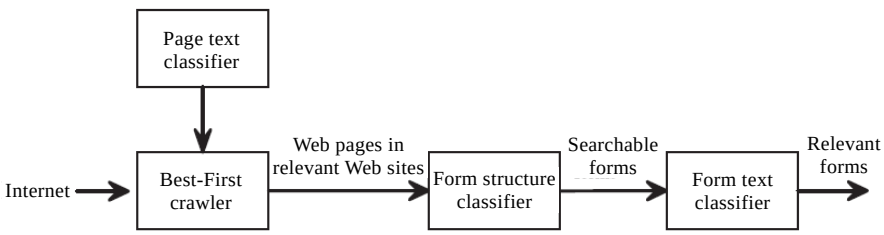

Fig.2 The high-level architecture proposed in this paper

Note that the best-first crawler used in this paper is a variation of the best-first crawler proposed in Ref.[22]. In Ref.[22], they make no difference about URLs which lie in a on-topic page; whereas we give URLs priorities according to the following formula

$$a \times page\_score + b \times anchor\_score.$$

Here, we let a and b take the same value one. The detailed procedure about our deep Web crawler is displayed in Fig.3.

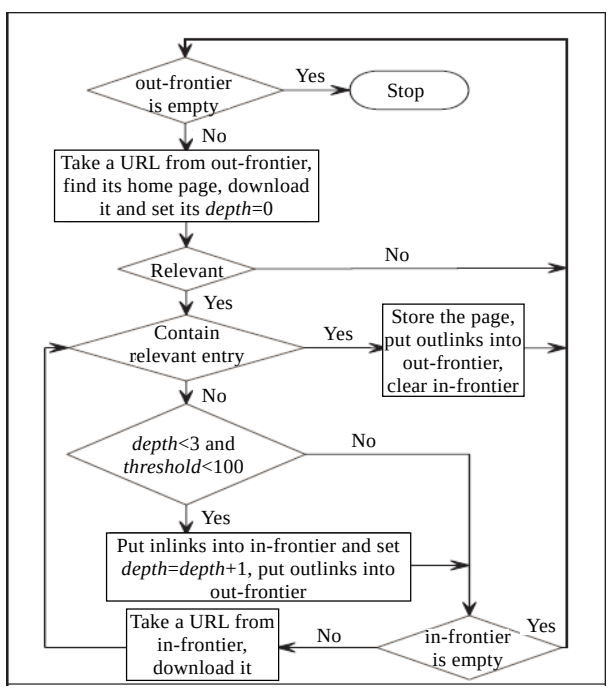

Fig.3 The detailed procedure of our three-step framework

Here we make an assumption: unless the home page of a Web site is relevant to a database domain, the Web site can not contain domain-specific deep Web entries. The reason why we make such an assumption is that we want our crawler to crawl in a promising search space and consequently improve its efficiency. In the following part, we will simply explain our three-step framework:

- First, given a URL, we find out its corresponding home page and decide whether this home page is domain-specific. Our crawler only finds deep Web entries in those sites which contain domain-specific home pages;
- Second, in each domain-specific Web site, our crawler crawls within it until *depth*>=3 or the total number of pages *threshold*>=100 is visited. Note that the reason why we set *depth*<3 is that Web databases tend to locate shallowly in their sites and the vast majority of them (approximately 94%) can be found at the top 3 levels[16]. Besides, in order to protect our crawler from getting trapped in some sites, we set a threshold for visiting maximum Web pages per site.

5 Experimental Results

TEL-8 Query Interfaces[25] dataset is used to train a form classifier. This dataset contains the original interfaces extracted from eight representative domains, which are Airfares, Automobiles, Books, CarRentals, Hotels, Jobs, Movies and MusicRecords. Table 1 shows the instances' distributions of the eight representative database domains.

Table 1 The instances' distributions of the eight database domains. 223 sources in all

Domain	Sources	Domain	Sources
Airfare	20	Auto	28
Book	43	Rental	13
Hotel	34	Job	20
Movie	32	Music	33

5.1 Training form structure classifier

In this paper, our form structure classifier is trained by using decision tree algorithm. The decision tree training data are collected as follows: we extract 223 searchable forms (see Table 1) from TEL-8 Query Interfaces as positive examples and manually gather 318 non-searchable forms as negative ones.

For each form in the sample dataset, we count the following features: number of checkboxes; number of file inputs; number of hidden tags; number of image inputs; number of submission methods (get and post); number of select elements; number of password tags; number of radio tags; number of word 'search' within form tag or submission button; number of text elements; number of textarea elements and number of word 'email' in input elements' name or value. The distributions about all those features in searchable and non-searchable forms are displayed in Table 2. From Table 2, we can draw the following conclusions:

- Searchable forms have a large number of checkboxes and items (options) in selection lists.
- No-Searchable forms have a large number of password tags and 'email' in input elements' name or value.

Using these structural features, we can train a decision tree classifier.

Two tools are used to construct a form structure classifier: Weka j48 algorithm[26] and Algorithm::SVMLight Perl package[23]. In Weka, the precision of the decision tree classifier is 0.948 718. The decision tree generated by Weka is displayed in Fig.4. In fact, we use Perl package Algorithm::SVMLight in this paper to train a decision tree classifier and its precision is 0.948 717 948. Obviously, these two tools have the similar results according to our experiments. Nevertheless, in Algorithm::SVMLight current implementation, only discrete-valued attributes are supported and consequently it outputs a large number of rules. In fact, it outputs 122 rules in all. Additionally, we

think that the decision tree misclassifies an instance if it can not decide, to which category this instance belongs.

Table 2 Features' distributions of searchable and non-searchable forms

Feature/Category	Searchable	Non-Searchable	Ratio
checkbox	2.39	0.18	**13.04:1**
email_yes	0.01	0.12	**1:13.13**
file	0.00	0.00	–
hidden	4.45	1.63	2.72:1
image	0.36	0.21	1.73:1
method_get	0.47	0.40	1.16:1
option	12.64	0.17	**74.23:1**
password	0.02	0.10	**1:5.86**
radio	0.48	0.10	4.76:1
search_yes	0.36	0.09	3.94:1
text	3.00	1.01	2.97:1
textarea	0.02	0.07	1:2.98

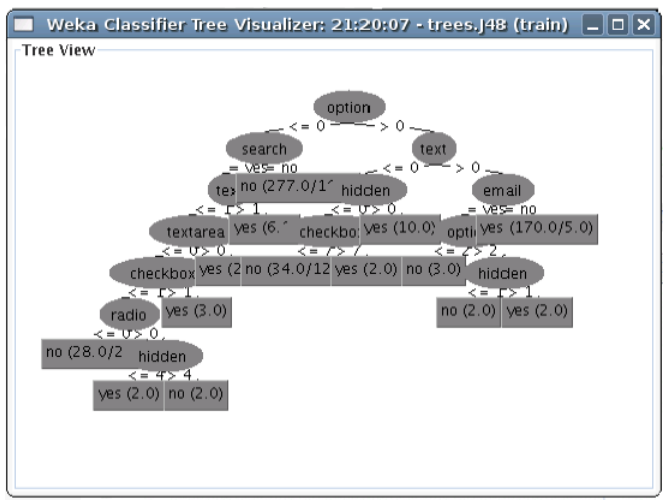

Fig.4 The decision tree generated by Weka j48 algorithm using form structural features

5.2 Training form text classifier

According to FT and PT methods (see Section 3.2), we can extract form textual features from forms. Five most frequent features obtained by FT and PT techniques are presented in Table 3. Table 3 shows that the PT method extracts more valuable features than the FT technique does.

Table 3 Five most frequent features extracted by FT and PT methods respectively

Method	Category	Textual features (Feature: Frequency)
FT	Airfare	option: 8113 value: 4161 *td*: 1181 id: 1069 class: 993
	Auto	option: 5673 value: 3002 *td*: 1520 *tr*: 716 class: 498
	Book	option: 10997 value: 5753 *td*: 1538 *tr*: 788 name: 421
	Rental	option: 6396 value: 3199 *td*: 892 pm: 550 class: 520
	Hotel	option: 13048 value: 6271 class: 1377 div: 1358 *td*: 1265
	Job	option: 7423 value: 3868 *u*: 775 td:680 *tr*: 413
	Movie	option: 6995 value: 3616 div: 1200 class: 1181 *td*: 734
	Music	option: 8090 value: 679 *td*:516 record: 456 font: 323
PT	Airfare	pm: 419 airlin: 279 air: 124 am: 102 airwai: 100
	Auto	docum: 108 car: 105 leas: 84 search: 63 make: 56
	Book	search: 130 titl: 110 book: 95 author: 75 new: 72
	Rental	pm: 402 option: 202 am: 168 airport: 144 car: 143
	Hotel	hotel: 234 pm: 228 island: 151 new: 135 room: 84
	Job	job: 207 new: 125 locat: 84 servic: 82 island: 81
	Movie	press: 211 book: 123 s: 109 video: 107 entertain: 107
	Music	record: 456 music: 226 sub: 156 search: 97 new: 80

Using those extracted textual features, we finally finish training eight SVM classifiers. The precisions of the eight SVM classifiers are shown in Table 4. Since PT method extracts more valuable textual features than FT technique does, we can use them train a more accurate classifier. Table 4 indicates that no matter what category is, using the PT method can always obtain a more accurate classifier than the FT technique can.

Table 4 Precisions of SVM classifiers trained with form textual features that are extracted by the FT and PT methods respectively

Category	FT method	PT method
Airfare	0.918 2	0.936 4
Auto	0.947 6	1.0
Book	0.927 3	0.963 6
Rental	0.954 5	0.963 6
Hotel	0.933 3	0.938 1
Job	0.940 9	0.968 2
Movie	0.927 3	0.977 3
Music	0.9	0.968 2

5.3 Training page text classifier

In order to train page text classifier, this paper gets its positive training data from the online open directory project (http://dmoz.org/). We use a Perl script program to fill out the searchable form and extract URLs from its returned result pages automatically. As for negative training data, we get them from DMOZ (http://rdf.dmoz.org/). DMOZ consists of sixteen categories, which are Arts, Business, Computers, Games, Health, Home, Kids_and_Teens, News, Recreation, Reference, Regional, Science, Shopping, Society, Sports and World. We get rid of four categories of them in our experiments. They are Kids_and_Teens, Reference, Regional and World. The number of URLs in each category is shown in Table 5.

Table 5 The number of URLs in each DMOZ category

Arts	Business	Computers	Games	Health	Home
585 924	511 620	285 336	123 758	131 050	33 555
News	Recreation	Science	Shopping	Society	Sports
235 704	120 308	213 014	235 160	269 864	154 921

In DMOZ, each example looks like this:

⟨ExternalPage about=
“http://www.airwise.com/airports/us/SLC/index.html”⟩
⟨*d*:Title⟩
Salt Lake City Airport - airwise.com
⟨/*d*:Title⟩
⟨*d*:Description⟩
Information about the airport including airlines,
ground transportation, parking, weather and airport
news.
⟨/*d*:Description⟩
⟨topic⟩
Top/Regional/North_America/United_States/Utah/
Localities/S/Salt_Lake_City/Transportation/Airports
⟨/topic⟩
⟨/ExternalPage⟩

This paper uses the content of ‘*d*:Description’ element and the Web page corresponding to the ‘about’

ExternalPage attribute to obtain a negative training example.

In order to be more representative, we derive URLs from each category according to its size. Since the 'Arts' category has the largest number of URLs, we get the most number of URLs from it. Excluded these URLs which can not be downloaded, the number of positive and negative examples which we use to train a page classifier for each category is listed in Table 6. Using these training data, we can finish training page classifiers. The precisions of these page classifiers are shown in Table 6.

Table 6 The number of positive and negative train data for each category as well as the precision of its corresponding page classifier

Category	Positive	Negative	Precision
Airfare	116	316	0.961 9
Auto	2 512	3 564	0.946 3
Book	1 568	3 325	0.913 5
Rental	913	2 282	0.973 7
Hotel	1 708	2 721	0.994 1
Job	1 706	3 178	0.979 1
Movie	1 601	3 124	0.900 4
Music	22	87	0.87

5.4 Using the three-step framework to find deep Web entries

We conduct eight large-scale experiments with our hidden Web crawler. For each category, we initialize our crawler with 100 seeds that are extracted from the DMOZ as the starting point. We save those pages and their corresponding URLs if the following two conditions are satisfied at the same time. First, they are judged to be relevant by page and form text classifiers. Second, each page contains at least one domain-specific deep Web entry. Since MusicRecords databases are very sparsely distributed, our best-first focused crawler only locates 50 deep Web entries for it. For other categories, our crawler finds 100 deep Web entries for each of them. Five deep Web entries about Airfares category which are located by our crawler are listed below:

http://www.aircanada.ca/

http://www.itn.net/

http://www.aircharter.com/

http://www.orbitz.com/

http://www.nwa.com/

At last, we manually verify whether the deep Web entries located by our crawler are what we want. The precisions of all these categories are shown in Table 7.

Table 7 The precisions of deep Web entries for each category

Domain	Precision	Domain	Precision
Airfare	0.90	Auto	0.88
Book	0.91	Rental	0.95
Hotel	0.94	Job	0.81
Movie	0.86	Music	0.80

6 Conclusion and Future Work

In this paper, a three-step framework is proposed to automatically identify domain-specific hidden Web entries. To verify its effectiveness and efficiency, eight large-scale experiments are conducted. Experimental results demonstrate that our method can find domain-specific deep Web entries accurately and efficiently. The average precision of the eight representative domains is 0.88.

In the near future, experiments on a larger number of categories are required in order to further assess the

effectiveness of our proposed technique. Additionally, with those obtained query interfaces, we will integrate them to obtain a unified interface and give it to users to query. Users don't have to hunt some domain-specific sources and learn the details for querying each resource.

Acknowledgement This work is sponsored by the Science and Technology Development Program of Jilin Province under Grant No.20070533 and the Natural Science Foundation of China under Grant No.60373099.

References:

[1] Rocco D, Caverlee J, Liu L, Critchlow T. Exploiting the deep Web with DynaBot: Matching, probing, and ranking. In: Ellis A, Hagino T, eds. Proc. of the World Wide Web Special Interest Tracks And Posters (WWW). Chiba: ACM, 2005. 1174−1175.

[2] BrightPlanet.com. The deep Web: Surfacing hidden value. http://brightplanet.com

[3] Bergman MK. The deep Web: Surfacing hidden value. Journal of Electronic Publishing, 2001,7(1):1174−1175. http://www.press.umich.edu/jep/07-01/bergman.html

[4] He B, Zhang Z, Chang KCC. Knocking the door to the deep Web: Integrating Web query interfaces. In: Weikum G, ed. Proc. of the SIGMOD Conf. Paris: ACM, 2004. 913−914.

[5] Chang KCC, He B, Zhang Z. MetaQuerier over the deep Web: Shallow integration across holistic sources. In: Nascimento MA, Özsu MT, Kossmann D, Miller RJ, Blakeley JA, Schiefer KB, eds. Proc. of the Int'l Conf. on Very Large Data Bases (VLDB). Morgan Kaufmann Publishers, 2004. 15−21.

[6] Wu W, Doan A, Yu CT. Merging interface schemas on the deep Web via clustering aggregation. In: Proc. of the Int'l Conf. on Data Mining (ICDM). IEEE Computer Society, 2005. 801−804.

[7] He H, Meng WY, Yu CT, Wu ZH. WISE-Integrator: A system for extracting and integrating complex Web search interfaces of the deep Web. In: Böhm K, Jensen CS, Haas LM, Kersten ML, Larson PA, Ooi BC, eds. Proc. of the Int'l Conf. on Very Large Data Bases (VLDB). ACM, 2005. 1314−1317.

[8] Chang KCC, Garcia-Molina H. Mind your vocabulary: Query mapping across heterogeneous information sources. In: Delis A, Faloutsos C, Ghandeharizadeh S, eds. Proc. of the SIGMOD Conf. Philadelphia: ACM Press, 1999. 335−346.

[9] He B, Zhang Z, Chang KCC. MetaQuerier: Querying structured Web sources on-the-fly. In: Özcan F, ed. Proc. of the SIGMOD Conf. ACM, 2005. 927−929.

[10] Nakatoh T, Yamada Y, Hirokawa S. Automatic generation of deep Web wrappers based on discovery of repetition. In: Proc. of the Asia Information Retrieval Symp. (AIRS). Beijing: Springer-Verlag, 2004. 269−272.

[11] Hedley YL, Younas M, James A, Sanderson M. A two-phase sampling technique for information extraction from hidden Web databases. In: Laender AHF, Lee D, Ronthaler M, eds. Proc. of the Int'l Workshop on Web Information and Data Management (WIDM). Washington: ACM, 2004. 1−8.

[12] Mundluru D, Katukuri JR, Celebi S. Automatically mining result records from search engine response pages. In: Proc. of the Int'l Conf. on Data Mining (ICDM). IEEE Computer Society, 2005. 749−752.

[13] Liu B, Grossman R, Zhai YH. Mining data records in Web pages. In: Getoor L, Senator TE, Domingos P, Faloutsos C, eds. Proc. of the Knowledge Discovery and Data Mining (KDD). Washington: ACM, 2003. 601−606.

[14] Hsieh W, Madhavan J, Pike R. Data management projects at Google. In: Chaudhuri S, Hristidis V, Polyzotis N, eds. Proc. of the SIGMOD Conf. Chicago: ACM, 2006. 725−726.

[15] Wu P, Wen JR, Liu H, Ma WY. Query selection techniques for efficient crawling of structured Web sources. In: Liu L, Reuter A, Whang KY, Zhang J, eds. Proc. of the Int'l Conf. on Data Mining (ICDE). IEEE Computer Society, 2006. 47.

[16] Raghavan S, Garcia-Molina H. Crawling the hidden Web. In: Apers PMG, Atzeni P, Ceri S, Paraboschi S, Ramamohanarao K, Snodgrass RT, eds. Proc. of the Int'l Conf. on Very Large Data Bases (VLDB). Rome: Morgan Kaufmann Publishers, 2001. 129−138.

[17] Cope J, Craswell N, Hawking D. Automated discovery of search interfaces on the Web. In: Schewe KD, Zhou X, eds. Proc. of the Australasian Database Conf. (ADC). Australian Computer Society, 2003. 181−189.

[18] Bergholz A, Chidlovskii B. Crawling for domain-specific hidden Web resources. In: Proc. of the Int'l Conf. on Web Information Systems Engineering (WISE). Roma: IEEE Computer Society, 2003. 125−133.

[19] Barbosa L, Freire J. Combining classifiers to identify online databases. In: Williamson CL, Zurko ME, Patel-Schneider PF, Shenoy PJ, eds. Proc. of the World Wide Web Conf. (WWW). ACM, 2007. 431−440.

[20] Barbosa L, Freire J. An adaptive crawler for locating hidden-Web entry points. In: Williamson CL, Zurko ME, Patel-Schneider PF, Shenoy PJ, eds. Proc. of the World Wide Web Conf. (WWW). ACM, 2007. 441−450.

[21] Barbosa L, Freire J. Searching for hidden-Web databases. In: Doan AH, Neven F, McCann R, Bex GJ, eds. Proc. of the 8th Int'l Workshop on the Web and Databases (WebDB). Baltimore: ACM Press, 2005. 1−6.

[22] Chang CC, Lin CJ. Libsvm—A library for support vector machines. http://www.csie.ntu.edu.tw/~cjlin/libsvm/

[23] CPAN. http://search.cpan.org/

[24] Torgo L, Gama J. Regression by classification. In: Borges D, Kaestner C, eds. Proc. of the Brasilian Artificial Intelligence Symp. Curitiba: Springer-Verlag, 1996. 51−60.

[25] The uiuc Web integration repository. http://metaquerier.cs.uiuc.edu/repository/

[26] Weka. http://www.cs.waikato.ac.nz/ml/weka/

WANG Hui was born in 1972. He received his Ph.D. degree from Jilin University. His research area is Web information mining.

LIU Yan-Wei was born in 1983. He is a graduate student at the Jilin University. His research area is Web information mining.

ZOU Wan-Li was born in 1957. He is a professor and doctoral supervisor at the Jilin University and a CCF senior member. His research areas are database, data mining and Web search engine.