Query Interfaces Understanding by Statistical Parsing

Weifeng Su
BNU-HKBU United International College
Zhuhai, Guangdong Prov., China
wfsu@uic.edu.hk

Yafei Li
BNU-HKBU United International College
Zhuhai, Guangdong Prov., China
yafeili@uic.edu.hk

Frederick H. Lochovsky
The Hong Kong University of Science and Technology
Hong Kong, China
fred@cse.ust.hk

ABSTRACT
Users submit queries to an online database via its query interface. Query interface parsing, which is important for many applications, understands the query capabilities of a query interface. Since most query interfaces are organized hierarchically, we present a novel query interface parsing method, StatParser (Statistical Parser), to automatically extract the hierarchical query capabilities of query interfaces. StatParser automatically learns from a set of parsed query interfaces and parses new query interfaces. StatParser starts from a small grammar and enhances the grammar with a set of probabilities learned from parsed query interfaces under the maximum-entropy principle. Given a new query interface, the probability-enhanced-grammar identifies the parse tree with the largest global probability to be the query capabilities of the query interface. Experimental results show that StatParser very accurately extracts the query capabilities and can effectively overcome the problems of existing query interface parsers.

Categories and Subject Descriptors
H.3.5 [Online Information Services]: Web-based services — Query Interface, H.5.2 [User Interfaces]: Natural language

Keywords
Query Interface, Maximum Entropy.

1. INTRODUCTION
A typical Web database comprises a back-end database and a query interface. The user obtains the data from a Web database by submitting a query via its query interface. Upon receiving the user query, the relevant data is retrieved from the back-end database and returned to the user embedded in HTML pages. Hence, the query interface serves as an intermediary between the user and the Web database. To submit a query, the user first has to understand the query capabilities of the query interface including (1) the semantics of each element in the form that comprises the query interface, (2) the metadata of each element (such as the data type) and (3) the element organization. Thereafter, the user submits his/her query by filling values into corresponding elements in the query interface.

Many applications require the understanding of query interfaces. In particular, many applications require a hierarchical representation of the query interface. Considering that there are many Web databases available nowadays, and that each domain has a large number of Web databases, understanding the query interface manually is an almost impossible task in many applications that interact with many Web databases. Therefore, automatic query interface understanding is an indispensible component in these applications.

Copyright is held by the International World Wide Web Conference Committee (IW3C2). IW3C2 reserves the right to provide a hyperlink to the author's site if the Material is used in electronic media.
WWW'14 Companion, April 7–11, 2014, Seoul, Korea.
ACM 978-1-4503-2745-9/14/04.
http://dx.doi.org/10.1145/2567948.2579702

In this paper, we propose a novel query interface understanding method, Statistical Parser (StatParser), which effectively combines the strengths of rule-based methods and learning-based methods, to automatically parse the query interface into a hierarchical representation. This paper is a short version of [3].

2. STATPARSER
2.1 Overview
A query interface includes a form, which is composed of a set of elements. The elements include text edit boxes, selection lists, radio buttons and check boxes.

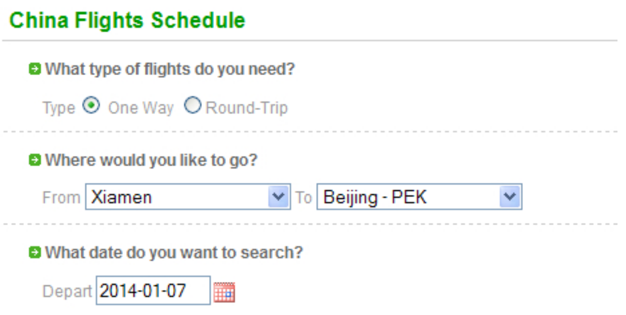
China Flights Schedule
What type of flights do you need?
Type
One Way
Round-Trip
Where would you like to go?
From
Xiamen
To
Beijing - PEK
What date do you want to search?
Depart
2014-01-07

Fig. 1(a) Example of a query interface from the Airline domain.

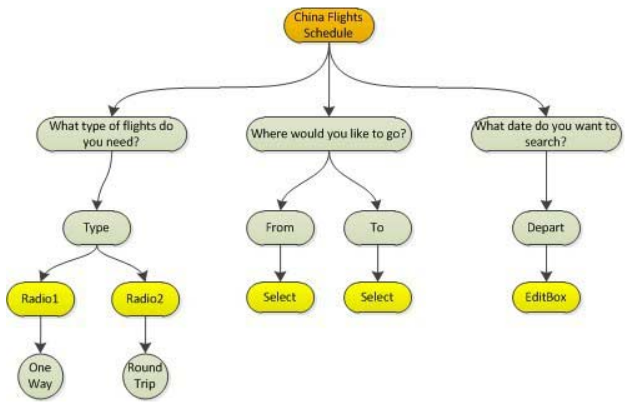
China Flights Schedule
What type of flights do you need?
Where would you like to go?
What date do you want to search?
Type
From
To
Depart
Radio1
Radio2
Select
Select
EditBox
One Way
Round Trip

Fig. 1(b) Semantic tree for Figure 1(a).

Definition 1 (Query Interface Understanding). Let $T=\{e_1, e_2, \ldots, e_m\}$ be a form in a query interface in which e_i is an element in the form and $L=\{l_1, l_2, \ldots, l_n\}$ is a set of labels embedded in T. Query interface understanding determines the semantic organization between elements and labels using a tree representation, i.e., a semantic tree. Each leaf node in the tree represents an element or a label and each internal node shows the description relationship between a label and an element, or between a label and a semantic unit.

Fig 1(a) shows an example of a query interface and Fig 1(b) shows its corresponding semantic tree.

StatParser consists of two stages: the training stage and the parsing stage. In the training stage, the important features are identified and their corresponding parameters are estimated with a set of parsed query interfaces. In the parsing stage, the features and parameters are used to understand a new query interface. Both stages have two steps: the preprocessing step and the understanding step.

The preprocessing step tries to process the radio buttons and checkboxes by combining each group of radio buttons or checkboxes that share the same name value and their corresponding labels into a single element. The training stage has two components in the preprocessing step: the *Feature Selection* component and the *Parameter Estimation* component. Given a set of parsed query interfaces, the *Feature Selection* component selects a set E_p of features that are supposed to be important for radio-button/check-box processing. Thereafter, the *Parameter Estimation* component learns the probability parameters for each of the selected features E_p. We handle radio buttons and check boxes first for two reasons. First, the pattern to combine radio buttons/check boxes and labels is different from other elements and labels. Second, the radio buttons/check boxes that have the same name value can be safely merged to simplify the query interface to facilitate further processing.

2.2 Grammars

Although query interfaces are designed autonomously, they are usually designed so that users can very easily understand them. Hence, there are some inherent patterns that almost all designers follow when designing query interfaces. These patterns may not be explicit to the designers, but they are used implicitly to construct query interfaces concisely and to make them easily understood by the users. We validated the nine patterns listed in [1] against more than 300 query interfaces from 10 domains both in Chinese and English and found that the first four patterns listed below are true in all query interfaces. In addition, we identified two new patterns (Patterns 5 and 6). The histogram in Fig. 2 shows how confident each pattern is in our survey.

Pattern 1: Query interfaces are presented in a top-down and left-to-right order.

Pattern 2: Elements and labels in a query interface are organized into semantic groups.

Pattern 3: Each label describes either an element or a semantic unit, but not both.

Pattern 4: The label for an element or a semantic unit is located either above, left, right or below the element or the semantic unit.

Pattern 5: Each element/semantic-unit is described by no more than one label in the semantic tree.

Pattern 6: Each radio-button/checkbox is combined with no more than one label.

Based on the patterns listed above, the grammars for preprocessing and parsing are presented in Section 2.4.

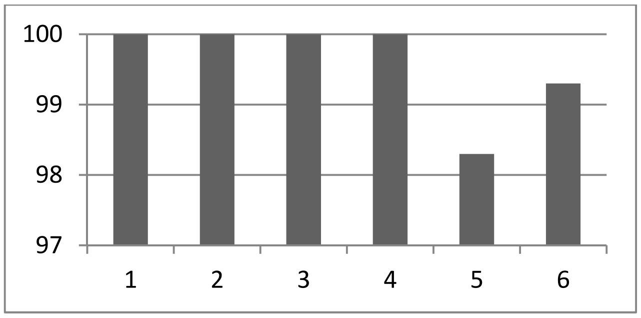

Fig. 2 Histogram of pattern confidence.

2.3 Context Information Extraction

Given a form for a query interface, StatParser extracts a set of context information for each element and label. According to our experiments, the context information is vital to the success of StatParser. Considering that forms are usually designed to be quickly understood by users, the relevant labels for an element are usually located around the element. To facilitate presentation, we use the term item to represent either an element or a label.

From observing several hundred query interfaces from various domains and various countries, we found that an item usually is most correlated with the items immediately around it (Pattern 4). Hence, we adopt the "field scope" concept in [1], which states that a label for an element must lie in one of the four positions: above the element, below the element, left to the element and right to the element. As shown in Fig. 3, for each item e_i, we consider the items in the following four positions around e_i as the features for e_i.

(1) The item just above e_i. If there are multiple items just above e_i, only the one that is left aligned to e_i is selected.

(2) The item just below e_i. If there are multiple items just below e_i, only the one that is left aligned to e_i is selected.

(3) The item just to the left of e_i. If there are multiple items to the left of e_i, only the one closest to e_i is selected.

(4) The item just to the right of e_i. If there are multiple items to the right of e_i, only the one closest to e_i is selected.

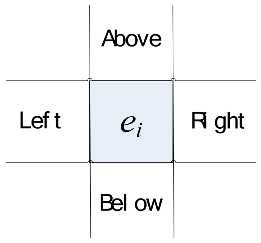

Fig. 3. Context information for an item e_i.

Each item generates two pieces of context information: the item type information and the distance information. The item type information represents the type of the item, whose value can be an element of {*label*, *editbox*, *selectionlist*, *radiobuttons*, *checkboxes*, *null*}. The *radiobuttons* or *checkboxes* represent a group of radio buttons or a group of checkboxes that has been constructed in the preprocessing step. The item type value is *null* if there is no item in the corresponding position.

The distance information of an item represents the distance between the item and e_i. Its value can be an element of {*far*, *near*, *null*}. In the experiments, the distance is *far* if there is at least one empty table cell or a line between the item and e_i and *near* otherwise. If the item type is *null*, its distance value is also *null*.

Consider the first three items in Fig. 1(a), i.e., (1) the label "What type of flight do you need?", (2) the label "type" and (3) the type radiobuttons. Table 1 shows the context information of these three items. In the table, the above item type of item_(1) is null because there is no item above it. The below item type of item_(1) is item_(2), which is a label and is near item_(1). Both the left items and right items of item_(1) are null because there are no items to the left or right of item_(1).

Table 1. Context information for elements in Figure 1(a).

Item id	Above item		Below item		Left of item		Right of item	
	type	dist.	type	dist.	type	dist.	type	dist.
(1)	null	null	label	near	null	null	null	null
(2)	label	near	label	far	null	null	Radio buttons	near
(3)	label	near	label	far	label	near	null	null

2.4 Maximum-Entropy-Based Parsing

MaxEnt is a modelling technique for approximating an observed distribution. We represent a single observation with *y*, whose value comes from *Y,* and conditional information with *x*, whose value comes from *X*. We assume that the value of *y* is influenced by the value of *x*. Therefore, the conditional probability $p(y|x)$ is estimated.

```
1. RC →LabelRadio* | LabelCheck*
2. LabelRadio → label radiobutton |
   radiobutton label
3. LabelCheck → label checkbox | checkbox
   label
```

Fig. 4. Production rules for preprocessing.

```
1. Root →Conditions Condition | Condition
2. Conditions → Conditions Condition |
   Condition
3. Condition → label Condition | Condition
   label| Condition Condition | Element
4. Element → editbox | selectionlist| RC
```

Fig. 5. Production rules for parsing.

In StatParser, the conditional information *x* refers to context information that is described in Section 2.3 and *y* refers to any production rule in Fig. 4 or Fig. 5, depending on the processing step. For example, if we observe that in a query interface and its corresponding parse tree a condition is combined with a label on its left, which corresponds to the production rule `Condition→label Condition` in Fig. 5, then we assign *x* to be "left item is a label" and *y* to be `Condition→label Condition`. As another example, if we observe that an input text box is combined with a near right label, which corresponds to the production rule `Condition→Condition label` in Fig. 5, then we assign *x* to be "near right item is a label" and *y* to be `Condition→Condition label`. We observe all the parsing trees and training query interfaces and collect all training pairs $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$.

Suppose that each feature f_i has a weight λ_i, i=1…k. Given an internal node *n*, the probability of a derivation a and its relevant features $h=\{f_1, \ldots, f_m\}$ is set as

$$p(h, a) = \prod_{j=1}^{k} \lambda_j^{f_j(h,a)}$$

Hence, the conditional probability of a derivation a given h and n is

$$p(a|h) = \frac{p(h,a)}{\sum_{a' \in A} p(h,a')}$$

in which *A* represents the set of all possible derivations of *n*.

Let T be a parse tree for a query interface I, and $Der(T)=\{a_1, \ldots, a_m\}$ be the set of derivations contained in *T*. The score of *T* is defined to be the product of the conditional probability

$$P(a_1, \ldots, a_m | d_1, \ldots, d_m) = \prod_{i=1}^{m} p(a_i|h_i).$$

Therefore, the query interface understanding problem is reduced to searching for the parse tree with the largest probability. That is, we find the best parse tree *T**, defined as

$$T^* = \underset{T \in \text{tree}(I)}{\arg\max}\ \text{score}(T)$$

3. EXPERIMENTS

We evaluate the performance of our approach and compare it with some state-of-the-art methods using query interface forms extracted from multiple domains. The cross-validation between query interfaces from different domains and the use of different languages is also reported.

3.1 Data Sets

Four datasets were used to perform the evaluation.

1. TEL-8 is first used in Zhang et al. [2004]. It contains 487 query interfaces from 8 domains: airlines, autos, books, car rentals, hotels, jobs, movies and music records. Each domain contains 20 to 80 query interfaces.
2. ICQ [Wu et al. 2004] contains 100 query interfaces from 5 domains: airlines, autos, books, jobs and real estate. Twenty query interfaces were extracted for each domain.
3. WISE is used in He et al. [2005b] and consists of 147 query interfaces collected from seven domains: books, electronics, games, movies, music, toys and watches.
4. CNW is a dataset containing 200 query interfaces in Chinese from 4 domains: books, movies, airlines and autos. Fifty query interfaces are extracted for each domain.

In all, 734 query interfaces in English and 200 query interfaces in Chinese are used.

3.2 Evaluation Metrics

Four metrics are used to evaluate StatParser and compare it with existing work.

The first metric, *parse precision*, is the number of correct non-terminal semantic units divided by the number of non-terminal semantic units in the semantic tree.

The second metric, *tree metric* [1], measures the correctness of the generated semantic tree. A tree edit distance, which counts the minimum number of insert, delete and relabeling operations needed to convert one tree into another, is used. For an interface, its precision is $P_t = (N_g - D_s)/\ N_g$, in which N_g is the number of nodes in the generated semantic tree and D_s is the edit distance to the gold standard tree. Its recall is $R_t = (N_g - D_s)/\ N_s$, in which N_s is the gold standard tree. Finally, the F-score $F_t = 2P_tR_t/(P_t + R_t)$ is calculated.

The third metric, *condition metric*, measures how well the query conditions are captured. A condition consists of three aspects of information: the name/label of the condition, the set of domain elements and the set of constraint elements. This metric has been applied in He et al. [2] and Zhang et al. [4]. Given a set of query interfaces, let Q_g be the gold standard query conditions and Q_a be the automatically extracted query conditions. The precision, recall and F-score are defined, respectively, as $P_c = |Q_g \cap Q_a|/Q_a$,$R_c = |Q_g \cap Q_a|/Q_g$ and $F_c = 2P_cR_c/(P_c + R_c)$.

The fourth metric, *element labeling correctness* (ELC), measures the correctness of assigning a label to each element. It is defined as the ratio of the number of correctly labeled elements to the total number of elements.

For each query interface in the above datasets, the gold standard semantic tree and query conditions are constructed manually.

3.3 Experiment Results

Table 2 shows the performance of StatParser on the four datasets that are trained using 20 randomly selected query interfaces from the corresponding dataset. It can be seen that StatParser has excellent performance on all of the four datasets.

Table 2. The performance of StatParser

	parse precision	*tree metric*	*condition metric*	*element labeling correctness*
TEL-8	96.1%	95.4%	95.4%	96.5%
ICQ	94.5%	93.6%	95.1%	96.4%
WISE	96.4%	96%	96.4%	97.3%
CNW	92.6%	90.8%	90.4%	92.9%

Figure 6 lists the tree metric F-scores of StatParser over the TEL-8, ICQ, WISE and CNW datasets given different numbers of training query interfaces. In the experiments, we ran a cross-validation on each dataset to make full use of it. That is, each dataset is divided into several subsets with an equal number of query interfaces. One of the subsets is selected as the training set to train StatParser and all other subsets are used as the test sets to evaluate the trained StatParser. In different experiments, the number of query interfaces in each subset is 5, 10, 15, 20, 25 or 30. On the one hand, the tree metric F-score of label assignment initially increases rapidly as the number of training Web sites increases to 15 because more features are identified and the probability parameters approach the real distribution. On the other hand, the tree metric F-score is fairly stable thereafter as most features have been identified and the probability parameters are very close to the real distribution.

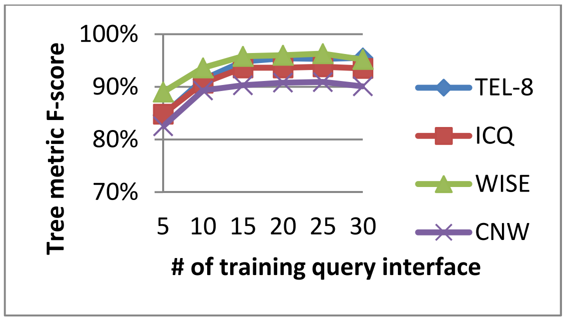

Fig. 6 The tree metric F-scores with different number of training interfaces

4. CONCLUSIONS

In this paper, we present a novel query interface understanding algorithm, StatParser, which effectively parses a query interface into a hierarchical representation. StatParser uses a simple grammar enhanced by probabilities that are learned from a set of parsed query interfaces using the maximum entropy model. The grammar with probabilities is then used to parse a new query interface into parse trees that depict the concept relationships in the query interface. The parse tree with the largest probability is identified as the one that represents the query capabilities of the query interface. StatParser has the advantages of both rule- and learning-based methods. Experimental results show that StatParser is very precise in capturing the element relationships in a query interface and is very effective at extracting the query conditions.

5. ACKNOWLEDGMENTS

This work is partially supported by NSFC grants (No. 61073017) and UIC internal research grants.

REFERENCES

[1] Dragut, E. C., Kabisch, T., Yu, C., and Leser, U. A hierarchical approach to model web query interfaces for web source integration. *Proceeding of the VLDB Endowment*, 2, 1, 325-336. 2009.

[2] He, H., Meng, W., Lu, Y., Yu, C., and WU, Z. Constructing interface schemas for search interfaces of web databases. In Proceedings of the 6th International Conference on Web Information Systems Engineering, 29-42. 2005.

[3] Su, W., Wu H., Li Y., Zhao J., Lochovsky F., Cai H. and Huang T. Understanding Query Interfaces by Statistical Parsing. *ACM Transaction on Web* (TWeb). 7(2), 1-22. May 2013.

[4] Zhang, Z., He, B., and Chang, K. C.-C. Understanding web query interfaces: Best-effort parsing with hidden syntax. In *Proceeding of ACM SIGMOD Conference*, 107-118. 2004.

[5] Wu, W., Yu, C., Doan, A., and Meng, W. An interactive clustering-based approach to integrating source query interfaces on the deep web. In *Proceedings of the ACM SIGMOD Conference*, 95-106, 2004.