

MOBIES: Mobile-Interface Enhancement Service for Hidden Web Database

Xin Jin
George Washington University
Washington, DC, USA
xjin@gwu.edu

Aditya Mone
University of Texas at Arlington
Arlington, TX, USA
aditya.mone@mavs.uta.edu

Nan Zhang*
George Washington University
Washington, DC, USA
nzhang10@gwu.edu

Gautam Das†
University of Texas at Arlington
Arlington, TX, USA
gdas@cse.uta.edu

ABSTRACT

Many web databases are hidden behind form-based interfaces which are not always easy-to-use on mobile devices because of limitations such as small screen sizes, trickier text entry, etc. In this demonstration, we have developed MOBIES, a third-party system that generates mobile-user-friendly interfaces by exploiting data analytics specific to the hidden web databases. Our user studies show the effectiveness of MOBIES on improving user experience over a hidden web database.

Categories and Subject Descriptors

H.2.7 [Database Administration]; H.3.5 [Online Information Services]: Web-based services

General Terms

Design, Experimentation, Performance

Keywords

Mobile HCI, Hidden web database, Data analytics

1. INTRODUCTION

A large number of hidden web databases provide proprietary form-based interfaces (consisting of textboxes, drop-down boxes, etc) for users to enter their desired values in a search query. For example, the web interface for NSF fastlane award database¹ provides a search form consisting of 22 control elements, including 6 drop-down boxes and 9 textboxes. Form-based interfaces is often difficult-to-use and error-prone on mobile devices (PDA/smart phones), mainly because of limitations such as smaller screen, trickier text input, etc.

Despite that many efforts (e.g., interface rendering) have been made to enhance mobile-user experience at the end of mobile OS

*Partly supported by NSF grants 0852673, 0852674, 0845644, 0915834 and a GWU Research Enhancement Fund.

†Partially supported by NSF grants 0812601, 0915834, 1018865, a NHARP grant from the Texas Higher Education Coordinating Board, and grants from Microsoft Research and Nokia Research.

¹<http://www.nsf.gov/awardsearch/tab.do?dispatch=4>

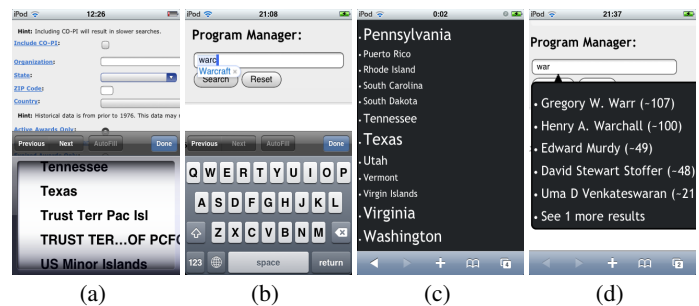


Figure 1: (a) Drop-down box on iPhone. (b) Auto-suggestion on iPhone. (c) Drop-down box with value visualization. (d) Textbox with auto-suggestion.

or browsers, we argue that existing solutions without considering *data analytics* do not suffice to alleviate the challenge on accessing hidden web databases from mobile devices. To see the reason, let us first consider an interface rendering example on the Apple iPhone. Figure 1a shows a spinning-wheel rendered to ease finger scrolling for the drop-down box element “PI State”. Given the limited space of the wheel, mistaken selections by rotation may occur every so often because “PI State” has as many as 74 option values. A closer look at the database, however, reveals that some values can be removed while others should be (somehow) highlighted because of popularity - e.g., values “US Minor Islands”, “Palau”, etc return no tuple whatsoever, while “Pennsylvania” returns 2,277 tuples.

Figure 1b shows another user-unfriendly scenario without learning the data analytics from the database being visited. As in the example, when people type “warc” into a textbox element “Program Manager”, a dictionary-based auto-suggestion method prompts “Warcraft”, which is almost impossible to be a person’s name.

In this demonstration, we bring into practice our previous work on hidden web databases [3,4]. We develop a novel third-party service called MOBIES (MOBILE Interface Enhancement System) to enhance mobile-access interface by exploiting data analytics (i.e., discovering attribute domain values, estimating aggregate information) from the hidden databases. For each supported hidden database, MOBIES issues a small number of search queries through the original form-based interfaces to retrieve the analytical information required for mobile interface construction. Then, MOBIES builds mobile-access interface based on the retrieved analytics and makes it available to mobile users.

To ensure responsiveness, MOBIES is not an “on-the-fly” service. Instead, it requires a pre-processing stage to calculate the data analytics by sampling. Each constructed interface is updated periodically to synchronize with the databases. Nonetheless, it is a *generic* solution in that the support for a hidden database can be easily added with minimal human intervention.

2. MOBIES ARCHITECTURE

Our MOBIES architecture is described in Figure 2 including two main components, *interface generator* (IG) and *mobile enhancement server* (MES).

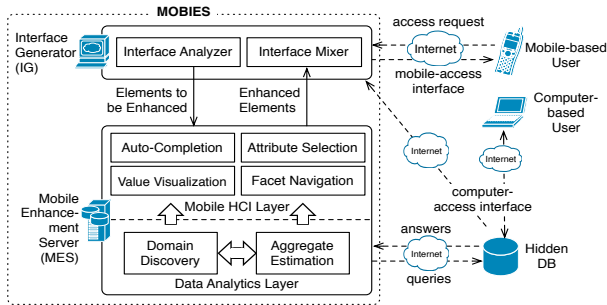


Figure 2: Detailed Architecture of MOBIES

2.1 Interface Generator (IG)

IG is mainly concerned with information extraction/integration issues. We acknowledge that there are many related works in the information retrieval community [1]. Hence, we intentionally design IG as a “plug-and-play” component for existing methods. Specifically, IG has two main modules: *interface analyzer* and *interface mixer*. Interface analyzer retrieves the computer-access interface of each supported hidden database and parses it to identify mobile-unfriendly elements (e.g., textboxes) which can be enhanced by MES. It also informs MES about these elements and requests for improvement. Interface mixer, on the other hand, receives from MES the enhanced elements and piece them together with the rest to generate the new mobile-user-friendly interface.

2.2 Mobile Enhancement Server (MES)

2.2.1 MobiHCI Layer

The MobiHCI (Mobile Human-Computer Interaction) layer receives the element-enhancement requests from IG, applies mobile-user-friendly design patterns, and transmits the enhanced elements back to IG. This layer currently supports 4 design patterns.

Value Visualization is illustrated by an example in Figure 1c. In contrast with its original interface in Figure 1a, ours can highlight those popular options (e.g., “Pennsylvania”, “Texas”) by enlarging them for the ease of scrolling selection. The intensity of popularity is provided by the data analytics layer through estimating COUNT (i.e., the number of tuples matched with the value) from the underlying hidden database. An accurate estimation is desirable to ensure that the most popular options would be the most noticeable. **Auto-suggestion** is to assist a user to enter the desirable text for an attribute by providing a number of values from its domain matching with the current input. Figure 1d is an example of MOBIES to auto-suggest the most popular “Program Manager” names for the NSF fastlane database, responding to the input “war”. The parenthetical numbers (e.g., (~107)) denote the respective name popularity estimated from the database. To enable this, a key prerequisite besides the earlier COUNT estimation is *domain discovery* - i.e., for a

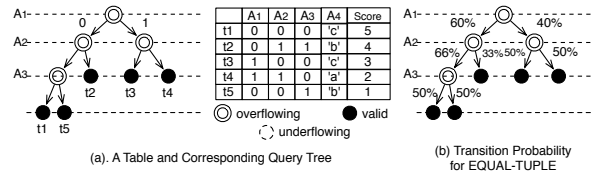


Figure 3: An example of query tree

textbox attribute, our data analytics layer ought to provide the most frequent (if not all) values in its domain. Once the domain has been discovered, auto-suggestion can be achieved by pre-retrieving a small set of values ranked by their estimated COUNT.

Output Attribute Selection intends to address the limitation of the mobile screen size. In particular, we select a “best” subset of attributes to form a snippet to be displayed on the screen, while values of the other attributes are available through a linked “detail” page. We adopt the method in [2], which suggests to display the attributes that are more closely correlated with the scoring function. To make such a decision, one requires the estimations from the data analytics layer to calculate the correlation coefficients.

Facet Navigation aims to select the most important facet (or a small number of important facets) to display on a mobile device. The basic idea is to select the attributes that can most effectively distinguish different tuples. To enable the existing facet selection algorithms, a critical support is again required from the data analytics layer to estimate marginal distributions of each attribute.

2.2.2 Data Analytics Layer

Domain Discovery Component: The domain discovery is performed by traversing a *query tree*, which is constructed from *pre-known attributes*. For example, “PI State” in Figure 1a is pre-known because all the 74 different domain values are available on its drop-down menu. Other pre-known attributes include radio-buttons, check boxes and so on. A more general case without any pre-known attributes was discussed in our work [4].

Figure 3a shows a query tree example for a hidden database D with 5 tuples, where A_i ($i \in [1, 3]$) are pre-known attributes while attribute A_4 is unknown. Note that **score** is only to “emulate” the concept of scoring function, which by no means can be known in practice. The i -th level represents A_i , while each edge represents a domain value of its parent level. Then, each node in the tree forms a query, with the root being $\text{SELECT } * \text{ FROM } D$ and each node at the i -th level containing $i - 1$ predicates - corresponding to every edge on the path from the root to the node. The left-most leaf-level node, for example, represents $\text{SELECT } * \text{ FROM } D \text{ WHERE } A_1 = 0 \text{ AND } A_2 = 0 \text{ AND } A_3 = 0$. Since the hidden database interface is usually restricted to return up to k tuples, one overly broad query (i.e., selects more than k tuples) will *overflow* and return only the top- k tuples selected according to the proprietary scoring function. For example, each node in Figure 3a is labeled by the outcome of its corresponding query when $k = 1$.

The main problem of a simple depth-first-search (DFS) traversal happens when the unknown attribute A_4 is strongly correlated with the attributes on the top of the tree. For example, if $A_1 \rightarrow A_4$ is a functional dependency, the same A_4 value is encountered by DFS when searching the subtree under $A_1 = 0$. On the other hand, a breath-first-search (BFS) traversal also wastes queries as A_4 is correlated with the scoring function. For example, consider a total order of values for A_4 and suppose that tuples with “larger” A_4 receive higher scores. Then, it is likely for BFS to discover only the larger values of A_4 .

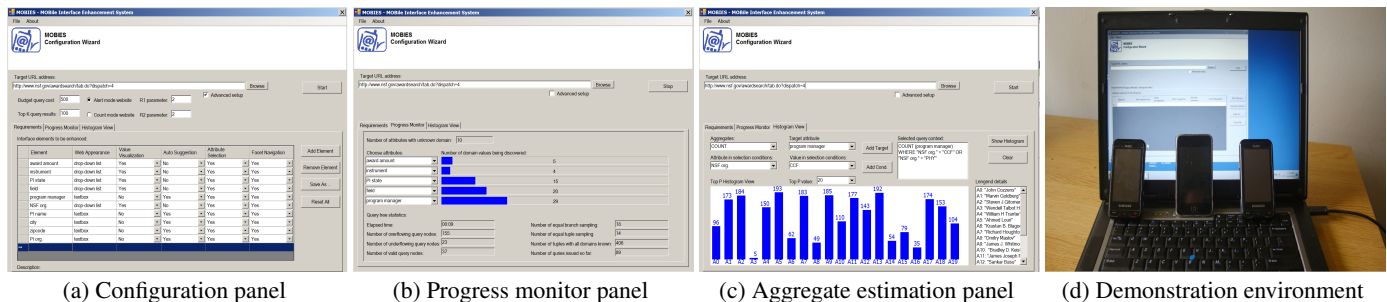


Figure 4: Snapshots of MOBIES configuration tool and real demonstration setup.

Therefore, we introduce randomness to the traversal for discovering domains, by mixing two random walks: *Equal-Branch* and *Equal-Tuple*. Specifically, *Equal-Branch* selects an outgoing branch uniformly at random and issues the query corresponding to the destination, while *Equal-Tuple* follows each branch with a transition probability *proportional* to its COUNT. Figure 3b shows an example of the probability for EQUAL-TUPLE to follow at each branch. Note that the COUNT can be either explicitly given by the website or estimated by the next Aggregate Query Processing Component. **Aggregate Query Processing Component:** We adopt our prior work [3], where the above-mentioned query tree is formed as the basis to enable a Horvitz-Thompson estimator such that aggregate estimations including COUNT, SUM and AVG can be accurately estimated with consuming merely a small number of queries.

3. DEMONSTRATION PLAN

The demonstration begins with a brief architecture introduction, followed by a short user study report. After that, we lead the audience through two scenarios to explore all MOBIES features: data analytics pre-processing and mobile online experience. We use the NSF fastlane award website as an example to demonstrate our MOBIES system.

3.1 User Study

We perform the user study to compare the mobile user-friendliness before and after applying MOBIES. Our preliminary study was conducted on iPhone with participation of 20 students at the University of Texas at Arlington. Each subject was required to access from iPhone a pair of NSF fastlane web interfaces for comparison. Both interfaces were composed of the same 4 drop-down menus: “PI State”, “Award Amount”, “Application Field”, “Award Instrument” and 1 textbox “Program Manager”, with the only difference that one had been enhanced by MOBIES. Then, we recorded the time of each subject to input the same workload of 20 queries. Every query contains 5 predicates corresponding to the 5 interface elements, respectively. Our finding was that the average input time was 323.3 seconds shorter after applying MOBIES.

3.2 Data Analytics Pre-Processing

We first provide the audience an chance to sneak preview the original NSF fastlane website on multiple mobile devices including iPhone, HTC Android and Palm Pre. Then, we switch to the MOBIES configuration panel (Figure 4a), where the audience can complete the following setup: 1) specify the interface element(s) of their own interest to be enhanced by auto-suggestion and/or value visualization; 2) choose to discover one or multiple attributes whose domains are unknown and 3) set the query cost (i.e., the maximum number of search queries to be issued) as the termination condition.

Our pre-processing starts following a click on the “Start” button but can stop anytime upon request.

During the pre-processing period, MOBIES collects a variety of real-time statistics, all of which are available to the audience through our progress monitor panel (Figure 4b) and aggregate estimation panel (Figure 4c). In particular, there are 3 parts for the demonstration. First is the *domain discovery progress*, where the audience can see a respective run-time progress bar for each unknown attribute (specified in the beginning setup). Second, since the entire data analytics layer hinges on sampling based on a concept of query tree (as in §2.2.2), we publish the query tree structural information in conjunction with the dynamic statistics in the hybrid sampling algorithm such as the number of overflowing/underflowing/valid nodes, elapsed time and so on. Third, to see the effectiveness of aggregate estimation, the audience are allowed to view the *histogram* of domain value popularity for both pre-known and unknown attributes. Selection conditions are supported when requesting the histogram. For example, Figure 4c shows a histogram to display all the program managers working for either CCF or PHY organizations. For those attributes with a large number of domain values, we allow the audience to set a threshold k such that the generated histogram shows up to k most frequent values.

3.3 Mobile Online Experience

Due to the low-connectivity and/or long waiting time for the previous scenario, we have prepared a completed pre-processing version for back-up. The audience are able to check from the configuration panel our default setup. In this scenario (Figure 4d), we allow the audience to experience the enhanced look-and-feel of the NSF fastlane interface by going through all the 4 patterns discussed in §2.2.1 on different mobile devices (i.e., iPhone, HTC Android and Palm Pre). After the audience submitting a search query, they will be directed into a facet navigation mode if the returning results are over a threshold (10 by default but can be changed to 30 or 50). Otherwise, the search results will be output in a concise form based on the output attribute selection pattern.

4. REFERENCES

- [1] K. Chang and J. Cho. Accessing the web: From search to integration. In *Tutorial, SIGMOD*, 2006.
- [2] G. Das, V. Hristidis, N. Kapoor, and S. Sudarshan. Ordering the attributes of query results. In *SIGMOD*, 2006.
- [3] A. Dasgupta, X. Jin, B. Jewell, N. Zhang, and G. Das. Unbiased estimation of size and other aggregates over hidden web databases. In *SIGMOD*, 2010.
- [4] X. Jin, N. Zhang, and G. Das. Attribute domain discovery for hidden web databases. In *SIGMOD*, 2011.