Find This For Me: Mobile Information Retrieval on the Open Web

Ifeyinwa Okoye
Institute of Cognitive Science
Univ of Colorado at Boulder
Boulder, CO 80309
USA
okoye@colorado.edu

Jalal Mahmud
IBM Research – Almaden
San Jose, CA 95120
jumahmud@us.ibm.com

Tessa Lau
IBM Research – Almaden
San Jose, CA 95120
tessalau@us.ibm.com

Julian Cerruti
IBM Argentina
Ing. Butty 275 - C1001AFA
Buenos Aires, Argentina
jcerruti@ar.ibm.com

ABSTRACT
With all the information available on the web, there is a growing need to provide mobile access to this information for the large, growing population of mobile internet users. In this paper, we propose a solution to the problem of *open web* mobile information retrieval, by conducting a dialogue with the user over a simple text-based interface. Using techniques from NLP, web page analysis, and information extraction, our approach automatically navigates web sites on the user's behalf and extracts specific information from those sites to present to the user textually. Empirical evaluation shows that our approach to open web information retrieval is feasible, and a qualitative evaluation validates that such a system meets user needs for mobile information access.

ACM Classification Keywords
H.5.2 Information Interfaces and Presentation: User Interfaces

General Terms
Algorithms, Design, Human Factors, Experimentation

Author Keywords
NLP, information extraction, information retrieval, mobile web

INTRODUCTION
Web browsing from a mobile phone continues to be challenging. The web is largely designed to be used from a PC, not a mobile device. Accomplishing information tasks from a mobile device can be difficult due to screen size limitations, slow page loads, and the need for excessive scrolling and navigation. Yet in many parts of the world, such as developing countries, mobile phones are the only means people have of connecting to the internet. These users typically don't have smart phones or data plans. How can we provide access to information for these users?

One of the main motivators for web access on mobile is for information retrieval tasks. For example: which LCD TV should I buy? what is the weather forecast for next week? can I get Green Day's latest album for less than $10? Our goal is to provide a mobile interface for these types of information retrieval tasks that is based on dynamically crawling and scraping the *open web* – the vast network of web sites designed solely for human consumption through a desktop browser.

Information retrieval tasks can be quite complex, including fetching information from multiple sites, collecting information about multiple items, and making comparisons between them. However, if we reduce these complicated tasks to their fundamentals, one basic building block for information retrieval consists of fetching a single property about a single item, such as the weather in Denver, the ISBN of the book *The Iliad*, or the price of a Samsung 40" LCD TV.

Furthermore, in our observations, we discovered that a majority of users' information retrieval needs fell into this simple item-property model. Therefore our approach has focused on solving this specific type of information retrieval over the open web.

Beginning with a natural-language query from the user, we find information about the specified item from a specific website, engage in a multiple-choice dialogue with the user to refine her query, and finally extract and present the item's property to the user. Figure 1 shows one sample usage scenario of our system.

Our approach combines natural language processing techniques, search automation, and information extraction techniques to solve the item-property retrieval problem.

In this paper, we make the following contributions:

- An algorithm for item-property information retrieval on the open web, consisting of the following steps: NLP for extracting an item and a property from user requests;

IUI 2011, February 13 - 16, 2011, Palo Alto, California.

user: get the price of a 42" lcd tv
system: please enter the website, for example: http://www.foo.com
user: http://www.compusa.com
system: I have found more than one option on the web-page. Which one would you like?
1: LG 42LD450 42" Full HD 1080p LCD HDTV - 1080p, 1920x1080, 16:9, 100000:1 Dynamic, 4ms, USB, 2 HDMI
2: Panasonic TCP42C2 Viera C2 Series 42" Plasma HDTV - 720p, 1024 x 768, 16:9, 600Hz, 2000000:1 Dynamic, 2 HDMI
3: LG 42LD550 42" LCD HDTV - 1080p, 1920 x 1080, 16:9, 120Hz, 150,000:1 Dynamic, USB, 4 HDMI
4: LG 42LD520 42" LCD Full HDTV - 1080p, 1920 x 1080, 16:9, 100000:1 Dynamic, 120Hz, 2.4ms, 3 HDMI
5: Panasonic TCL42U22 Viera Link 42" Class LCD HDTV - 1080p, 1920x1080, 16:9, 20000:1 Dynamic, 3 HDMI
user: 1
system: Result:
#1 Price: $599.99
#2 List Price: $749.99
#3 CompUSA is not responsible for typographical or photographical errors. Prices and specifications are subject to change without notice.*Price after manufacturer mail-in rebate. Restrictions apply.

Figure 1. Example interaction with our system

search form detection and automation; search result detection; and information extraction to return the desired property;

- An implementation of our algorithm;
- Empirical evaluations of each of the components in our implemented system; and
- A qualitative user study that validates our approach.

The rest of the paper is organized as follows. We begin with an overview of related work and how our work compares in context. We then detail the problem of mobile information retrieval, including the approach we have chosen. Next, we describe the main building blocks of our algorithm, interleaved with empirical evaluations of each part. We then present the results of a user study that validate the approach we have taken with users in the developing world. Finally, we conclude with a discussion of future work.

RELATED WORK

Our work is related to research on question answering, interactive information retrieval systems, and intelligent agents for mobile.

Question Answering and Interactive Information Retrieval Systems

There exist a large number of question answering and interactive information retrieval systems [16, 6, 7, 4, 18]. SmartWeb [16] combines intelligent question answering with composable semantic web services to provide a multi-modal user interface for the Web. Google 411 [6] is a speech recognition based business directory assistance service which allows searching and placing a call to the local businesses from a mobile phone. Google SMS [7] allows searching for and retrieving specific types of information from a mobile phone.

These and many more question answering and information retrieval systems work with pre-existing semantic web services. These web services are typically annotated with their semantic content and thus finding the appropriate web service for a task is straightforward. However, the number of web service providers that provide web service APIs to their information is relatively small. Therefore, people often need to retrieve information from websites that do not provide a web service API. In contrast to existing approaches, our solution attempts to solve a more general problem, i.e. retrieving information from any website. Our problem is more difficult because we have to find information from complex webpages which are not semantically labeled.

IBM's DeepQA [4] is a question answering system which uses natural language processing, information retrieval, machine learning, knowledge representation and reasoning, as well as massive parallel computation to solve the question answering problem. While DeepQA attempts to solve the general QA problem using a preprocessed data set, our work focuses specifically on a very small subproblem of QA and aims to deliver realtime results from the live web. RIA [18] is an interactive context-sensitive information retrieval system for large and complex data sets. It uses natural language dialogue, context sensitive information retrieval, and multimodal output generation to perform the information seeking task. RIA presents information from a pre-existing dataset, compared to our approach which works over the open web. However, we would like to explore integrating more features from systems such as RIA and DeepQA into our own work to improve our accuracy and coverage.

Intelligent Agents for Mobile

HP's SiteOnMobile [14] allows end users to create "tasklets" to scrape data from websites and make it available via SMS or voice calls. Similarly, CoCo [10] lets users create scripts to automate web tasks and invoke them from a mobile interface. However, both of these systems require the user to have previously instructed the system how to do the task. In contrast, our system doesn't require any pre-programming or the existence of pre-recorded scripts. Siri [13] is an intelligent assistant for the iPhone that allows delegation of certain preprogrammed tasks such as making restaurant reservations, getting movie tickets, and hailing a taxi. It works with a number of web service providers to deliver answers and actions from them. Like the QA and information retrieval systems above, Siri relies on an existing set of web services to provide its functionality, and cannot be extended to open web information retrieval.

MOBILE INFORMATION RETRIEVAL

Preliminary study

We conducted a preliminary study to explore what sort of tasks users would feel comfortable delegating to a system.

For this, we set up a Wizard of Oz study where users thought they were directly interacting with a system via Internet Relay Chat (IRC). Twelve software developers in Argentina were recruited as study participants. The users were told to ask the system to perform any web task and we had someone posing as the system and doing the tasks. In all, we collected a total of 51 tasks.

The tasks ranged from basic questions (date and argentinian time for Argentina vs. Greece?) to information synthesis (send me the state of my current ebay bids) to even more complicated actions (write on my Twitter "I'm going to Tandil").

Overall, 54% of the tasks were information retrieval tasks (e.g., what is the weather forecast for next week in Almaden), while 46% were requests to take action (e.g., forward my work phone to my cell). Given that more than half of the tasks required information retrieval, we decided to focus our efforts on providing a system for mobile information retrieval.

The information retrieval problem

Information retrieval tasks can be relatively complex. For example, a user might want to plan a fishing trip for the upcoming weekend, deciding between multiple locations based on their weather forecasts, recent rainfall, and access to fishing spots.

Because information needs such as these typically span multiple websites, one of the goals of our work is to provide a general-purpose system that can work across arbitrary sites on the open web. We specifically do not want to limit our approach to working only with sites that provide semantic web interfaces, even though that would make the problem we are solving significantly easier. Moreover, the proportion of websites providing semantic web interfaces is significantly lower in the developing world. Therefore we have formulated our approach to work over the open web, websites designed for use by people using standard web browsers.

While many information needs are complex, one of the fundamental building blocks for solving those needs consists of one basic operation: fetching a single property about a single item from a specific website. Therefore, we designed our system to solve this specific information retrieval task. Furthermore, We observed in our preliminary study that many of the information retrieval tasks could in fact be formulated as an item-property retrieval problem.

Item-property information retrieval

Finding information about a specific item is generally done using web search. Therefore, our algorithm for item-property retrieval is based around using the web search built in to specific websites to retrieve the desired information. Although there are other means to find item information without using search, such as browsing the site's taxonomy, this taxonomic information would either have to be known to the system, or collected from the user in an interaction. The former requires more domain-specific information to be provided to the system, and the latter requires undue interaction on the user's part. So we decided to rely only on the website's search functionality.

At a high level, our algorithm follows the following approach: given a user's information query, determine which website to use, automatically interact with that website using its search functionality to find the desired item, dialogue with the user to narrow down the search results if necessary, and then extract the desired property of the item on the item's detail page to return to the user.

We chose to use NLP techniques to interpret the user's query, in order to make the interface as general as possible. A menu-driven interface would have limited us to only the set of concepts designed into the system, rather than generalizing to arbitrary concepts defined by users. Our NLP-based algorithms are described in more detail in the *Natural Language Query Processing* section.

Currently we explicitly ask the user to specify which website to use to satisfy their information need. Alternatively, we could ask the user to specify where they wanted to search, e.g. "Best Buy", and then do a Google search for the term and use the top-ranked website as the starting point for our search. Or, we could remember the last-used website and use it for future queries. Another way could be suggesting a default website on which to run the query, and let the users override this if they wish.

Given a search term and a website, the *Search Form Detection* subsection below describes how we locate the search form and automatically interact with it to conduct the search. The user's search term is frequently not specific enough to describe exactly the item they want. For example, they might say "lcd tv" instead of "LG 42LD450 42" Full HD 1080p LCD HDTV - 1080p". In these cases, websites typically return a list of search results rather than taking the user directly to the item detail page for their item. Our approach to handling this situation is to detect the search result page, extract summaries of each of the results, and present these options to the user. The *Search Result Detection* subsection below describes our approach in more detail.

Finally, after the user has selected an option, the system automically navigates to the detail page for that item and tries to extract the desired property (e.g., price, temperature, ISBN) for that item. For this task we use a variety of information extraction heuristics, described in the section *Information Extraction*.

NATURAL LANGUAGE QUERY PROCESSING

In this section we discuss how we used natural language processing (NLP) as a first step towards helping a user retrieve information on the web. We used NLP to help us figure out what exactly the user was trying to accomplish given the user's task description i.e we wanted to understand the semantics of a user's query. FrameNet [2] is a logical resource to use for such queries since the frame name describes what the main concept is and the frame elements indicate the semantic role for each lexical unit in the user's query. How-

ever, FrameNet does not cover all the predicates in English and FrameNet's accuracy in assigning the lexical units in a sentence to the right frame element is less than 60% for their own data set [3]. It is very likely that the accuracy will go further down when using a different data set which their classifier wasn't trained on. Hence, due to the coverage and accuracy limits of FrameNet, we did not use FrameNet for our task. Instead of using the FrameNet roles, we used the three basic roles of *item*, *property* and *web location*. Another insight from the preliminary study was that most users tended to not include the web location as part of their task description so we explicitly ask them for the web location. Hence our work in this section was geared towards deducing only the item and property from the task description.

We used a dependency parser to first understand how the lexical units in the sentence relate to each other and then applied several heuristics to determine the item and property, given the dependency parse of the user's task description. We used the Stanford Parser [9] because it is freely available, easy to install and use, is continually being updated, and has a very active community.

To extract the item and the property, we do the following (in order):

1. We replace quoted strings with a dummy non-quoted string. So, *who is the author of "What is the what"* will be replaced with *who is the author of QuotedString*.
2. We parse the string using the Stanford Parser.
3. We find the head noun in the parsed string. From our preliminary study described above, we found that most of what we considered to be the item in a sentence turned out to be a noun or a noun compound. Hence, we look for the head noun in the parsed string and that becomes the beginning of the item. For example,*3g* is the head noun in *what is the price of the white iphone 3g*.
4. We resolve the head noun to include its modifiers. So *what is the price of the white iphone 3g* will have *white iphone 3g* as the item, although only *3g* is the head noun for the sentence.
5. Next, we find another noun that is in a preposition or possession relationship with the head noun and this becomes the property. For example, price is in a prep_of relationship with 3g in the sentence *what is the price of the white iphone 3g* while number is in a possession relationship with Doe in the sentence *what is John Doe's cell phone number?*
6. Finally, we resolve the property such that its modifiers are attached to it. So the full property for *what is John Doe's cell phone number* will be *cell phone number*.

Empirical Evaluation of Query Processing

We evaluated how well the system was able to determine the item and the property of the item given a sentence. For example, given the sentence *what are some user reviews of bangkok haunts*, the item would be *bangkok haunts* and the property would be *user reviews*. To get the data set, we asked people in our department to send us ten tasks they had tried to do on the web expressed in the form of an English sentence. Twenty people responded. We removed duplicates (from 200 sentences) and ended up with about 130 sentences. Next, we asked two people to decompose each of the sentences into an item and a property. Then we had the system do the same for each of the sentences. We used the agreement between the two experts as the ground truth, *i.e.*, the correct value for the item and property of a sentence are the values both experts agreed on. For the item, the experts agreed 75% of the time and for the property they agreed 69% of the time. Making the ground truth to be only the data points for which both experts were in agreement, the system's recall for identifying the item was 69% while the recall for identifying the property was 83%. We didn't compute the precision of our query processing algorithm since we were more interested in seeing how correctly our system could identify item name and property when they were actually present in the query and when both annotators agreed on their values. During error analysis, we found that most of the errors were a result of the system not being able to parse the user's text correctly. For example in the sentence *who is the author of what is the what*, the system does not recognize *what is the what* as one entity. In the future, we hope that with enough training examples, we can retrain the parser model to deal with these kind of user inputs. The fact that experts only agreed part of the time emphasizes the difficulty in identifying the item and property in an unconstrained user's textual input; in that light, the system's performance is relatively good considering the difficulty of the problem.

SEARCH AUTOMATION

In this section, we describe our algorithm to automate search in a website. It takes a website URL and item as input and outputs a list of search results options. Our algorithm identifies a search form from the home page of that website, uses a browser automation service such as CoCo [10] to search that website using the item as the search term, identifies search results from the next returned webpage, extracts options from the search results, and returns them. Next, we will describe two key components of our algorithm: search form detection and search results detection.

Search Form Detection

Search form detection on webpages has been discussed in a number of previous papers [15, 11]. For example [15] used a supervised machine learning based approach to detect search forms. [11] used a simpler approach by using a shallow knowledge-base with keywords to detect search forms. We use the second approach (knowledge-base with keywords) to detect search forms for its simplicity and to avoid the amount of effort needed to collect and label training data for a machine learning solution. In [11], the authors proposed a knowledge-base with the words *search*, *find*, and *go*. We augmented that knowledge-base with additional words (e.g., look it up, go!) after inspecting a number of websites. [11] identified all instances of search forms from the webpage. However, we identify a single search form to conduct

a search. If there are multiple search forms in a page, we select the one which gets the highest word overlap with the keywords from knowledge-base.

Search Results Detection

Existing Approaches

There are existing algorithms to detect search results [15, 5, 11]. Previous approaches can be broadly categorized as machine learning based solutions (e.g., [15]), template-based solutions (e.g., [5]) or contextual browsing solutions [11]. Template-based solutions [5] require the system to store a set of search result templates provided by users, which is neither scalable nor practical for our problem. Contextual browsing solutions [11] use the context of a followed link or the search terms to detect the most relevant geometric segment from the next webpage. However, this approach assumes that search results appear in one geometric segment which is not necessarily the case because of different granularity of geometric segments in webpages across different sites. We did not use a machine learning based solution (as proposed in [15]) to avoid the amount of effort required to train the classifier for search results detection. Although such a solution would work well for the domains on which the classifier is trained, incorporating a new domain would require additional training, which we wished to avoid.

Our Algorithm

To fit our needs, we developed our own algorithm to detect search results from a webpage. In this section, we refer to each individual search result as a *search result item*. We use the term *search result node* to refer to the DOM tree node which contains those search result items.

Our algorithm uses the observation that search result items have similarity in their presentation patterns (e.g., matching XPaths) in the webpage. Figure 2 shows results for a search for "calcium citrate" on the Walgreens.com website. Each search result item is highlighted using a dotted rectangle. As seen from the figure, the items have similar presentation styles and patterns.

To detect search results, it is important to capture such similarities in patterns. A simple approach to detect search results by using pattern similarities could be to visit the DOM tree of the page in a top down order, for each node check if its child nodes are similar (matching tags and similar structure) and identify the parent node as the search result node if child nodes are similar. However, such an approach won't work because it may also detect item taxonomies (categories of items, e.g., in Figure 2, the menu items on top of the page) as a search result. In addition, in a webpage there are many lists with repeating patterns, e.g. list of links, combobox, etc. If we only look at pattern similarity of the nodes to detect search results, we may end up detecting many repeating patterns which are actually not search results. So how can we differentiate search results from other lists or repeating patterns?

From previous studies and our observations of many websites, in addition to repeating patterns of search result items, we deduced the following additional criteria for detecting search results:

- Each search result item typically has some structure as opposed to consisting of a single link, which doesn't have any subtree or structure in its DOM tree representation. For example, in figure 3, the search result item has a structure (subtree).

- There are some similarities in the tag distributions of the child nodes of each search result item in the DOM tree. For example, in figure 3, child nodes of the search result item have similar tag distributions.

We may apply the above criteria to the simple approach (to detect repeating patterns) to detect search results. Thus, if the childnodes of a given node are similar (similar tag-distri butions), we can do further checking for each of them to see:

- if they have some structure

- if there are similarities in tag distributions of their childnodes.

This approach can detect more (actual) search results than the previous simpler approach because it prevents single lists of links or images, which are most often used to display categories of items or other lists, from being identified as search results. However, in a webpage there may be multiple patterns which satisfy all of the criteria of search results (i.e. repeating patterns and the other two criteria listed above). To prevent other lists that satisfy the above criteria from being identified as a search result, we add the following heuristics to our algorithm:

- Search results should be the list where each search result item will be associated with a feature which represents the pattern of that search result item. We call it a *Pattern Feature*. Multiple items can have the same value of that feature. A weight (importance) of a particular pattern feature will represent its frequency of occurrence. We compute a weight (importance) of each potential search result item (identified using the previous criteria) using the weight of its pattern feature, and the match of the search terms (since in many cases, a search result item has matching terms with search words) with the words from it. For (actual) search result items, such weight will be maximum among items in any other lists which satisfy our other criteria of search results detection.

We designed our algorithm which takes into account all of the criteria discussed above. We design our algorithm as a bottom up approach so that we generate all the candidate search result items from the webpage, compute their features and weights, find the list of items which have the maximum weight, and return them as search results.

Now that we have discussed our design decisions, we will set up few definitions which will help to understand our algorithm. We refer to each DOM tree node which has a pattern inside it as a *pattern node*. Figure 3 shows one such search

result item and the corresponding pattern node in the DOM tree. As seen from the figure, child nodes of the pattern node have similarity in their structure. Algorithm *isPatternNode* determines if a node in the DOM tree is a pattern node. We won't discuss each line of the algorithm for lack of space, but conceptually this algorithm detects a node as a pattern node if it has child nodes with similar structure. However, we like to mention some additional heuristics we have applied to detect a pattern node:

- while computing child-tag distribution of a node (line 2), we ignore the tags which don't affect the structure, examples of these tags are `br`, `hr`, `style`, `script`, `b`, `text`.
- while computing if two child nodes are similar (line 11), we only compute similarity of their tag distributions by using cosine similarity and a threshold of 0.3 (determined experimentally).
- while deciding if a node is a pattern node by comparing number of matching child nodes (line 13), we use another threshold which is 1 (determined experimentally).
- while checking if a pattern feature is valid, we use the observation that a pattern node typically doesn't contain any of the following tags in its XPath: `script`, `input`, `option`, `select`, `map`, `area`, `strong`. Thus, a node with any of those tags in its pattern feature is not a valid pattern node.

Algorithm *isPatternNode* identifies the search result item shown in figure 3 as a pattern node. The pattern feature computed for this node is: `html.body.div.div.div.div.div.div.div.div`. The above algorithm is the core part of the search results detection algorithm *findSearchResults* which computes a weight for each pattern feature, finds pattern nodes with maximum weight, their common ancestor node (i.e. search result node) and all the search result items which descend from that node.

Algorithm *isPatternNode*
Input: $Node$: A Node in the DOM Tree
Output: *TRUE*: if the node is a pattern node, *FALSE* otherwise

1. **if** $Node.isLeaf$ **return** $FALSE$
2. $Node.ChTagList \leftarrow$Child Tag List of the $Node$
3. $CmTag \leftarrow$Most Frequent Tag in $Node.ChTagList$
4. $Index \leftarrow$Index of First Child Node which has $CmTag$
5. $CmChild \leftarrow Node.ChildList.Get(Index)$
6. $MatchCount \leftarrow 0$
7. **for** $i \leftarrow 1$ **to** $Node.ChTagList.Length$
8. **do** $CurChildTag \leftarrow Node.ChTagList.Get(i)$
9. $CurChild \leftarrow Node.ChildList.Get(i)$
10. **if** $CurChildTag = CmTag$
11. **then if** $isSimilar(CmChild, CurChild)$
12. **then** increment $MatchCount$
13. **if** $MatchCount > Threshold$
14. **then** $Node.Pattern \leftarrow Concat$(Node.$XPath$,CmTag)
15. **if** $isValidPattern(Node.Pattern)$
16. **then return** $TRUE$
17. **else return** $FALSE$
18. **return** $FALSE$

Figure 2. Search Results in a webpage (highlighted using dotted rectangle). Each of them are are presented using similar pattern. Section of the webpage corresponding to the Search Result Node is highlighted using solid rectangle

Algorithm *findSearchResults*
Input: $Node$: a Node in the DOM Tree
Input: q: Search Terms
Output: $searchResults$: Search Result Items

1. $DescendentList \leftarrow$Nodes in the subtree of $Node$
2. $pNodes \leftarrow \emptyset$
3. $pFeatures \leftarrow \emptyset$
4. **for** $i \leftarrow 1$ **to** $DescendentList.Length$
5. **do** $DescNode \leftarrow DescendentList.Get(i)$
6. **if** $isPatternNode(DescNode)$
7. **then** add $DescNode$ to $pNodes$
8. add $DescNode.Pattern$ to $pFeatures$
9. $MaxWeight \leftarrow 0$
10. $srNode \leftarrow NULL$
11. $searchResults \leftarrow \emptyset$
12. **for** $i \leftarrow 1$ **to** $pFeatures.Length$
13. **do** $Pattern \leftarrow pFeatures.Get(i)$
14. $pNode_1 \leftarrow pNodes.Get(i)$
15. $Weight \leftarrow pFeatures.Freq(Pattern)$
16. $Words \leftarrow pNode_1.Words$
17. $Similarity \leftarrow Sim(Words, q)$
18. $Weight \leftarrow Weight + Weight * Similarity$
19. **if** $Weight > 0$
20. **then** $j \leftarrow$index of next occurrence of $Pattern$ in $pFeatures$
21. $pNode_2 \leftarrow pNodes.Get(j)$
22. **if** $Weight > MaxWeight$
23. **then** $MaxWeight \leftarrow Weight$
24. $srNode \leftarrow comAnces(pNode_1, pNode_2)$
25. $searchResults \leftarrow findItems(srNode, Pattern, pNodes)$

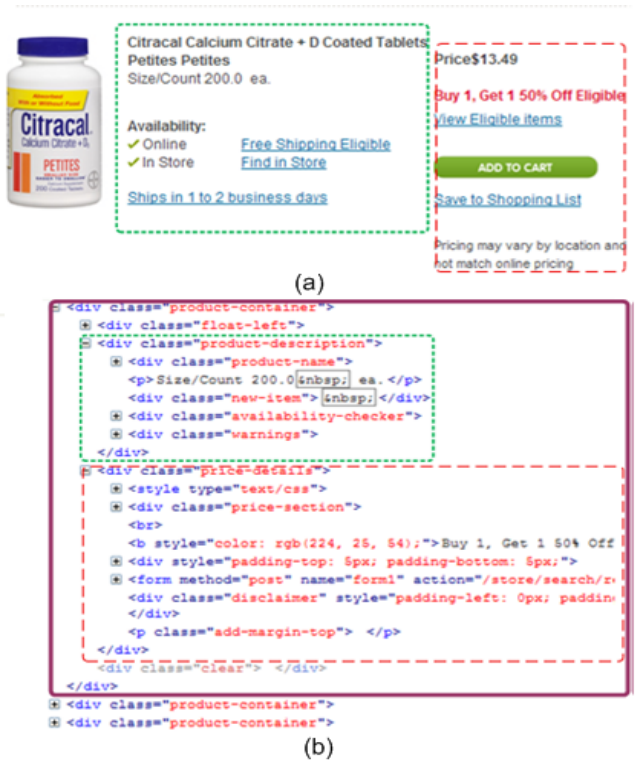

Figure 3. (a) A Search Result Item. (b) DOM tree node (highlighted using solid rectangle) for the search result item. Childnodes of this search result item with similar structure are highlighted using green and red dotted rectangle. This DOM tree node is a pattern node since its child nodes have similar tag distributions. Each of the search result items in Figure 2 are identified as pattern node, and their common ancestor becomes search result node containing search result items.

26. **return** $searchResults$

Option Detection from Search Result

The input to this algorithm is a search result item, and output is an option. To do that, it identifies all the links in the search result item, and assigns a score to each of them depending on the similarity between the text of that link and the search terms. The link with the highest similarity is identified as the option to present to the user. If no link is similar (similarity score is zero), then it returns the first link (in the DOM order) as an option.

Empirical Evaluation of Search Automation

The key algorithms of search automation are search form detection and search results detection. Here we describe the experiment we conducted to evaluate the performance of these two algorithms.

Experimental Setup

We used 58 websites for our experiment. The websites were selected from surveys we conducted both in our department and outside. The surveys asked people to send us the name of the websites they visit frequently for information retrieval tasks. We visited those websites and saved the homepage of each of them. We identified the homepages which contained a search form. There were 48 websites in that category. For those websites, we conducted a search (2 searches per website) and saved the search result pages. In total, we collected 96 search result pages.

Performance of Search Form Detection

For this experiment, we used all of the homepages as input to the search form detection algorithm and verified the output. We computed how many search forms were identified correctly, how many were identified incorrectly and how many were not identified. From this computation, we measured the recall/precision of the search form detection algorithm. We found that search forms were correctly identified for 35 of those pages, which yields a recall value of 73% (35/48). The algorithm identified the wrong form as the search form for 5 websites, which results in 87.5% (35/40) precision.

One of the reasons for such incorrect identification is the presence of multiple search forms in a webpage, where only one of them is the correct search form with which an item should be searched. For example, in the Walmart website (http://www.walmart.com), there are two search forms, one for product search and the other one for store locations search. In addition, there are websites which have general web search forms in addition to a search form for internal search. However, the current algorithm cannot identify the correct search form for such cases. In the future, we need to develop a more sophisticated algorithm to deal with those cases.

Recall performance of search form detection may be improved by adding more keywords to our knowledge-base. However, that may drop precision. We plan to conduct a large study across more websites to collect the keywords which typically appear in search forms.

Performance of Search Results Detection

For this experiment, we used 96 webpages with search results as input to our search results detection algorithm and verified the output. We found that search results were correctly identified for 67 of those pages, which yields a recall value of 70%. For 18 of those 29 pages where the algorithm didn't identify the correct search results, it identified incorrect search results. This results in 67/(67 + 18) or 79% precision.

We analyzed the webpages where our algorithm was not able to detect search results or detected wrong results and summarized the reasons for those failures. The main reasons are:

- Our algorithm detects search results when there are at least two search results, and fails otherwise. The algorithm needs further improvement to deal with single search result case.
- While detecting pattern nodes, our algorithm uses some thresholds which may affect the accuracy of pattern node detection and that affects search results detection.
- Our algorithm doesn't work for very shallow search results without much structure in them. However it was a design choice. In future, we will develop algorithm to

identify all types of search results (with or without structure).

INFORMATION EXTRACTION

This section discusses our approach to extract the property of an item from a detailed web page (e.g., the page which contains information of properties of an item). This is an application of information extraction from webpages which is a well researched topic and there are many papers in this topic, such as [8, 12, 1]. Most of the prior art uses analysis algorithms and heuristics on DOM trees to extract certain information. However, they were developed for specific applications, e.g., reproducing the web page content for viewing on a mobile device [12], structured data extraction [1]. None of the existing techniques were readily applicable for our problem, which is to find the property of an item. As a result, we have developed our own heuristics to solve this problem.

Given a item detail web page, we do a number of pre-processing steps before we search for the desired property of the item in that page. The pre-processing steps are:

- We remove advertisements from the page using the approach outlined in [8].
- Next, we remove the text contained within the *option* tag because these are usually a drop-down menu. We also remove all script tags and the text contained within them.

After the above pre-processing is done, we use the algorithm *findProperty* to find the property of the item.

Algorithm *findProperty*
Input: *item* : item
Input: *property* : property
Input: *webpage* : the webpage to be searched
Output: *top3Result*: a list containing the top three pieces of information on the webpage that matched the property to be extracted, otherwise it returns NULL

1. $HNodes \leftarrow findHNodes(webpage, item)$
2. **if** $HNodes \neq NULL$
3. **then** $LNodes \leftarrow getLNodes(Siblinigs(HNodes))$
4. **else** $LNodes \leftarrow getLNodes(webpage)$
5. $top3Result \leftarrow searchLNodes(LNodes, property)$
6. **if** $top3Result = NULL$
7. **then** $synSet \leftarrow getSynset(property)$
8. **for** $i \leftarrow 1$ **to** $synSet.Length$
9. **do** $top3Result \leftarrow searchLNodes(LNodes, synSet.Get(\text{i}))$
10. **if** $top3Result \neq NULL$
11. **then return** $top3Result$
12. **return** $top3Result$

Line 1 of the algorithm retrieves all the header nodes (*h1* or *h2*) which contain the item. We look for the item in these nodes because a webpage which contains a detailed information about the item typically highlights the item by increasing its font size relative to the other text on the page. Thus items typically appear in header nodes. If an item appears in header nodes, then we get all of the leaf nodes from the siblings of such header nodes (line 3). Otherwise, we get all the leaf nodes from the entire web page (line 4).

Next, we use the vector space model to construct an index of the text content of the leaf nodes and then search for the *property* in that index (line 5). We retrieve the top 3 results from the search and return as output. However, if the system does not find the *property*, we use WordNet [17] to retrieve the words that are similar to the property and then search for these words.

This approach is limited in three ways. First, we can only extract information that is in plain text from the page. Second, we can only extract information that is actually present in the HTML source of the webpage. If the information is obtained dynamically from a database, such as in Google Maps, then we cannot extract it. Lastly, we don't deduce the semantic meaning of the property. So if a user wants the author of a book, we search for *author* in the document. If the keyword *author* or its synonyms are not on the page, for example if the page simply contains *book by John Doe*, then we will not be able to identify *John Doe* as the author.

Empirical Evaluation of Information Extraction

For this experiment, we wanted to evaluate how well our algorithm was able to extract the correct information from a webpage, given that the information to be extracted was in plain text. To get the data set, we asked twenty people to send us the top 10 websites they visited in the last couple of days and what they were trying to do on those websites. We removed duplicates and ended up with 9 categories and about 3 websites for each category. The categories were *sports news, technews, general news, research papers, multimedia, product, conferences, non-english language* and *requires authentication*. We dropped the non-english websites and the websites that required authentication. Then for each of the remaining categories, we visited those sites and noted down what would be considered a *item detail page* for the website. For example, for research papers from the ACM website, a item detail page would be a page containing the description of one research paper.

Next, using the information we collected about what users were trying to do on the websites, we identified certain properties we felt would be reflected in textual form on the webpage. For example, for research papers, we had *DOI*, *author* and *abstract* while for products, we had *price*, *model number*, *product features* and *reviews*.

Then for each of the item detail pages, we had a user note down what they would select from the page if asked to extract the associated properties. After this, we let the system extract the top 3 relevant pieces of text on the webpage that were associated with each property. Next, the user was asked to identify which, if any, of the three pieces of text the system extracted, matched what the user had extracted. We assigned the score 0 if none of them matched, 1 if the first one matched, 2 for the second and 3 for the third. If a text the system extracted contained significantly more text in addition to the text extracted by the user, then the system was

considered to have failed to extract the required information.

On analyzing the result, 50% of the requested properties were extracted in the first result slot, 12% in the second slot and 2%, in the third result slot while the sytem failed to extract 36% of the properties requested. Using the user's extracted text as the ground truth, the recall was 0.63 while the mean reciprocal rank was 0.57.

The 36% that the system failed to extract correctly can be classified into four groups. The first group contains extractions in which the system gave much more text than the user wanted. We can improve this by further limiting the amount of text sent back to the user. We can have the system output only one sentence as opposed to all the text contained in a leaf node.

The second group contains extractions in which the system only extracted the text that exactly matched the property but didn't extract more information associated with the property that the user wanted. This was a problem with the way the webpage was organized. For example a table on the webpage might have price as a column header and then have different prices listed in the columns but our algorithm was looking for the lexical price to be *next to* the numerical price. We can add better heuristics to define *next to* when dealing with webpages.

The third group contains extractions in which the property was of a different form on the page, for example the user wanted information about the *contributors* to an article but the web page only had information about people that *contributed* to the article. We can improve this by lemmatizing the index and the search term.

The last group contains extractions in which the search term occurred multiple times on the page and the one the user wanted was not in the top 3, extracted by the system. To fix this problem, we need to come up with better heuristics to rank such multiple occurences.

USER EVALUATION

We conducted a user study to collect feedback on our overall approach. Specifically, we wanted to test the following hypotheses:

- H1. Users have complex information retrieval needs.
- H2. The item-property model is a valid building block for complex information queries.
- H3. Users can express their needs as an item-property query.

Experimental Methodology

The user study consisted of a semi-structured interview of six participants. All of the participants were software developers in Argentina, which allowed us to evaluate the application of our approach in the context of a developing country. Only two of the participants owned a smart phone and none of the participants had a data plan subscription. Half of the participants occasionally paid for data minutes for particular operations. One of the participants didn't own a cellphone. All of the participants had seen virtual assistants before. Two of them were frequent users of CoCo [10].

The interviews were conducted via a combination of telephone, instant messaging, and remote shell (SSH). Each of the interviews lasted for 45 minutes, during which the interviewer took notes from the responses and remarks of the participants.

The flow of each interview was as follows:

- We introduced the problem of open web mobile information retrieval to the user and collected demographic information. We asked the user to provide examples of information queries they normally do on the web.
- We presented the item-property model and then asked the user to produce information queries following that model.
- For one of these queries, we asked the user to explain in detail how she would solve it using a computer, a mobile phone, and a human assistant.
- We walked the user through an interaction with our system using the scenario shown in Figure 1. We then asked the user to comment on the interaction experience and results seen.

Results and observations

All of the users mentioned that they perform information queries on the web multiple times per day while on their desktop computers. On the other hand, only half of them performed information queries on their phones, and very rarely. The main reason given is that the browser in their phones is not comfortable enough.

Most of the queries that users expressed involved complex operations such as comparing prices of products in different websites, getting weather conditions for multiple cities while arranging for a fishing trip, or finding the most convenient price for a hotel in a given city. This result confirms hypothesis H1.

After we explained our item-property model, all users were able to formulate queries with its characteristics, and many of these corresponded to portions of the more complex queries they had formulated before. Some users required extra effort to map from their information need to the item-property model and a few of the produced queries did not strictly follow this model. These results support hypothesis H2, and provide evidence for H3.

After watching the demonstration, all users agreed that the output matched what they expected. Most users were surprised to get more than one result back, but appreciated the additional information provided (different types of prices). All users were confused by the third, invalid answer.

Several users identified ways to improve the system. One user indicated he would have preferred the list of options to

include the requested property for each of the listed items; two users commented they would like to be able to express search filters (e.g., used or new, category of room); and one user also suggested that after the result is returned, the agent allows querying for additional properties of the item found.

The users that had used CoCo [10] before indicated that, in comparison, this system represents an improvement. The main argument for this statement was that many times an information need arises in mobile situations, before CoCo has been trained to complete it. Having a mobile IR system such as this one would enable the user to solve these queries without prior training.

When asked if they would use a system with these characteristics, the response was almost unanimously positive. All participants but one would prefer it over using a phone's web browser. Two even mentioned they would like to use it on their desktop PCs. Only two users would not prefer it over asking a friend on the phone (one had no data plan, and the other did not trust the information on the web for her query).

CONCLUSION AND FUTURE WORK

We have presented an algorithm for the problem of mobile information retrieval on the open web. Specifically, we have addressed the problem of item-property retrieval from arbitrary web sites. Our approach combines NLP, search form detection, search result detection and summarization, and information extraction in order to provide a textual dialogue-based interface for finding information on the web. We present empirical evidence that each of the components in our approach works well, and a user evaluation validating the use of such a system for information access in the developing world. There are many possible avenues of future research. First, we will improve our algorithms to support more information retrieval needs that follow the item-property model. For example, we may use similar heuristics for processing similarly organized sites, especially to detect search results. Second, we will explore whether exploiting domain-specific constraints can help our algorithms to deliver a better result. Third, we will address more complex information retrieval tasks (e.g., fetching information from multiple sites, collecting information about multiple items, making comparisons between items) that can use the algorithms presented in this paper as a basic building block. Furthermore, we would like to develop algorithms for doing tasks that require taking actions on the open web, e.g., updating twitter status, paying bills. Fourth, we will integrate this system as an extension to the CoScripter Concierge (CoCo) [10] to leverage the same transport mechanisms that CoCo uses. Finally, we will deploy this system to users, especially mobile users in developing countries and conduct field studies to determine how well our system meets their needs.

ACKNOWLEDGEMENT

We thank Jeffrey Nichols for his insightful comments about this paper, Luis Mariano Guerra for his help in maintaining our server and all of our study participants.

REFERENCES

1. A. Arasu and H. Garcia-Molina. Extracting structured data from web pages. In *SIGMOD '03*, pages 337–348, 2003.
2. C. Baker, C. Fillmore, and J. Lowe. The berkeley framenet project. In *Proceedings of the 17th international conference on Computational linguistics-Volume 1*, pages 86–90, 1998.
3. D. Das, N. Schneider, D. Chen, and N. Smith. Probabilistic frame-semantic parsing. *Proc. of NAACLHLT*, 2010.
4. `http://www.research.ibm.com/deepqa/`.
5. M. Dontcheva, S. M. Drucker, D. Salesin, and M. F. Cohen. Relations, cards, and search templates: user-guided web data integration and layout. In *UIST '07*, pages 61–70, 2007.
6. `http://www.google.com/goog411/`.
7. `http://www.google.com/sms`.
8. S. Gupta, G. Kaiser, D. Neistadt, and P. Grimm. Dom-based content extraction of html documents. In *WWW '03*, pages 207–214, 2003.
9. D. Klein and C. Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 423–430, 2003.
10. T. Lau, J. Cerruti, G. Manzato, M. Bengualid, J. P. Bigham, and J. Nichols. A conversational interface to web automation. In *To Appear at UIST '10*, 2010.
11. J. Mahmud, Y. Borodin, and I. V. Ramakrishnan. Assistive browser for conducting web transactions. In *IUI '08*, pages 365–368, 2008.
12. A. Rahman, H. Alam, and R. Hartono. Content extraction from html documents. In *WDA '01: Proceedings of the 1st Int. Workshop on Web Document Analysis*, 2001.
13. `http://www.siri.com`.
14. `http://www.siteonmobile.com`.
15. Z. Sun, J. Mahmud, S. Mukherjee, and I. V. Ramakrishnan. Model-directed web transactions under constrained modalities. In *WWW '06*, pages 447–456, 2006.
16. W. Wahlster. Smartweb: multimodal web services on the road. In *MULTIMEDIA '07*, pages 16–16, 2007.
17. `http://wordnet.princeton.edu`.
18. M. X. Zhou, K. Houck, S. Pan, J. Shaw, V. Aggarwal, and Z. Wen. Enabling context-sensitive information seeking. In *IUI '06*, pages 116–123, 2006.