Generalizing Web Pages for Deep Web Information Extraction

ABSTRACT

Web sites have scores of useful information hidden behind forms (deep web). Automatic access to information on the deep web is difficult because of its dynamic nature. The information in the deep web holds useful data for the common person. Most approaches to accessing this information stop at simply retrieving the data. On the other hand, efforts which seek to answer questions on the web do not consider much of the deep web.

In our implementation we propose a community-based approach to answering queries using information from the deep web. We record a user process of extracting information based on a question. This yields what we call a query resolution method (QRM), that is, a method of resolving the query. We can generalize QRMs with other compatible, extraction processes that are submitted by users. When a new similar query is posed, one can locate an appropriate QRM to resolve it.

Categories and Subject Descriptors
H.3.3 [Information Search and Retrieval]: Information Search and Retrieval --Query formulation, Retrieval models, Search process, Selection process.

General Terms
Algorithms, Design, Theory,

Keywords
Deep Web, Information Extraction, Composition, Query Resolution, Community

1. INTRODUCTION

Researchers have estimated that the amount of quality information covered by traditional search engines is a thousandth of the actual useful data on the World Wide Web [6]. Much of this data is not indexable by search engines because it is created dynamically and updated frequently, thus any indexed page may quickly become out of date. In order for humans to reach some of this data they must submit information through HTML forms. This content is commonly referred to as the deep or hidden web [6]. Pages that are part of the deep web are teeming with useful information. Thus, producing methods to easily obtain this information is an important endeavor.

Several researchers have published methods of systematically obtaining and using this information [1,9,12,16,20]. However, few solutions seek to natively use the temporal information produced by deep web pages to respond to user queries directly. One reason researchers do not use this information is the acknowledgement that one cannot form a single monolithic schema for all content on the hidden web. Our approach hinges on the idea that one can formalize common occurrences of query types and, in addition, leverage human resourcefulness to identify those parts of the deep web that are useful to answering such queries.

Tim Berners-Lee discusses link data [7], also known as the semantic web. Linked data is a standardized method of associating semantic annotations in HTML documents and other online data so machines are able to semantically describe a document's entities. The goal of the semantic web is to join data across multiple web sites and treat the web like a large database [23]. In order to provide a general benefit to users, web publishers and developers must conform to the standards described by Berners-Lee [21].

Currently, according to DBpedia, there are 4.5 billion resource description framework (RDF) triples with 180 millions links on the semantic web. While this is impressive, it is only a small percentage of the useful data on the web. The requirement for web publishers and developers to annotate their data impedes the progress of semantic web adoption. If a user wants to combine data for her own purposes, she must manually extract information offline into a data store and then process that data. If such information must be obtained from a deep web source, then this task becomes difficult because extra work must be performed to extract data from behind forms.

The XXXX 3.0 system enables a community of users to efficiently create their own extraction processes and share, exchange, and compose new such processes. Subsequent users may pose queries that are answerable through such extraction processes. The extraction process requires a wrapper for individual and multi-page queries. Each extraction process is constructed based on structured input and output parameters, posed as a question we call a semi-structured query (SSQ). We use a tool called the Query Resolution Recorder (QRR) to assist users in retrieving information from deep web sites.

The QRR is an application that works with a browser plug-in to assist the user in the query answering process. The tool allows a user to visit a series of pages, in particular, deep web pages, to gather the data necessary to answer the SSQ and construct the answer. The user’s interactions are recorded by the QRR and stored, along with the associated SSQ and other relevant data, as a Query Resolution Mechanism (QRM). The user is able to evaluate a QRM with appropriate input arguments, to perform similar extractions.

The rest of the paper is organized as follows. Section 2 presents related work. Sections 3, 4, and 5 describe the different phases of our system – constructing QRMs, generalizing QRMs, and answering user queries. Section 6 describes example usage scenarios and Section 7 concludes.

2. RELATED WORK

The semantic web has produced many applications with similar goals. Methods have been created to automatically extract entity

\^1 Applications which seek to draw information for deep web pages have been called Mashups.

\^2 DBpedia refers to a website DBpedia.org a community effort for extracting relationships from Wikipedia.

\^3 The RDF semantics specification is available on the w3 website http://www.w3.org/TR/rdf-mt/.
information from documents [8,10,14,22,24]. RDF is used to describe the semantics of entities that may exist on a webpage; therefore it is possible to query this semantic data across web pages using SPARQL [23]. SPARQL provides a method of expressing SQL-like queries over silos with RDF triples. However, this only works on websites that publish their data in RDF or similar formats.

Raghavan and Garcia-Molina describe a method of crawling hidden web pages using a variety of techniques [14]. Their work allows a crawler to obtain access to multiple instances of deep web resources. Relevant resources are added to a global index allowing users of the search engine to see instances of dynamic deep web sites. They also use a method to automatically find meaningful information in the response pages. Their focus is on indexing deep web sources and their solution is limited to a keyword search.

Thresher [16] implemented a browser plug-in allowing users to assign semantic information to web pages by extracting patterns from the same web pages. The current implementation does not allow users to query the semantic information over non-annotated deep websites.

Piggy Bank [11] allows one to annotate semantics from web pages and add them to a piggy bank published on the web. This allows subsequent users the ability to use the piggy bank store. The data in the piggy bank is static, having been defined by a single annotation.

The START project [5] intelligently parses natural language queries and answers questions based on wrapped sources of information stored in their knowledge base. Additionally, START focuses on factual information instead of spatiotemporal information such as events.

System T [18] is a declarative information extraction system developed at IBM Almaden. This system has a SQL like syntax allowing the user to express what they would like to extract and applying the extractor over several instances of web pages. Query performance is improved using a text extraction algebra and optimization techniques. The authors mention nothing about processing extractions from the deep web.

The Purple SOX system [8] allows users to declaratively specify a path of operators for information extraction. Purple SOX extracts entities from input files, which may be static snapshots of HTML pages. Their system uses a learning process in entity extractors to retrieve content from each page. Finally, Purple SOX allows users to give feedback about the content being extracted.

Our system differentiates itself by combining three features. First, users are allowed to create the actual extractors by example rather than relying on externally created scripts. Second, users are able to query deep web sources and obtain real-time content. Finally, user feedback improves the quality of results.

3. **QRM CONSTRUCTION**

Weld et al. [29] explain four methods to create wrappers in order to achieve the high precision extraction we require. First, humans may write extractions based on rules [14]. Second humans may label training data for supervised learning extractors. Third, humans may validate candidate extractions, regardless of how they are produced. Finally, humans may manually aggregate examples of the structured information. We require precise extractions over many pages so we use a mixture of the first and second methods to produce extractors. Instead of writing the extraction rules, as in method one, the user creates them by example using the QRR. We also use user created labels to better understand the data extracted.

Similar to the method proposed by Muezzinoglu et al. [22], our QRR asks the user to define a question she will subsequently answer. The user enters both the inputs to the query, posed as a question, as well as the kind of results she expects to receive. This information is represented as an SSQ. The SSQ is used to link the semantic information in the web form with both the inputs and outputs of each form interaction. The SSQ is similar to the model used in Query-by-Example [30]. We formally define a SSQ as follows:

**Definition 1.** A fully expressed SSQ is a 5-tuple $S = (\Omega, R, I, C, O)$ such that:

1. $\Omega$ is a free text, natural language query posed in the form of a question.
2. $R$ is an ontological realm which may be thought of as a category or concept.
3. $I$ is the set of inputs $I_1, I_2, I_3, I_4, I_5$ where $I_1$ is the class or class instance associated with the context who, $I_2$ is the class or class instance associated with the context what, $I_3, I_4$, and $I_5$ are associated with the contexts when, where, and how respectively.
4. $C$ is a set of context classes associated with the $I$.
5. $O$ is the set of outputs $O_1, O_2, O_3, O_4, O_5$ where $O_1$ is the class associated with the context who, $O_2$ is the class associated with the context what, $O_3, O_4$, and $O_5$ are associated with the contexts when, where, and how respectively.
6. The users supplies $\Omega$ and maps the terms in $\Omega$ to both $I$ and $O$. The user also labels inputs with classes in $C$. $R$ is inferred from the terms in $\Omega$.

![Figure 1 - The Screen for entering a SSQ](image)

Consider the example shown in Figure 1. The natural language query, or $\Omega$, is *Where is the Rolling Stones next concert?*. An appropriate ontological realm, $R$, is Musical Event. The inputs who, when, and what are *Rolling Stones, Next Date, and Concert.*
Classes corresponding to the inputs are Band, Date, and Performance. The output class Venue is associated with the context where.

During the answer retrieval process, a user discovers an element of the query's result on a web page. Next, she highlights that element, typically a text string. This highlight allows the QRR to record the source of the queries answer. The QRR stores all information from the user query in our data store. When the user submits web forms she may associate each of the form’s inputs with one of our stored context classes.

The discovery process can be broken down into three cases:

1) The user typed in a URL and highlighted part of or all the answer.

2) The user typed in a URL, clicked a link, and found part or all of the answer.

3) The user typed in a URL, filled out a form, and found part or all of the answer on a subsequent page.

Any discovery process consists of a sequence of the above transactions. Additionally, data collected for inputs and outputs may be found during any stage of the complete discovery process. An output collected during one stage of an extraction may be the input to a subsequent stage in the extraction.

We store the location and content of each URL visited. For each element selected we store both the URL and xpath string to the element containing the start and end of the highlight location. Additionally, we store a span, that is, the character offset from the nearest opening document object model (DOM) node of the start highlight and the character offset from the enclosing DOM node of the end highlight. The term span is also used in SystemT [18].

When all the information from the completed QRR is obtained and the SSQ has been wrapped, a QRM can be generated. This QRM extractor may be evaluated to retrieve data specified by its input arguments. The arguments provided to the extractor are used, as default values, for the parameters identified during creation by the QRR. We formally define a QRM as follows:

**Definition 2.** A QRM is a 5-tuple $Z = (I_C, O, P, M, R, A)$ such that:

1) $I_C$ is the sequence of input classes $I_{C1}, I_{C2}, I_{C3}, I_{C4}, I_{C5}$ where $I_{C1}$ represents the input class associated with the context who, $I_{C2}$ represents the input class associated with the context what, $I_{C3}, I_{C4}, I_{C5}$ represents the input class associated with the context when, where, and how respectively.

2) $O$ is the set of outputs as defined in the SSQ above.

3) $P$ is an ordered list of web pages $P = (P_1, P_2, ..., P_n)$ where $P_h = (U_h, (I_{h1}, I_{h2}, ..., I_{hn}), (O_{h1}, O_{h2}, ..., O_{hn}))$ for $1 \leq h \leq n$, $u = |I_h|$, $v = |O_h|$. $P_h$ is a triple comprising a URL $U_h$ a sequence of input arguments, and sequence of output results.

Let $I_b = U_{j=1}^b I_{ij}$ be the set of the selector expressions for the inputs of a query and $O_b = U_{j=1}^b O_{ij}$ be the set of selectors for the outputs of a query.

Let $K$ be the set of all string constants.

Then $U_1 \in K$ and for all $1 < g < |I_1|, I_{1g} \in K \cup I_b \cup O_b$, and for $n > 1$ and $m < n$, $U_2 \in K \cup (U_{m=1}^{n-1} O_{mn})$ where $1 \leq h \leq |O_m|$, and for all $1 < g < |I_n|, I_{ng} \in K \cup I_b \cup O_b \cup O_{mh}$.

(4) $M$ is a map between $I_i, K, O$ to page list inputs $I_h$.

(5) $R$ is an ontological realm.

(6) $A$ is the sequence of outputs from $I_{C3}, K$, and $O$ representing the answer.

![Figure 2 – The process for query generation](image)

4. Generalization

Generalization is the process of combining multiple similar QRMs, producing a single QRM with a broader range of acceptable inputs and outputs. Similar QRMs are those whose SSQs have matching input and output classes and equivalent query realms.

Each QRM, $Q_i$, can be considered to be a function mapping input SSQs to answers, i.e. $Q_i : S \rightarrow A$. We consider a QRM to be characterized by a set of SSQ and answer pairs:

$Q_i \equiv \{(s, a)| Q_i(s) = a\}$

A QRM $Q$ generalizes $Q_1, Q_2, ..., Q_n$ if:

$Q \equiv \bigcup_{i=1}^n Q_i$

that is, if the input-to-output mappings of each $Q_i$ are contained within $Q$.

Given two QRMs $Q_1$ and $Q_2$, with page references $P_{11}, P_{12}, ..., P_{1n}$ and $P_{21}, P_{22}, ..., P_{2m}$ such that $n \leq m$, we choose $Q_1$ as a frame of reference for page generalization because it requires fewer page visitations to execute. Pages in $Q_1$ and $Q_2$ are paired and generalized based on URL. The generalized pages are placed into $Q_3$ along with any unpaired pages from $Q_1$. 
Figure 3 – Page generalization

Given two paired web pages, the xpaths to the outputs on each page are the elements being generalized. These outputs may be highlighted text or a dynamically created link which is then used to navigate to a subsequent page. Between two visitations to the same page, the content layout may change. Thus, it is necessary to have a general extraction path to the target. This process requires additional user input or computation to determine which specific item will be the correct one for the current query. This process is repeated for all matching pages; any pages in the smaller QRM which do not have corresponding pages in the larger query are generalized with themselves.

Typically, the response pages created by querying web forms, dynamically created and employ a templated structure. Such web pages are highly susceptible to structural changes [25,26]. Generalizing the extraction paths overcomes small structural changes in visited web pages. This per-page generalization is performed using the GenPath algorithm of Badica et al. [3].

When a user highlights a portion of an HTML page, submits a form, or follows a link we convert the xpath(s) of the appropriate DOM node(s) to an extraction path. The extraction path is a richer representation of the xpath allowing a mapping function Epath-to-XPath() to produce a contextualized xpath. A contextualized xpath is a query of the DOM nodes that identifies the context of the object node to be extracted. Aside from allowing the extractor to withstand small structural changes to the affected web page, this method is also resistant to random HTML injections designed to inhibit screen scrapers, such as those described by Bhagwan et al. [27].

5. QUERY ANSWERING

In order to answer a query in an automated fashion, we must either find a single QRM that fits the user’s question or create a new QRM by composing several QRMs. The problem of chaining QRMs is difficult because it requires users to explore the permutations of existing QRMs. Thus, it is necessary to prune our set of QRMs before attempting to compose a new QRM. We perform Ontological Pruning.

Ontological Pruning lets us consider only those QRMs that are semantically related to the given SSQ. For example, if the query realm is Musical Events, we will not consider QRMs that have the realm Sporting Events.

We ask the user to associate a realm with the SSQ and if there is no appropriate realm, she may create one. Thus, real association requires no extra effort on the part of the system. We are investigating several machine-learning approaches to identifying the appropriate classification for SSQ input terms. Our strategy is to extend ontology-matching methods [17, 27] that operate at the class level. This strategy allows us to perform matching between tuples of instances to tuples of classes, and then to rank QRMs.

In order to produce a result we consider three steps. First, if the signature of the SSQ matches the signature of an existing QRM, we have our result. Second, if no matching SSQ exists we perform a QRM construction algorithm. The query construct5ion algorithm is a variation of the bidirectional A* algorithm. We attempt to create a path of QRMS in order to satisfy the input and output requirements of the SSQ. More information on this algorithm is similar to the algorithm in TransformScout [12]. The third step is to ask the assistance of the user. We ask the user to manually create the new QRM.

We show the user how we attempted to answer a query, by showing the query plan (sequence of QRMs) we constructed and the scores in ranking and similarity. A user may review other answering options if she does not believe the result provided is correct.

Figure 4 – Query answering process

6. APPLICATION

Our system is a useful application to both professionals in vertical markets and casual answer seekers.

A local auto shop employee may be interested in discovering the best price for a part the customer requires. This company may simply create an extractor for the products of local auto parts stores. By running the query she is able to find the location of the part that is most inexpensive. This could subsequently help business relations between stores improve.

We have been contacted by a student organization at our university who wishes to obtain a large collection of professor evaluation results from a university website. Once the extraction process of a single professor and her evaluation results is obtained, the resulting QRMs can be executed for additional professors.

The sports fan regularly asks questions such as What hotel is closest to the next Los Angeles Lakers basketball game? To answer this query the user must find the location of the next Lakers game. Using this information, she can determine the hotel closest to basketball arena. When posed to our system for the first time, this query is answered by a composition of QRMs. The name of the team is a variable input parameter. This provides for the ability to search for the same information with a different team, e.g., the Miami Heat. Also modifying the sport type input parameter allows for a more diverse search, e.g. Los Angeles Dodgers baseball game or Florida Gators football game.

7. CONCLUSION

This paper proposes a method for wrapping the deep web information extraction process and using the extracted process to
answer complicated questions. The system allows a community of users to submit independently answered queries. It uses the extraction process of answered queries to solve new queries. It sorts and ranks the QRM provided to the user. Additionally, we use generalization techniques to ensure we have robust extraction methods.

We are presently looking into several issues to improve the querying process, scraping process, and the execution of QRMs. We are running tests to determine how a user naturally categorizes their queries into who, what, when, where, and how context. This will provide intuition about how users conceptualize search queries. In addition, this research will assist us in developing a natural language processor to translate a user free text query into our SSQ representation.

During the extraction process, we reach many HTML lists and tables. Extracting information from the web is difficult because there is no formalized order of displaying information. These structures contain important, ordered information. We are researching the best methods to dynamically extract correct answers from these structures. For example, the context how may contain a qualifier average. In this case, we may want to retrieve the median element in the list. On the other hand, if the qualifier is cheapest, we may want to retrieve the first element.

8. REFERENCES


