DOI:10.1145/1897816.1897839

Google's WebTables and Deep Web Crawler identify and deliver this otherwise inaccessible resource directly to end users.

BY MICHAEL J. CAFARELLA, ALON HALEVY, AND JAYANT MADHAVAN

Structured Data on the Web

THOUGH THE WEB is best known as a vast repository of shared documents, it also contains a significant amount of structured data covering a complete range of topics, from product to financial, public-record, scientific, hobby-related, and government. Structured data on the Web shares many similarities with the kind of data traditionally managed by commercial database systems but also reflects some unusual characteristics of its own; for example, it is embedded in textual Web pages and must be extracted prior to use; there is no centralized data design as there is in a traditional database; and, unlike traditional databases that focus on a single domain, it covers everything. Existing data-management systems do not address these challenges and assume their data is modeled within a well-defined domain.

This article discusses the nature of Web-embedded structured data and the challenges of managing it. To begin, we present two relevant research projects developed at Google over the past five years. The first, WebTables, compiles a huge collection of databases by crawling the Web to find small relational databases expressed using the HTML table tag. By performing data mining on the resulting extracted information, WebTables is able to introduce new data-centric applications (such as schema completion and synonym finding). The second, the Google Deep Web Crawler, attempts to surface information from the Deep Web, referring to data on the Web available only by filling out Web forms, so cannot be crawled by traditional crawlers. We describe how this data is crawled by automatically submitting relevant queries to a vast number of Web forms. The two projects are just the first steps toward exposing and managing structured Web data largely ignored by Web search engines.

Web Data

Structured data on the Web exists in several forms, including HTML tables, HTML lists, and back-end Deep Web databases (such as the books sold on Amazon.com). We estimate in excess of one billion data sets as of February 2011. More than 150 million sources come from a subset of all English-language HTML tables,[4,5] while Elmeleegy et al[11] suggested an equal number from HTML lists, a total that does not account for the non-English Web. Finally, our experience at Google

» key insights

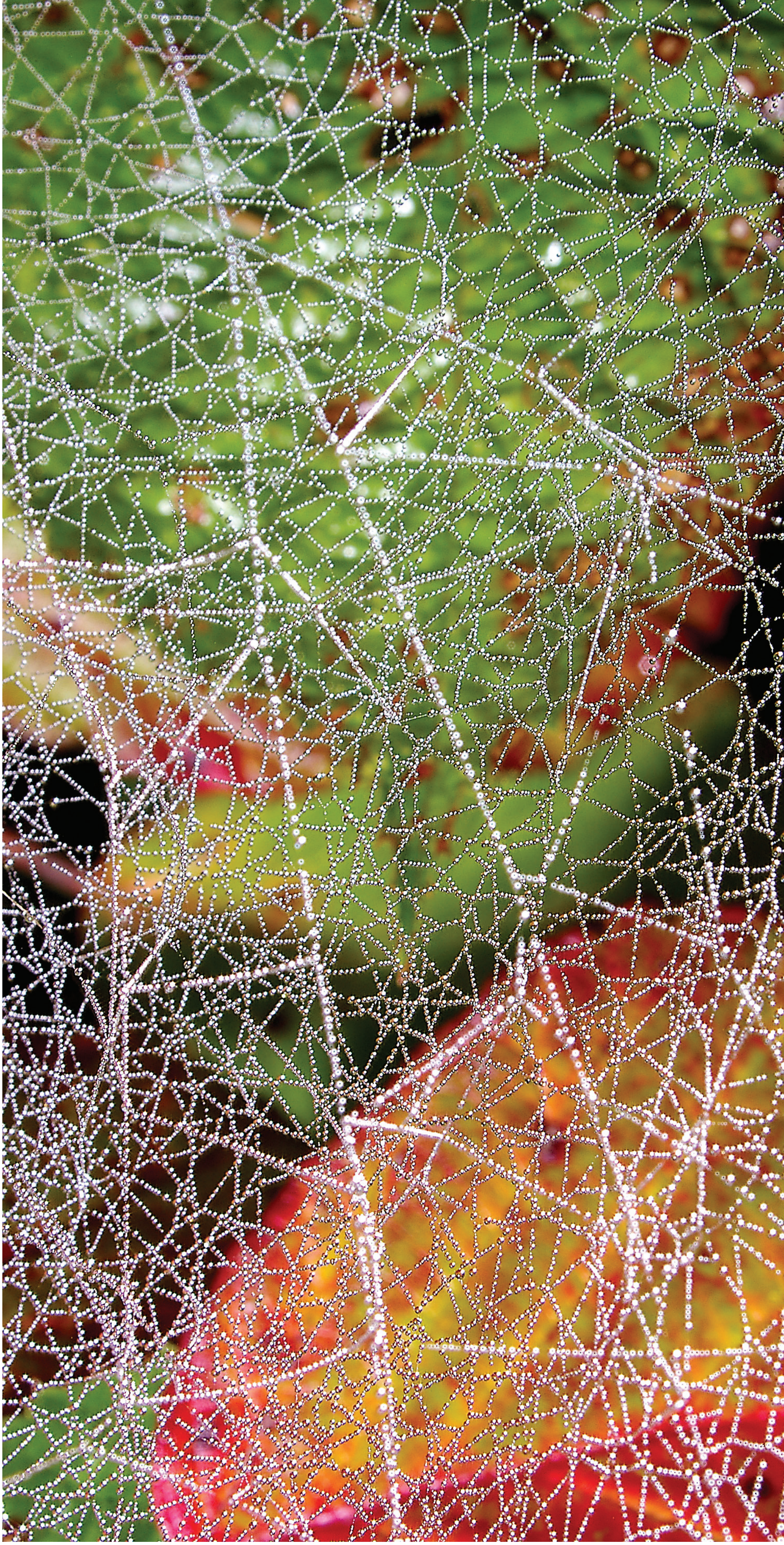
- Because data on the Web is about everything, any approach that attempts to leverage it cannot rely on building a model of the data ahead of time but on domain-independent methods instead.
- The sheer quantity and heterogeneity of structured data on the Web enables new approaches to problems involving data integration from multiple sources.
- While the content of structured data is typically different from what is found in text on the Web, each content collection can be leveraged to better understand other collections.

suggests the Deep Web alone can generate more than one billion pages of valuable structured data. The result is an astounding number of distinct structured data sets, most still waiting to be exposed more effectively to users.

This structured data differs from data stored in traditional relational databases in several ways:

Data in "page context" must be extracted. Consider a database embedded in an HTML table (such as local coffeehouses in Seattle and the U.S. presidents in Figure 1). To the user the data set appears to be structured, but a computer program must be able to automatically distinguish it from, say, a site's navigational bar that also uses an HTML table. Similarly, a Web form that gives access to an interesting Deep Web database, perhaps containing all Starbucks locations in the world, is not that different from a form offering simple mailing-list signup. The computer program might also have to automatically extract schema information in the form of column labels sometimes appearing in the first row of an HTML table but that sometimes do not exist at all. Moreover, the subject of a table may be described in the surrounding text, making it difficult to extract. There is nothing akin to traditional relational metadata that leaves no doubt as to how many tables there are and the relevant schema information for each table.

No centralized data design or data-quality control. In a traditional database, the relational schema provides a topic-specific design that must be observed by all data elements. The database and the schema may also enforce certain quality controls (such as observing type consistency within a column, disallowing empty cells, and constraining data values to a certain legal range). For example, the set of coffeehouses may have a column called `year-founded` containing integers constrained to a relatively small range. Neither data design nor quality control exists for Web data; for



PHOTOGRAPH BY MAX WESTBY

example, if a `year-founded` string is in the first row, there is nothing to prevent the string *macchiatone* from appearing beneath it. Any useful application making use of Web data must also be able to address uncertain data design and quality.

Vast number of topics. A traditional database typically focuses on a particular domain (such as products or proteins) and therefore can be modeled in a coherent schema. On the Web, data covers everything, and is also one of its appeals. The breadth and cultural variations of data on the Web make it inconceivable that any manual effort would be able to create a clean model of all of it.

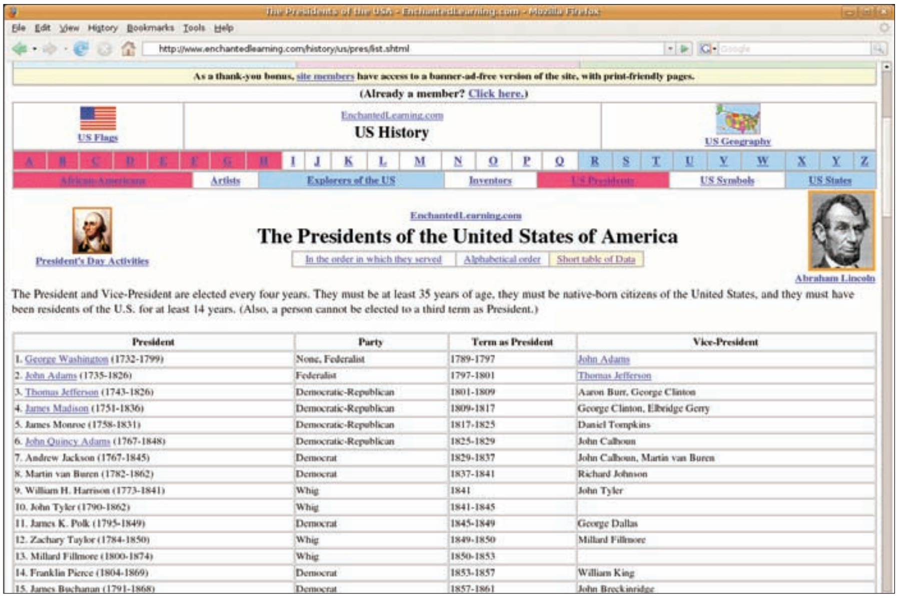


Figure 1. Typical use of the `table` tag to describe relational data that has structure never explicitly declared by the author, including metadata consisting of several typed and labeled columns, but that is obvious to human observers. The navigation bars at the top of the page are also implemented through the table tag but do not contain relational-style data.

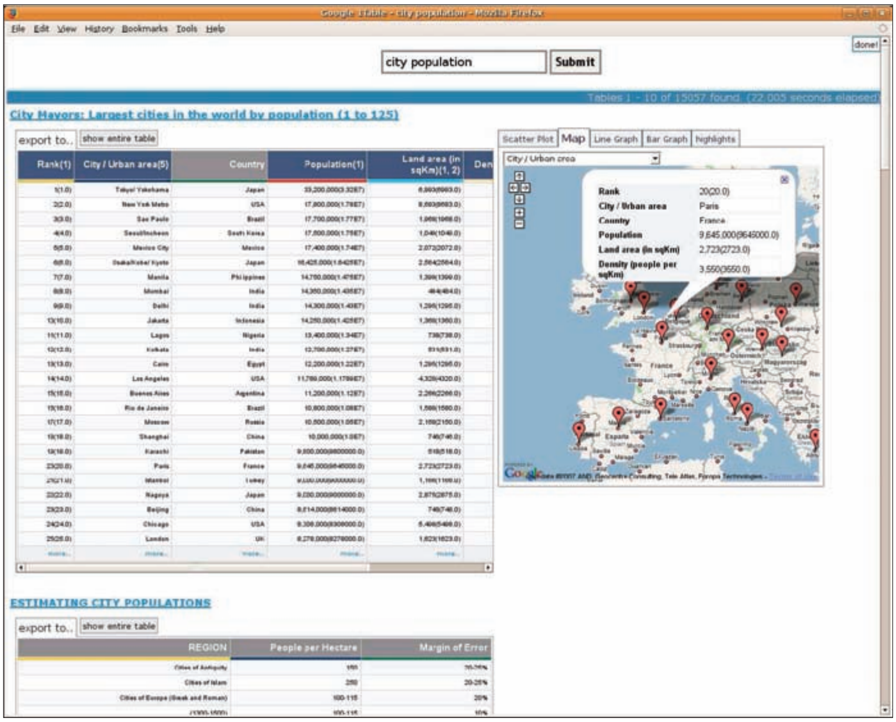


Figure 2. Results of a keyword query search for "city population," returning a relevance-ranked list of databases. The top result contains a row for each of the most populous 125 cities and columns for "City/Urban Area," "Country," "Population," and "rank" (by population among all the cities in the world). The system automatically generated the image at right, showing the result of clicking on the "Paris" row. The title ("City Mayors…") links to the page where the original HTML table is located.

Before addressing the challenges associated with accessing structured data on the Web, it is important to ask what users might do with such data. Our work is inspired by the following example benefits:

Improve Web search. Structured Web data can help improve Web search in a number of ways; for example, Deep Web databases are not generally available to search engines, and, by surfacing this data, a Deep Web exploration tool can expand the scope and quality of the Web-search index. Moreover, the layout structure can be used as a relevance signal to the search ranker; for example, an HTML table-embedded database with a column `calories` and a row `latte`, should be ranked fairly high in response to the user query `latte calories`. Traditionally, search engines use the proximity of terms on a page as a signal of relatedness; in this case, the two terms are highly related, even though they may be distant from each other on the page.

Enable question answering. A long-standing goal for Web search is to return answers in the form of facts; for example, in the `latte calories` query, rather than return a URL a search engine might return an actual numerical value extracted from the HTML table. Web search engines return actual answers for very specific query domains (such as weather and flight conditions), but doing so in a domain-independent way is a much greater challenge.

Enable data integration from multiple Web sources. With all the data sets available on the Web, the idea of combining and integrating them in ad hoc ways is immensely appealing. In a traditional database setting, this task is called data integration; on the Web, combining two disparate data sets is often called a "mashup." While a traditional database administrator might integrate two employee databases with great precision and at great cost, most combinations of Web data should be akin to Web search—relatively imprecise and inexpensive; for example, a user might combine the set of coffeehouses with a database of WiFi hotspots, where speed is more important than flawless accuracy. Unlike most existing mashup tools, we do not want users to be limited to data that has been prepared for integration (such as already available in XML).

The Web is home to many kinds of structured data, including embedded in text, socially created objects, HTML tables, and Deep Web databases. We have developed systems that focus on HTML tables and Deep Web databases. WebTables extracts relational data from crawled HTML tables, thereby creating a collection of structured databases several orders of magnitude larger than any other we know of. The other project surfaces data obtained from the Deep Web, almost all hidden behind Web forms and thus inaccessible. We have also constructed a tool (not discussed here) called Octopus that allows users to extract, clean, and integrate Web-embedded data.[3] Finally, we built a third system, called Google Fusion Tables,[13] a cloud-based service that facilitates creation and publication of structured data on the Web, therefore complementing the two other projects.

Any useful application making use of Web data must also be able to address uncertain data design and quality.

WebTables

The WebTables system[4,5] is designed to extract relational-style data from the Web expressed using the HTML table tag. Figure 1 is a table listing American presidents (http://www.enchantedlearning.com/history/us/pres/list.shtml) with four columns, each with topic-specific label and type (such as **President** and **Term as President**) as a date range; also included is a tuple of data for each row. Although most of the structured-data metadata is implicit, this Web page essentially contains a small relational database anyone can crawl.

Not all table tags carry relational data. Many are used for page layout, calendars, and other nonrelational purposes; for example, in Figure 1, the top of the page contains a `table` tag used to lay out a navigation bar with the letters A–Z. Based on a human-judged sample of raw tables, we estimate up to 200 million true relational databases in English alone on the Web. In general, less than 1% of the content embedded in the HTML table tags represents good tables. Indeed, the relational databases in the WebTables corpus form the largest database corpus we know of, by five orders of decimal magnitude.[a]

WebTables focuses on two main problems surrounding these databases: One, perhaps more obvious, is how to extract them from the Web in the first place, given that 98.9% of tables carry no relational data. Once we address this problem, we can move to the second—what to do with the resulting huge collection of databases.

Table extraction. The WebTables table-extraction process involves two steps: First is an attempt to filter out all the nonrelational tables. Unfortunately, automatically distinguishing a relational table from a nonrelational table can be difficult. To do so, the system uses a combination of handwritten and statistically trained classifiers that use topic-independent features of each table; for example, high-quality data tables often have relatively few empty cells. Another useful feature is whether each column contains a uniform data type (such as all dates or all integers). Google Research has found that finding a column toward the left side of the table with values drawn from the same semantic type (such as country, species, and institution) is a valuable signal for identifying high-quality relational tables.

The second step is to recover metadata for each table passing through the first filter. Metadata is information that describes the data in the database (such as number of columns, types, and names). In the case of the presidents, the metadata contains the column labels `President`, `Party`, and so on. For coffeehouses, it might contain `Name`, `Speciality`, and `Roaster`. Although metadata for a traditional relational database can be complex, the goal for WebTables metadata is modest—determine whether or not the first row of the ta-

a The second-largest collection we know is due to Wang and Hu,[22] who also tried to gather data from Web pages but with a relatively small and focused set of input pages. Other research on table extraction has not focused on large collections.[10,12,23] Our discussion here refers to the number of distinct databases, not the number of tuples. Limaye et al[16] described techniques for mapping entities and columns in tables to an ontology.

ble includes labels for each column. When inspecting tables by hand, we found 70% of good relational-style tables contain such a metadata row. As with relational filtering, we used a set of trained classifiers to automatically determine whether or not the schema row is present.

The two techniques together allowed WebTables to recover 125 million high-quality databases from a large general Web crawl (several billion Web pages). The tables in this corpus contained more than 2.6 million unique "schemas," or unique sets of attribute strings. This enormous data set is a unique resource we explore in the following paragraphs.

Leveraging extracted data. Aggregating data over the extracted WebTables data, we can create new applications previously difficult or impossible through other techniques. One such application is structured data search. Traditional search engines are tuned to return relevant documents, not data sets, so users searching for data are generally ill-served. Using the extracted WebTables data, we implemented a search engine that takes a keyword query and returns a ranked list of databases instead of URLs; Figure 2 is a screenshot of the prototype system. Because WebTables extracted structural information for each object in the search engine's index, the results page can be more interesting than in a standard search engine. Here, the page of search results contains an automatically drawn map reflecting the cities listed in the data set; imagine the system being used by knowledge workers who want to find data to add to a spreadsheet.

In addition to the data in the tables, we found significant value in the collection of the tabular schemata we collected. We created the Attribute Correlation Statistics Database (ACSDb) consisting of simple frequency counts for each unique piece of metadata WebTables extracts; for example, the database of presidents mentioned earlier adds a single count to the four-element set `president`, `party`, `term-as-president`, `vice-president`. By summing individual attribute counts over all entries in the ACSDb, WebTables is able to compute various attribute probabilities, given a randomly chosen database; for example, the probability of seeing the `name` attribute is far higher than seeing the `roaster` attribute.

An important lesson we learned is there is significant value in analyzing collections of metadata on the Web, in addition to the data itself.

WebTables also computes conditional probabilities, so, for example, we learn that `p(roaster | house-blend)` is much higher than `p(roaster | album-title)`. It makes sense that two coffee-related attributes occur together much more often than a combination of a coffee-related attribute and, say, a music-related attribute. Using these probabilities in different ways, we can build interesting new applications, including these two:

Schema autocomplete. The database schema auto-complete application is designed to assist novice database designers. Like the tab-complete feature in word processors, schema autocomplete takes a few sample attributes from the user and suggests additional attributes to complete the table; for example, if a user types `roaster` and `house-blend`, the auto-complete feature might suggest `speciality`, `opening-time` and other attributes to complete the coffeehouse schema. Table 1 lists example outputs from our auto-complete tool, which is also useful in scenarios where users should be encouraged to reuse existing terminologies in their schemas.

The auto-complete algorithm is easily implemented with probabilities from the ACSDb. The algorithm repeatedly emits the attribute from the ACSDb to yield the highest probability, when conditioned on the attributes the user (or algorithm) has already suggested. The algorithm terminates when the attribute yielding the highest probability is below a tunable threshold.

Synonym finding. The WebTables synonym-finding application uses ACSDb probabilities to automatically detect likely attribute synonyms; for example, `phone-number` and `phone-#` are two attribute labels that are semantically equivalent. Synonyms play a key role in data integration. When we merge two databases on the same topic created by different people, we first need to reconcile the different attribute names used in the two databases. Finding these synonyms is generally done by the application designer or drawn automatically from a pre-compiled linguistic resource (such as a thesaurus). However, the task of synonym finding is complicated by the fact that attribute names are often acronyms or word combinations, and their meanings are highly contextual. Unfortunately, manually computing a set of synonyms is burdensome and error-prone.

WebTables uses probabilities from the ACSDb to encode three observations about good synonyms:

- Two synonyms should not appear together in any known schema, as it would be repetitive on the part of the database designer;
- Two synonyms should share common co-attributes; for example, `phone-number` and `phone-#` should both appear along with name and address; and
- The most accurate synonyms are popular in real-world use cases.

WebTables can encode each of these observations in terms of attribute probabilities using ACSDb data. Combining them, we obtain a formula for a synonym-quality score WebTables uses to sort and rank every possible attribute pair; Table 2 lists a series of input domains and the output pairs of the synonym-finding system.

Deep Web Databases

Not all structured data on the Web is published in easily accessible HTML tables. Large volumes of data stored in back-end databases are often made available to Web users only through HTML form interfaces; for example, a large chain of coffeehouses might have a database of store locations that are retrieved by zip code using the HTML form on the company's Web site, and users retrieve data by performing valid form submissions. On the back-end, HTML forms are processed by either posing structured queries over relational databases or sending keyword queries over text databases. The retrieved content is published on Web pages in structured templates, often including HTML tables.

While WebTables-harvested tables are potentially reachable by users posing keyword queries on search engines, the content behind HTML forms was for a long time believed to be beyond the reach of search engines; few hyperlinks point to Web pages resulting from form submissions, and Web crawlers did not have the ability to automatically fill out forms. Hence, the names "Deep," "Hidden," and "Invisible Web" have all been used to refer to the content accessible only through forms. Bergman[2] and He et al[14] have speculated that the data in the Deep Web far exceeds the data indexed by contemporary search engines. We estimate at least 10 million potentially useful distinct forms[18]; our previous work[17] has a more thorough discussion of the Deep Web literature and its relation to the projects described here.

The goal of Google's Deep Web Crawl Project is to make Deep Web content accessible to search-engine users. There are two complementary approaches to offering access to it: create vertical search engines for specific topics (such as coffee, presidents, cars, books, and real estate) and surface Deep Web content. In the first, for each vertical, a designer must create a mediated schema visible to users and create semantic mappings from the Web sources to the mediated schema. However, at Web scale, this approach suffers from several drawbacks:

- A human must spend time and effort building and maintaining each mapping;
- When dealing with thousands of domains, identifying the topic relevant to an arbitrary keyword query is extremely difficult; and
- Data on the Web reflects every topic in existence, and topic boundaries are not always clear.

The Deep Web Crawl project followed the second approach to surface DeepWeb content, pre-computing the most relevant form submissions for all interesting HTML forms. The URLs resulting from these submissions can then be added to the crawl of a search engine and indexed like any other HTML page. This approach leverages the existing search-engine infrastructure, allowing the seamless inclusion of Deep Web pages into Web-search results. The system currently surfaces content for several million Deep Web databases spanning more than 50 languages and several hundred domains, and the surfaced pages contribute results to more than 1,000 Web-search queries per second on Google.com. For example, as of the writing of this article, a search query for `citibank atm 94043` will return in the first position a parameterized URL surfacing

Table 1. Sample output from the schema autocomplete tool. To the left is a user's input attribute; to the right are sample schemas.

Input attribute	Auto-completer output
name	name, size, last-modified, type
instructor	instructor, time, title, days, room, course
elected	elected, party, district, incumbent, status, opponent, description
ab	ab, h, r, bb, so, rbi, avg, lob, hr, pos, batters
sqft	sqft, price, baths, beds, year, type, lot-sqft, days-on-market, stories

Table 2. Sample output from the synonym-finding tool. To the left are the input context attributes; to the right are synonymous pairs generated by the system.

Input context	Synonym-finder outputs
name	e-mail|email, phone|telephone, e-mail address|email address, date|last-modified
instructor	course-title|title, day|days, course|course-#, course-name|course-title
elected	candidate|name, presiding-officer|speaker
ab	k|so, h|hits, avg|ba, name|player
sqft	bath|baths, list|list-price, bed|beds, price|rent

results from a database of ATM locations—a very useful search result that would not have appeared otherwise.

Pre-computing the set of relevant form submissions for any given form is the primary difficulty with surfacing; for example, a field with label `roaster` should not be filled in with value `toyota`. Given the scale of a Deep Web crawl, it is crucial there be no human involvement in the process of pre-computing form submissions. Hence, previous work that either addressed the problem by constructing mediator systems one domain at a time[8,9,21] or needed site-specific wrappers or extractors to extract documents from text databases[1,19] could not be applied.

Surfacing Deep Web content involves two main technical challenges:

▸ Values must be selected for each input in the form; value selection is trivial for select menus but very challenging for text boxes; and

▸ Forms have multiple inputs, and using a simple strategy of enumerating all possible form submissions can be wasteful; for example, the search form on cars.com has five inputs, and a cross product will yield more than 200 million URLs, even though cars.com lists only 650,000 cars for sale.[7]

The full details on how we addressed these challenges are in Madhavan et al.[18] Here, we outline how we approach the two problems:

Selecting input values. A large number of forms have text-box inputs and require valid input values for the retrieval of any data. The system must therefore choose a good set of values to submit in order to surface useful result pages. Interestingly, we found it is not necessary to have a complete understanding of the semantics of the form to determine good candidate text inputs. To understand why, first note that text inputs fall into one of two categories: generic search inputs that accept most keywords and typed text inputs that accept only values in a particular topic area.

For search boxes, the system predicts an initial set of candidate keywords by analyzing text from the form site, using the text to bootstrap an iterative probing process. The system submits the form with candidate keywords; when valid form submissions result, the system extracts more keywords from the resulting pages. This iterative process continues until either there are no new candidate keywords or the system reaches a prespecified target number of results. The set of all candidate keywords can then be pruned, choosing a small number that ensures diversity of the exposed database content. Similar iterative probing approaches have been used to extract text documents from specific databases.[1,6,15,19,20]

For typed text boxes, the system attempts to match the type of the text box against a library of types common across topics (such as U.S. zip codes). Note that probing with values of the wrong type results in invalid submissions or pages with no results. We found even a library of just a few types can cover a significant number of text boxes.

Selecting input combinations. For HTML forms with more than one input, a simple strategy of enumerating the entire cross-product of all possible values for each input will result in a huge number of output URLs. Crawling too many URLs drains the resources of a search engine Web crawler while posing an unreasonable load on Web servers hosting the HTML forms. Choosing a subset of the cross-product that yields results that are nonempty, useful, and distinct is an algorithmic challenge.[18] The system incrementally traverses the search space of all possible subsets of inputs. For a given subset, it tests whether it is informative, or capable of generating URLs with sufficient diversity in their content. As we showed in Madhavan et al,[18] only a small fraction of possible input sets must be tested, and, for each subset, the content of only a sample of generated URLs must be examined. Our algorithm is able to extract large fractions of underlying Deep Web databases without human supervision, using only a small number of form submissions. Furthermore, the number of form submissions the system generates is proportional to the size of the database underlying the form site, rather than the number of inputs and input combinations in the form.

Limitations of surfacing. By creating Web pages, surfacing does not preserve the structure or semantics of the data gathered from the underlying DeepWeb databases. But the loss in semantics is also a lost opportunity for query answering; for example, suppose a user searched for "used ford focus 1993" and a surfaced used-car listing page included Honda Civics, with a 1993 Honda Civic for sale, but also said "has better mileage than the Ford Focus." A traditional search engine would consider such a surfaced Web page a good result, despite not being helpful to the user. We could avoid this situation if the surfaced page had a search-engine-specific annotation that the page was for used-car listings of Honda Civics. One challenge for an automated system is to create a set of structure-aware annotations textual search engines can use effectively.

Next Steps

These two projects represent first steps in retrieving structured data on the Web and making it directly accessible to users. Searching it is not a solved problem; in particular, search over large collections of data is still an area in need of significant research, as well as integration with other Web search. An important lesson we learned is there is significant value in analyzing collections of metadata on the Web, in addition to the data itself.

Specifically, from the collections we have worked with—forms and HTML tables—we have extracted several artifacts:

▸ A collection of forms (input names that appear together and values for select menus associated with input names);

▸ A collection of several million schemata for tables, or sets of column names appearing together; and

▸ A collection of columns, each with values in the same domain (such as city names, zip codes, and car makes).

Semantic services. Generalizing from our synonym finder and schema auto-complete, we build from the schema artifacts a set of semantic services that form a useful infrastructure for many other tasks. An example of such a service is that, given a name of an attribute, return a set of values for its column; such a service can automatically fill out forms in order to surface Deep Web content. A second example is, given an entity, return a set of possible properties—attributes and relationships—that may be associated with it. Such a service would be useful for both information-extraction tasks and query expansion.

Structured data from other sources. Some of the principles of our previous projects are useful for extracting structured data from other growing sources on the Web:

Socially created data sets. These data sets (such as encyclopedia articles, videos, and photographs) are large and interesting and exist mainly in site-specific silos, so integrating them with information extracted from the wider Web would be useful;

Hypertext-based data models. These models, in which page authors use combinations of HTML elements (such as a list of hyperlinks), perform certain data-model tasks (such as indicate that all entities pointed to by the hyperlinks belong to the same set); this category can be considered a generalization of the observation that HTML tables are used to communicate relations; and

Office-style documents. These documents (such as spreadsheets and slide presentations) contain their own structured data, but because they are complicated, extracting information from them can be difficult, though it also means they are a tantalizing target.

Creating and publishing structured data. The projects we've described are reactive in the sense that they try to leverage data already on the Web. In a complementary line of work, we created Google Fusion Tables,[13] a service that aims to facilitate the creation, management, and publication of structured data, enabling users to upload tabular data files, including spreadsheets and CSV, of up to 100MB. The system provides ways to visualize the data—maps, charts, timelines—along with the ability to query by filtering and aggregating the data. Fusion Tables enables users to integrate data from multiple sources by performing joins across tables that may belong to different users. Users can keep the data private, share it with a select set of collaborators, or make it public. When made public, search engines are able to crawl the tables, thereby providing additional incentive to publish data. Fusion Tables also includes a set of social features (such as collaborators conducting detailed discussions of the data at the level of individual rows, columns, and cells). For notable uses of Fusion Tables go to https://sites.google.com/site/fusiontablestalks/stories.

Conclusion

Structured data on the Web involves several technical challenges: difficult to extract, typically disorganized, and often messy. The centralized control enforced by a traditional database system avoids all of them, but centralized control also misses out on the main virtues of Web data—that it can be created by anyone and covers every topic imaginable. We are only starting to see the benefits that might accrue from these virtues. In particular, as illustrated by WebTables synonym finding and schema auto-suggest, we see the results of large-scale data mining of an extracted (and otherwise unobtainable) data set.

It is often argued that only select Web-search companies are able to carry out research of the flavor we've described here. This argument holds mostly for research projects involving access to logs of search queries, but the research described here was made easier by having access to a large Web index and computational infrastructure, and much of it can be conducted at academic institutions as well, in particular when it involves such challenges as extracting the meaning of tables on the Web and finding interesting combinations of such tables. ACSDb is freely available to researchers outside of Google (https://www.eecs.umich.edu/ michjc/acsdb.html); we also expect to make additional data sets available to foster related research.

References

1. Barbosa, L. and Freire, J. Siphoning Hidden-Web data through keyword-based interfaces. In *Proceedings of the Brazilian Symposium on Databases*, 2004 , 309–321.
2. Bergman. M.K. The Deep Web: Surfacing hidden value. *Journal of Electronic Publishing 7*, 1 (2001).
3. Cafarella, M.J., Halevy, A.Y., and Khoussainova, N. Data integration for the relational Web. *Proceedings of the VLDB Endowment 2*, 1 (2009), 1090–1101.
4. Cafarella, M.J., Halevy, A.Y., Wang, D.Z., Wu, E., and Zhang, Y. WebTables: Exploring the power of tables on the Web. *Proceedings of the VLDB Endowment 1*, 1 (Aug. 2008), 538–549.
5. Cafarella, M.J., Halevy, A.Y., Zhang, Y., Wang, D.Z., and Wu, E. Uncovering the relational Web. In *Proceedings of the 11th International Workshop on the Web and Databases* (Vancouver, B.C., June 13, 2008).
6. Callan, J.P. and Connell, M.E. Query-based sampling of text databases. *ACM Transactions on Information Systems 19*, 2 (2001), 97–130.
7. Cars.com (faq); http://siy.cars.com/siy/qsg/faqgeneralinfo.jsp#howmanyads
8. Cazoodle apartment search; http://apartments.cazoodle.com/
9. Chang, K.C.-C., He, B., and Zhang, Z. Toward large-scale integration: Building a metaquerier over databases on the Web. In *Proceedings of the Conference on Innovative Data Systems Research* (Asilomar, CA, Jan. 2005).
10. Chen, H., Tsai, S., and Tsai, J. Mining tables from large-scale html texts. In *Proceedings of the 18th International Conference on Computational Linguistics* (Saarbrucken, Germany, July 31–Aug. 4, 2000), 166–172.
11. Elmeleegy, H., Madhavan, J., and Halevy, A. Harvesting relational tables from lists on the Web. *Proceedings of the VLDB Endowment 2*, 1 (2009), 1078–1089.
12. Gatterbauer, W., Bohunsky, P., Herzog, M., Krüupl, B., and Pollak, B. Towards domain-independent information extraction from Web tables. In *Proceedings of the 16th International World Wide Web Conference* (Banff, Canada, May 8–12, 2007), 71–80.
13. Gonzalez, H., Halevy, A., Jensen, C., Langen, A., Madhavan, J., Shapley, R., Shen, W., and Goldberg-Kidon, J. Google Fusion Tables: Web-centered data management and collaboration. In *Proceedings of the SIGMOD ACM Special Interest Group on Management of Data* (Indianapolis, 2010). ACM Press, New York, 2010, 1061–1066.
14. He, B., Patel, M., Zhang, Z., and Chang, K.C.-C. Accessing the Deep Web. *Commun. ACM 50*, 5 (May 2007), 94–101.
15. Ipeirotis, P.G. and Gravano, L. Distributed search over the Hidden Web: Hierarchical database sampling and selection. In *Proceedings of the 28th International Conference on Very Large Databases* (Hong Kong, Aug. 20–23, 2002), 394–405.
16. Limaye, G., Sarawagi, S., and Chakrabarti, S. Annotating and searching Web tables using entities, types, and relationships. *Proceedings of the VLDB Endowment 3*, 1 (2010), 1338–1347.
17. Madhavan, J., Ko, D., Kot, L., Ganapathy, V., Rasmussen, A., and Halevy, A.Y. Google's Deep Web Crawl. *Proceedings of the VLDB Endowment 1*, 1 (2008), 1241–1252.
18. Madhavan, J., Cohen, S., Dong, X.L., Halevy, A.Y., Jeffery, S.R., Ko, D., and Yu, C. Web-scale data integration: You can afford to pay as you go. In *Proceedings of the Second Conference on Innovative Data Systems Research* (Asilomar, CA, Jan. 7–10, 2007). 342–350.
19. Ntoulas, A., Zerfos, P., and Cho, J. Downloading textual Hidden Web content through keyword queries. In *Proceedings of the Joint Conference on Digital Libraries* (Denver, June 7–11, 2005), 100–109.
20. Raghavan, S. and Garcia-Molina, H. Crawling the Hidden Web. In *Proceedings of the 27th International Conference on Very Large Databases* (Rome, Italy, Sept. 11–14, 2001), 129–138.
21. Trulia; http://www.trulia.com/
22. Wang, Y. and Hu, J. A machine-learning-based approach for table detection on the Web. In *Proceedings of the 11th International World Wide Web Conference* (Honolulu, 2002), 242–250.
23. Zanibbi, R., Blostein, D., and Cordy, J. A survey of table recognition: Models, observations, transformations, and inferences. *International Journal on Document Analysis and Recognition 7*, 1 (2004), 1–16.

Michael J. Cafarella (michjc@umich.edu) is an assistant professor of computer science and engineering at the University of Michigan, Ann Arbor, MI.

Alon Halevy (halevy@google.com) is Head of the Structured Data Management Research Group, Google Research, Mountain View, CA.

Jayant Madhavan (jayant@google.com) a senior software engineer at Google Research, Mountain View, CA.