Web-Scale Extraction of Structured Data

Michael J. Cafarella
University of Washington
mjc@cs.washington.edu

Jayant Madhavan
Google Inc.
jayant@google.com

Alon Halevy
Google Inc.
halevy@google.com

ABSTRACT

A long-standing goal of Web research has been to construct a unified Web knowledge base. Information extraction techniques have shown good results on Web inputs, but even most domain-independent ones are not appropriate for Web-scale operation. In this paper we describe three recent extraction systems that can be operated on the entire Web (two of which come from Google Research). The TEXTRUNNER system focuses on raw natural language text, the WEBTABLES system focuses on HTML-embedded tables, and the deep-web surfacing system focuses on "hidden" databases. The domain, expressiveness, and accuracy of extracted data can depend strongly on its source extractor; we describe differences in the characteristics of data produced by the three extractors. Finally, we discuss a series of unique data applications (some of which have already been prototyped) that are enabled by aggregating extracted Web information.

1. INTRODUCTION

Since the inception of the Web, the holy grail of web information extraction has been to create a knowledge base of all facts represented on the Web. Even if such a knowledge base were imperfect (and it certainly would be), its contents can be used for a variety of purposes, including answering factual queries (possibly performing simple inferences), expansion of keyword queries to improve recall, and assisting more specific extraction efforts. The vast majority of the work on information extraction has focused on more specific tasks, either by limiting the extraction to particular domains (e.g., extracting seminar announcement data from email in an academic department, or extracting corporate intelligence from news articles), or training extractors that apply to specific web sites. As such, these techniques cannot be applied to the Web as a whole.

In this paper we describe three extraction systems that began with the goal of being domain and site independent, and therefore apply to the entire Web. The systems target different kinds of resources on the Web: text, HTML tables and databases behind forms. Each of these systems has extracted a portion of what can some day become the unified Web knowledge base, and teaches us important lessons on the way to that goal. The first project is the TEXTRUNNER system [3], which operates on very large amounts of unstructured text, making very few assumptions about its target data. The second project is the more recent WEBTABLES system [8, 9], which extracts relational tables from HTML structures. The third project is the deep-web crawling project [21], which surfaces contents of backend databases that are accessible via HTML forms. We describe how the data extracted from each of these systems differ in content and the types of techniques that needed to be applied in each case. We also describe how each system contributes to computing a set of entities and relationships that are represented on the Web.

Finally, we look beyond the specific goal of creating a knowledge base of the Web and consider what kind of *semantic services* can be created in order to assist with a wide collection of tasks. We argue that by *aggregating* vast amounts of structured data on the Web we can create valuable services such as synonym finding, schema auto-complete and type prediction. These services are complimentary to creating a knowledge base of the Web and can be created even if the knowledge base is still under construction. We show early examples of such services that we created in our recent work.

2. TEXTUAL EXTRACTION

The first extraction system that we consider operates over very large amounts of unstructured text. Banko et al.'s TEXTRUNNER consumes text from a Web crawl and emits n-ary tuples [3]. It works by first linguistically parsing each natural language sentence in a crawl, then using the results to obtain several candidate tuple extractions. For example, TEXTRUNNER might process the sentence "Albert Einstein was born in 1879", find two noun phrases and a linking verb phrase, then create the tuple ⟨`Einstein, 1879`⟩ in the `was_born_in` relation. Finally, TEXTRUNNER applies extraction-frequency-based techniques to determine whether the extraction is accurate. Table 1 lists a few example TEXTRUNNER extractions.

There are a number of extractors that emit tuples from raw text inputs, including DIPRE, SNOWBALL, and KNOWITALL [1, 7, 14]. However, TEXTRUNNER has two additional unusual qualities that make it es-

Object 1	Relation	Object 2
einstein	discovered	relativity
1848	was_year_of	revolution
edison	invented	phonograph
einstein	died_in	1955

Table 1: A few example binary tuple extractions from TEXTRUNNER.

pecially apt for creating a Web-scale knowledge base. First, TEXTRUNNER was designed to operate in *batch* mode, consuming an entire crawl at once and emitting a large amount of data. In contrast, other systems have been on-demand query-driven systems that choose pages based on the user's revealed interest. The on-demand approach may seem more appealing because it promises to only perform relevant work, and because the system may be able to use the query to improve the output quality. However, the batch-oriented technique allows us to pre-compute good extractions before any queries arrive, and then index these extractions aggressively. In the query-driven approach, all of this work (possibly including downloading the text itself) must happen at query-time. Also, note that traditional search engines follow a similar batch-processing approach.

Second, TEXTRUNNER does not attempt to populate a given target relation or schema, but rather discovers them during processing. KNOWITALL, DIPRE, and SNOWBALL all require some sort of target guidance from the user (e.g., a set of seed pairs or a set of extraction phrases). This kind of *open information extraction* is necessary if we want to extract the Web's worth of data without a strong model of Web contents (in the form of either a query load or perhaps some kind of general ontology). Further, this approach allows the extractor to automatically obtain brand-new relations as they appear over time.

This system has been tested in two large deployments. The initial TEXTRUNNER project ran on a general corpus of 9 million Web pages, extracting 1 million concrete tuples (of which 88% were accurate) [3]. In a more recent experiment, TEXTRUNNER was run on a much larger 500M page crawl, yielding more than 200M tuples that occurred at least two times (though detailed accuracy figures are not yet available) [2]. H-CRF is an extension to the TEXTRUNNER system that uses an approach based on conditional random fields to improve output quality [4].

TEXTRUNNER's open IE technique differs substantially from previous work. For example, the DIPRE and SNOWBALL projects accept a set of binary seed tuples, learn patterns that textually connect the two elements, then extend the set by downloading relevant Web pages and applying the learned patterns. The KNOWITALL system extracts binary *is-a* relations (and in later work, other *n-ary* relations) from downloaded Web pages [14]. It worked by applying a handful of generic extraction phrases (due to Hearst [17]) like "X such as Y" and "X, including Y,..." to the pages, then distinguishing true from spurious extractions with a trained classifier that took as input frequency-statistics of the extraction phrases.

Several other systems occupy an interesting middle ground between raw text and structured data. The Yago system produced ontologies consisting of hypernyms and binary relations, extracted from WordNet and the structured parts of Wikipedia (such as the specialized "list pages" that put other Wikipedia entities into broad sets) [24]. It did not extract information from the Wikipedia article content. The Kylin system used Wikipedia infoboxes (essentially, small lists of attribute/value pairs associated with a single entity) to train textual extractors that were then applied to the article text [27]. The primary goal of the system was to make the infobox data more complete, especially for versions of Wikipedia in unpopular languages.

The data that TEXTRUNNER and other text-centric extractors emit differs in several ways from traditional relational-style data. First, the cardinality of text-derived tuples is heavily reliant on extractable natural language constructs: binary relations (generally two nouns and a linking relation) are much easier to extract and are more common than 3-ary or larger tuples. In contrast, relational tables make it easy to express data with many dimensions.

Second, the domains that appear in text-derived data will be different from data found in relations. In part, this difference is due to the above point about binary tuples being more "natural" than other tuple sizes in natural language text. But this difference in domains is also due to issues of human reading and writing style: it would be very boring to read a piece of text that exhaustively enumerates movie times, but scanning a table is quite easy.

Third, unlike the WEBTABLES extractor we describe next, where each extraction is based on a *single* occurrence of a table on the Web, the TEXTRUNNER (as well as the KNOWITALL) extractor outputs extractions that are found in *multiple* locations on the Web. This is necessary since the extractors that TEXTRUNNER relies on are linguistic extractors that are quite fallible. In contrast, the tables extracted by WEBTABLES are "cleaner" after extraction than text-based tuples. Further, because an extracted relation is a fairly complicated object consisting of multiple tuples, it is harder to find exact replicas elsewhere on the Web and so frequency-based techniques are not obviously applicable.

Finally, the output data model from TEXTRUNNER differs slightly from the relational model. The individual dimensions of a TEXTRUNNER extraction are generally not labeled (e.g., we do not know that `Einstein` is in the `human` attribute, and that `1879` is in `birthyear`). But most relations (and extractions from the "hard-coded" text systems) do have this kind of attribute metadata. The situation is reversed in the case of the name of the relation. The `was_born_in` extraction serves as a rough name for a relation in which the

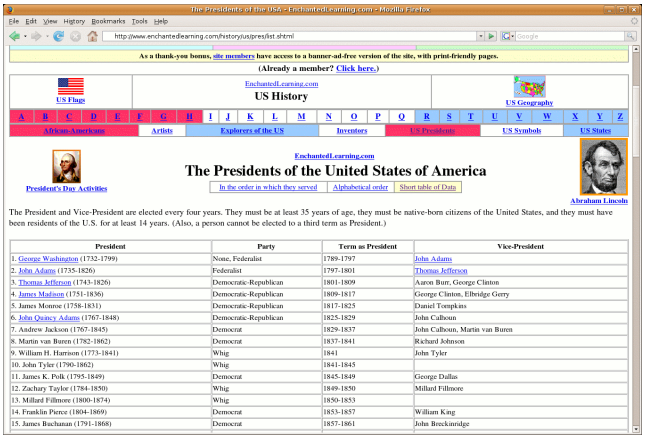

Figure 1: Example HTML Table.

derived tuple belongs to, but relational tables on the Web do not usually offer their relation name.

3. THE RELATIONAL WEB

The WebTables system [8, 9] is designed to extract structured data from the Web that is expressed using the HTML `table` tag. For example, the Web page shown in Figure 1 contains a table that lists American presidents[1]. The table has four columns, each with a domain-specific label and type (e.g., **President** is a person name, **Term as President** is a date range, etc) and there is a tuple of data for each row. Even though much of its metadata is implicit, this Web page essentially contains a small relational database that anyone can crawl.

Of course, not all `table` tags carry relational data. A huge number are used for page layout, for calendars, or other non-relational 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 several thousand raw tables, we estimate that our general Web crawl of 14.1B tables contains about 154M true relational databases, or about 1.1% of the total. While the percentage is fairly small, the vast number of tables on the Web means that the total number of relations is still enormous. Indeed, the relational databases in our crawl form the largest database corpus we know of, by five orders of decimal magnitude.

Unfortunately, distinguishing a *relational* table from a *non-relational* one can be difficult to do automatically. Obtaining a set of good relational tables from a crawl can be considered a form of *open information extraction*, but instead of raw unstructured text the extractor consumes (messy) structured inputs. Our WebTables system uses a combination of hand-written and statistically-trained classifiers to recover the relational tables from the overall set of HTML tables. After running the resulting classifiers on a general Web crawl, WebTables obtains a huge corpus of structured materialized relational databases. These databases are a very useful source of information for the unified Web knowledge base.

Recovering relational databases from the raw HTML tables consists of two steps. First, WebTables attempts to filter out all the non-relational tables. Second, for all the tables that we believe to be relational, WebTables attempts to recover *metadata* for each.

WebTables executes the following steps when filtering out relational tables:

Step 1. Throw out *obviously* non-relational HTML tables, such as those consisting of a single row or a single column. We also remove tables that are used to display calendars or used for HTML form layout. We can detect all of these cases with simple hand-written detectors.

Step 2. Label each remaining tables as *relational* or *non-relational* using a trained statistical classifier. The classifier bases its decision on a set of hand-written table features that seem to indicate a table's type: the number of rows, the number of columns, the number of empty cells, the number of columns with numeric-only data, etc.

Step 1 of this process removes more than 89% of the total table set. The remaining HTML tables are trickier. Traditional schema tests such as constraint checking are not appropriate in a messy Web context, so our test for whether a table is *relational* is necessarily somewhat subjective. Based on a human-judged sample of several thousand examples, we believe a little more than 10% of the remaining tables should be considered relational.

We used a portion of this human-emitted sample to train the classifier in Step 2. That step's output (and hence, the output of the entire filtering pipeline) is an imperfect but still useful corpus of databases. The output retains 81% of the truly relational databases in the input corpus, though only 41% of the output is relational. That means WebTables emits a database of 271M relations, which includes 125M of the raw input's estimated 154M true relations (and, therefore, also includes 146M false ones).

After relational filtering, WebTables tries to recover each relation's metadata. Because we do not recover multi-table databases, and because many traditional database constraints (e.g., key constraints) cannot be expressed using the `table` tag, our target metadata is fairly modest. It simply consists of a set of labels (one for each column) that is found in the table's first row. For example, the table from Figure 1 has metadata that consists of `President`, `Party`, `Term as President`, and `Vice President`.

Because table columns are not explicitly typed, the metadata row can be difficult to distinguish from a table's data contents. To recover the metadata, we used a second trained statistical classifier that takes a table and emits a *hasmetadata*/ *nometadata* decision. It uses features that are meant to "reconstruct" type information for the table: the number of columns that have non-string data in the first row versus in the table's body, and various tests on string-length meant to detect whether the first value in a column is drawn from

[1] `http://www.enchantedlearning.com/history/us/pres/list.shtml`

the same distribution as the body of the column.

A human-marked sample of the relational filtering output indicates that about 71% of all true relations have metadata. Our metadata-recovery classifier performs well: it achieves precision of 89% and recall of 85%.

The materialized relational tables that we obtain from WebTables are both relatively clean and very expressive. First, the tabular structure makes individual elements easy to extract, and hence provides a very rich collection of entities for the Web knowledge base. Other data sources (such as the text collections used in TextRunner) require that we demarcate entities, relations, and values in a sea of surrounding text. In contrast, a good relational table explicitly declares most label endpoints through use of the table structure itself. Further, a relational table carries many pieces of metadata at once: each tuple has a series of dimensions that usually have attribute labels, and the mere presence of tuples in a table indicate set membership. However, unlike a TextRunner tuple, the label of the relation is rarely explicit. Finally, unlike the HTML form interfaces to deep-web databases (described in the next section), but like tuples derived by TextRunner, all of the dimensions in a relational table can be queried directly (forms may allow queries on only a subset of the emitted data's attributes).

Interestingly, in [25] it is shown that tables extracted by WebTables can be used to successfully expand instance sets that were originally extracted from free text using techniques similar to those in TextRunner. We also have initial results indicating that entities in the extracted tables can be used to improve the segmentation of HTML lists – another source for structured data on the Web.

4. ACCESSING DEEP-WEB DATABASES

Not all the structured data on the Web is published in easily-accessible HTML tables. Large volumes of data stored in backend databases are often made available to web users only through HTML form interfaces. For example, US census data can be retrieved by zipcode using the HTML form on the US census website[2]. Users retrieve data by performing valid form submissions. HTML forms either pose structured queries over relational databases or keyword queries over text databases, and the retrieved contents are published in structured templates, e.g., HTML tables, on web pages.

While the tables harvested by WebTables can potentially be reached by users by posing keyword queries on search engines, the contents behind HTML forms were for a long time believed to be beyond the reach of search engines – there are not many hyperlinks pointing to web pages that are results of form submissions and web crawlers did not have the ability to automatically fill out forms. Hence, the names Deep, Hidden, or Invisible Web have been used to collectively refer to the contents accessible only through forms. It has been speculated that the data in the Deep Web far exceeds that currently indexed by search engines [6, 16]. We estimate that there are at least 10 million potentially useful forms [20].

Our primary goal was to make the data in deep-web databases more easily accessible to search engine users. Our approach has been to *surface* the contents into web pages that can then be indexed by search engines (like any other web page). As in the cases of TextRunner and WebTables our goal was to develop techniques that would apply efficiently on large numbers of forms. This is in contrast with much prior work that have either addressed the problem by constructing mediator systems one domain at a time [12, 13, 26], or have needed site-specific wrappers or extractors to extract documents from text databases [5, 22]. As we discuss, the pages we surface contain tables from which additional data can be extracted for the Web knowledge base.

Over the past few years we have developed and deployed a system that has surfaced the contents of over a million of such databases, which span over 50 languages and over 100 domains. The surfaced pages contribute results to over a thousand web search queries per second on Google.com. In the rest of this section, we present an overview of our system. Further details about our system can be found in [21].

4.1 Surfacing Deep-Web databases

There are two complementary approaches to offering access to deep-web databases. The first approach, essentially a data integration solution, is to create vertical search engines for specific domains (e.g., cars, books, real-estate). In this approach we could create a mediator form for the domain at hand and semantic mappings between individual data sources and the mediator form. At web-scale, this approach suffers from several drawbacks. First, the cost of building and maintaining the mediator forms and the mappings is high. Second, it is extremely challenging to identify the domain (and the forms within the domain) that are relevant to a keyword query. Finally, data on the web is about everything and domain boundaries are not clearly definable, not to mention the many different languages – creating a mediated schema of everything will be an epic challenge, if at possible.

The second approach, surfacing, pre-computes the most relevant form submissions for all interesting HTML forms. The URLs resulting from these submissions can then be indexed like any other HTML page. Importantly, this approach leverages the existing search engine infrastructure and hence allows the seamless inclusion of Deep-Web pages into web search results; we thus prefer the surfacing approach.

The primary challenge in developing a surfacing approach lies in pre-computing the set of form submissions for any given form. First, values have to be selected for each input in the form. Value selection is trivial for select menus, but is very challenging for text boxes. Second, forms have multiple inputs and using a simple

[2] http://factfinder.census.gov/

strategy of enumerating all possible form submissions can be very wasteful. For example, the search form on cars.com has 5 inputs and a Cartesian product will yield over 200 million URLs, even though cars.com has only 650,000 cars on sale [11]. We present an overview of how we address these challenges in the rest of this section.

An overarching challenge in developing our solution was to make it scale and be domain independent. As already mentioned, there are millions of potentially useful forms on the Web. Given a particular form, it might be possible for a human expert to determine through laborious analysis the best possible submissions for that form, but such a solution would not scale. Our goal was to find a completely automated solution that can be applied to any form in any language or domain.

Selecting Input Values: A large number of forms have text box inputs and require valid inputs values for any data to be retrieved. Therefore, the system needs to choose a good set of values to submit in order to surface the most useful result pages. Interestingly, we found that it is not necessary to have a complete understanding of the semantics of the form in order to determine good candidate values for text inputs. We note that text inputs fall in one of two categories: generic search inputs that accept most keywords and typed text inputs that only accept values in a particular domain.

For search boxes, we start by making an initial prediction for good candidate keywords by analyzing the text on pages from that the form site that might be already indexed by the search engine. We use the initial set of keywords to bootstrap an iterative probing process. We test the form with candidate keywords and when valid form submissions result, we extract more keywords from the resulting pages. This iterative process continues until either new candidate keywords cannot be extracted or a pre-specified target is reached. The set of all candidate keywords can then be pruned to select a smaller subset that ensures diversity of the exposed database contents. Similar iterative probing approaches have been used in the past to extract text documents from specific databases [5, 10, 18, 22].

For typed text boxes, we attempt to match the type of the text box against a library of types that are extremely common across domains, e.g., zip codes in the US. Note that probing with values of the wrong type results in invalid submissions or pages with no results. We observed that 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 Cartesian product of all possible inputs will result in a very large number of URLs being generated. Crawling too many URLs will drain the resources of a search engine web crawler while also posing an unreasonable load on web servers hosting the HTML forms. Interestingly, when the Cartesian product is very large, it is likely that a large number of the form submissions result in empty result sets that are useless from an indexing standpoint.

To only generate a subset of the Cartesian product, we developed an algorithm that intelligently traverses the search space of possible input combinations to identify only the subset of input combinations that are likely to be useful to the search engine index. We introduced the notion of an input template: given a set of *binding* inputs, the template represents that set of all form submissions using only the Cartesian product of values for the binding inputs. We showed that only input templates that are *informative*, i.e., generate web pages with sufficiently distinct retrieved contents, are useful to a search engine index. Hence, given a form, our algorithm searches for informative templates in the form, and only generates form submissions from them.

Based on the algorithms outlined above, we find that we only generate a few hundred form submissions per form. Furthermore, we believe the number of form submissions we generate is proportional to the size of the database underlying the form site, rather than the number of inputs and input combinations in the form. In [21], we also show that we are able to extract large fractions of underlying deep-web databases automatically (without any human supervision) using only a small number of form submissions.

4.2 Extracting the extracted databases

By creating web pages, surfacing does not preserve the structure and hence the semantics of the data exposed from the underlying deep-web databases. The loss in semantics is also a lost opportunity for query answering. For example, suppose a user were to search for "used ford focus 1993". Suppose there is a surfaced used-car listing page for Honda Civics, which has a 1993 Honda Civic for sale, but with a remark "has better mileage than the Ford Focus". A simple IR index can very well consider such a surfaced web page a good result. Such a scenario can be avoided if the surfaced page had the annotation that the page was for used-car listings of Honda Civics and the search engine were able to exploit such annotations. Hence, the challenge in the future will be to find the right kind of annotation that can be used by the IR-style index most effectively.

When contents from a deep-web database are surfaced onto a web page, they are often laid out into HTML tables. Thus, a side-effect of surfacing is that there are potentially many more HTML tables that can then be recovered by a system like WebTables. In fact, HTML tables generated from the deep-web can potentially be recovered in a more informed way than in general. Surfacing leads to not one, but multiple pages with overlapping data laid out in identical tables. Recovering the tables collectively, rather than each one separately, can potentially lead to a complete extraction of the the deep-web database. In addition, mappings can be predicted from inputs in the form to columns in the recovered tables thereby resulting in recovering more of the semantics of the underlying data. Such deeper extraction and recovery is an area of future work.

In addition, the forms themselves contribute to interesting structured data. The different forms in a do-

main, just like different tables, are alternate representations for metadata within a domain. The forms can be aggregated and analyzed to yield interesting artifacts. Resources such as the ACSDb [9] can be constructed based on the metadata in forms. For example, as works such as [15, 23, 28] have shown, mediated schemas can be constructed for domains by clustering forms belonging to that domain. Likewise, form matching within a domain can be improved by exploiting other forms in the same same domain [19].

5. WEB-SCALE AGGREGATES

So far our discussion has focused on a knowledge base of facts from the Web. However, our experience has shown that significant value can be obtained from analyzing collections of metadata on the Web. Specifically, from the collections we have been working with (forms and HTML tables) we can extract several artifacts, such as: (1) a collection of forms (input names that appear together, values for select menus associated with input names), (2) a collection of several million schemata for tables, i.e., sets of column names that appear together, and (3) a collection of columns, each having values in the same domain (e.g., city names, zipcodes, car makes).

We postulate that we can build from these artifacts a set of *semantic services* that are useful for many *other* tasks. In fact, these are some of the services we would expect to build from a Web knowledge base, but we argue that we do not need a complete Web knowledge base in order to do so. Examples of such services include:

- Given an attribute name (and possibly values for its column or attribute names of surrounding columns) return a set of names that are often used as synonyms. In a sense, such a service is a component of a schema matching system whose goal is to help resolve heterogeneity between disparate schemata. A first version of such a service was described in [9].

- Given a name of an attribute, return a set of values for its column. An example of where such a service can be useful is to automatically fill out forms in order to surface deep-web content.

- Given an entity, return a set of possible properties (i.e, attributes and relationships) that may be associated with the entity. Such a service would be useful for information extraction tasks and for query expansion.

- Given a few attributes in a particular domain, return other attributes that database designers use for that domain (akin to a schema auto-complete). A first version of such a service was described in [9]. Such a service would be of general interest for database developers and in addition would help them choose attribute names that are more common and therefore avoid additional heterogeneity issues later.

6. CONCLUSION

We described three systems that perform information extraction in a domain-independent fashion, and therefore can (and have been) applied to the entire Web. The first system, TEXTRUNNER, extracts binary relationships between entities from arbitrary text and therefore obtains a very wide variety of relationships. TEXTRUNNER exploits the power of redundancy on the Web by basing its extractions on *multiple occurrences* of facts on the Web. The WEBTABLES system targets tabular data on the Web and extracts structured data that typically requires multiple attributes to describe and often includes numerical data that would be cumbersome to describe in text. In WEBTABLES, the fact that we have *multiple rows* in a table can provide further clues about the semantics of the data. Our deep-web crawl extracts an additional type of structured data that is currently stored in databases and available behind forms. The data extracted by the deep-web crawl requires additional effort to be fully structured, but the potential arises from the fact that we have *multiple tables* resulting from the same form. In all three cases, a side result of the extraction is a set of entities, relationships and schemata that can be used as building blocks for the Web knowledge base and for additional semantic services.

7. REFERENCES

[1] E. Agichtein, L. Gravano, J. Pavel, V. Sokolova, and A. Voskoboynik. Snowball: A Prototype System for Extracting Relations from Large Text Collections. In *SIGMOD*, 2001.

[2] M. Banko. Personal Communication, 2008.

[3] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open Information Extraction from the Web. In *IJCAI*, 2007.

[4] M. Banko and O. Etzioni. The Tradeoffs Between Open and Traditional Relational Extraction. In *ACL*, 2008.

[5] L. Barbosa and J. Freire. Siphoning hidden-web data through keyword-based interfaces. In *SBBD*, 2004.

[6] M. K. Bergman. The Deep Web: Surfacing Hidden Value. *Journal of Electronic Publishing*, 2001.

[7] S. Brin. Extracting Patterns and Relations from the World Wide Web. In *WebDB*, 1998.

[8] M. J. Cafarella, A. Halevy, Y. Zhang, D. Z. Wang, and E. Wu. Uncovering the Relational Web. In *WebDB*, 2008.

[9] M. J. Cafarella, A. Halevy, Y. Zhang, D. Z. Wang, and E. Wu. WebTables: Exploring the Power of Tables on the Web. In *VLDB*, 2008.

[10] J. P. Callan and M. E. Connell. Query-based sampling of text databases. *ACM Transactions on Information Systems*, 19(2):97–130, 2001.

[11] Cars.com FAQ. http://siy.cars.com/siy/qsg/faqGeneralInfo.jsp#howmanyads.

[12] Cazoodle Apartment Search. http://apartments.cazoodle.com/.

[13] K. C.-C. Chang, B. He, and Z. Zhang. MetaQuerier over the Deep Web: Shallow Integration across Holistic Sources. In *VLDB-IIWeb*, 2004.

[14] O. Etzioni, M. Cafarella, D. Downey, S. Kwok, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates. Web-scale information extraction in KnowItAll (preliminary results). In *WWW*, 2004.

[15] B. He and K. C.-C. Chang. Statistical Schema Matching across Web Query Interfaces. In *SIGMOD*, 2003.

[16] B. He, M. Patel, Z. Zhang, and K. C.-C. Chang. Accessing the Deep Web: A survey. *Communications of the ACM*, 50(5):95–101, 2007.

[17] M. A. Hearst. Automatic Acquisition of Hyponymns from Large Text Corpora. In *COLING*, 1992.

[18] P. G. Ipeirotis and L. Gravano. Distributed Search over the Hidden Web: Hierarchical Database Sampling and Selection. In *VLDB*, 2002.

[19] J. Madhavan, P. A. Bernstein, A. Doan, and A. Y. Halevy. Corpus-based Schema Matching. In *ICDE*, 2005.

[20] J. Madhavan, S. Jeffery, S. Cohen, X. Dong, D. Ko, C. Yu, and A. Halevy. Web-scale Data Integration: You can only afford to Pay As You Go. In *CIDR*, 2007.

[21] J. Madhavan, D. Ko, L. Kot, V. Ganapathy, A. Rasmussen, and A. Halevy. Google's Deep-Web Crawl. In *VLDB*, 2008.

[22] A. Ntoulas, P. Zerfos, and J. Cho. Downloading Textual Hidden Web Content through Keyword Queries. In *JCDL*, 2005.

[23] A. D. Sarma, X. Dong, and A. Halevy. Bootstrapping pay-as-you-go data integration systems. In *SIGMOD*, 2008.

[24] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: a core of semantic knowledge. In *WWW*, 2007.

[25] P. P. Talukdar, J. Reisinger, M. Pasca, D. Ravichandran, R. Bhagat, and F. Pereira. Weakly Supervised Acquisition of Labeled Class Instances using Graph Random Walks. In *EMNLP*, 2008.

[26] Trulia. http://www.trulia.com/.

[27] F. Wu and D. S. Weld. Autonomously Semantifying Wikipedia. In *CIKM*, 2007.

[28] W. Wu, C. Yu, A. Doan, and W. Meng. An Interactive Clustering-based Approach to Integrating Source Query Interfaces on the Deep Web. In *SIGMOD*, 2004.